5420 Anomaly Detection, Fall 2020

Assignment 4

Submitted by: Harsh Dhanuka, hd2457

Objectives

It is important in any data science project to define the objective as specific as possible. Below let's write it from general to specific. This will direct your analysis.

- Identify hospitals that may abuse the resources or conduct fraud.
- Identify hospitals that may abuse the resources or conduct fraud compared to its peers.
- Identify hospitals that may abuse the resources or conduct fraud compared to the average (mean, median, etc) of its peers.
- Identify hospitals that may abuse the resources or conduct fraud compared to the average (mean, median, etc) of its peers of the same DRG and State.
- Identify hospitals that may abuse the resources or conduct fraud through different clustering techniques - kNN and PCA
- Identify hospitals that may abuse the resources or conduct fraud through different clustering techniques - kNN and PCA, by tuning the hyper paramters, and using the 'Average' Aggregate method to cross validate

Please click Section 5 to directly go to the kNN and PCA analysis.

Table of Contents

Section 1: Initial Steps

- Section 1: Initial Steps
  - Section 1.1: Load Data
  - Section 1.2: Basic Summary Check and considerations

Section 2: Data Cleaning and Preparation

- Section 2: Data Cleaning and Preparation
  - Section 2.1: Rename all column names
  - Section 2.2: Convert amount variables to appropriate numeric format, strip off the dollar symbol
  - Section 2.3: Convert Provider_Id to appropriate object format
  - Section 2.4: Convert Provided_Zip_Code from integer to appropriate Zipcode format
  - Section 2.5: Check NA's
  - Section 2.6: Add a new "states" dataset to match Regions with the exising Provider_State variable
  - Section 2.7: Add Median Income by zipcode dataset to match by zipcode in original dataset

Section 3: EDA of all variables

- Section 3: EDA
  - Section 3.1: Showing the Distribution of X
    - Section 3.1.1: DRG Definition Distribution .....
    - Section 3.1.7: Total Discharges Distribution
  - Section 3.2: Showing the Distribution of Y by another Categorical Variable X
    - Section 3.2.1: Average_Total_Payments by DRG_Definition .....
    - Section 3.2.5: Total Discharges by Region
  - Section 3.3: Showing interaction of two or three variables
    - Section 3.3.1: interaction between Average_Total_Payments and Average_Medicare_Payments .....
    - Section 3.3.7: Region wise-distribution of the three numerical variables

I have commented this EDA section as detailed EDA and all the graphs was already shown in previous 2 submissions.

Section 4: Feature Engineering

- Section 4: Feature engineering, and explain a potential fraud case using Median Scores
  - Section 4.1: Popular DSG_Definitions's by state .....
  - Section 4.17: Ratio of Total Discharges to Zip_Population

Section 5: kNN, PCA

- Section 5: Modeling
  - Section 5.1: kNN
  - Section 5.2: PCA

Understanding the two methods:

kNN Method

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method proposed by Thomas Cover used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

PCA Method

The principal components of a collection of points in a real p-space are a sequence of p direction vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first i−1 vectors. Here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line. These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The i-th principal component can be taken as a direction orthogonal to the first i-1 principal components that maximizes the variance of the projected data.

From either objective, it can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset. Robust and L1-norm-based variants of standard PCA have also been proposed.

1. Initial Steps

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import scipy
import time
import seaborn as sns
sns.set(style="whitegrid")
import warnings
warnings.filterwarnings("ignore")
import missingno as msno

from sklearn.impute import SimpleImputer
from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

from pprint import pprint
import zipcodes

import plotly
import plotly.express as px

from pyod.models.knn import KNN
from pyod.models.combination import aom, moa, average, maximization
from pyod.utils.utility import standardizer
from pyod.utils.data import generate_data, get_outliers_inliers
from pyod.models.pca import PCA
from pyod.utils.data import evaluate_print
from pyod.utils.example import visualize
```

1.1. Load Data

In [2]:
```
# Read the data

df = pd.read_csv('/Users/harshdhanuka/Desktop/Columbia Class Matter/SEM 3/5420 Anomaly Detection/Assignment 6 PyOD/inp
atientCharges.csv')
df.head()
```

Out[2]:

	DRG Definition	Provider Id	Provider Name	Provider Street Address	Provider City	Provider State	Provider Zip Code	Hospital Referral Region Description	Total Discharges	Average Covered Charges	Average Total Payments	Average Medicare Payments
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan	91	$32963.07	$5777.24	$4763.73
1	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10005	MARSHALL MEDICAL CENTER SOUTH	2505 U S HIGHWAY 431 NORTH	BOAZ	AL	35957	AL - Birmingham	14	$15131.85	$5787.57	$4976.71
2	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10006	ELIZA COFFEE MEMORIAL HOSPITAL	205 MARENGO STREET	FLORENCE	AL	35631	AL - Birmingham	24	$37560.37	$5434.95	$4453.79
3	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10011	ST VINCENT'S EAST	50 MEDICAL PARK EAST DRIVE	BIRMINGHAM	AL	35235	AL - Birmingham	25	$13998.28	$5417.56	$4129.16
4	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10016	SHELBY BAPTIST MEDICAL CENTER	1000 FIRST STREET NORTH	ALABASTER	AL	35007	AL - Birmingham	18	$31633.27	$5658.33	$4851.44

1.2. Basic Summary Check

In [3]:
```
# Rows and  Columns

print(" ")
print("Number of rows and columns in the dataset:")
df.shape
```

```
Number of rows and columns in the dataset:
```

Out[3]:
```
(163065, 12)
```

In [4]:
```
print(" ")
print("Basic statistics of the numerical columns are as follows:")

# Check basic statistics
df.describe()
```

```
Basic statistics of the numerical columns are as follows:
```

Out[4]:

	Provider Id	Provider Zip Code	Total Discharges
count	163065.000000	163065.000000	163065.000000

mean	255569.865428	47938.121908	42.776304
std	151563.671767	27854.323080	51.104042
min	10001.000000	1040.000000	11.000000
25%	110092.000000	27261.000000	17.000000
50%	250007.000000	44309.000000	27.000000
75%	380075.000000	72901.000000	49.000000
max	670077.000000	99835.000000	3383.000000

In [5]:
```
# Check the column dtypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 163065 entries, 0 to 163064
Data columns (total 12 columns):
DRG Definition                         163065 non-null object
Provider Id                            163065 non-null int64
Provider Name                          163065 non-null object
Provider Street Address                163065 non-null object
Provider City                          163065 non-null object
Provider State                         163065 non-null object
Provider Zip Code                      163065 non-null int64
Hospital Referral Region Description   163065 non-null object
 Total Discharges                      163065 non-null int64
 Average Covered Charges               163065 non-null object
 Average Total Payments                163065 non-null object
Average Medicare Payments              163065 non-null object
dtypes: int64(3), object(9)
memory usage: 14.9+ MB
```

Considerations from eyeballing the data:

1. Check for categorical/object fields from the data variable descriptions. Convert the relevant numeric fields to their respective categorical/object fields: `Provider Id`
2. The `Zipcode` column is represented as an integer. Convert it to zipcode format.
3. Variable `Hospital Referral Region Description` comprises of the State and the city, which I see is the nearest metro city.
4. `Average Covered Charges` is not significant for our analysis, it will be for other purposes such claims fraud, insurance premiums, etc.
5. The two payments columns need to be converted to proper numeric formats.

In [6]:
```
# Basic Sort of Provider ID and DRG Definition

df = df.sort_values(['Provider Id', 'DRG Definition'])
df.head(2)
```

Out[6]:

	DRG Definition	Provider Id	Provider Name	Provider Street Address	Provider City	Provider State	Provider Zip Code	Hospital Referral Region Description	Total Discharges	Average Covered Charges	Average Total Payments	Average Medicare Payments
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan	91	$32963.07	$5777.24	$4763.73
1079	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan	38	$20312.78	$4894.76	$3865.50

2. Data Cleaning and Preparation

2.1. Rename all column names

The given column names have a lot of spaces, trailing spaces, etc. I will rename all the columns as per appropriate naming convention.

In [7]:
```
df.columns = ['DRG_Definition', 'Provider_Id', 'Provider_Name',
       'Provider_Street_Address', 'Provider_City', 'Provider_State',
       'Provider_Zip_Code', 'Hospital_Referral_Region_Description',
       'Total_Discharges', 'Average_Covered_Charges',
       'Average_Total_Payments', 'Average_Medicare_Payments']
df.columns
```

Out[7]:
```
Index(['DRG_Definition', 'Provider_Id', 'Provider_Name',
       'Provider_Street_Address', 'Provider_City', 'Provider_State',
       'Provider_Zip_Code', 'Hospital_Referral_Region_Description',
       'Total_Discharges', 'Average_Covered_Charges', 'Average_Total_Payments',
       'Average_Medicare_Payments'],
      dtype='object')
```

2.2. Convert amount variables to appropriate numeric format, strip off the dollar symbol

In [8]:
```
df["Average_Total_Payments"] = df["Average_Total_Payments"].str[1:].astype(float)
df["Average_Medicare_Payments"] = df["Average_Medicare_Payments"].str[1:].astype(float)
df["Average_Covered_Charges"] = df["Average_Covered_Charges"].str[1:].astype(float)
```

2.3. Convert Provider_Id to appropriate object format

In [9]:
```
df["Provider_Id"] = df["Provider_Id"].astype(object)
```

In [10]:
```
df.head(2)
```

Out[10]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Descriptio
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dotha
1079	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dotha

2.4. Convert `Provided_Zip_Code` from integer to appropriate Zipcode format

In [11]:
```
# Zipcode to 5 character integer zipcode format

df['Provider_Zip_Code'] = df['Provider_Zip_Code'].astype(str).str.zfill(5)
```

2.5. Check NA's

In [12]:
```
df.isnull().sum().sum()
```

Out[12]: 0

There are no NA's, which is good for our analysis.

2.6. Add a new "states" dataset to match 'Regions' with the exising `Provider_State` variable.

Regions will be a a very useful feature when performing the Exploratory Data Analysis.

In [13]:
```
states = pd.read_csv('/Users/harshdhanuka/Desktop/Columbia Class Matter/SEM 3/5420 Anomaly Detection/Assignment 6 PyOD
/states.csv')
states.head(5)
```

Out[13]:

	State	State Code	Region	Division
0	Alaska	AK	West	Pacific
1	Alabama	AL	South	East South Central
2	Arkansas	AR	South	West South Central
3	Arizona	AZ	West	Mountain
4	California	CA	West	Pacific

In [14]:
```
# Left join the new dataset

df = pd.merge(left = df, right = states, left_on = 'Provider_State', right_on = 'State Code', how = 'left')
df.head(2)
```

Out[14]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

In [15]:
```
# Remove duplicate 'state' column

df = df.drop(columns = ['State', 'State Code'])
```

2.7. Add 'Median Income' by zipcode dataset to match by zipcode in original dataset

Dataset source: https://www.psc.isr.umich.edu/dis/census/Features/tract2zip/

This has the zipcode wise mean and median income data for 2006 to 2010

In [16]:
```
income_df = pd.read_excel('/Users/harshdhanuka/Desktop/Columbia Class Matter/SEM 3/5420 Anomaly Detection/Assignment 6
PyOD/MedianZIP-3.xlsx')
income_df['Zip'] = income_df['Zip'].astype(str).str.zfill(5)
income_df.head(3)
```

Out[16]:

	Zip	Median	Mean	Pop
0	01001	56662.5735	66687.8	16445
1	01002	49853.4177	75062.6	28069
2	01003	28462.0000	35121	8491

In [17]:
```
income_df.isnull().sum()
```

Out[17]:
```
Zip        0
Median     0
Mean       0
Pop        0
dtype: int64
```

In [18]:
```
df = pd.merge(left = df, right = income_df, left_on = 'Provider_Zip_Code', right_on = 'Zip', how = 'left')
df = df.drop(columns = ['Zip','Mean'])

df.rename(columns={'Median':'Zip_Median_Income',
                   'Pop':'Zip_Population'}, inplace=True)

df.head(2)
```

Out[18]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

.

---------- SECTION BREAK ----------

.

3. EDA

3.1. Showing the Distribution of X

3.1.1. `DRG_Definition` Distribution

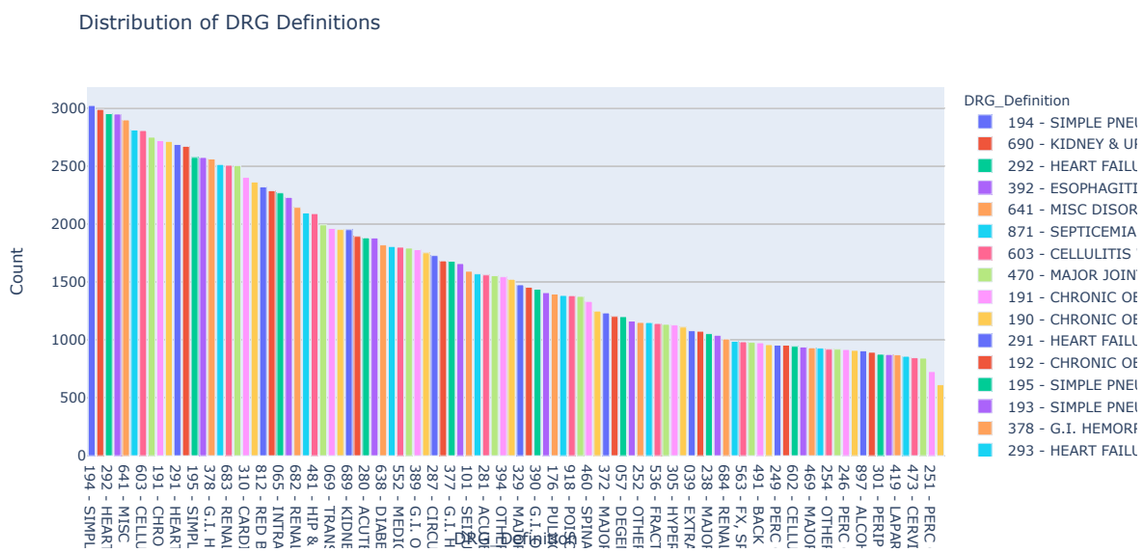
Explore the total number of DRG Definitions, and the count of how many times they appear.

In [19]:
```
df_count = df['DRG_Definition'].value_counts()
df_count = pd.DataFrame(df_count).reset_index()
df_count.columns = ['DRG_Definition','Count']
df_count.head()
```

Out[19]:

	DRG_Definition	Count
0	194 - SIMPLE PNEUMONIA & PLEURISY W CC	3023
1	690 - KIDNEY & URINARY TRACT INFECTIONS W/O MCC	2989
2	292 - HEART FAILURE & SHOCK W CC	2953
3	392 - ESOPHAGITIS, GASTROENT & MISC DIGEST DIS...	2950
4	641 - MISC DISORDERS OF NUTRITION,METABOLISM,F...	2899

In [20]:
```
fig = px.bar(df_count, x = 'DRG_Definition', y = 'Count', color = 'DRG_Definition',
             width=1450, height=500,
            title = "Distribution of DRG Definitions")
fig.show()
```



Observation:

The DRG Definition has a seemingly good distribution. The counts of DRG Definitons range from around 3000 to 600. All other DRG Definition counts lie within this range.

Here, any DRG Definition count doesnt seem like an outlier and all behave normally.

3.1.2. `Provider_Name` Distribution, see popular Providers

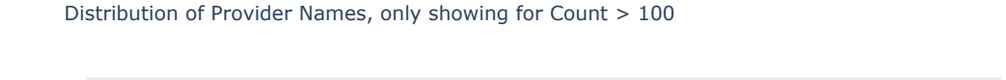
Explore the total number Provider Names and the count of how many times each one appears.

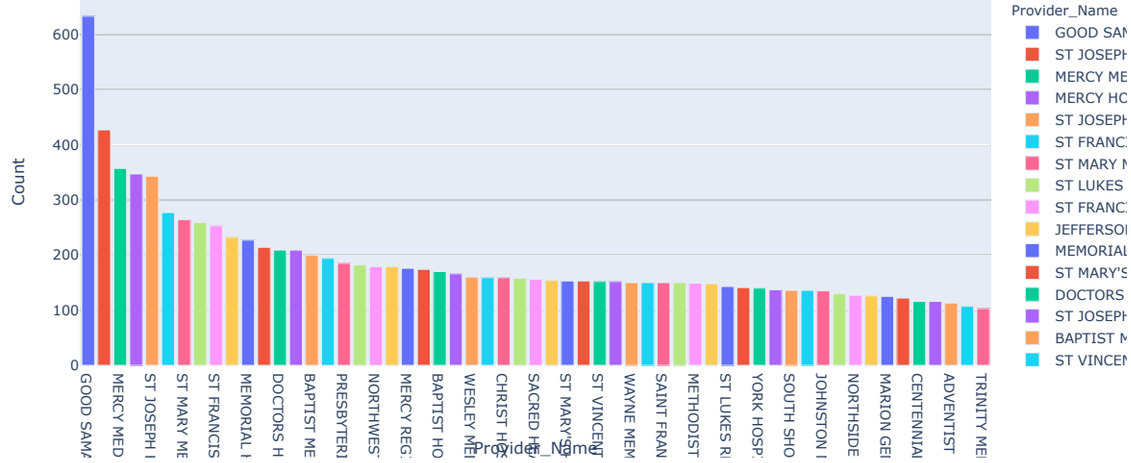
In [21]:
```
df_count = df['Provider_Name'].value_counts()
df_count = pd.DataFrame(df_count).reset_index()
df_count.columns = ['Provider_Name','Count']
df_count.head()
```

Out[21]:

	Provider_Name	Count
0	GOOD SAMARITAN HOSPITAL	633
1	ST JOSEPH MEDICAL CENTER	427
2	MERCY MEDICAL CENTER	357
3	MERCY HOSPITAL	347
4	ST JOSEPH HOSPITAL	343

In [22]:
```
# Show only those Provider_Names whose total count is above 100

df_count1 = df_count.loc[df_count['Count'] > 100]
fig = px.bar(df_count1, x='Provider_Name', y='Count',
             width=1200, height=500,
             color = 'Provider_Name',
            title = "Distribution of Provider Names, only showing for Count > 100")
fig.show()
```





In [23]:
```
# Show only those Provider_Names whose total count is below 3

df_count1 = df_count.loc[df_count['Count'] < 3]
fig = px.bar(df_count1, x='Provider_Name', y='Count',
             width=1200, height=500,
            color = 'Provider_Name',
            title = "Distribution of Provider Names, only showing for Count < 3")
# fig.show()
```

Observation:

From the above two count charts, it is clear than some Providers are extremely popular, and have around 600 entries. They seem to be the ones who provide services under multiple DRG Definitions.

While, some Providers are very unpopular, and have only 1 entry. Now, this depends on the DRG Definition, as some hospitals be a single specialty hospital, and hence everyone goes there only.

3.1.3. `Provider_City` Distribution, see popular Cities

Explore the total number Cities and the count of how many times each one appears.

In [24]:
```
df_count = df['Provider_City'].value_counts()
df_count = pd.DataFrame(df_count).reset_index()
df_count.columns = ['Provider_City','Count']
df_count.head()
```

Out[24]:

	Provider_City	Count
0	CHICAGO	1505
1	BALTIMORE	1059
2	HOUSTON	950
3	PHILADELPHIA	898
4	BROOKLYN	877

In [25]:
```
# Show only those Provider_Cities whose total count is above 500

df_count1 = df_count.loc[df_count['Count'] > 500]
fig = px.bar(df_count1, x='Provider_City', y='Count',width=1000, height=500,
            color = 'Provider_City',
            title = "Distribution of Provider Cities, only showing for Count >500")
# fig.show()
```

In [26]:
```
# Show only those Provider_Cities whose total count is below 5

df_count1 = df_count.loc[df_count['Count'] < 5]
fig = px.bar(df_count1, x='Provider_City', y='Count',width=1000, height=500,
            color = 'Provider_City',
            title = "Distribution of Provider Cities, only showing for Count > 5")
# fig.show()
```

Observation:

Chicago is the most popular city with around 1500 entries. There are also a lot of other cities which have only 1 entry.

3.1.4. `Provider_State` Distribution, see popular States

Explore the total number States and the count of how many times each one appears.

In [27]:
```
df_count = df['Provider_State'].value_counts()
df_count = pd.DataFrame(df_count).reset_index()
df_count.columns = ['Provider_State','Count']
df_count.head()
```

Out[27]:

	Provider_State	Count
0	CA	13064
1	TX	11864
2	FL	11155
3	NY	9178
4	IL	7909

In [28]:
```
fig = px.bar(df_count, x='Provider_State', y='Count',
             width=1000, height=500,
             color = 'Provider_State',
            title = "Distribution of Provider State")
# fig.show()
```

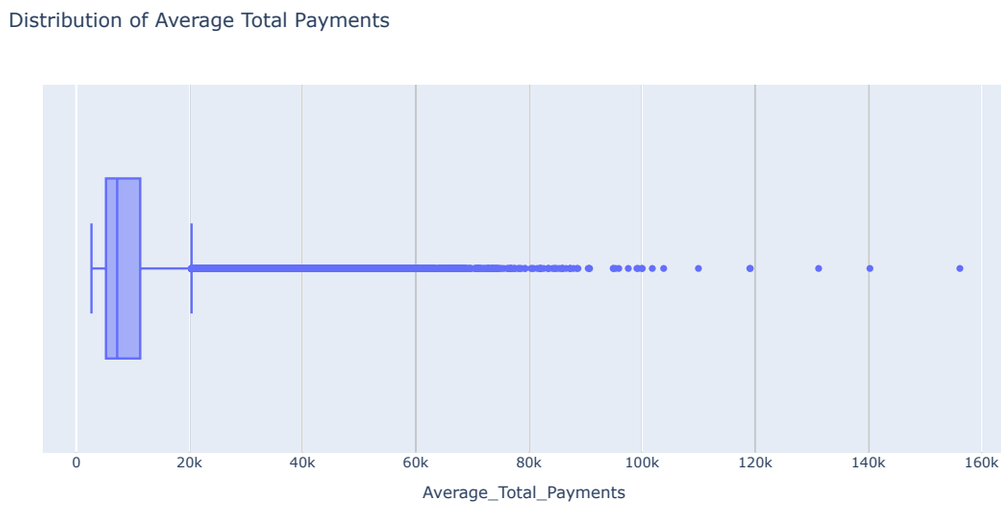
Observation:

The states seem to have a good distribution. There seems to be no outliers or staes requiring special attention.

3.1.5. `Average_Total_Payments` Distribution

In [29]:
```
fig = px.histogram(df, x="Average_Total_Payments",
            width=1000, height=500,
            title = "Distribution of Average Total Payments")
# fig.show()
```

In [30]:
```
fig = px.box(df, x="Average_Total_Payments",width=1000, height=500,
            title = "Distribution of Average Total Payments")
fig.show()
```



In [31]:
```
fig = px.violin(df, y="Average_Total_Payments", box=True,
                points='all',width=800, height=500,
               title = "Distribution of Average Total Payments")
# fig.show()
```

Observation:

Most of the Average payments are less than USD 11,000. So, any average payment above that might be a reason for futher investigation.

There are some extreme high values, more than USD 150,000 which may need investigation.

- Median: 7214.1

3.1.6. `Average_Medicare_Payments` Distribution

In [32]:
```
fig = px.histogram(df, x="Average_Medicare_Payments",
            width=1000, height=500,
                  title = "Distribution of Average Medicare Payments")
# fig.show()
```

In [33]:
```
fig = px.box(df, x="Average_Medicare_Payments",width=1000, height=500,
            title = "Distribution of Average Medicare Payments")
# fig.show()
```

Observation:

Most of the Average Medicare payments are less than USD 10,000. So, any average payment above that might be a reason for futher investigation.

There are some extreme high values, more than USD 150,000 which may need investigation.

- Median: 6158.46

3.1.7. `Total_Discharges` Distruibution

In [34]:
```
fig = px.histogram(df, x="Total_Discharges",
            width=800, height=500,
                  title = "Distribution of Total Discharges")
# fig.show()
```

In [35]:
```
fig = px.box(df, x="Total_Discharges",width=1000, height=500,
            title = "Distribution of Total Discharges")
# fig.show()
```

Observation:

Most of the Total Discharges are less than 49.

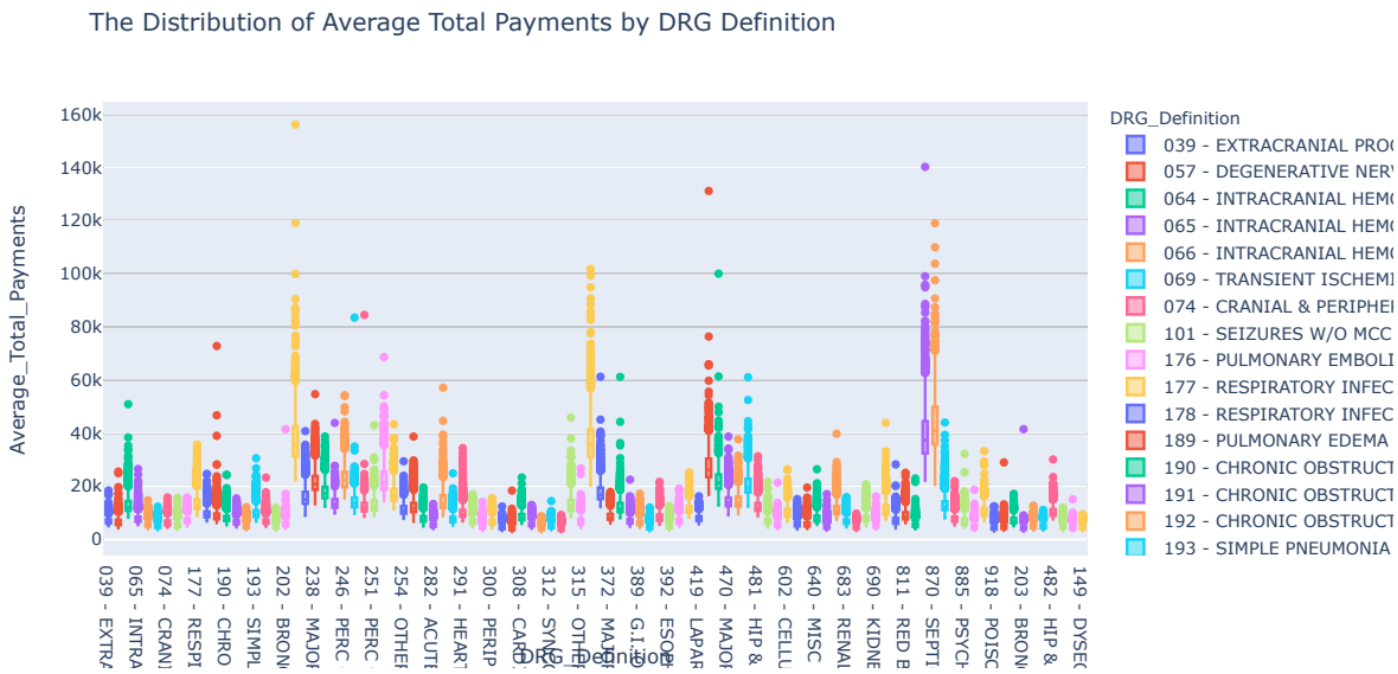
There are some extreme high values, such as 3383, which may need investigation.

- Median: 27

3.2. Showing the Distribution of Y by another Categorical Variable X

3.2.1. `Average_Total_Payments` by `DRG_Definition`

In [36]:
```
fig = px.box(df, x="DRG_Definition", y="Average_Total_Payments",width=1400, height=500,
             color = "DRG_Definition",
            title = "The Distribution of Average Total Payments by DRG Definition")
fig.show()
```



Observation:

Some DRG's have a very high Average Total Payments, these may be critical operations, which bear high cost.

3.2.2. `Average_Total_Payments` by `State`

In [37]:
```
fig = px.box(df, x="Provider_State", y="Average_Total_Payments",width=1000, height=500,
             color = "Provider_State",
            title = "The Distribution of Average Total Payments by Provider State")
# fig.show()
```

Observation:

The Average Total Payments are more or less similar, but some states such as NY and CA are very expensiove overall.

3.2.3. `Average_Total_Payments` by `Region`

In [38]:
```
fig = px.box(df, x="Region", y="Average_Total_Payments",width=1000, height=500,
             color = "Region",
            title = "The Distribution of Average Total Payments by Region")
# fig.show()
```

In [39]:
```
# px.violin(df,x='Average_Total_Payments', y = "Region", color='Region',
#           title = "The Distribution of Average Total Payments by Region",
#           orientation='h').update_traces(side='positive',width=2)
```

Observation:

The West region seems to be generally high in terms of Total Average Payments. This was verified earlier as we saw the state CA was extremely high as well.

It is followed by Northeast, which includes the state NY.

3.2.4. `Total_Discharges` by `DRG_Definition`

In [40]:
```
fig = px.box(df, x="DRG_Definition", y="Total_Discharges",width=1400, height=500,
             color = "DRG_Definition",
            title = "The Distribution of Total Discharges by DRG Definition")
# fig.show()
```

Observation:

The Discharge rate for some DRG's is very high, while most others have a balanced discharged rate.

3.2.5. `Total_Discharges` by `Region`

In [41]:
```
fig = px.box(df, x="Region", y="Total_Discharges",width=1000, height=500,
             color = "Region",
            title = "The Distribution of Total Discharges by Region")
# fig.show()
```

In [42]:
```
# px.violin(df,x='Total_Discharges', y = "Region", color='Region',
#           title = "The Distribution of Total Discharges by Region",
#           orientation='h').update_traces(side='positive',width=2)
```

Observation:

Most regions have a similar total discharged pattern. However, the Northeast region has an outlier.

3.3. Showing interaction of two or three variables

3.3.1. See interaction between `Average_Total_Payments` and `Average_Medicare_Payments`

In [43]:
```
fig = px.scatter(df, x="Average_Total_Payments", y="Average_Medicare_Payments",
                 size = "Average_Total_Payments", color = 'Average_Total_Payments',
                size_max=60,width=800, height=600)
# fig.show()
```

Observation:

As the average total payments increase, the average medicare payments also increase, which shows that there is a very high collinearity between these two variables.

3.3.2. Understand features such as mean, median, min, max, etc of `Average_Total_Payments`

I will group the entire data by `DRG_Definition` and then calculate the statistics for each group overall.

In [44]:
```
agg_columns = ['mean', 'median', 'var', 'std', 'count', 'min', 'max']
groupby_drg = df[['DRG_Definition', 'Average_Total_Payments']].groupby(by='DRG_Definition').agg(agg_columns)

groupby_drg.columns = [header + '-' + agg_column
                       for header, agg_column in zip(groupby_drg.columns.get_level_values(0), agg_columns)]

groupby_drg.columns = groupby_drg.columns.get_level_values(0)
```

In [45]:
```
groupby_drg.reset_index(inplace=True)
groupby_drg['Average_Total_Payments-range'] = groupby_drg['Average_Total_Payments-max'] - groupby_drg['Average_Total_P
ayments-min']

groupby_drg.head(2)
```

Out[45]:

DRG_Definition	Average_Total_Payments-mean	Average_Total_Payments-median	Average_Total_Payments-var	Average_Total_Payments-std	Average_Total_Payments-count	Average_To

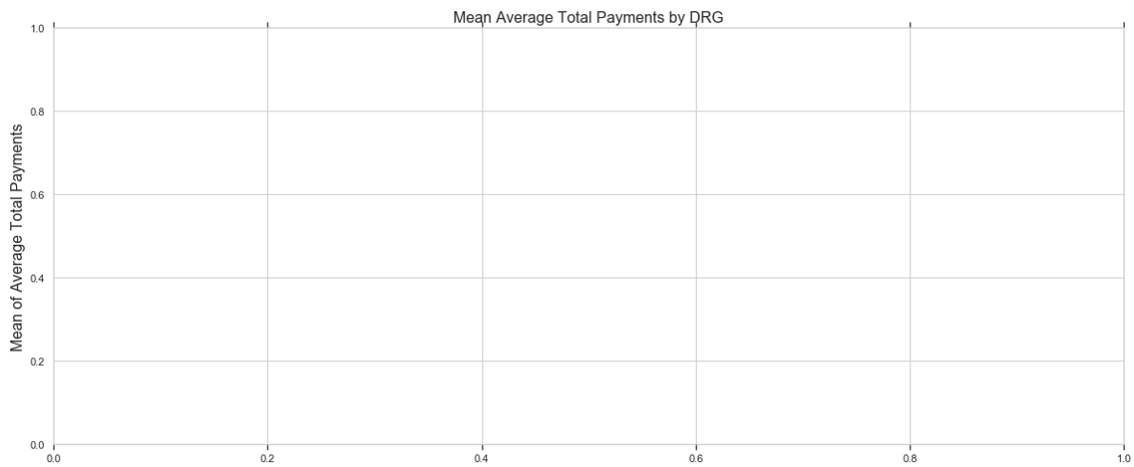
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	6960.534004	6582.89	2.184111e+06	1477.873952	1079
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	6706.276445	6093.75	4.137017e+06	2033.965862	1201

In [46]:
```
def plt_setup(_plt):
    _plt.tick_params(
    axis='x',          # changes apply to the x-axis
    which='both',      # both major and minor ticks are affected
    bottom='off',      # ticks along the bottom edge are off
    top='off',         # ticks along the top edge are off
    labelbottom='off')
```

See the 'mean' of the respective DRG_Definition groups

In [47]:
```
plt.figure(figsize=(20,8))
# sns.barplot(x='DRG_Definition', y='Average_Total_Payments-mean',
#            data=groupby_drg.sort_values('Average_Total_Payments-mean'))
plt_setup(plt)
plt.title('Mean Average Total Payments by DRG', fontsize=16)
plt.ylabel('Mean of Average Total Payments', fontsize=16)
```

Out[47]:
```
Text(0, 0.5, 'Mean of Average Total Payments')
```



Observation:

Some DRG groups have very high mean, which implies that there are some DRG groups which generally charge a very high amount for treatment in terms of 'Total Payments'.

3.3.3. Sum of Average Total Payments by DRG_Definition

In [48]:
```
pyt_by_drg = df.groupby('DRG_Definition').sum().reset_index()
```

In [49]:
```
pyt_by_drg = pyt_by_drg.sort_values('Average_Total_Payments', ascending=False)
# pyt_by_drg.head()
```

In [50]:
```
# Extract only rows with amount > 40,000,000

pyt_by_drg = pyt_by_drg.loc[pyt_by_drg['Average_Total_Payments'] > 40000000]
```

In [51]:
```
# plt.figure(figsize=(20,4))
# fig = sns.barplot(x='DRG_Definition', y='Average_Total_Payments',
#             data=pyt_by_drg)
# fig.set_xticklabels(fig.get_xticklabels(), rotation=15)

# plt.title('Mean Average Total Payments by DRG, for total > 40,000,000', fontsize=16)
# plt.ylabel('Mean of Average Total Payments', fontsize=16)
```

Observation:

The DRG 329 is the highest in terms of total sum fom the Average Total Payments.

3.3.4. Unique ids, unique names, and unique cities for Providers

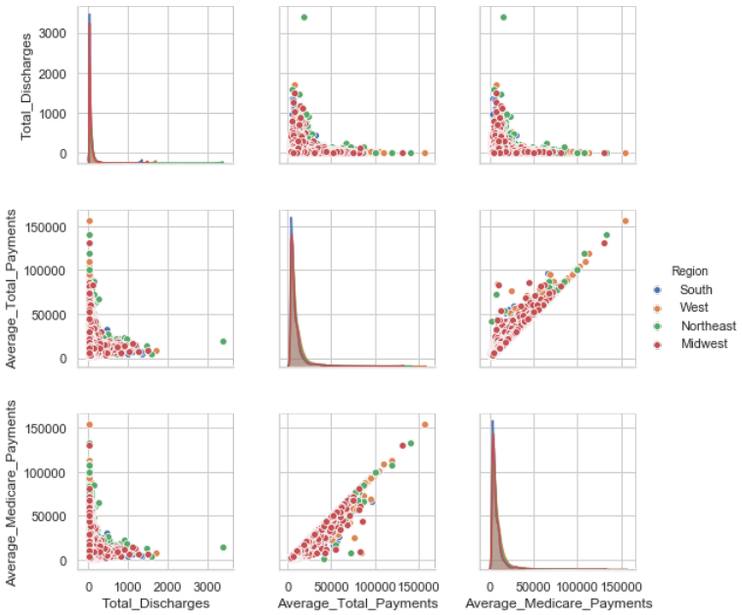
In [52]:
```
unique_ids = len(df.groupby('Provider_Id').count())
unique_providers = len(df.groupby('Provider_Name').count())
unique_cities = len(df.groupby('Provider_City').count())
unique_states = len(df.groupby('Provider_State').count())

print(" ")
print(f'There are {unique_ids} unique provider id values in the data, and {unique_providers} unique provider names in
a total of {unique_cities} unique cities, and {unique_states} states.')
print(" ")
```

```
There are 3337 unique provider id values in the data, and 3201 unique provider names in a total of 1977 unique cities
, and 51 states.
```

3.3.5. Check correlations between the three numerical/integer variables, region wise

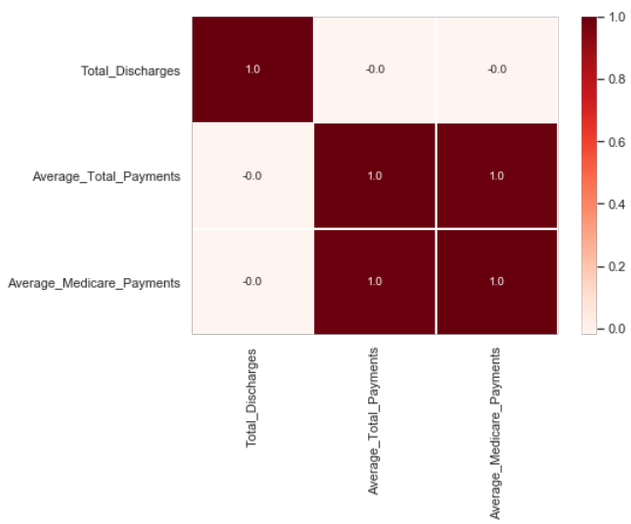
In [53]:
```
fig = sns.pairplot(df[['Region', 'Total_Discharges', 'Average_Total_Payments','Average_Medicare_Payments']],
             hue= 'Region')
fig
```

Out[53]: <seaborn.axisgrid.PairGrid at 0x7fac46bc5450>



In [54]:
```
corr = df[['Total_Discharges', 'Average_Total_Payments', 'Average_Medicare_Payments']].corr()
f,ax = plt.subplots(figsize=(7,5))
sns.heatmap(corr, annot=True, cmap='Reds', linewidths=.4, fmt= '.1f',ax=ax)
# plt.show()
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac434001d0>



Observation:

From above graphs, there are some variables that are highly correlated such as Average Total Payment and Average Medicare Payment. Average total payment has a long tail distribution, which could indicate potential fraud.

From corr matrix: Total payment is correlated with medicare payment.

We can conclude that those variables are indeed related, for modeling purposes, it more make sense to include only one or two of the three variables.

3.3.6. State wise-distribution of the three numerical variables

In [55]:
```
plt.figure(figsize=(20,20))
g = sns.PairGrid(df,
                 x_vars = ['Total_Discharges', 'Average_Total_Payments', 'Average_Medicare_Payments'],
                 y_vars = ['Provider_State'],
                 height=10, aspect=.25)

# Draw plot
g.map(sns.stripplot, size=10, orient="h",
      palette="ch:s=1,r=-.1,h=1_r", linewidth=1.10, edgecolor="w")

# Use the same x axis limits on all columns and add better labels
g.set(xlabel="", ylabel="")

titles = ["Total Discharges", "Average Total Payments",
          "Average Medicare Paymens"]

for ax, title in zip(g.axes.flat, titles):

    # Set a different title for each axes
    ax.set(title = title)

    # Make the grid horizontal instead of vertical
    ax.xaxis.grid(False)
    ax.yaxis.grid(True)

# plt.tight_layout()
# plt.show()
```

<Figure size 1440x1440 with 0 Axes>

3.3.7. Region wise-distribution of the three numerical variables

In [56]:
```
#plt.figure(figsize=(20,20))
#g = sns.PairGrid(df,
#                  x_vars = ['Total_Discharges', 'Average_Total_Payments', 'Average_Medicare_Payments'],
#                  y_vars = ['Region'],
#                  height=10, aspect=.25)

# Draw plot
#g.map(sns.stripplot, size=10, orient="h",
#      palette="ch:s=1,r=-.1,h=1_r", linewidth=1.10, edgecolor="w")

# Use the same x axis limits on all columns and add better labels
#g.set(xlabel="", ylabel="")

#titles = ["Total Discharges", "Average Total Payments",
#          "Average Medicare Paymens"]

#for ax, title in zip(g.axes.flat, titles):

    # Set a different title for each axes
#    ax.set(title = title)

    # Make the grid horizontal instead of vertical
#    ax.xaxis.grid(False)
#    ax.yaxis.grid(True)

# plt.tight_layout()
# plt.show()
```

.

---------- SECTION BREAK ----------

.

4. Feature Engineering

`Feature 1`

4.1. Popular DSG_Definitions's by state

By understanding the top DSG's per state, we esablish a baseline for what people in the state normally get treated for. This information is useful in one of two ways:

- Can be used to flag non-common and expensive procedures
- Focus fraud detection on the common procedures.

Assuming that the fraudulent users would try to get treated for the same conditions as the population.

In [57]:
```
drg_by_state = df.groupby(['Provider_State', 'DRG_Definition']).agg({'DRG_Definition': 'count'})
drg_by_state.head()
```

Out[57]:

		DRG_Definition
Provider_State	**DRG_Definition**	
	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	1
	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/O MCC	1
AK	064 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W MCC	2
	065 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W CC	6

066 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W/O CC/MCC	4

In [58]:
```
drg_by_state.tail()
```

Out[58]:

Provider_State	DRG_Definition	DRG_Definition
	872 - SEPTICEMIA OR SEVERE SEPSIS W/O MV 96+ HOURS W/O MCC	2
	897 - ALCOHOL/DRUG ABUSE OR DEPENDENCE W/O REHABILITATION THERAPY W/O MCC	1
WY	917 - POISONING & TOXIC EFFECTS OF DRUGS W MCC	1
	918 - POISONING & TOXIC EFFECTS OF DRUGS W/O MCC	2
	948 - SIGNS & SYMPTOMS W/O MCC	3

Note:

I am not merging this with the original dataset as this is a new data table created to be used as a reference to check the most common DRG's by state

`Feature 2`

4.2. Number of Provider_Names per city

This information can be used in one of two ways:

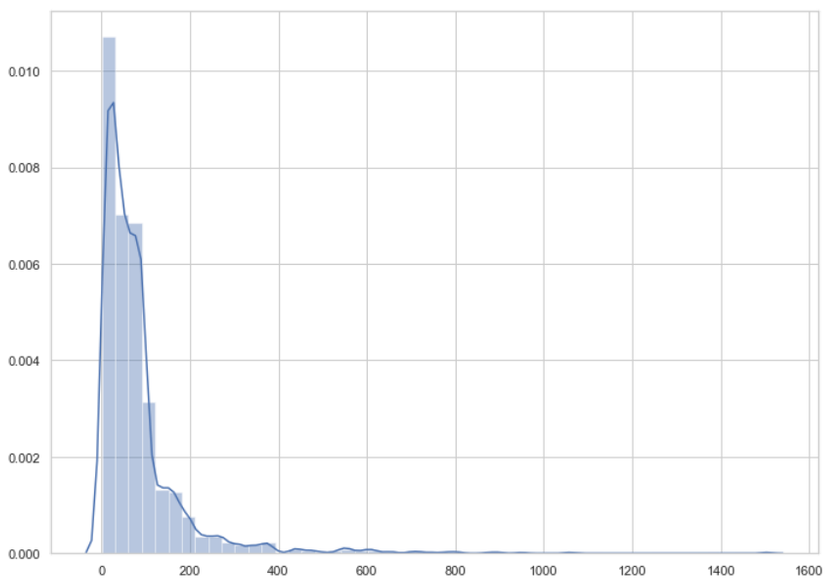
- Assuming that the city with the most providers have a higher probablity of being a victim fraud. It allows to focus on the cities with the highest concentration.
- Assuming the cities with the lowest provider per population densitity are a higher risk. The reason is that fraudsters have less options to cheat, and they are forced to choose from a select few.

In [59]:
```
providers_per_city = df.groupby(['Provider_City']).agg({'Provider_Name':'count'})
providers_per_city.head()
```

Out[59]:

Provider_City	Provider_Name
ABBEVILLE	18
ABERDEEN	107
ABILENE	152
ABINGDON	63
ABINGTON	99

In [60]:
```
fig = plt.figure(figsize=(10,7))
providers_per_city.sort_values(by = 'Provider_Name', ascending = False).head()
sns.distplot(providers_per_city)
plt.tight_layout()
```



Note:

I am not merging this with the original dataset as this is a new data table created to be used as a reference to check the most common hospitals.

`Feature 3`

4.3. Average cost per procedure/treatment

Assuming that people/hospitals would likely cheat on the most expensive procedures, this feature would allow us to focus on the ones that would have the highest likelihood.

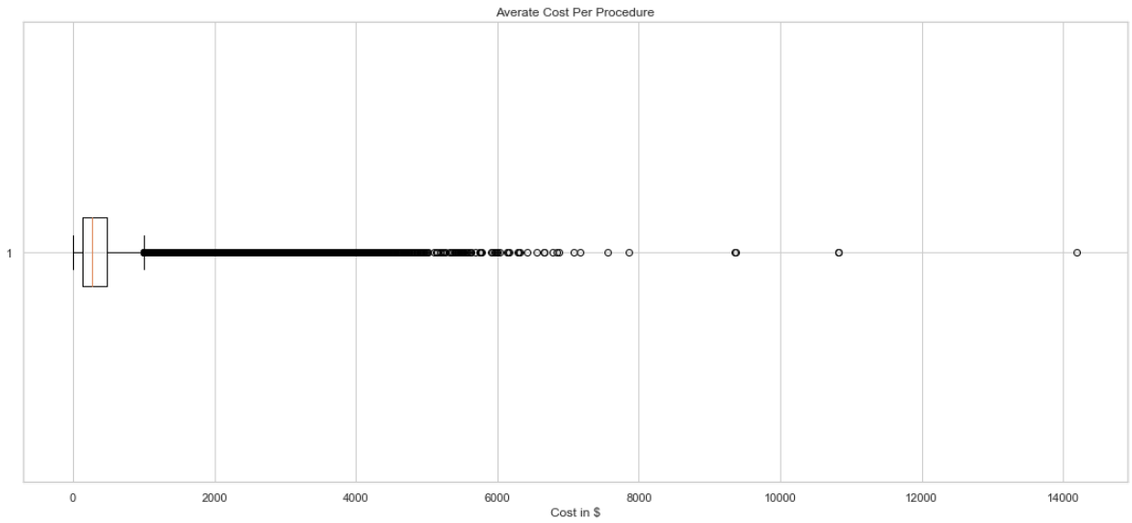
In [61]:
```
df['Average_Cost_Per_Procedure'] = df['Average_Total_Payments']/df['Total_Discharges']
df.head(2)
```

Out[61]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

In [62]:
```
fig = plt.figure(figsize=(15,7))
plt.boxplot(df['Average_Cost_Per_Procedure'], vert=False)
plt.title('Averate Cost Per Procedure')
plt.xlabel("Cost in $")
plt.tight_layout()
```



Feature 4

4.4. Average Medicare Payments, converted to % of Average Total Payments

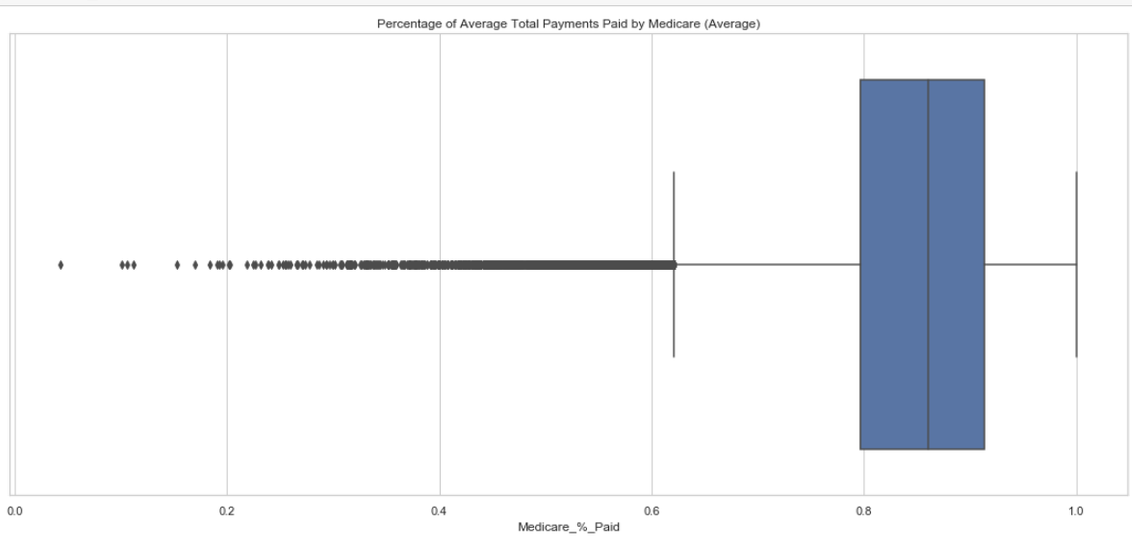
Medicare % paid varies by states, hospitals, and procedure. This feature will allow us to determing which hospitals, treatment and procedures are viewed favorably by medicare.

In [63]:
```
df['Medicare_%_Paid'] = df['Average_Medicare_Payments']/df['Average_Total_Payments']
df.head(2)
```

Out[63]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

In [64]:
```
fig = plt.figure(figsize=(15,7))
sns.boxplot(df['Medicare_%_Paid'])
plt.title('Percentage of Average Total Payments Paid by Medicare (Average)')
plt.tight_layout()
```



Observation:

Most average medicare payments are around 80-90% of the average total payments.

Feature 5

4.5. Medicare Payments, converted to % of Average Total Payments, State-wise

I am trying to understand if there are any states with 99% and greater medicare payout averages. Hypothesis is that those states are more attractive for fraud.

In [65]:
```
medicare_pct_state = df.groupby('Provider_State').agg({'Medicare_%_Paid': 'mean'}).reset_index()
medicare_pct_state.head(5)
```

Out[65]:

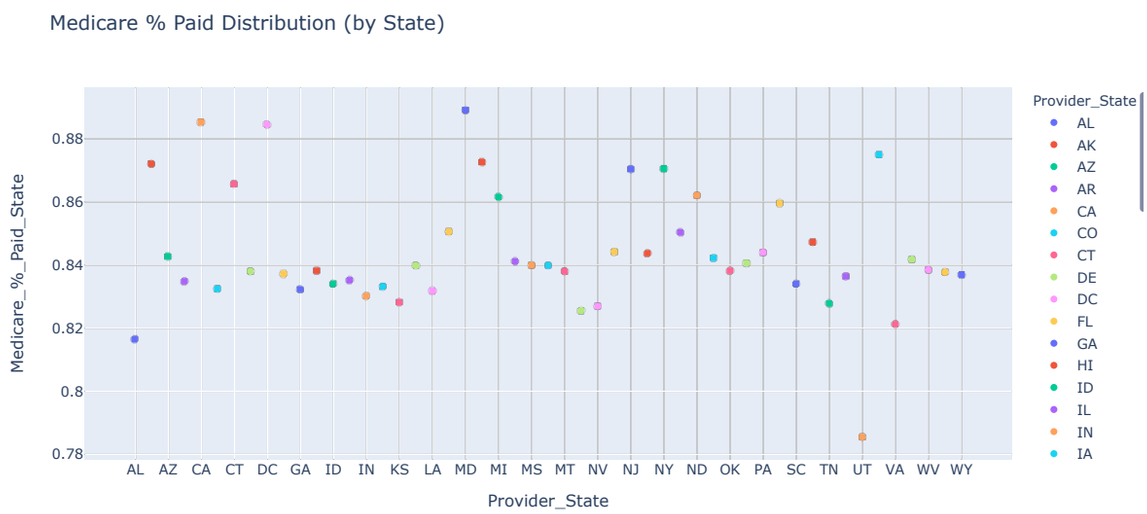
	Provider_State	Medicare_%_Paid
0	AK	0.871982
1	AL	0.816622
2	AR	0.834876
3	AZ	0.842718
4	CA	0.885084

In [66]:
```
df = pd.merge(left = df, right = medicare_pct_state, left_on = 'Provider_State', right_on = 'Provider_State',
              how = 'left')

df.rename(columns = {'Medicare_%_Paid_x':'Medicare_%_Paid',
                     'Medicare_%_Paid_y':'Medicare_%_Paid_State'}, inplace=True)

df.head(2)
```

Out[66]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

In [67]:
```
fig = px.scatter(df, x="Provider_State", y="Medicare_%_Paid_State", width=1000, height=500,
             color = "Provider_State",
            title = "Medicare % Paid Distribution (by State)")
fig.show()
```



In [68]:
```
#fig = plt.figure(figsize=(15,7))
#sns.distplot(df['Medicare_%_Paid_State'])
#plt.title('Medicare % Paid Distribution (by State)')
#plt.tight_layout()
```

Feature 6

4.6. Out-of-pocket expense, difference between 'Average_Total_Payments' & 'Average_Medicare_Payments'

Out of pocket highlight procedures/treatments that are most expensive. The hypothesis is that the procedures with the highest out of pocket cost are the least likely to be a target for fraud.

In [69]:
```
df['Out_of_Pocket_Payment'] = df['Average_Total_Payments'] - df['Average_Medicare_Payments']
df.head(2)
```

Out[69]:

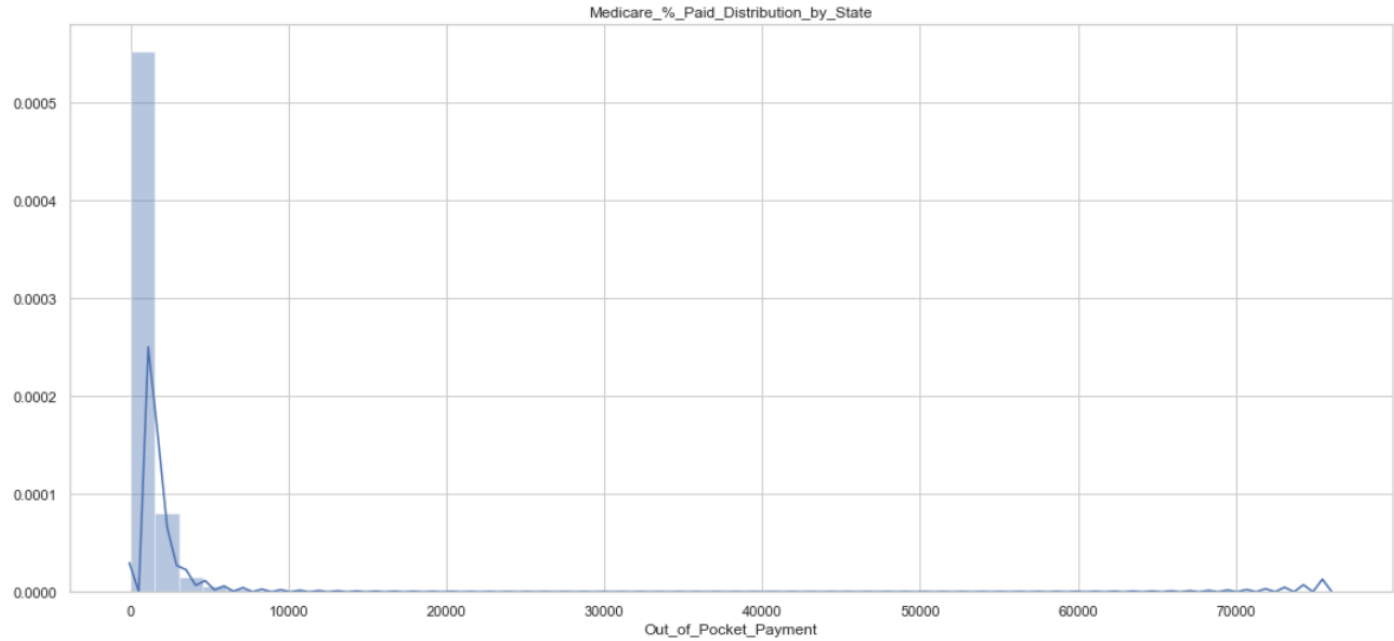
	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

In [70]:
```
sorted_avg_out_of_pocket = df.groupby(['DRG_Definition']).agg({'Out_of_Pocket_Payment': 'mean'})
sorted_avg_out_of_pocket.sort_values(by = 'Out_of_Pocket_Payment',ascending=False).head()
```

Out[70]:

DRG_Definition	Out_of_Pocket_Payment
460 - SPINAL FUSION EXCEPT CERVICAL W/O MCC	3735.070150
473 - CERVICAL SPINAL FUSION W/O CC/MCC	2594.714232
247 - PERC CARDIOVASC PROC W DRUG-ELUTING STENT W/O MCC	2582.521719
207 - RESPIRATORY SYSTEM DIAGNOSIS W VENTILATOR SUPPORT 96+ HOURS	2559.372528
853 - INFECTIOUS & PARASITIC DISEASES W O.R. PROCEDURE W MCC	2497.221490

In [71]:
```
fig = plt.figure(figsize=(15,7))
sns.distplot(df['Out_of_Pocket_Payment'])
plt.title('Medicare_%_Paid_Distribution_by_State')
plt.tight_layout()
```



Feature 7

4.7. Out-of-Pocket expense per discharge

Hospital that have high out of pocket cost can be useful to narrow down the ones unattractive to fraudsters.

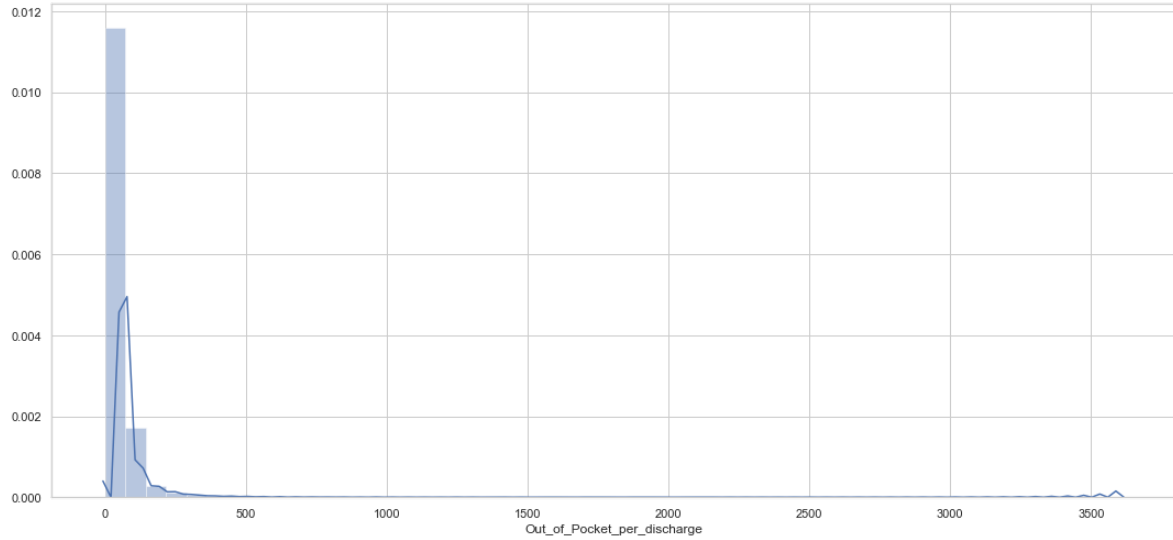
In [72]:
```
df['Out_of_Pocket_per_discharge'] = df['Out_of_Pocket_Payment']/df['Total_Discharges']
df.head(2)
```

Out[72]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 21 columns

In [73]:
```
fig = plt.figure(figsize=(15,7))
sns.distplot(df['Out_of_Pocket_per_discharge'])
plt.tight_layout()
```



Feature 8

4.8. Total treatments per state

The nomimal amount of treatments per state can be a misleading number to look at, given that the states with the most people will bubble to the top. This feature can be useful when compared against the population.

In [74]:
```
patients_states = df['Provider_State'].value_counts()
patients_states = pd.DataFrame(patients_states).reset_index()
patients_states.columns = ['Provider_State','Count']
patients_states.head()
```

Out[74]:

	Provider_State	Count
0	CA	13064
1	TX	11864
2	FL	11155
3	NY	9178
4	IL	7909

In [75]:
```
df = pd.merge(left = df, right = patients_states, left_on = 'Provider_State', right_on = 'Provider_State',
              how = 'left')

df.rename(columns = {'Count':'State_Total'}, inplace=True)

df.head(2)
```

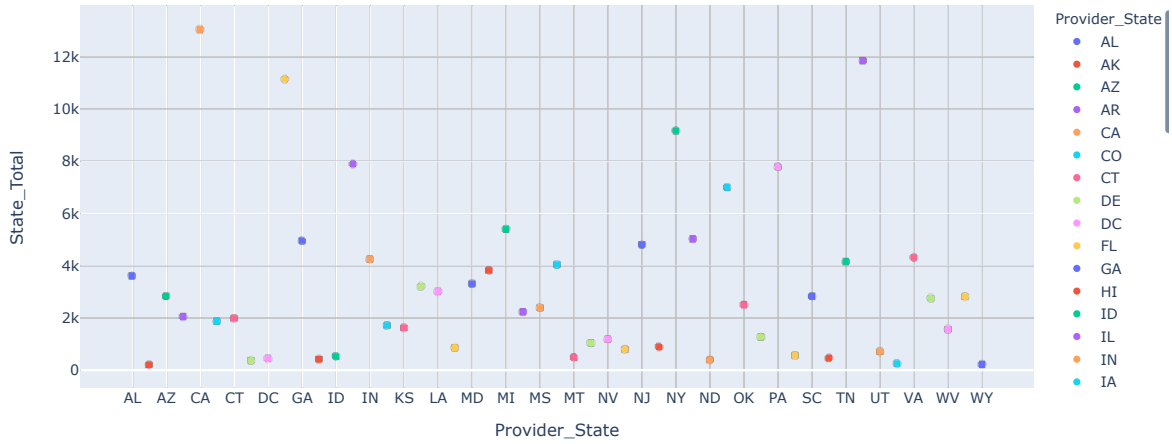
Out[75]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 22 columns

In [76]:
```
fig = px.scatter(df, x="Provider_State", y="State_Total", width=1000, height=500,
             color = "Provider_State",
            title = "Total Procedures/Treatments  per state")
fig.show()
```

Total Procedures/Treatments per state



In [77]:
```
#fig = plt.figure(figsize=(15,7))
#sns.distplot(df['State_Total'])
#plt.tight_layout()
```

Feature 9

4.9. Average treatments/patients by State (mean values, as grouped by State)

States have differents norms and rules. This features allows us to capture the normal state of each. Result can also be used to compare against the mean.

In [78]:
```
patient_avg_state = df.groupby('Provider_State').mean()[['Total_Discharges',
                                                         'Average_Total_Payments',
                                                         'Average_Medicare_Payments']].reset_index()
patient_avg_state.head()
```

Out[78]:

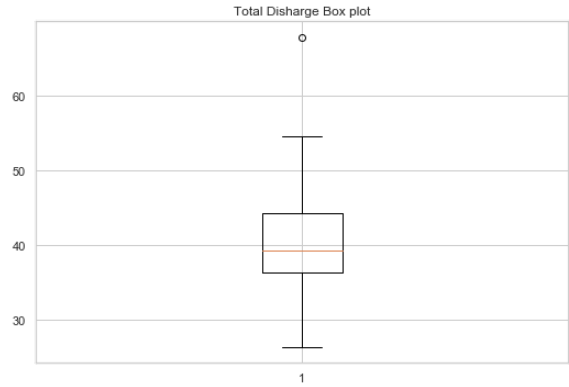
	Provider_State	Total_Discharges	Average_Total_Payments	Average_Medicare_Payments
0	AK	26.588745	14572.391732	12958.969437
1	AL	39.258322	7568.232149	6418.007120
2	AR	41.978229	8019.248805	6919.720832
3	AZ	36.690284	10154.528211	8825.717240
4	CA	36.357854	12629.668472	11494.381678

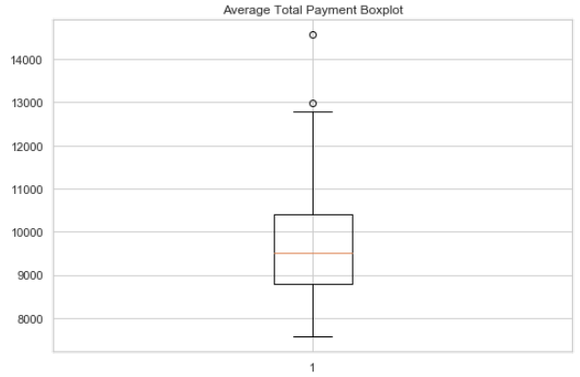
In [79]:
```
patient_avg_state.loc[:,'Total_Discharges':'Average_Medicare_Payments'].corr()
```

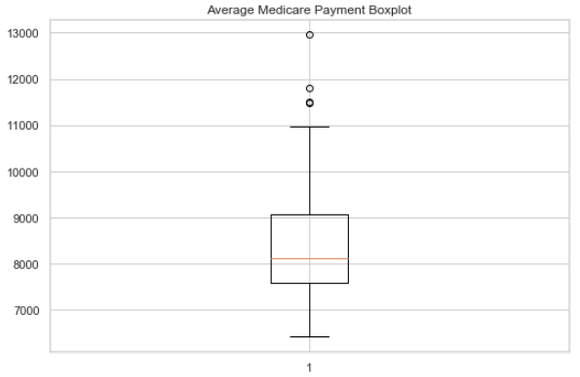
Out[79]:

	Total_Discharges	Average_Total_Payments	Average_Medicare_Payments
Total_Discharges	1.000000	-0.124043	-0.060745
Average_Total_Payments	-0.124043	1.000000	0.991735
Average_Medicare_Payments	-0.060745	0.991735	1.000000

In [80]:
```
fig = plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
plt.boxplot(patient_avg_state['Total_Discharges'])
plt.title('Total Disharge Box plot')
plt.xlabel('')

plt.subplot(2, 2, 3)
plt.boxplot(patient_avg_state['Average_Total_Payments'])
plt.title('Average Total Payment Boxplot')
plt.xlabel('')

plt.subplot(2, 2, 4)
plt.boxplot(patient_avg_state['Average_Medicare_Payments'])
plt.title('Average Medicare Payment Boxplot')
plt.tight_layout()
plt.show()
```







In [81]:
```
# States with highest discharges

patient_avg_state.sort_values(by = 'Total_Discharges', ascending = False).head()
```

Out[81]:

	Provider_State	Total_Discharges	Average_Total_Payments	Average_Medicare_Payments
8	DE	67.901015	10360.072411	8959.673274
22	MI	54.539952	9754.420406	8662.157756
31	NJ	52.052839	10678.988647	9586.940056
20	MD	51.955255	12608.947664	11480.121829
27	NC	51.043841	9089.435711	7998.649702

In [82]:
```
# States with highest Average Total Payments

patient_avg_state.sort_values(by = 'Average_Total_Payments', ascending = False).head()
```

Out[82]:

	Provider_State	Total_Discharges	Average_Total_Payments	Average_Medicare_Payments
0	AK	26.588745	14572.391732	12958.969437
7	DC	43.954545	12998.029416	11811.967706
11	HI	26.497738	12775.739525	10967.475045
4	CA	36.357854	12629.668472	11494.381678
20	MD	51.955255	12608.947664	11480.121829

Note:

These new mean columns are useful, but I believe Median columns will be more handy. So, I will not add the mean columns to the original dataset yet.

`Feature 10`

4.10. Calculate the median of average total payment amount by DRG, by state.

'Median Average Total Payment'

To catch payments for treaments that exceed a normal amount, I will first create a feature that calculates a score for each data row. The score will indicate the size the payment amount for a particular treament relative to the median size of the payment amount for the DRG code by state level.

In [83]:
```
median_drg_state = df.groupby(['DRG_Definition','Provider_State'])['Average_Total_Payments'].median().reset_index()
median_drg_state.head()
```

Out[83]:

	DRG_Definition	Provider_State	Average_Total_Payments
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	AK	8401.95
1	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	AL	5658.33
2	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	AR	5890.00
3	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	AZ	6959.89
4	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	CA	7863.14

In [84]:
```
df = pd.merge(left = df, right = median_drg_state, left_on = ['DRG_Definition','Provider_State'], right_on = ['DRG_Def
inition','Provider_State'], how = 'left')
df.rename(columns={'Average_Total_Payments_y':'Median_Avg_Total_Pyt',
                   'Average_Total_Payments_x':'Average_Total_Payments'}, inplace=True)
df.head(2)
```

Out[84]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 23 columns

In [85]:
```
# Check for one particular state and one particular DRG, to see median

df[(df['Provider_State'] == 'NV') & (df['DRG_Definition'] == '194 - SIMPLE PNEUMONIA & PLEURISY W CC')].head(2)
```

Out[85]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Descripti
89303	194 - SIMPLE PNEUMONIA & PLEURISY W CC	290001	RENOWN REGIONAL MEDICAL CENTER	1155 MILL STREET	RENO	NV	89502	NV - Re

89400	194 - SIMPLE PNEUMONIA & PLEURISY W CC	290003	SUNRISE HOSPITAL AND MEDICAL CENTER	3186 S MARYLAND PKWY	LAS VEGAS	NV	89109	NV - Las Veg

2 rows × 23 columns

`Feature 11`

4.11. Creating a common median multiple score for Average Total Payments

Explain a potential fraud case

'Median Score`

I will now take the average total payment and divide it by the median payment to generate a simple score that indicates how many times the current payment amount is larger than the median amount.

In [86]:
```
df['Median_Score'] = df['Average_Total_Payments']/df['Median_Avg_Total_Pyt']
df.head(2)
```

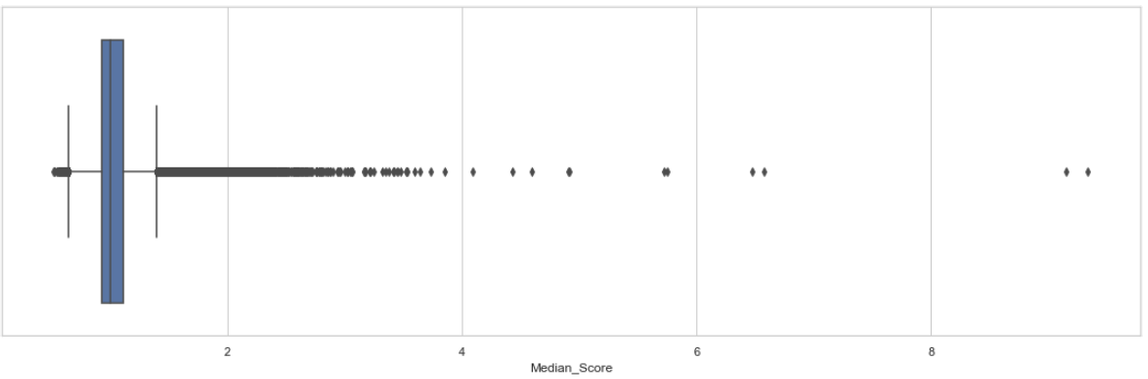
Out[86]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 24 columns

In [87]:
```
# fig = plt.figure(figsize=(10,5))
# sns.distplot(df['Median_Score'])
# plt.tight_layout()
```

In [88]:
```
fig = plt.figure(figsize=(15,5))
sns.boxplot(df['Median_Score'])
plt.tight_layout()
```



In [89]:
```
df['Median_Score'].describe()
```

Out[89]:
```
count    163065.000000
mean          1.050746
std           0.211465
min           0.517695
25%           0.925511
50%           1.000000
75%           1.112126
max           9.338775
Name: Median_Score, dtype: float64
```

Observation:

It would appear that most treatment payment amounts for the same DRG within the same state are within 90% to 110% of the median price. This is expected as normally doctors should be charging similar prices for similar treatment in simlar areas. However, we see instances where the payment amount is many times that of the median.

As we see in the box plot above, in two specific cases, treament cost over 9 times the median amount.

Let us investigate further.

Below are the cases where the payment made was over 6 times the median score amount.

In [90]:
```
df[df['Median_Score'] >= 6]
```

Out[90]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Descri
69869	203 - BRONCHITIS & ASTHMA W/O CC/MCC	220008	STURDY MEMORIAL HOSPITAL	211 PARK STREET	ATTLEBORO	MA	02703	RI - Provid
125467	189 - PULMONARY EDEMA & RESPIRATORY FAILURE	390096	ST JOSEPH MEDICAL CENTER	2500 BERNVILLE ROAD	READING	PA	19605	PA - Rea
130307	948 - SIGNS & SYMPTOMS W/O MCC	390312	CANCER TREATMENT CENTERS OF AMERICA	1331 EAST WYOMING AVENUE	PHILADELPHIA	PA	19124	PA - Philade

156606	249 - PERC CARDIOVASC PROC W NON-DRUG-ELUTING ...	500051	OVERLAKE HOSPITAL MEDICAL CENTER	1035-116TH AVE NE	BELLEVUE	WA	98004	WA - Se

4 rows × 24 columns

Observation:

The highest median score is 9.33 which is for:

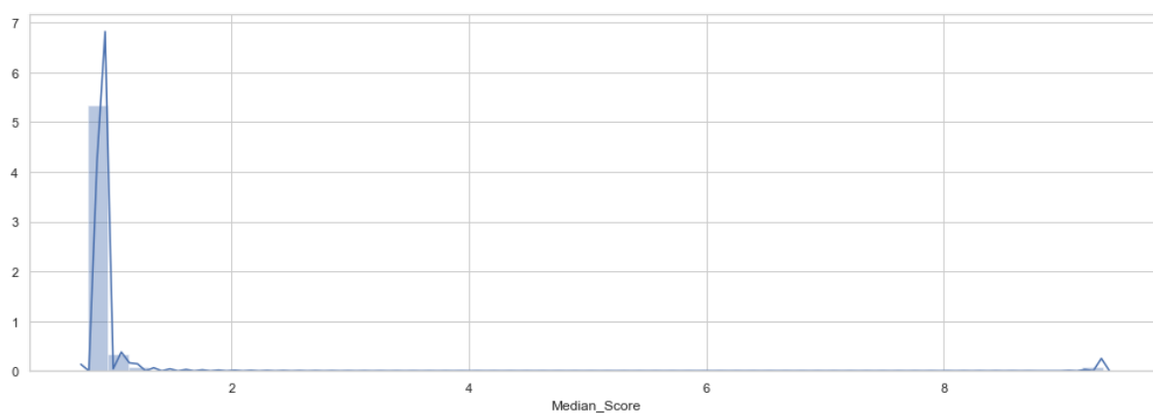
- index number - 69869
- DRG - 203 - BRONCHITIS & ASTHMA W/O CC/MCC
- City - ATTLEBORO
- State - MA

This individual was charged USD 41,482 for a treament that had median amount of just USD 4,441.92. This particular treatment was performed at Provider - STURDY MEMORIAL HOSPITAL.

Now, Let's examine this particular hospital more closely.

In [91]:
```
suspect_hospital1 = df[df['Provider_Name'] == 'STURDY MEMORIAL HOSPITAL']['Median_Score']

fig = plt.figure(figsize=(14,5))
sns.distplot(suspect_hospital1)
plt.tight_layout()

print(" ")
print("Median Score distribution for Provider - STURDY MEMORIAL HOSPITAL is as follows")
print(" ")
print(suspect_hospital1.describe())
```

```
Median Score distribution for Provider - STURDY MEMORIAL HOSPITAL is as follows

count    69.000000
mean      1.027317
std       1.016937
min       0.786568
25%       0.873115
50%       0.903796
75%       0.928759
max       9.338775
Name: Median_Score, dtype: float64
```



Observation:

The graphical representation is strange. The hospital typically charges less than the median amount for its treatments as we see that the highest number of observations fall beloew the median score of 1. This makes the treament with the median mulitple score of over 9 highly unusal.

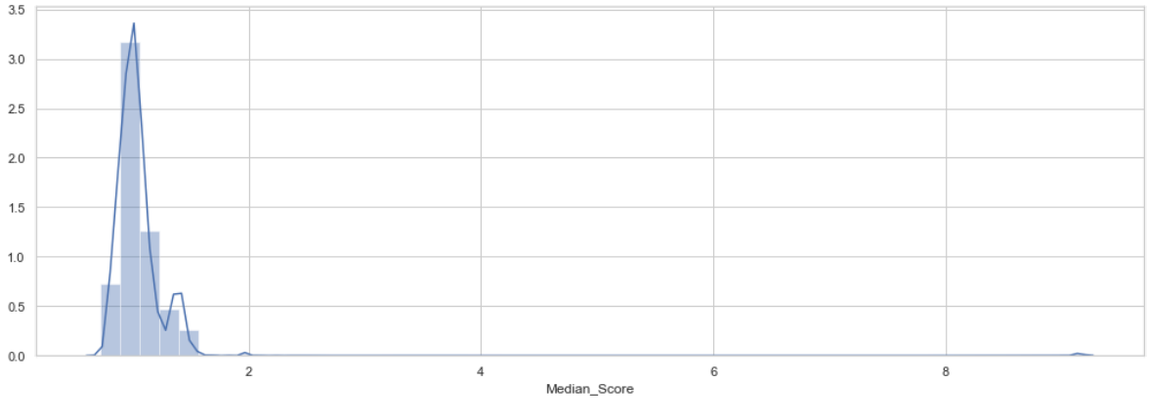
Now, another hospital with a treatment which cost way over the median amount is 'ST JOSEPH MEDICAL CENTER', which is:

- index number - 125467
- DRG - 189 - PULMONARY EDEMA & RESPIRATORY FAILURE
- City - READING
- State - PA

Lets examine this hosptial's track record too.

In [92]:
```
suspect_hospital2 = df[df['Provider_Name'] == 'ST JOSEPH MEDICAL CENTER']['Median_Score']

fig = plt.figure(figsize=(14,5))
sns.distplot(suspect_hospital2)
plt.tight_layout()

print(" ")
print("Median Score distribution for Provider - ST JOSEPH MEDICAL CENTER is as follows")
print(" ")
print(suspect_hospital2.describe())
```

```
Median Score distribution for Provider - ST JOSEPH MEDICAL CENTER is as follows

count    427.000000
mean       1.060935
std        0.424824
min        0.720824
25%        0.933857
50%        1.007073
75%        1.099725
max        9.150067
Name: Median_Score, dtype: float64
```



Observation:

Again, this is a hospital that typically charges resonable prices for treatment, most observations are below the median score of 2, which is fine. But, this makes the one treatment that is over 9 times the median price an extreme outlier deserving of extra attention.

Making the broad assumption that around 1% of medical payments are fraudulent. I will tag any medical treatment that paid more than the top 99th percentile of median_scores in the dataset.

In [93]:
```
np.percentile(df['Median_Score'], 99)
```

Out[93]:
```
1.790032745554255
```

1% of medical treatments cost more than 1.79 times the median payment of that treament by state. Treaments that paid more than this shall be flagged.

`Feature 12`

4.12. Boolean Flag for Providers, based on the Median Score, if not in 99% percentile of data.

'Median Score Flag'

To catch payments for treaments that exceed a normal amount, I will first create a feature that calculates a score for each data row. The score will indicate the size the payment amount for a particular treament relative to the median size of the payment amount for the DRG code by state level.

In [94]:
```
df['Median_Score_Flag'] = df['Median_Score'] >= np.percentile(df['Median_Score'], 99)
df.head(2)
```

Out[94]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 25 columns

This approach is good for finding providers that overcharge substancially. However, it is not so useful in find hospitals that overcharge slightly but over the course of many treatments. One way to find these providers is to find which ones have the highest average median_score.

`Feature 13`

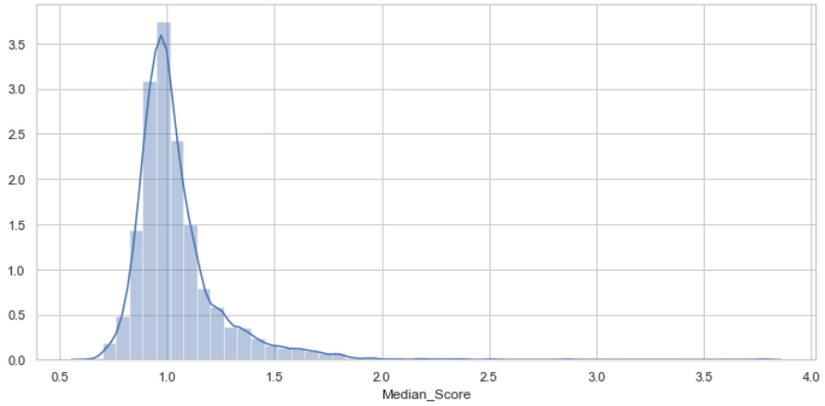
4.13. Calculate the Median Score, this time based in individual Provider level, previously we did on individual row level.

'Median Score by Provider'

To catch payments for treaments that exceed a normal amount, I will first create a feature that calculates a score for each data row. The score will indicate the size the payment amount for a particular treament relative to the median size of the payment amount for the DRG code by state level.

In [95]:
```
Median_Score_by_Provider = df.groupby(['Provider_Name']).mean()['Median_Score'].reset_index()
print(Median_Score_by_Provider.head(2))

fig = plt.figure(figsize=(10,5))
sns.distplot(Median_Score_by_Provider['Median_Score'])
plt.tight_layout()

print(" ")
print("Median Score distribution for Provider -  is as follows")
print(" ")
print(Median_Score_by_Provider.describe())
```

```
                   Provider_Name  Median_Score
0    ABBEVILLE GENERAL HOSPITAL      1.033378
1  ABBOTT NORTHWESTERN HOSPITAL      1.056686

Median Score distribution for Provider -  is as follows

       Median_Score
count   3201.000000
mean       1.043923
std        0.200934
min        0.635737
25%        0.927984
50%        0.995734
75%        1.096960
max        3.777786
```



Some providers consistantly charge multiple times the median payment amount. However, before we blow the horn on these hospitals, we have to consider the fact that perhaps these hospitals are charging more than their state counter parts because they are either high-end/luxury hosptials and/or they are located in expensive cities. Lets see what hospitals these are.

In [96]:
```
df = pd.merge(left = df, right = Median_Score_by_Provider, left_on = 'Provider_Name',
              right_on = 'Provider_Name', how = 'left')

df.rename(columns={'Median_Score_x':'Median_Score',
                   'Median_Score_y':'Median_Score_by_Provider'}, inplace=True)
df.head(2)
```

Out[96]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 26 columns

Display the hospitals, that on average charge more than double the median price for the treatments they provide copmared to others within their state.

I had expected these hospitals to either be luxury hospitals or located in expensive cities.

In [97]:
```
df[df['Median_Score_by_Provider'] >= 2][['Provider_Name', 'Provider_State','Provider_City']].drop_duplicates()
```

Out[97]:

	Provider_Name	Provider_State	Provider_City
15017	CONTRA COSTA REGIONAL MEDICAL CENTER	CA	MARTINEZ
19110	MEMORIAL HOSPITAL LOS BANOS	CA	LOS BANOS
23176	KEEFE MEMORIAL HOSPITAL	CO	CHEYENNE WELLS
23353	VAIL VALLEY MEDICAL CENTER	CO	VAIL
27595	JACKSON MEMORIAL HOSPITAL	FL	MIAMI
46304	MIDWESTERN REGION MED CENTER	IL	ZION
96910	GUADALUPE COUNTY HOSPITAL	NM	SANTA ROSA
130305	CANCER TREATMENT CENTERS OF AMERICA	PA	PHILADELPHIA
138799	UNIVERSITY OF TEXAS MEDICAL BRANCH GAL	TX	GALVESTON
159340	WELCH COMMUNITY HOSPITAL	WV	WELCH

`Feature 14`

4.14. Create a boolean benchmark for hospitals who overcharge

'Provider Flag by Median Score'

I set the benchmark at an upper level of 2 or higher median score, this will avoid situations where the cost of the city impacts the cost of treatment as even in the most expensive cities, the average cost of treament will in an expensive city will never be double the average cost of treament outside the city within the same state.

In [98]:
```
df['Provider_Flag_by_Median_Score'] = df['Median_Score_by_Provider'] >= 2
df.head(2)
```

Out[98]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 27 columns

See a the list of providers that are flagged as their treamtent payments are on average, more than double the median amount for the state they are in.

In [99]:
```
df[df['Provider_Flag_by_Median_Score']][['Provider_Id','Provider_Name','Provider_City',
                                         'Provider_State','Median_Score_by_Provider']].drop_duplicates()
```

Out[99]:

	Provider_Id	Provider_Name	Provider_City	Provider_State	Median_Score_by_Provider
15017	50276	CONTRA COSTA REGIONAL MEDICAL CENTER	MARTINEZ	CA	2.254486
19110	50528	MEMORIAL HOSPITAL LOS BANOS	LOS BANOS	CA	2.508609
23176	60043	KEEFE MEMORIAL HOSPITAL	CHEYENNE WELLS	CO	2.376034
23353	60096	VAIL VALLEY MEDICAL CENTER	VAIL	CO	2.030422
27595	100022	JACKSON MEMORIAL HOSPITAL	MIAMI	FL	2.173011
46304	140100	MIDWESTERN REGION MED CENTER	ZION	IL	2.861803
96910	320067	GUADALUPE COUNTY HOSPITAL	SANTA ROSA	NM	2.187486
130305	390312	CANCER TREATMENT CENTERS OF AMERICA	PHILADELPHIA	PA	3.777786
138799	450018	UNIVERSITY OF TEXAS MEDICAL BRANCH GAL	GALVESTON	TX	2.336738
159340	510086	WELCH COMMUNITY HOSPITAL	WELCH	WV	2.085069

I will now pick the hospital with the highest score, which is - CANCER TREATMENT CENTERS OF AMERICA.

Now, I assume this hospital constantly overcharges on procedures that are much cheaper in other hospitals within the same state. Lets see it below.

In [100]:
```
df[df['Provider_Name'] == 'MEMORIAL HOSPITAL LOS BANOS'][['Median_Score']]
```

Out[100]:

	Median_Score
19110	2.624709
19111	2.473714
19112	2.490096
19113	2.748748
19114	2.305088
19115	2.489811
19116	2.471841
19117	2.464865

Observation:

As we see, this is an effective method to track and find hospitals who overcharge or committ fraud.

`Feature 15`

4.15. Sum total of 'Total Discharged' grouped by the Provider Id and Provider Name

I will check the grand total of the number of discharges made by ech Provider. The hypothesis is that the hospitals with the highest number of diacharges are more susceptible to fraud, having false patients and fake claims.

In [101]:
```
discharges_provider = df.groupby(['Provider_Id','Provider_Name'])['Total_Discharges'].sum().reset_index(name='Grand To
tal of Discharges')
discharges_provider = discharges_provider.sort_values(by='Grand Total of Discharges', ascending=False)
discharges_provider.head()
```

Out[101]:

	Provider_Id	Provider_Name	Grand Total of Discharges
599	100007	FLORIDA HOSPITAL	25828
1970	330101	NEW YORK-PRESBYTERIAN HOSPITAL	16834
2882	450388	METHODIST HOSPITAL	15921
583	80001	CHRISTIANA CARE HEALTH SERVICES, INC.	14542
1534	230130	WILLIAM BEAUMONT HOSPITAL	14469

Observation:

It is seen that the highest discharges are by 'Florida Hospital', at 25,828.

However, it would be ideal to check the total discharges as group be Provider State to get better inference.

In [102]:
```
discharges_provider_state = df.groupby(['Provider_State','Provider_Id', 'Provider_Name'])['Total_Discharges'].sum().re
set_index(name='Grand Total of Discharges')
discharges_provider_state = discharges_provider_state.sort_values(by='Grand Total of Discharges', ascending=False)
discharges_provider_state.head()
```

Out[102]:

	Provider_State	Provider_Id	Provider_Name	Grand Total of Discharges
599	FL	100007	FLORIDA HOSPITAL	25828
2065	NY	330101	NEW YORK-PRESBYTERIAN HOSPITAL	16834
2882	TX	450388	METHODIST HOSPITAL	15921
590	DE	80001	CHRISTIANA CARE HEALTH SERVICES, INC.	14542
1534	MI	230130	WILLIAM BEAUMONT HOSPITAL	14469

`Feature 16`

4.16. Ratio of 'Average Total Payments' to 'Zip_Median_Income'

'Avg_Payment_by_Median_Income'

This ratio will give the an idea as to whether the average payments are higher or lower than the median income of the population. The hypothesis is that if the ratio is high, then the persons in that zip code are paying much higher than their median income for the treatment, which might be a fraud case, as the patient/hospital might have inflated bills.

Even the same hospital may have lower ratio for a particular DRG or State, but higher for another one, so I will not group the data. I need individual ratio for each row entry.

In [103]:
```
df['Avg_Payment_by_Median_Income'] = df['Average_Total_Payments'] / df['Zip_Median_Income']
df.head(2)
```

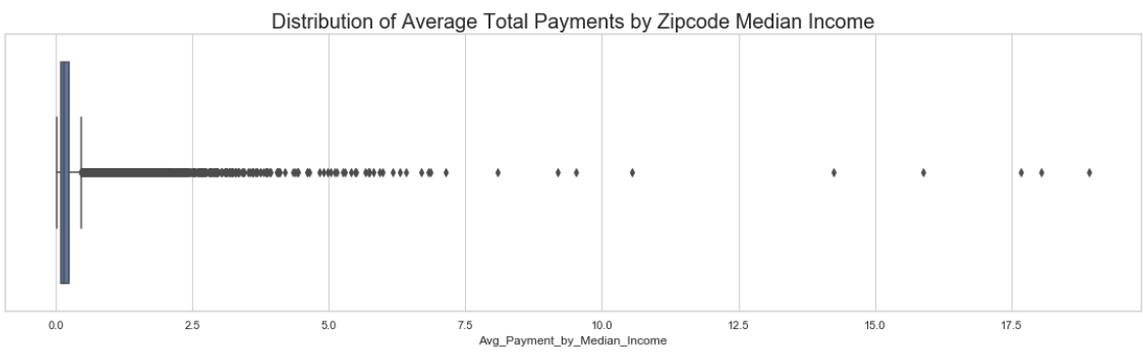
Out[103]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 28 columns

In [104]:
```
fig, ax = plt.subplots(figsize= (20,5))
sns.boxplot(df["Avg_Payment_by_Median_Income"])
plt.title("Distribution of Average Total Payments by Zipcode Median Income", fontsize=20)
```

Out[104]:
```
Text(0.5, 1.0, 'Distribution of Average Total Payments by Zipcode Median Income')
```



The ones which are over 5 might need some investigation.

`Feature 17`

4.17. Ratio of 'Total Discharges' to 'Zip_Population'

'Total_Disc_by_Pop'

Hospitals havings very ratio are likely to be showing false patients.

Those with higher ratio, might need further investigation. Even the same hospital may have lower ratio for a particular DRG or State, but higher for another one, so I will not group the data. I need individual ratio for each row entry.

In [105]:
```
df['Total_Disc_by_Pop'] = df['Total_Discharges'] / df['Zip_Population']
df.head(2)
```

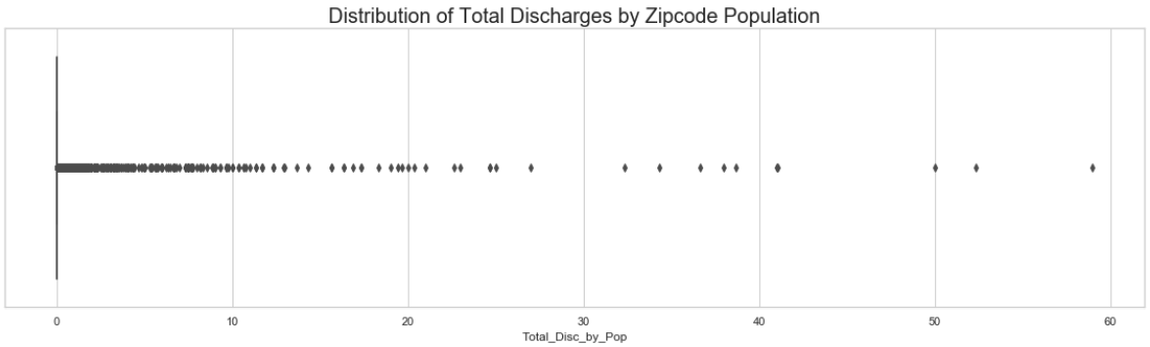
Out[105]:

	DRG_Definition	Provider_Id	Provider_Name	Provider_Street_Address	Provider_City	Provider_State	Provider_Zip_Code	Hospital_Referral_Region_Description
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan

2 rows × 29 columns

In [106]:
```
fig, ax = plt.subplots(figsize= (20,5))
sns.boxplot(df["Total_Disc_by_Pop"])
plt.title("Distribution of Total Discharges by Zipcode Population", fontsize=20)
```

Out[106]:
```
Text(0.5, 1.0, 'Distribution of Total Discharges by Zipcode Population')
```



The ratio's over 10 might need investigation.

.

---------- SECTION BREAK ----------

.

5. Modelling

I will create a new copy of the data to perform the analysis, and call it 'df1'.

In [107]:
```
# Create a copy of the original dataset

df1 = df.copy()
```

In [108]:
```
# Check nulls

df1.isnull().sum().sum()
```

Out[108]:
```
48452
```

These NA's are created as a reuslt of merging the Income and the Population data by zipcode. There are certain zip codes in our data set, which do not have an entry n the zipcode dataset.

So, I will replace the NA's with the median of the respective column.

Impute missing Income and Population values with respective median

In [109]:
```
df1 = df1.fillna(df1.median())
```

In [110]:
```
df1.isnull().sum().sum()
```

Out[110]:
```
0
```

Drop irrelavant columns

There a lot of columns which will not be useful in the kmeans clustering analysis, most of which are categorical columns. I will drop them.

In [111]:
```
df1 = df1.drop(columns = ['Provider_Id', 'Provider_Name', 'Provider_Street_Address',
                          'Provider_City', 'Provider_Zip_Code',
                          'Hospital_Referral_Region_Description',
                          'Region',
                          'Division', 'Zip_Median_Income',
                          'Zip_Population', 'Median_Score_Flag',
                          'Provider_Flag_by_Median_Score'])
df1.head(2)
```

Out[111]:

	DRG_Definition	Provider_State	Total_Discharges	Average_Covered_Charges	Average_Total_Payments	Average_Medicare_Payments	Average_Cost_Per_Procedu
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	AL	91	32963.07	5777.24	4763.73	63.4861
1	057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/...	AL	38	20312.78	4894.76	3865.50	128.8094

Categorical Variables

I will drop the categrical columns, since I have alrwady used them to create meaningful new features

Label en-coding will be misleading for the clustering, and one-hot encoding creates many different new binary features, which is not ideal for a kNN and PCA clustering.

In [112]:
```
df1 = df1.drop(columns = ['DRG_Definition', 'Provider_State'])
```

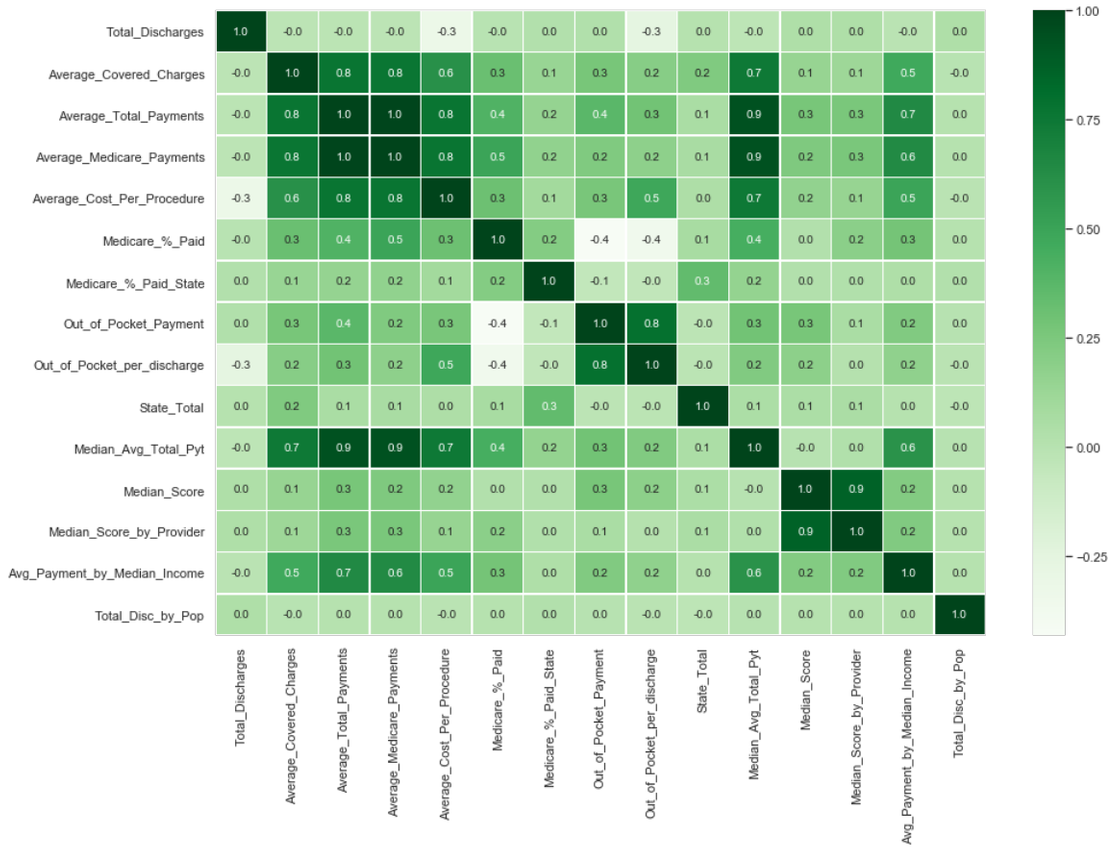
Create a alias for numerical variables

In [113]:
```
# All numeric / float variables in thedataset

num_variables_all = ['Total_Discharges', 'Average_Covered_Charges', 'Average_Total_Payments', 'Average_Medicare_Paymen
ts',
                 'Average_Cost_Per_Procedure', 'Medicare_%_Paid',
       'Medicare_%_Paid_State', 'Out_of_Pocket_Payment',
       'Out_of_Pocket_per_discharge', 'State_Total', 'Median_Avg_Total_Pyt',
       'Median_Score', 'Median_Score_by_Provider',
       'Avg_Payment_by_Median_Income',
       'Total_Disc_by_Pop']
```

Check multi-collinearity

Drop one column from a pair with a ratio of above 0.7

In [114]:
```
corr = df1.corr()
f,ax = plt.subplots(figsize=(15,10))
sns.heatmap(corr, annot=True, cmap='Greens', linewidths=.4, fmt= '.1f', ax=ax)
plt.show()
```



In [115]:
```
# Remove one each from the pair of highly collinear variables

df1 = df1.drop(columns = ['Average_Medicare_Payments', 'Average_Covered_Charges',
                          'Median_Avg_Total_Pyt', 'Out_of_Pocket_per_discharge',
                       'Average_Cost_Per_Procedure', 'Median_Score_by_Provider',
                      'Avg_Payment_by_Median_Income'])
```

In [116]:
```
# Final numeric variables selected

num_variables = ['Total_Discharges', 'Average_Total_Payments',
                 'Medicare_%_Paid',
       'Medicare_%_Paid_State', 'Out_of_Pocket_Payment',
       'State_Total', 'Median_Score',
       'Total_Disc_by_Pop']
```

Scaling/Standardizing all float or integer variables

I will use Standard Scalar/Standardize to scale all the numerical variables

In [117]:
```
X = StandardScaler().fit_transform(df1)
X = pd.DataFrame(data = X, columns = num_variables)
X.head()
```

Out[117]:

	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_Pop
0	0.943640	-0.512776	-0.249983	-1.562935	-0.173743	-0.656043	-0.140595	-0.028289
1	-0.093463	-0.627913	-0.646365	-1.562935	-0.160024	-0.656043	-0.258383	-0.031520
2	0.806665	0.072115	0.533658	-1.562935	-0.104393	-0.656043	-0.321171	-0.028715
3	2.469943	-0.412988	-0.173291	-1.562935	-0.095291	-0.656043	-0.093474	-0.023532
4	-0.191303	-0.666841	-0.673441	-1.562935	-0.205142	-0.656043	-0.166491	-0.031825

In [118]:
```
X.isnull().sum().sum()
```

Out[118]: 0

5.1. kNN clustering

Split into Train Test, to perform the kNN clustering

For this, I will first split my data to train and test. This is essential to train the model.

Note:

Even the train data has many outliers, so instead of using the smalle test data, I will use the entire data (X) inplace of the test data.

In [119]:
```
X.shape[0] * 0.75
```

Out[119]:
```
122298.75
```

In [120]:
```
X_train = X.iloc[:122300,:]
X_test = X.iloc[122301:,:]
print("Shape of new dataframes - {} , {}".format(X_train.shape, X_test.shape))
```

```
Shape of new dataframes - (122300, 8) , (40764, 8)
```

kNN Model

The k-NN algorithm is a non-parametric method that identifies the k closest training examples. Any isolated data points can potentially be classified as outliers. The following lines completes the training for the k-NN model and stores the model as clf.

KNN does not have any learning involved,

i.e., there are no parameters we can tune to make the performance better. Computing neighbors does not require the target variable.

In [121]:
```
# train kNN detector

clf_name = 'KNN'
clf = KNN()
clf.fit(X_train)
```

Out[121]:
```
KNN(algorithm='auto', contamination=0.1, leaf_size=30, method='largest',
  metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2,
  radius=1.0)
```

For the purpose of testing the trained set model,

I will use the entire dataset, which X as the Test data.

This is because even the train dataset containes outliers, and I need to idenify outliers in the entire dataset.

In [122]:
```
# Now we have the trained K-NN model, let's apply to the test data

y_test_pred = clf.predict(X) # outlier labels (0 or 1)

# Because it is '0' and '1', we can run a count statistic.
# There are 16986 '1's and 146079 '0's.
# The number of anomalies is roughly 12 percent:

unique, counts = np.unique(y_test_pred, return_counts=True)
dict(zip(unique, counts))

#{0: 146079, 1: 16986}
```

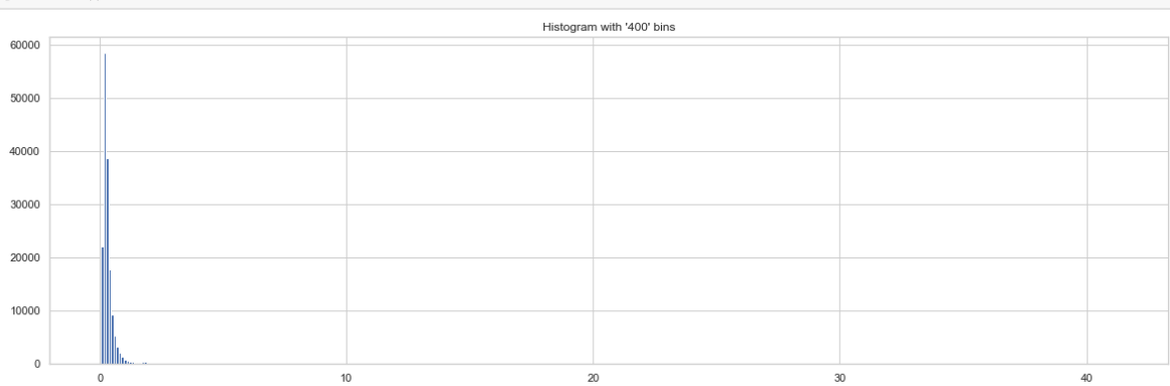
Out[122]:
```
{0: 146079, 1: 16986}
```

Generate the anomaly score using clf.decision_function and visualize

In [123]:
```
x_test_scores = clf.decision_function(X)
```

Plotting the Scores

In [124]:
```
fig, ax = plt.subplots(figsize= (20,6))
plt.hist(x_test_scores, bins = 400)  # arguments are passed to np.histogram
plt.title("Histogram with '400' bins")
plt.show()
```

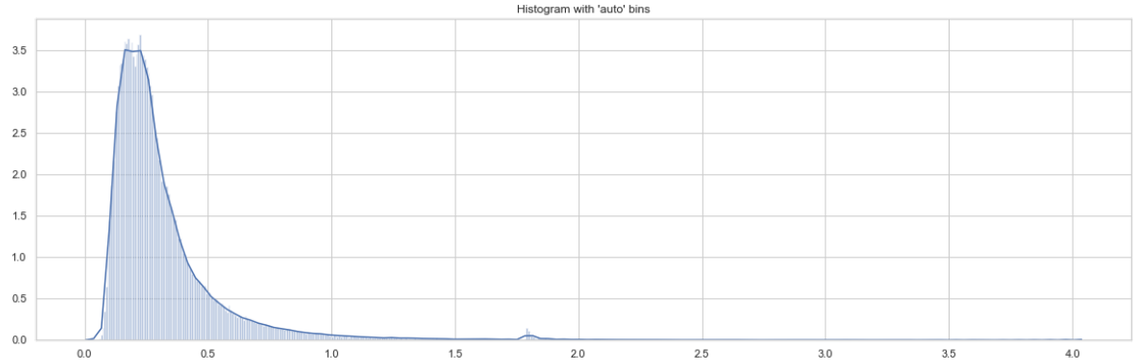


In the above graph, it is very difficult to see the distribution, as the range is very vast. So, I will cut the range to 4.0 and then re-plot

In [125]:
```
x_test_scores_small = x_test_scores[x_test_scores[:, ] < 4.0]
x_test_scores_small
```

Out[125]:
```
array([0.25519257, 0.16854206, 0.29718556, ..., 0.27110163, 0.71855941,
       0.38147397])
```

In [126]:
```
fig, ax = plt.subplots(figsize= (20,6))
sns.distplot(x_test_scores_small, bins = 'auto')  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")
plt.show()
```



Observation:

A high anomaly score probably means more abnormal transaction/score. The histogram above shows there are outliers. I will now chose a range of cut points, to check the distribution of clusters, and then identify some as suspicious.

Reasonable Boundary

Generally, looking at the graph, 2.0 seems the ideal number to be a cut point or the reasonable boundary. If we choose 2.0 to be the cut point, we can suggest those >= 2.0 to be outliers.

But, I want to investigate deeper, and I will chose 4 diifferent cut points, which are:

- 0.5
- 1.0
- 2.0
- 5.0

Now, lets evaluate all these 4 different cut points, and build 5-cluster model.

Build a 5 cluster model as per above reasonable boundaries

In [127]:
```
df_test = pd.DataFrame(X)
df_test['score'] = x_test_scores

df_test['cluster'] = (np.where(df_test['score'] > 5.0, 5,
                                        (np.where(df_test['score'] > 2.0, 4,
                                            (np.where(df_test['score'] > 1.0, 3,
                                                     (np.where(df_test['score'] > 0.5, 2, 1))))))))

df_test['cluster'].value_counts()
```

Out[127]:
```
1    141804
2     17065
3      3575
4       567
5        54
Name: cluster, dtype: int64
```

Percentage of Data points in each cluster

In [128]:
```
temp = pd.DataFrame()

temp['Percentage of total'] = (df_test['cluster'].value_counts() / df_test['cluster'].value_counts().sum()) * 100
temp['Cluster'] = ['1','2','3','4','5']

temp
```

Out[128]:

	Percentage of total	Cluster
1	86.961641	1
2	10.465152	2
3	2.192377	3
4	0.347714	4
5	0.033116	5

I will provde the description and business insight later below, when I aggregate with the 'Average' method

Summary statistics

In [129]:
```
# Now let's show the summary statistics:
df_test.groupby('cluster').mean()
```

Out[129]:

cluster	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_
1	-0.118737	-0.152158	0.050007	0.013037	-0.179234	0.024999	-0.132081	-0.02
2	0.660132	0.871102	-0.224984	-0.004699	0.835457	-0.157285	0.719945	-0.00
3	1.209147	1.419581	-0.683052	-0.457927	2.204629	-0.214455	1.305438	0.40
4	1.814985	2.498838	-1.248673	-0.281906	4.678498	-0.138857	2.410171	2.62
5	4.081403	4.062423	-1.886511	0.525202	11.569548	-0.285665	7.597254	23.26

Note:

Outlier Detection Models Are Prone and sensitive to Outliers/Anomalies

Also, Outlier detection models tend to be very sensitive to outliers. In our case the unsupervised k-NN model can easily commit overfitting. We need to produce stable models.

Achieve Model Stability by Aggregating Multiple Models

The solution is to train multiple models then aggregate the scores. By aggregating multiple models, the chance of overfitting is greatly reduced and the prediction accuracy will be improved. The PyOD module offers four methods to aggregate the outcome. I will use the 'Average' method for my analysis.

Aggregate by 'Average' Method to get better results

In [130]:
```
new_X = X.drop(columns = ['score', 'cluster'])
new_X.head(2)
```

Out[130]:

	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_Pop
0	0.943640	-0.512776	-0.249983	-1.562935	-0.173743	-0.656043	-0.140595	-0.028289
1	-0.093463	-0.627913	-0.646365	-1.562935	-0.160024	-0.656043	-0.258383	-0.031520

In [131]:
```
# Test a range of k-neighbors from 40 to 80. There will be 2 k-NN models.
n_clf = 3
k_list = [40, 60, 80]

# Just prepare data frames so we can store the model results
train_scores = np.zeros([X_train.shape[0], n_clf])
test_scores = np.zeros([new_X.shape[0], n_clf])
print(train_scores.shape)
print(test_scores.shape)

# Modeling
for i in range(n_clf):
    k = k_list[i]
    clf = KNN(n_neighbors=k, method='largest')
    clf.fit(X_train)

# Store the results in each column:
    train_scores[:, i] = clf.decision_scores_
    test_scores[:, i] = clf.decision_function(new_X)

# Decision scores have to be normalized before combination
train_scores_norm, test_scores_norm = standardizer(train_scores,test_scores)
```

```
(122300, 3)
(163065, 3)
```

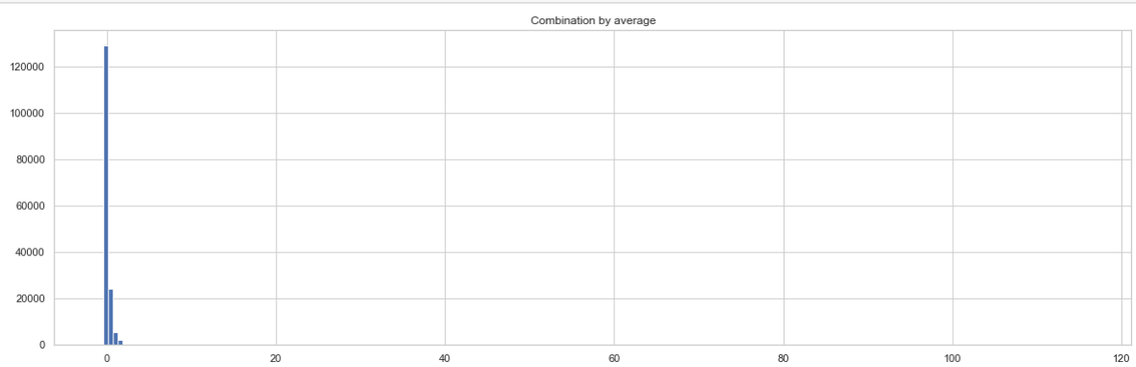
Average Method

In [132]:
```
y_by_average = average(test_scores_norm)
y_by_average
```

Out[132]:
```
array([-0.05351814, -0.27708158,  0.04388025, ..., -0.22499327,
        0.39168762,  0.02768605])
```

Plotting the scores

In [133]:
```
fig, ax = plt.subplots(figsize= (20,6))
plt.hist(y_by_average, bins=200) # arguments are passed to np.histogram
plt.title("Combination by average")
plt.show()
```

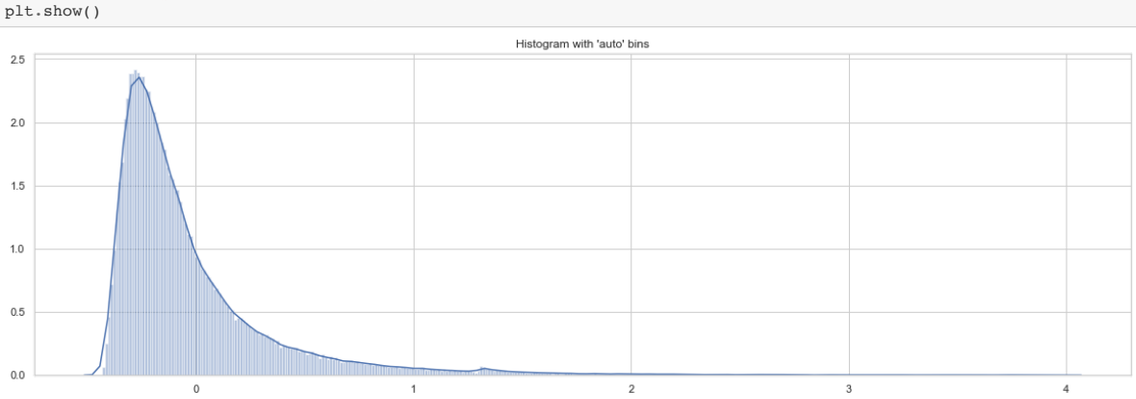


In the above graph, it is very difficult to see the distribution, as the range is very vast. So, I will cut the range at 4.0 and then re-plot

In [134]:
```
y_by_average_small = y_by_average[y_by_average[:, ] < 4.0]
y_by_average_small
```

Out[134]:
```
array([-0.05351814, -0.27708158,  0.04388025, ..., -0.22499327,
        0.39168762,  0.02768605])
```

In [135]:
```
fig, ax = plt.subplots(figsize= (20,6))
sns.distplot(y_by_average_small, bins = 'auto')  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")
plt.show()
```



Observation:

A high anomaly score probably means more abnormal transaction/score. The histogram above shows there are outliers. I will now chose a range of cut points, to check the distribution of clusters, and then identify some as suspicious.

Reasonable Boundary

Generally, looking at the graph, 1.0 seems the ideal number to be a cut point or the reasonable boundary. If we choose 2.0 to be the cut point, we can suggest those >= 2.0 to be outliers.

But, I want to investigate deeper, and I will chose 3 diifferent cut points, which are:

- 0.0
- 1.0
- 5.0

Now, lets evaluate all these 3 different cut points, and build 4-cluster model.

Build a 4 cluster model as per above reasonable boundaries

In [136]:
```
df_test = pd.DataFrame(new_X)
df_test['y_by_average_score'] = y_by_average
df_test['y_by_average_cluster'] = (np.where(df_test['y_by_average_score'] > 5.0, 4,
                                        (np.where(df_test['y_by_average_score'] > 1.0, 3,
                                            (np.where(df_test['y_by_average_score'] > 0.0, 2, 1))))))

df_test['y_by_average_cluster'].value_counts()
```

Out[136]:
```
1    112570
2     44630
3      5609
4       256
Name: y_by_average_cluster, dtype: int64
```

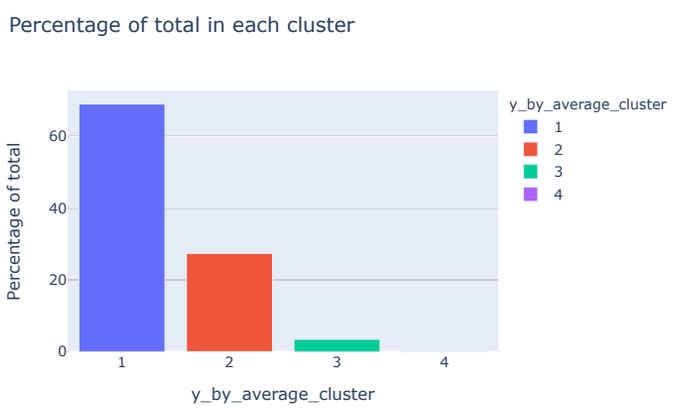
Percentage of data points in each cluster

In [137]:
```
temp = pd.DataFrame()

temp['Percentage of total'] = (df_test['y_by_average_cluster'].value_counts() / df_test['y_by_average_cluster'].value_
counts().sum()) * 100
temp['y_by_average_cluster'] = ['1','2','3','4']

temp
```

Out[137]:

	Percentage of total	y_by_average_cluster
1	69.033821	1
2	27.369454	2
3	3.439733	3
4	0.156993	4

In [138]:
```
fig = px.bar(temp, x = 'y_by_average_cluster', y = 'Percentage of total', color = 'y_by_average_cluster',
             width=600, height=400,
            title = "Percentage of total in each cluster")
fig.show()
```



Final Summary statistics

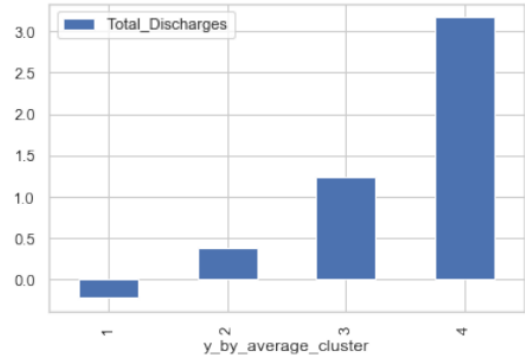
In [139]:
```
cluster_df = df_test.groupby('y_by_average_cluster').mean().reset_index()
cluster_df
```

Out[139]:

	y_by_average_cluster	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Scor
0	1	-0.221173	-0.283433	0.061537	-0.022363	-0.256388	0.033268	-0.25252
1	2	0.384266	0.492422	-0.066014	0.078355	0.325327	-0.077728	0.43998
2	3	1.236714	1.643783	-0.652100	-0.200777	2.240825	-0.039097	1.40880
3	4	3.167817	2.770374	-1.263044	0.572458	6.927906	-0.221593	3.46687

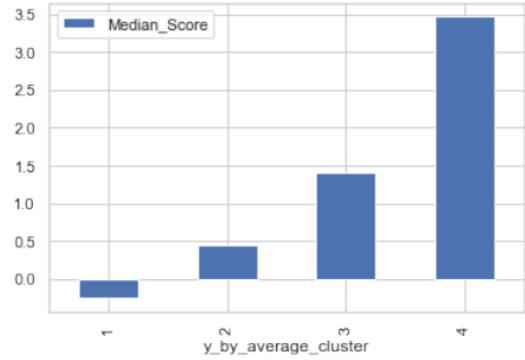
In [140]:
```
cluster_df.plot(x = 'y_by_average_cluster', y = 'Total_Discharges', kind = 'bar')
```

Out[140]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabed2ae810>
```



In [141]:
```
cluster_df.plot(x = 'y_by_average_cluster', y ='Median_Score' , kind ='bar')
```

Out[141]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabed6a6610>
```



Observation:

From the above 4 cluster analysis, the following can be explained:

A. Clusters 3, and 4 have a very small percentage of total data in their cluster pool. So, there are two possible options for these clusters:

- Either they are true anomalies
- They are provider of rare services, or very expensive operations, surgeries or services, performed only by very limited number of providers in the country.

B. Cluster 4 has a roughly 0.15%, and Cluster 3 has roughly 3.4% of the data in its cluster pool. These are the critical cut-off points, as I believe anything near or less than 10% of the entire data, represents a different but normal cluster group.

C. The other clusters which have over 10% of the total data in their cluster pools, seem normal and fine. The differences lie probably due to state or the drg definition or the state. But they are not not likely to be a cause for anomalies.

ANOMALIES or SUSPICIOUS clusters:

The above table gives a comprehensive view of the means, as per the different clusters which is useful to identofy anomalies.

From the final 8 features used for clustering, I generated 4 different clusters using the aggregated average method. The following are the key takeaways for anomalies and suspicious clusters:

`Total Discharges`

1. Clusters 4 has an extremely high value, away from the normal mean of other clusters. Cluster 4 has a mean almost 3-4 times the mean of other clusters, which is very a big concern, even though this cluster has like 0.15% of the total data points.

`Average Total Payments`

1. Cluster 4 has extremely high values. Cluster 4, which has around 0.15% of the total data points, has a mean of USD 2.77, which is over 2-3 times the means of the other clusters.

`Out of Pocket Payment`

1. Cluster 4 has extremely high values. Cluster 4, which has around 0.15% of the total data points, has a mean of over USD 6.2, which is over 6-7 times the means of the other clusters.

`Median Score`

1. Cluster 4 has the highest mean amongst all other clusters means.

From the above observations, it is clear that there is 1 distinct cluster, which has a very high cluster mean for a lot of variables, and emerges as suspicious cases or has anomalies. This is cluster 4.

Combining these results with the Percentage results obtained in the previous step, I can concule that Clusters 4 and maybe 3 are suspicious and need further investigation.

BUSINESS INSIGHTS

I performed the kNN clustering for the data, and also used the 'Average' aggregate method to build a stable model. On building the average model, I found that the original model I built was stable and had good insights.

My analysis gave me 4 clusters, and out of those 4, Cluster 4 seems to be the one which has a lot a potential anomalies, and is suspicious because the means for the variables, as explained above, are far away from the means of the variables of other clusters.

Further, on evaluating the y-by-average-score for all the clusters, which gives us insights about the clusters which are anomalies, as the anomalies might have a very very high score comapred to others, I see that cluster 4 has a score almost 13-14 times higher than all other clusters. So, I can safely conclude that Cluster 4 is highly suspicious.

So, I would pass on the 256 specific entries of the Cluster 4 to the relevant authorities, and call for further investigation on each of the entries, to understand of they are true anomalies. I will provide all the reasoning as I have highlighted above, as to the differences in the means, and walk through the process I have done.

5.2. PCA

Scaling all float or integer variables

I will use Standard Scalar to scale all the numerical variables

In [142]:
```
X = StandardScaler().fit_transform(df1)
X = pd.DataFrame(data = X, columns = num_variables)
X.head()
```

Out[142]:

	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_Pop
0	0.943640	-0.512776	-0.249983	-1.562935	-0.173743	-0.656043	-0.140595	-0.028289
1	-0.093463	-0.627913	-0.646365	-1.562935	-0.160024	-0.656043	-0.258383	-0.031520
2	0.806665	0.072115	0.533658	-1.562935	-0.104393	-0.656043	-0.321171	-0.028715
3	2.469943	-0.412988	-0.173291	-1.562935	-0.095291	-0.656043	-0.093474	-0.023532
4	-0.191303	-0.666841	-0.673441	-1.562935	-0.205142	-0.656043	-0.166491	-0.031825

In [143]:
```
X.isnull().sum().sum()
```

Out[143]: 0

Split into Train Test, to perform the PCA clustering

In [144]:
```
X.shape[0] * 0.75
```

Out[144]: 122298.75

In [145]:
```
X_train = X.iloc[:122300,:]
X_test = X.iloc[122301:,:]
print("Shape of new dataframes - {} , {}".format(X_train.shape, X_test.shape))
```

```
Shape of new dataframes - (122300, 8) , (40764, 8)
```

PCA Model

In [146]:
```
# train PCA detector

clf_name = 'PCA'
clf = PCA()
clf.fit(X_train)
```

Out[146]:
```
PCA(contamination=0.1, copy=True, iterated_power='auto', n_components=None,
  n_selected_components=None, random_state=None, standardization=True,
  svd_solver='auto', tol=0.0, weighted=True, whiten=False)
```

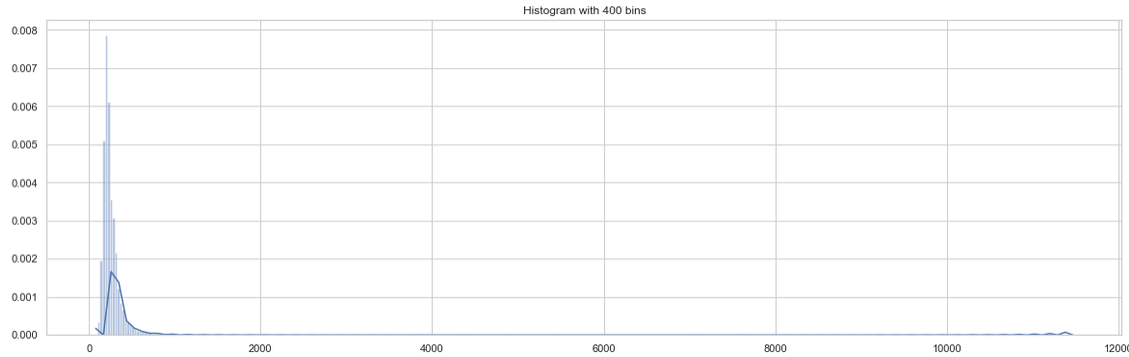
For the purpose of testing the trained set, I will use the entire dataset, which X. This is because even the train dataset containes outliers, and I need to idenify outliers in the entire dataset.

In [147]:
```
# And you can generate the anomaly score using clf.decision_function:

x_test_scores = clf.decision_function(X)
x_test_scores
```

Out[147]:
```
array([235.32912495, 233.10543446, 226.53901866, ..., 239.11148843,
       239.83864843, 338.76485958])
```

Plotting the Scores

In [148]:
```
fig, ax = plt.subplots(figsize= (20,6))
sns.distplot(x_test_scores, bins = 400)  # arguments are passed to np.histogram
plt.title("Histogram with 400 bins")
plt.show()
```

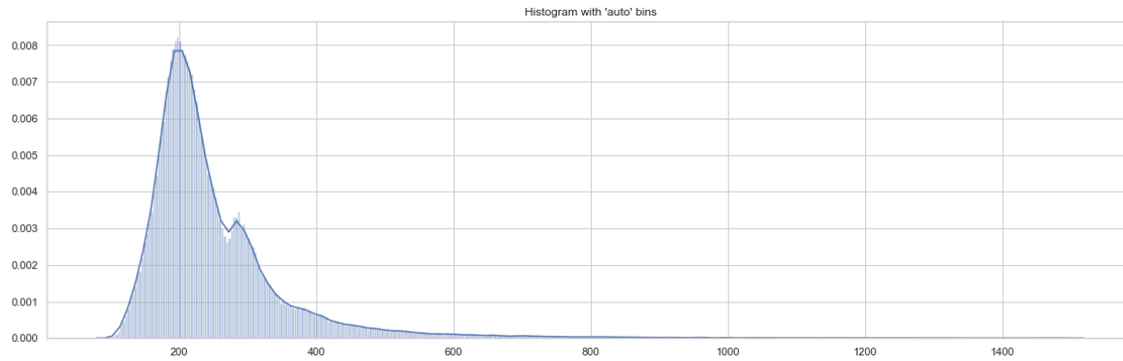


In the above graph, it is very difficult to see the distribution, as the range is very vast. So, I will cut the range and then re-plot

In [149]:
```
x_test_scores_small = x_test_scores[x_test_scores[:, ] <1500]
x_test_scores_small
```

Out[149]:
```
array([235.32912495, 233.10543446, 226.53901866, ..., 239.11148843,
       239.83864843, 338.76485958])
```

In [150]:
```
fig, ax = plt.subplots(figsize= (20,6))
sns.distplot(x_test_scores_small, bins = 'auto')  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")
plt.show()
```



Observation:

A high anomaly score probably means more abnormal transaction/score. The histogram above shows there are outliers. I will now chose a range of cut points, to check the distribution of clusters, and then identify some as suspicious.

Reasonable Boundary

Generally, looking at the graph, 600 seems the ideal number to be a cut point or the reasonable boundary. If we choose 600 to be the cut point, we can suggest those >= 600 to be outliers.

But, I want to investigate deeper, and I will chose 4 diifferent cut points, which are:

- 400
- 800
- 2000

Now, lets evaluate all these 3 different cut points, and build 4-cluster model.

Build a 4 cluster model as per above boundaries

In [151]:
```
df_test = pd.DataFrame(X)
df_test['score'] = x_test_scores

df_test['cluster'] = (np.where(df_test['score'] > 2000, 4,
                                        (np.where(df_test['score'] > 800, 3,
                                            (np.where(df_test['score'] > 400, 2, 1))))))

df_test['cluster'].value_counts()
```

Out[151]:
```
1    151001
2     10655
3      1298
4       111
Name: cluster, dtype: int64
```

In [152]:
```
temp = pd.DataFrame()

temp['Percentage of total'] = (df_test['cluster'].value_counts() / df_test['cluster'].value_counts().sum()) * 100
temp['Cluster'] = ['1','2','3','4']

temp
```

Out[152]:

	Percentage of total	Cluster
1	92.601723	1
2	6.534204	2
3	0.796002	3

4	0.068071	4

Summary Statistics of the clusters

In [153]:
```
# Now let's show the summary statistics:
df_test.groupby('cluster').mean()
```

Out[153]:

cluster	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_
1	-0.067901	-0.147091	0.028336	-0.012953	-0.125839	-0.010172	-0.095256	-0.02
2	0.643581	1.683242	-0.255198	0.158533	1.130013	0.133454	1.057700	0.02
3	2.307686	3.034456	-1.050996	0.153992	4.460853	0.112102	2.069451	0.98
4	3.607354	3.037997	-1.760167	0.602407	10.552194	-0.283137	3.854493	20.33

Note:

Outlier Detection Models Are Prone to Outliers

Also, Outlier detection models tend to be very sensitive to outliers. In our case the unsupervised k-NN model can easily commit overfitting. We need to produce stable models.

Achieve Model Stability by Aggregating Multiple Models

The solution is to train multiple models then aggregate the scores. By aggregating multiple models, the chance of overfitting is greatly reduced and the prediction accuracy will be improved. The PyOD module offers four methods to aggregate the outcome. I will use the 'Average' method for my analysis.

Aggregate by 'Average' Method to get better results

In [154]:
```
new_X = X.drop(columns = ['score', 'cluster'])
new_X.head(2)
```

Out[154]:

	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Score	Total_Disc_by_Pop
0	0.943640	-0.512776	-0.249983	-1.562935	-0.173743	-0.656043	-0.140595	-0.028289
1	-0.093463	-0.627913	-0.646365	-1.562935	-0.160024	-0.656043	-0.258383	-0.031520

In [155]:
```
new_X.shape
```

Out[155]:
```
(163065, 8)
```

In [156]:
```
X_train.shape
```

Out[156]:
```
(122300, 8)
```

In [157]:
```
# Standardize data
# X_train_norm, X_test_norm = standardizer(X_train, X)

# Test a range of k-neighbors from 40 to 80. There will be 2 k-NN models.
n_clf = 4
k_list = [2, 3, 4, 5]

# Just prepare data frames so we can store the model results
train_scores = np.zeros([X_train.shape[0], n_clf])
test_scores = np.zeros([new_X.shape[0], n_clf])
train_scores.shape

# Modeling
for i in range(n_clf):
    k = k_list[i]
    clf = PCA(n_components=k)
    clf.fit(X_train)

# Store the results in each column:
    train_scores[:, i] = clf.decision_scores_
    test_scores[:, i] = clf.decision_function(new_X)

# Decision scores have to be normalized before combination
train_scores_norm, test_scores_norm = standardizer(train_scores,test_scores)
```

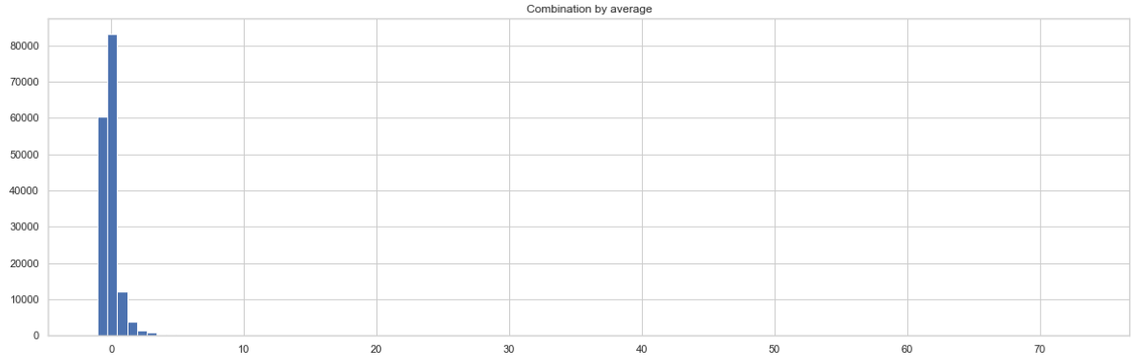
Average Method

In [158]:
```
# Combination by average

y_by_average = average(test_scores_norm)
y_by_average
```

Out[158]:
```
array([-0.03994419, -0.04359613, -0.07156694, ..., -0.02900731,
       -0.05461565,  0.45274169])
```

Plotting the scores

In [159]:
```
fig, ax = plt.subplots(figsize= (20,6))
plt.hist(y_by_average, bins=100) # arguments are passed to np.histogram
plt.title("Combination by average")
plt.show()
```

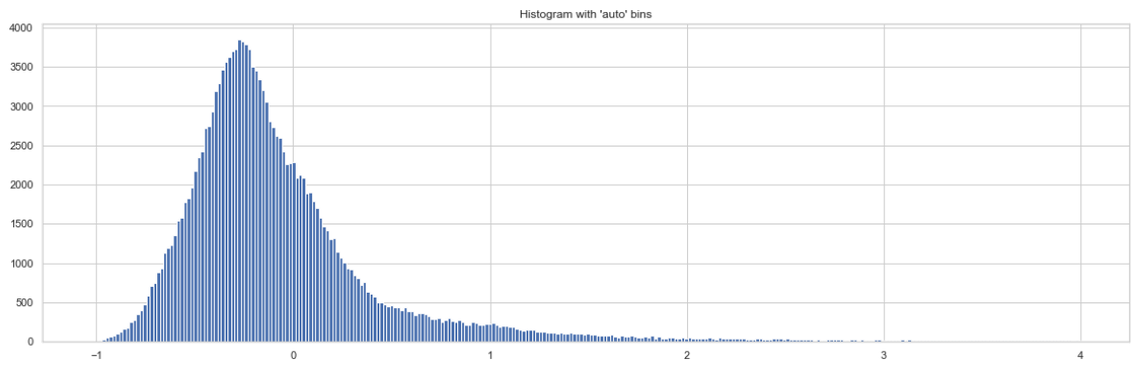


In the above graph, it is very difficult to see the distribution, as the range is very vast. So, I will cut the range and then re-plot

In [160]:
```
y_by_average_small = y_by_average[y_by_average[:, ] < 4.0]
y_by_average_small
```

Out[160]:
```
array([-0.03994419, -0.04359613, -0.07156694, ..., -0.02900731,
       -0.05461565,  0.45274169])
```

In [161]:
```
fig, ax = plt.subplots(figsize= (20,6))
plt.hist(y_by_average_small, bins = 'auto')  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")
plt.show()
```



Observation:

A high anomaly score probably means more abnormal transaction/score. The histogram above shows there are outliers. I will now chose a range of cut points, to check the distribution of clusters, and then identify some as suspicious.

Reasonable Boundary

Generally, looking at the graph, 2.0 seems the ideal number to be a cut point or the reasonable boundary. If we choose 2.0 to be the cut point, we can suggest those >= 2.0 to be outliers.

But, I want to investigate deeper, and I will chose 3 diifferent cut points, which are:

- 1.0
- 2.0
- 5.0

Now, lets evaluate all these 3 different cut points, and build 4-cluster model.

Build a 4 cluster model as per above reasonable boundaries

In [162]:
```
df_test = pd.DataFrame(new_X)
df_test['y_by_average_score'] = y_by_average
df_test['y_by_average_cluster'] = (np.where(df_test['y_by_average_score'] > 5.0, 4,
                                    (np.where(df_test['y_by_average_score'] > 2.0, 3,
                                        (np.where(df_test['y_by_average_score'] > 1.0, 2, 1))))))

df_test['y_by_average_cluster'].value_counts()
```

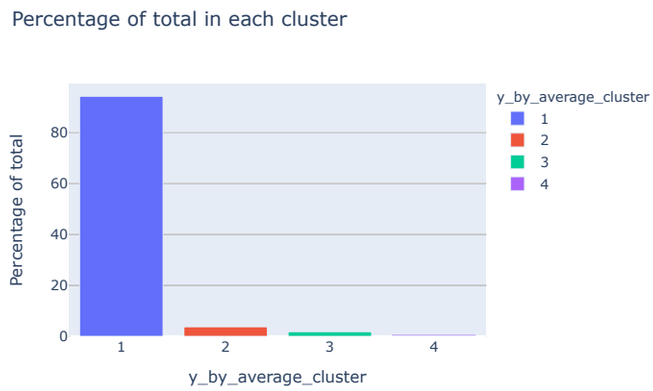
Out[162]:
```
1    153469
2      6059
3      2905
4       632
Name: y_by_average_cluster, dtype: int64
```

In [163]:
```
temp = pd.DataFrame()

temp['Percentage of total'] = (df_test['y_by_average_cluster'].value_counts() / df_test['y_by_average_cluster'].value_
counts().sum()) * 100
temp['y_by_average_cluster'] = ['1','2','3','4']

temp
```

Out[163]:

	Percentage of total	y_by_average_cluster
1	94.115230	1
2	3.715696	2
3	1.781498	3
4	0.387576	4

In [164]:
```
fig = px.bar(temp, x = 'y_by_average_cluster', y = 'Percentage of total', color = 'y_by_average_cluster',
             width=600, height=400,
            title = "Percentage of total in each cluster")
fig.show()
```



Summary Statistics of the clusters

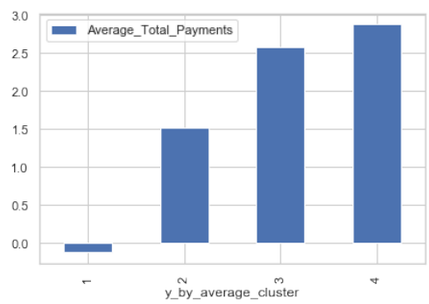
In [165]:
```
cluster_df = df_test.groupby('y_by_average_cluster').mean().reset_index()
cluster_df
```

Out[165]:

	y_by_average_cluster	Total_Discharges	Average_Total_Payments	Medicare_%_Paid	Medicare_%_Paid_State	Out_of_Pocket_Payment	State_Total	Median_Scor
0	1	-0.061422	-0.120624	0.026056	-0.004790	-0.112412	-0.005606	-0.08040
1	2	0.590131	1.522103	-0.208337	0.007671	0.944993	0.062190	1.00408
2	3	1.311465	2.572240	-0.606691	0.187180	2.530000	0.169234	1.57154
3	4	3.229441	2.875316	-1.541189	0.229214	6.608139	-0.012786	2.67493

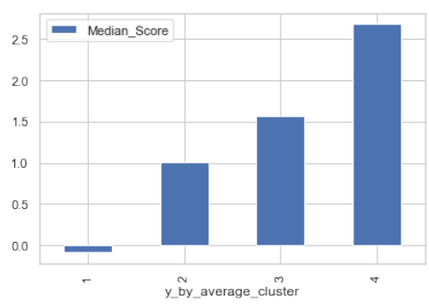
In [166]:
```
cluster_df.plot(x = 'y_by_average_cluster', y ='Average_Total_Payments' , kind ='bar')
```

Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac45a42ad0>



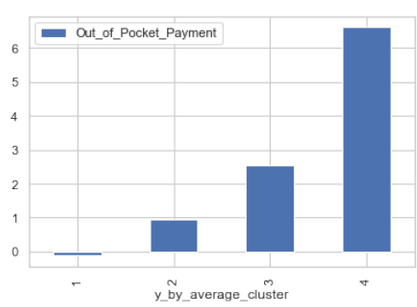
In [167]:
```
cluster_df.plot(x = 'y_by_average_cluster', y ='Median_Score' , kind ='bar')
```

Out[167]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac465bea10>



In [168]:
```
cluster_df.plot(x = 'y_by_average_cluster', y ='Out_of_Pocket_Payment' , kind ='bar')
```

Out[168]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac465e7690>



Observation:

From the above 4 cluster analysis, the following can be explained:

A. Clusters 3, and 4 have a very small percentage of total data in their cluster pool. So, there are two possible options for these clusters:

- Either they are true anomalies
- They are provider of rare services, or very expensive operations, surgeries or services, performed only by very limited number of providers in the country.

B. Cluster 4 has a roughly 0.38%, and Cluster 3 has roughly 1.78% of the data in its cluster pool. These are the critical cut-off points, as I believe anything near or less than 10% of the entire data, represents a different but normal cluster group.

C. The other clusters which have over 10% of the total data in their cluster pools, seem normal and fine. The differences lie probably due to state or the drg definition or the state. But they are not not likely to be a cause for anomalies.

ANOMALIES or SUSPICIOUS clusters:

The above table gives a comprehensive view of the means, as per the different clusters which is useful to identofy anomalies.

From the final 8 features used for clustering, I generated 4 different clusters using the aggregated average method. The following are the key takeaways for anomalies and suspicious clusters:

`Total Discharges`

1. Clusters 4 has an extremely high value, away from the normal mean of other clusters. Cluster 4 has a mean almost 3-4 times the mean of other clusters, which is very a big concern, even though this cluster has like 0.38% of the total data points.

`Out of Pocket Payment`

1. Cluster 4 has extremely high values. Cluster 4, which has around 0.38% of the total data points, has a mean of over USD 66, which is over 6-7 times the means of the other clusters.

`Median Score`

1. Cluster 4 has the highest mean amongst all other clusters means, almost 6-7 times higher.

From the above observations, it is clear that there is 1 distinct cluster, which has a very high cluster mean for a lot of variables, and emerges as suspicious cases or has anomalies. This is cluster 4.

Combining these results with the Percentage results obtained in the previous step, I can concule that Clusters 4 and maybe 3 are suspicious and need further investigation.

BUSINESS INSIGHTS

I performed the PCA clustering for the data, and also used the 'Average' aggregate method to build a stable model. On building the average model, I found that the original model I built was stable and had good insights.

My analysis gave me 4 clusters, and out of those 4, Cluster 4 seems to be the one which has a lot a potential anomalies, and is suspicious because the means for the variables, as explained above, are far away from the means of the variables of other clusters.

Further, on evaluating the y-by-average-score for all the clusters, which gives us insights about the clusters which are anomalies, as the anomalies might have a very very high score comapred to others, I see that cluster 4 has a score almost 9-10 times higher than all other clusters. So, I can safely conclude that Cluster 4 is highly suspicious.

So, I would pass on the 638 specific entries of the Cluster 4 to the relevant authorities, and call for further investigation on each of the entries, to understand of they are true anomalies. I will provide all the reasoning as I have highlighted above, as to the differences in the means, and walk through the process I have done.

Conclusion:

kNN and PCA are are very good models to identify cluster who might be suspicious or have anomalies. For performing the analysis, I used 4 clusters as the optimum number of clusters. An important observation was that:

- Even though some clusters have avery high in cluster mean, but they contaian a good portion of the total data. Hence we cannot call that cluster as suspicious. So, finally, per my anslysis, I can say that clusters 4 and maybe 3, in both models were suspicious, as they contained fewer than 5% of the total data points, and also, they had high in-cluster means for a lot of variables, which makes them very suspicious.