`Machinehack-ML-Merchandise-Popularity-Prediction`

We need to predict the popularity from all the other data from dataset like Store_Ratio, Basket Ratio, Store Score.

Big Brands spend a significant amount on popularizing a product. Nevertheless, their efforts go in vain while establishing the merchandise in the hyperlocal market. Based on different geographical conditions same attributes can communicate a piece of much different information about the customer. Hence, insights this is a must for any brand owner.

`ABOUT DATASET`

Dataset link: https://www.kaggle.com/datasets/oossiiris/machinehack-ml-merchandise-popularity-prediction Train.csv - 18208 x 12 (Includes popularity Column as Target variable)

Test.csv - 12140 x 11

`COLUMNS IN THE DATASET`

storeratio

basketratio

category1

storescore

category2

storepresence

score1

score2

score3

score4

time

popularity (Target Column)

`Importing required libraries`

In [ ]:

```
# Import all necessary libraries
import seaborn as sns
import numpy as np
from sklearn.metrics import log_loss
import pandas as pd
from scipy.stats import skew,kurtosis,zscore
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stat
from matplotlib import pylab
from pylab import *
from imblearn.over_sampling import SMOTE
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
```

Importing the dataset

In [ ]:

```
# Load the dataset using pandas
train_df =
test_df =
```

In [ ]:

```
# print train head
```

Out[ ]:

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	Score_1	Score_2	Score_3	Score_4	time	popularity
0	0.407	0.00380	2	-35.865	1	0.9920	0.944	0.0988	0.1100	113.911	189125	4
1	0.234	0.10500	0	-19.884	1	0.9440	0.900	0.1290	0.0382	76.332	186513	4
2	0.668	0.72600	9	-7.512	1	0.0387	0.000	0.1340	0.4530	124.075	172143	4
3	0.184	0.00561	4	-34.357	1	0.8490	0.931	0.1110	0.0641	79.037	184000	5
4	0.231	0.13100	6	-22.842	1	0.9370	0.000	0.1090	0.0677	109.560	186507	3

In [ ]:

```
# print test head
```

Out[ ]:

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	Score_1	Score_2	Score_3	Score_4	time
0	0.164	0.994	1	-23.718	0	0.12400	0.99100	0.7890	0.00001	127.602	236436
1	0.586	0.636	4	-7.710	1	0.00208	0.32200	0.1070	0.35500	136.337	253631
2	0.457	0.743	0	-7.591	1	0.40400	0.00931	0.2010	0.68800	154.902	215669
3	0.423	0.748	5	-9.832	1	0.03000	0.22100	0.1690	0.42800	93.977	325200
4	0.802	0.756	2	-10.791	0	0.08430	0.00765	0.0521	0.96300	131.715	288293

In [ ]:

```
# Make a copy of the dataset
df =
```

Identifying the number of features or columns

In [ ]:

```
# Check the shape of train dataset
```

Out[ ]:

```
(18208, 12)
```

In [ ]:

```
# Check the shape of test dataset
```

Out[ ]:

```
(12140, 11)
```

Know all the names of the columns

In [ ]:

```
# Check the columns in the train dataset
```

Out[ ]:

```
Index(['Store_Ratio', 'Basket_Ratio', 'Category_1', 'Store_Score',
       'Category_2', 'Store_Presence', 'Score_1', 'Score_2', 'Score_3',
       'Score_4', 'time', 'popularity'],
      dtype='object')
```

Knows more about the data in the columns like data type it contains and total samples of each

In [ ]:

```
# Check which columns are having categorical, numerical or boolean values of train dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18208 entries, 0 to 18207
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Store_Ratio     18208 non-null  float64
 1   Basket_Ratio    18208 non-null  float64
 2   Category_1      18208 non-null  int64
 3   Store_Score     18208 non-null  float64
 4   Category_2      18208 non-null  int64
 5   Store_Presence  18208 non-null  float64
 6   Score_1         18208 non-null  float64
 7   Score_2         18208 non-null  float64
 8   Score_3         18208 non-null  float64
 9   Score_4         18208 non-null  float64
 10  time            18208 non-null  int64
 11  popularity      18208 non-null  int64
dtypes: float64(8), int64(4)
memory usage: 1.7 MB
```

After checking the Dtypes of all the columns

object - String values

float64 - Numerical values

Observation: There are no String values so there are no categorical data

In [ ]:

```
# Check which columns are having categorical, numerical or boolean values of test dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12140 entries, 0 to 12139
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Store_Ratio     12140 non-null  float64
 1   Basket_Ratio    12140 non-null  float64
 2   Category_1      12140 non-null  int64
 3   Store_Score     12140 non-null  float64
 4   Category_2      12140 non-null  int64
 5   Store_Presence  12140 non-null  float64
 6   Score_1         12140 non-null  float64
 7   Score_2         12140 non-null  float64
 8   Score_3         12140 non-null  float64
 9   Score_4         12140 non-null  float64
 10  time            12140 non-null  int64
dtypes: float64(8), int64(3)
memory usage: 1.0 MB
```

After checking the Dtypes of all the columns

object - String values

float64 - Numerical values

Observation: There are no String values so there are no categorical data

Know more mathematical relations of the dataset like count, min, max values, standarad deviation values, mean and different percentile values

In [ ]:

```
# For more information on the train dataset like the total count in all the columns
# min, max values and more information of the respective columns
```

Out[ ]:

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	Score_1	Score_2	Score_3	
count	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	1
mean	0.544283	0.483585	5.155536	-12.198086	0.648506	0.477702	0.322109	0.164888	0.421440	
std	0.202709	0.302010	3.535068	8.370566	0.477450	0.380634	0.413493	0.136531	0.271922	
min	0.000000	0.000216	0.000000	-47.576000	0.000000	0.000000	0.000000	0.011900	0.000000	
25%	0.411000	0.200000	2.000000	-16.496250	0.000000	0.086175	0.000001	0.095300	0.184750	
50%	0.573000	0.517000	5.000000	-9.166500	1.000000	0.430000	0.002245	0.112000	0.393000	
75%	0.699000	0.742000	8.000000	-5.943750	1.000000	0.895000	0.859000	0.176000	0.640000	
max	0.998000	1.000000	11.000000	-0.079000	1.000000	0.996000	1.000000	0.991000	0.999000	

In [ ]:

```
# For more information on the test dataset like the total count in all the columns
# min, max values and more information of the respective columns
```

Out[ ]:

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	Score_1	Score_2	Score_3	
count	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	1
mean	0.543776	0.488879	5.121417	-12.062847	0.642916	0.474675	0.321641	0.165942	0.420485	
std	0.200109	0.301217	3.528765	8.300385	0.479160	0.377582	0.414438	0.139109	0.270508	
min	0.000000	0.000000	0.000000	-46.847000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.414750	0.213000	2.000000	-16.066000	0.000000	0.087775	0.000001	0.095900	0.190000	
50%	0.570000	0.521000	5.000000	-9.046500	1.000000	0.430500	0.002050	0.112000	0.392000	
75%	0.696000	0.745000	8.000000	-5.891750	1.000000	0.882000	0.861000	0.173250	0.633000	
max	0.978000	1.000000	11.000000	0.662000	1.000000	0.996000	1.000000	0.994000	1.000000	

Get the total number of samples in the dataset using the len() function

In [ ]:

```
# check the lenght of test and train dataset
```

```
train data length:  18208
test data length:  12140
```

Counting the total number of missing value

In [ ]:

```
# Check for missing values in all the columnns of the train dataset
```

Out[ ]:

```
Store_Ratio       0
Basket_Ratio      0
Category_1        0
Store_Score       0
Category_2        0
Store_Presence    0
Score_1           0
Score_2           0
Score_3           0
Score_4           0
time              0
popularity        0
dtype: int64
```

There is no missing values in this dataset

In [ ]:

```
# Check for missing values in all the columnns of the test dataset
```

Out[ ]:

```
Store_Ratio       0
Basket_Ratio      0
Category_1        0
Store_Score       0
Category_2        0
Store_Presence    0
Score_1           0
Score_2           0
Score_3           0
Score_4           0
time              0
dtype: int64
```

There is no missing values in this dataset

Get unique values

In [ ]:

```
# get unique values in train dataset
```

```
Store_Ratio : 1053
Basket_Ratio : 1993
Category_1 : 12
Store_Score : 10758
Category_2 : 2
Store_Presence : 2832
Score_1 : 3668
Score_2 : 1385
Score_3 : 1591
Score_4 : 12943
time : 12755
popularity : 5
```

In [ ]:

```
# get unique values in test dataset
```

```
Store_Ratio : 1010
Basket_Ratio : 1811
Category_1 : 12
Store_Score : 8300
Category_2 : 2
Store_Presence : 2507
Score_1 : 3064
Score_2 : 1275
Score_3 : 1491
Score_4 : 9463
time : 9301
```

EDA

CORRELATION MATRIX

Why ?

A correlation matrix is a table showing correlation coefficients between variables

There are three broad reasons for computing a correlation matrix:

1. To summarize a large amount of data where the goal is to see patterns. In our example above, the observable pattern is that all the variables highly correlate with each other.
2. To input into other analyses. For example, people commonly use correlation matrixes as inputs for exploratory factor analysis, confirmatory factor analysis, structural equation models, and linear regression when excluding missing values pairwise.
3. As a diagnostic when checking other analyses. For example, with linear regression, a high amount of correlations suggests that the linear regression estimates will be unreliable

In [ ]:

```
#correlation
#perfom correlation matrix Using pandas
```

```
corr =
```

Out[ ]:

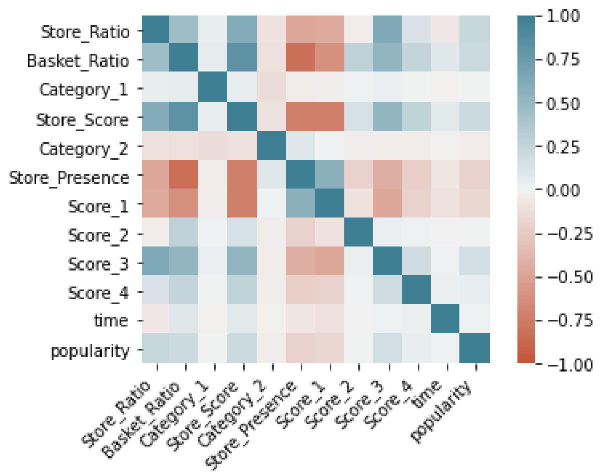
	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	Score_1	Score_2	Score_3	Score_4	time
Store_Ratio	1.00	0.47	0.04	0.60	-0.10	-0.47	-0.45	-0.04	0.62	0.12	-0.06
Basket_Ratio	0.47	1.00	0.05	0.83	-0.11	-0.82	-0.61	0.28	0.53	0.25	0.10
Category_1	0.04	0.05	1.00	0.04	-0.14	-0.05	-0.03	0.02	0.04	0.00	-0.01
Store_Score	0.60	0.83	0.04	1.00	-0.09	-0.72	-0.73	0.15	0.52	0.26	0.08
Category_2	-0.10	-0.11	-0.14	-0.09	1.00	0.10	0.02	-0.04	-0.04	-0.05	-0.01
Store_Presence	-0.47	-0.82	-0.05	-0.72	0.10	1.00	0.57	-0.19	-0.43	-0.22	-0.07
Score_1	-0.45	-0.61	-0.03	-0.73	0.02	0.57	1.00	-0.10	-0.47	-0.20	-0.11
Score_2	-0.04	0.28	0.02	0.15	-0.04	-0.19	-0.10	1.00	0.03	0.02	-0.00
Score_3	0.62	0.53	0.04	0.52	-0.04	-0.43	-0.47	0.03	1.00	0.18	0.01
Score_4	0.12	0.25	0.00	0.26	-0.05	-0.22	-0.20	0.02	0.18	1.00	0.03
time	-0.06	0.10	-0.01	0.08	-0.01	-0.07	-0.11	-0.00	0.01	0.03	1.00
popularity	0.23	0.20	0.00	0.20	-0.03	-0.20	-0.16	0.00	0.18	0.04	0.02

Observations from above correlation matrix

1. store ratio is strongly correlated with basket ratio , store score, score3 ,store presence score1 , score3
2. basket ratio is correlatd to store score, store presence, score1 score3.
3. category1 is correlated with category2
4. score presence is correlated with score1, score2, score3, popularity
5. score1 is correlated with score3 , score4, time, popularity

In [ ]:

```
# Using seaborn
ax =
```



SCATTER PLOT

1. A scatter plot is a type of plot using Cartesian coordinates to display values for typically two variables for a set of data.
2. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.
3. Scatter plot's are used to observe and show relationships between two numeric variables.

In [ ]:

```
# perform scatterplot
# Make a list of all the columns of train dataset
# Loop through the different columns
```

```
def plot_scatter(x, y):

  # code here
```

Observations from above scatter plot

1. As value of store presence increases value of basket ratio decreases.
2. As value of store scrore increases value of basket ratio increases.
3. Increase in value of store ratio results in the increase of basket ratio's value

HISTOGRAM

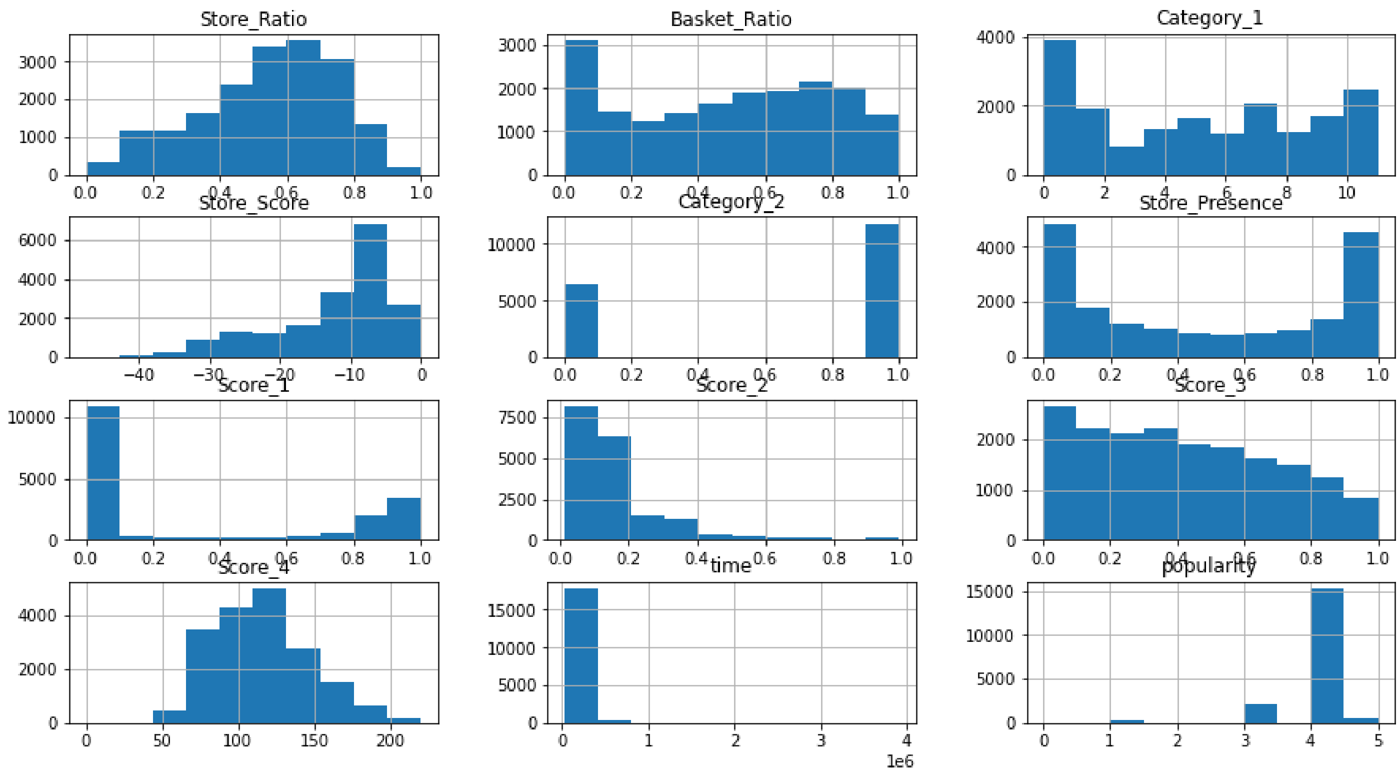
1. A histogram is an approximate representation of the distribution of numerical data.

2. To construct a histogram, the first step is to "bin" (or "bucket") the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval.

3. The words used to describe the patterns in a histogram are: "symmetric", "skewed left" or "right", "unimodal", "bimodal" or "multimodal".

In [ ]:

```
# perform histogram using pandas for all columns of train dataset
```



observation from above histogram

1. The data distribution of store presence is bimodal
2. Score_2, score 1, score3 and time data distribution is skewed left.
3. Store score and popularity data distribution is skewed right.
4. distribution of basket ratio, carwgort 1 is multimodal.
5. distribution of store eatio is bimodal and skewed

VIF - Variance inflation factor

1. The variance inflation factor (VIF) quantifies the extent of correlation between one predictor and the other predictors in a model.
2. It is used for diagnosing collinearity/multicollinearity.
3. Higher values signify that it is difficult to impossible to assess accurately the contribution of predictors to a model.

In [ ]:

```
#import statsmodel.api
```

In [ ]:

```
# creating a dataframe of just numerical values
train_for_vif =


# target values
target =


# numerical values column names
names =


#print names
```

```
['Store_Ratio', 'Basket_Ratio', 'Category_1', 'Store_Score', 'Category_2', 'Store_Presence', 'Score_1', '
Score_2', 'Score_3', 'Score_4', 'time']
```

In [ ]:

```
# droping rows with from new dataframe empty cells
```

Out[ ]:

```
array([False,  True, False, False, False, False, False, False, False,
       False, False])
```

In [ ]:

```
# Calculating VIF for each feature.

  # taking one column as target variable

  # taking all other remaining columns as fetaure variable

  # firting the OLS model on y and x


  # geting the r^2 value of results.

  # calculating vif value

for i in range(0, len(names)):
  y =

  x =

  model =

  results =

  rsq =

  vif =

  print("R Square value of {} columns is {} keeping all other columns as features".format(names[i],(round
  print("Variance inflation Factor of {} columns is {} \n".format(names[i], vif))
```

```
R Square value of Store_Ratio columns is 0.91 keeping all other columns as features
Variance inflation Factor of Store_Ratio columns is 11.74

R Square value of Basket_Ratio columns is 0.92 keeping all other columns as features
Variance inflation Factor of Basket_Ratio columns is 12.13

R Square value of Category_1 columns is 0.68 keeping all other columns as features
Variance inflation Factor of Category_1 columns is 3.11

R Square value of Store_Score columns is 0.9 keeping all other columns as features
Variance inflation Factor of Store_Score columns is 9.87

R Square value of Category_2 columns is 0.65 keeping all other columns as features
Variance inflation Factor of Category_2 columns is 2.83

R Square value of Store_Presence columns is 0.85 keeping all other columns as features
Variance inflation Factor of Store_Presence columns is 6.69

R Square value of Score_1 columns is 0.72 keeping all other columns as features
Variance inflation Factor of Score_1 columns is 3.59

R Square value of Score_2 columns is 0.64 keeping all other columns as features
Variance inflation Factor of Score_2 columns is 2.79

R Square value of Score_3 columns is 0.85 keeping all other columns as features
Variance inflation Factor of Score_3 columns is 6.48

R Square value of Score_4 columns is 0.92 keeping all other columns as features
Variance inflation Factor of Score_4 columns is 12.88

R Square value of time columns is 0.83 keeping all other columns as features
Variance inflation Factor of time columns is 5.87
```

Observations:

there is colinearity/multicolinearity between variables as the VIF value is almost upto 2.5

Store_Ratio, Basket_Ratio, Category_1, Store_Score, Category_2, Store_Presence, Score_1, Score_2, Score_3, Score_4, time they all have colinearity with all the variables.

BOX PLOT

A boxplot is a standardized way of displaying the dataset based on a five-number summary:

1. Minimum (Q0 or 0th percentile): the lowest data point excluding any outliers.

2. Maximum (Q4 or 100th percentile): the largest data point excluding any outliers.

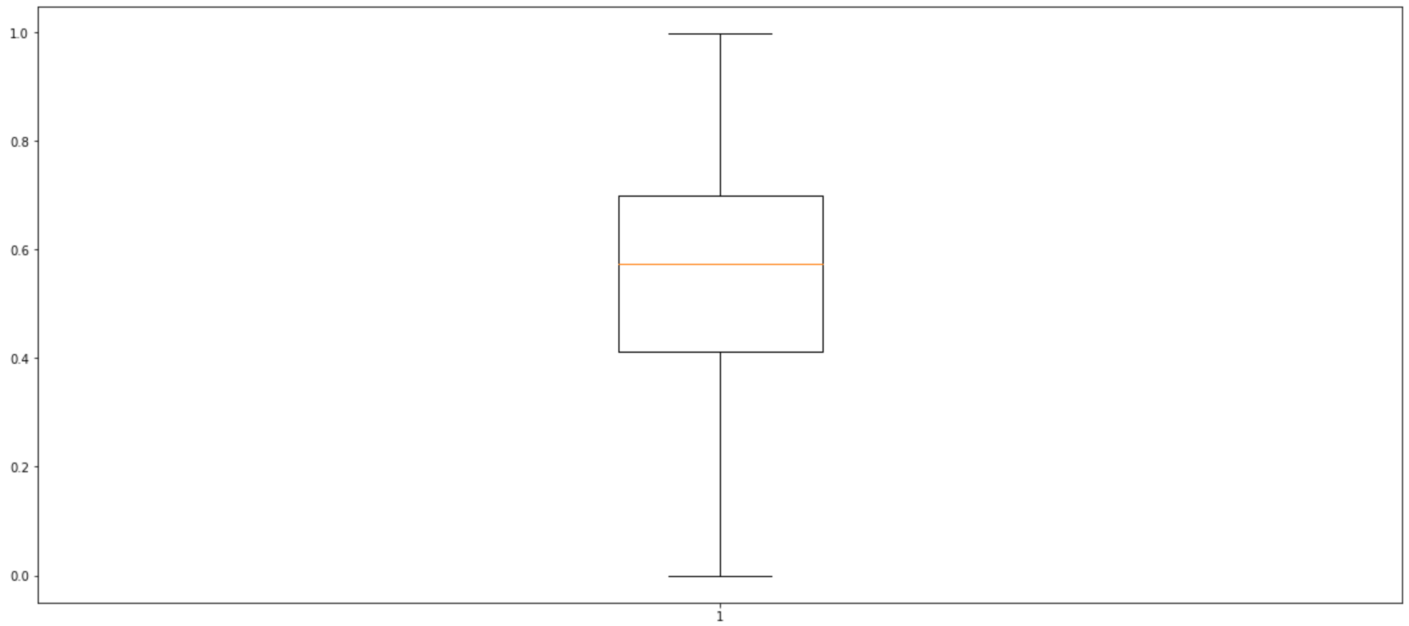
3. Median (Q2 or 50th percentile): the middle value of the dataset.

4. First quartile (Q1 or 25th percentile): also known as the lower quartile qn(0.25), is the median of the lower half of the dataset.

5. Third quartile (Q3 or 75th percentile): also known as the upper quartile qn(0.75), is the median of the upper half of the dataset

In [ ]:

```
# Perform a box plot on Store_Ratio
```



from above box plot graph:

Store ratio

1. 25% of store ratio have value between range 0 to 0.4.

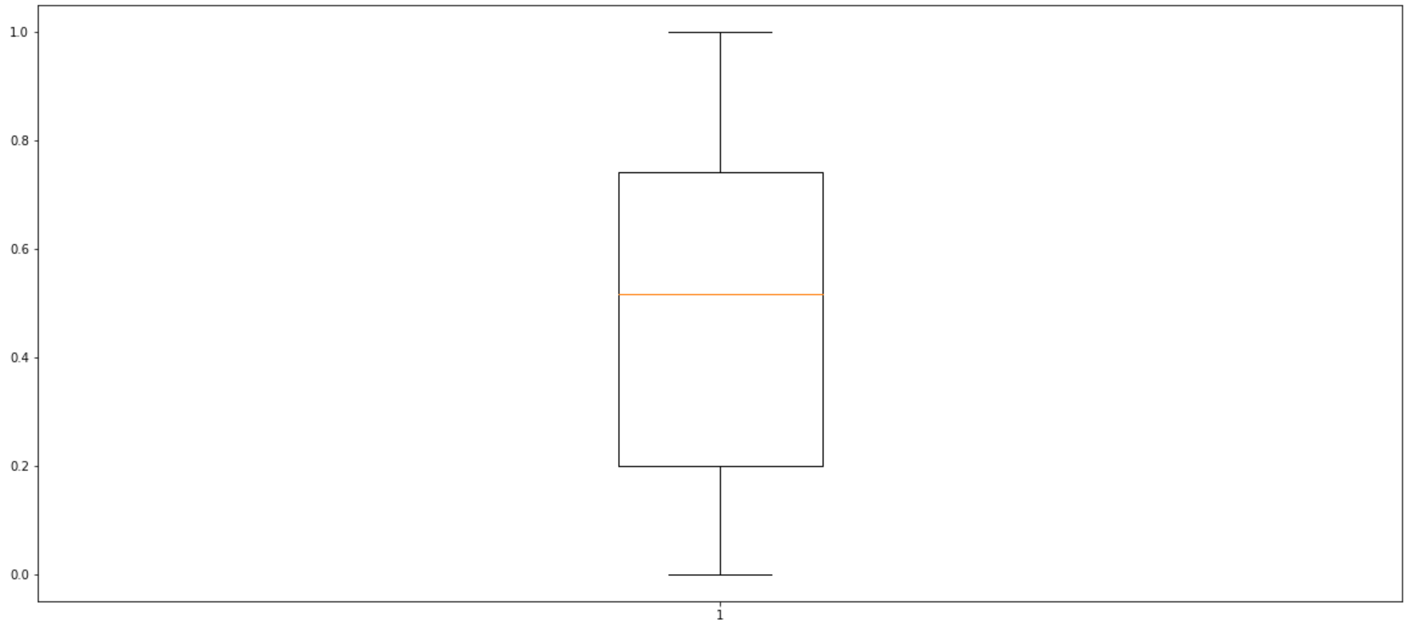
2. 25% of store ratio have value between range 0.4 to 0.6.

3. 25% of store ratio have value between range 0.6 to 0.7.
4. 25% of store ratio have value between range 0.7 to 1.

The mean store ratio is around 6.

In [ ]:

```
# Perform a box plot on Basket_Ratio
```



from above box plot graph:

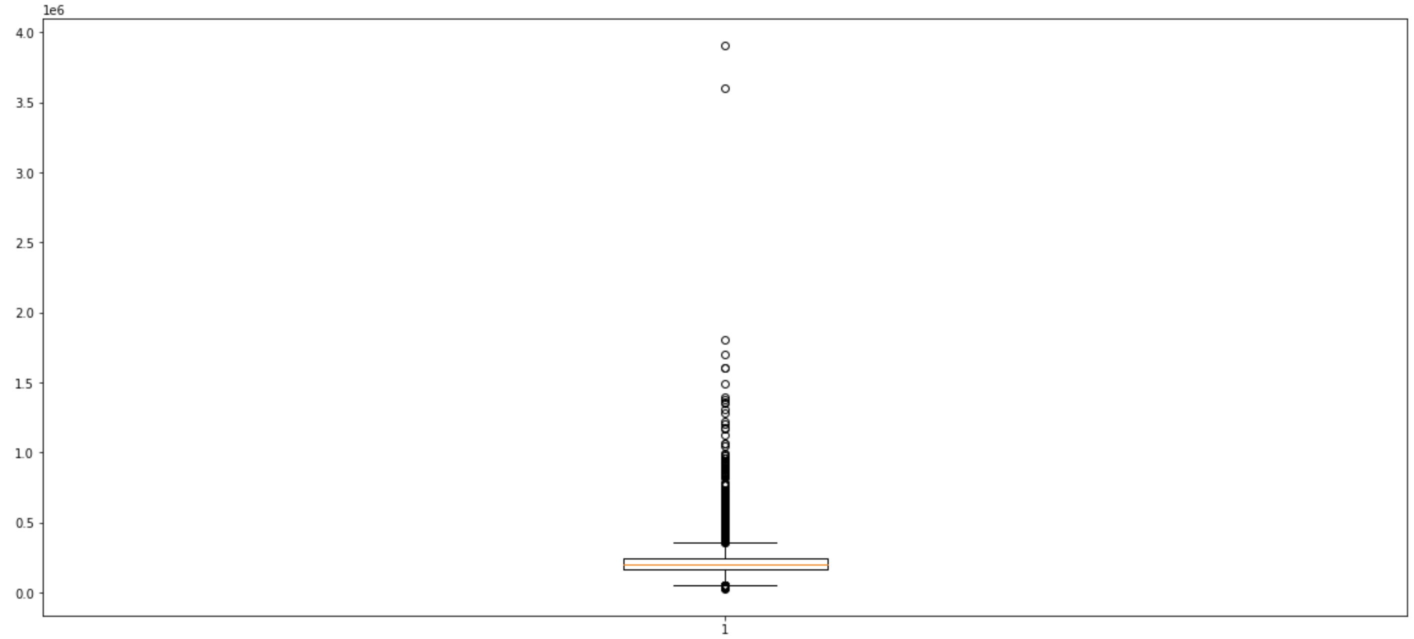
basket ratio

1. 25% of basket ratio have value between range 0 to 0.2.
2. 25% of basket ratio have value between range 0.4 to 0.52
3. 25% of basket ratio have value between range 0.52 to 0.78.
4. 25% of basket ratio have value between range 0.78 to 1.

The mean basket ratio is around 0.52

In [ ]:

```
# Perform a box plot on time
```



from above box plot graph:

time

1. 25% of time have value between range 0 to 0.2.

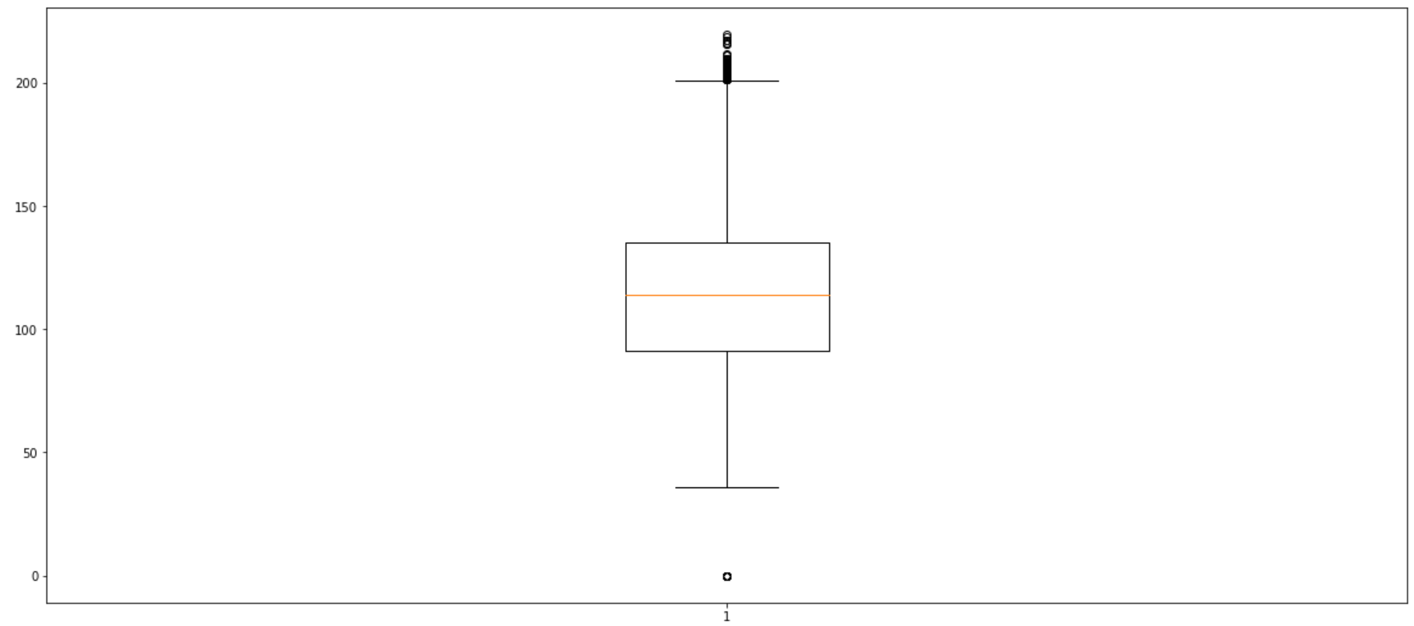
2. 25% of time have value between range 0.2 to 0.25

3. 25% of time have value between range 0.25 to 0.3.
4. 25% of time have value between range 0.3 to 0.4

The mean time is around 0.25

In [ ]:

```
# Perform a box plot on Score_4
```



from above box plot graph:

score 4

1. 25% of score 4 have value between range 40 to 90.

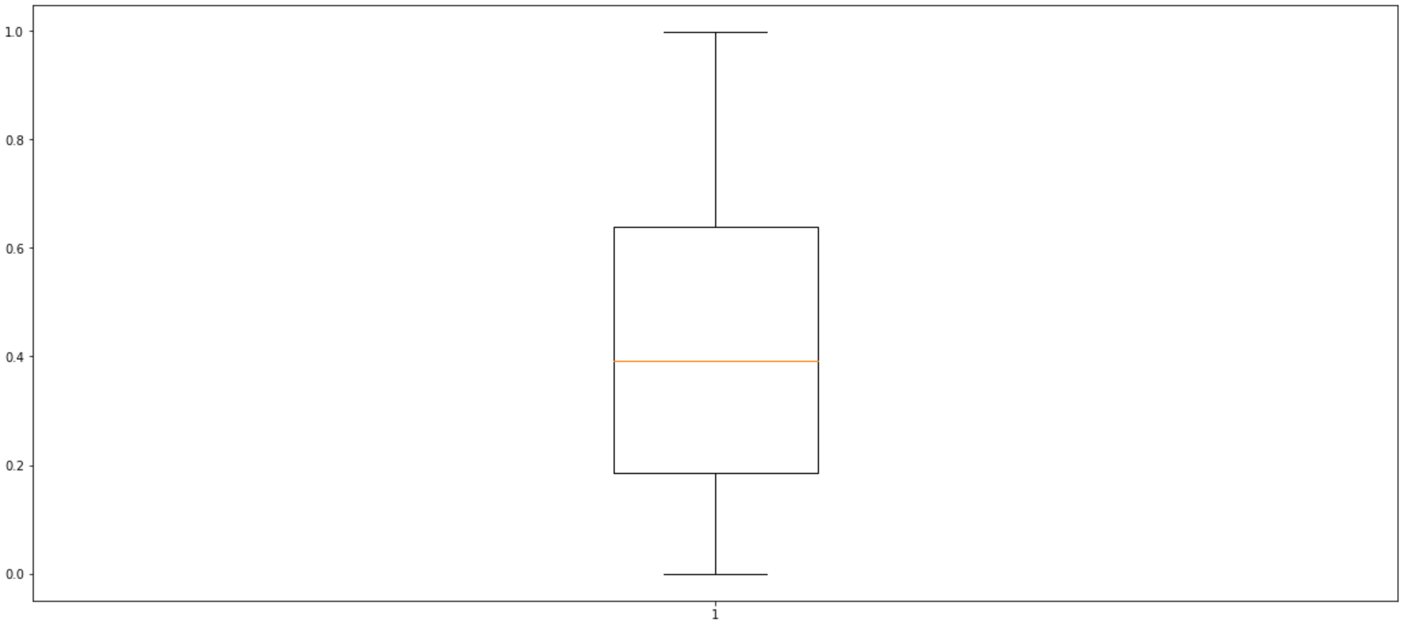
2. 25% of score 4 have value between range 90 to 125

3. 25% of score 4 have value between range 125 to 140.
4. 25% of score 4 have value between range 140 to 200.

The mean score 4 is around 125

In [ ]:

```
# Perform a box plot on Score_3
```



from above box plot graph:

score3

1. 25% of score3 have value between range 0 to 0.2.

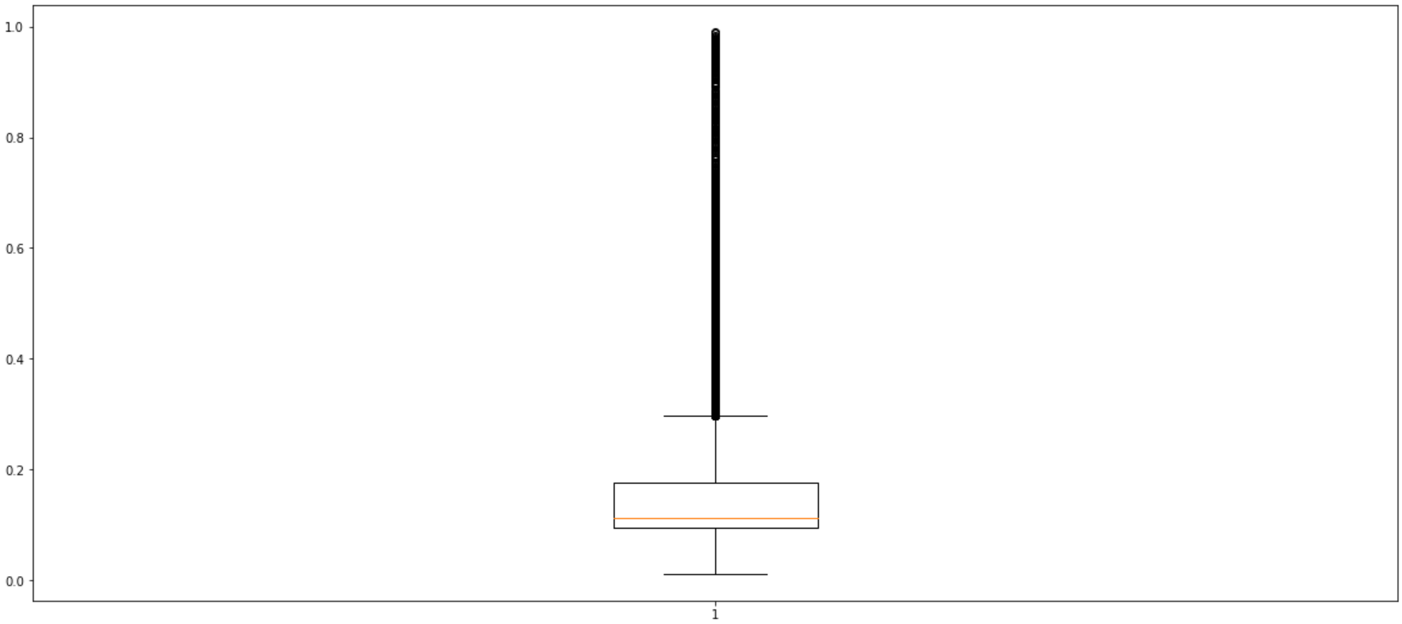
2. 25% of score3 have value between range 0.2 to 0.4

3. 25% of score3 have value between range 0.4 to 0.62.
4. 25% of score3 have value between range 0.62 to 1.

The mean score3 is around 0.4

In [ ]:

```
# Perform a box plot on Score_2
```



from above box plot graph:

score2

1. 25% of score2 have value between range 0 to 0.1.

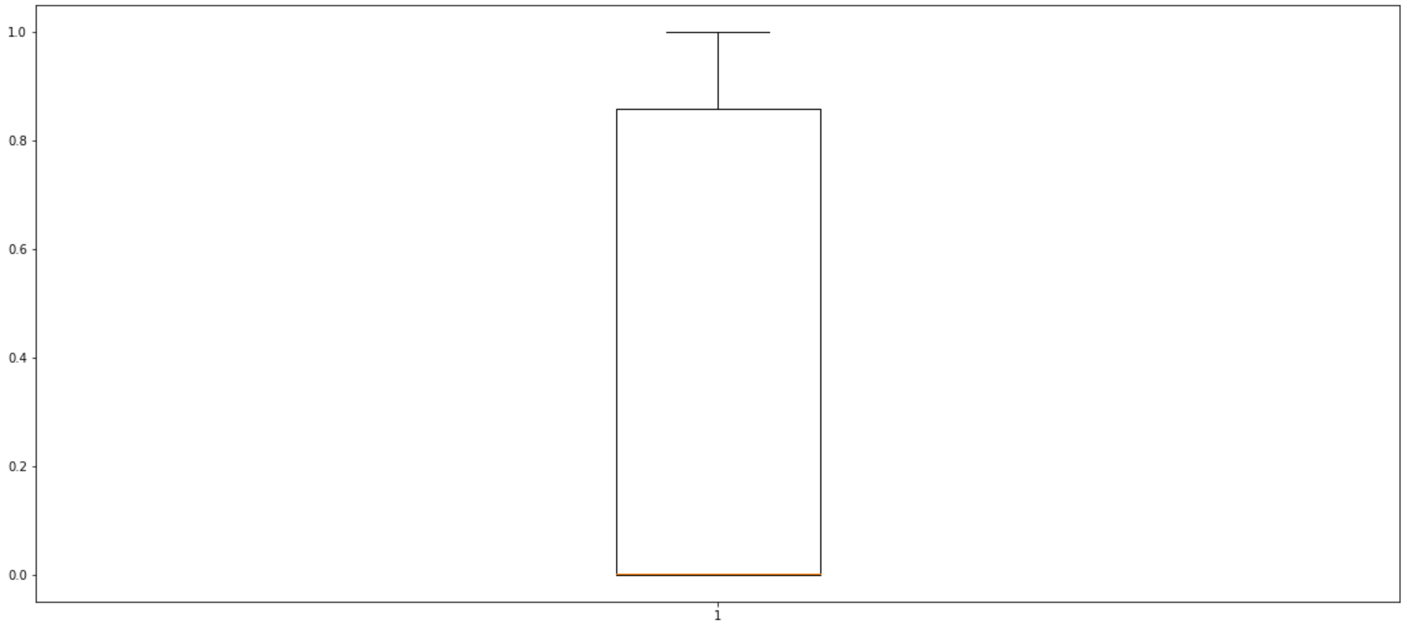
2. 25% of score2 have value between range 0.1 to 0.12

3. 25% of score2 have value between range 0.12 to 0.18.
4. 25% of score2 have value between range 0.18 to 1.

The mean score2 is around 0.12

In [ ]:

```
# Perform a box plot on Score_1
```



from above box plot graph:

score1

1. 25% of score1 have value between range 0 to 0.0.

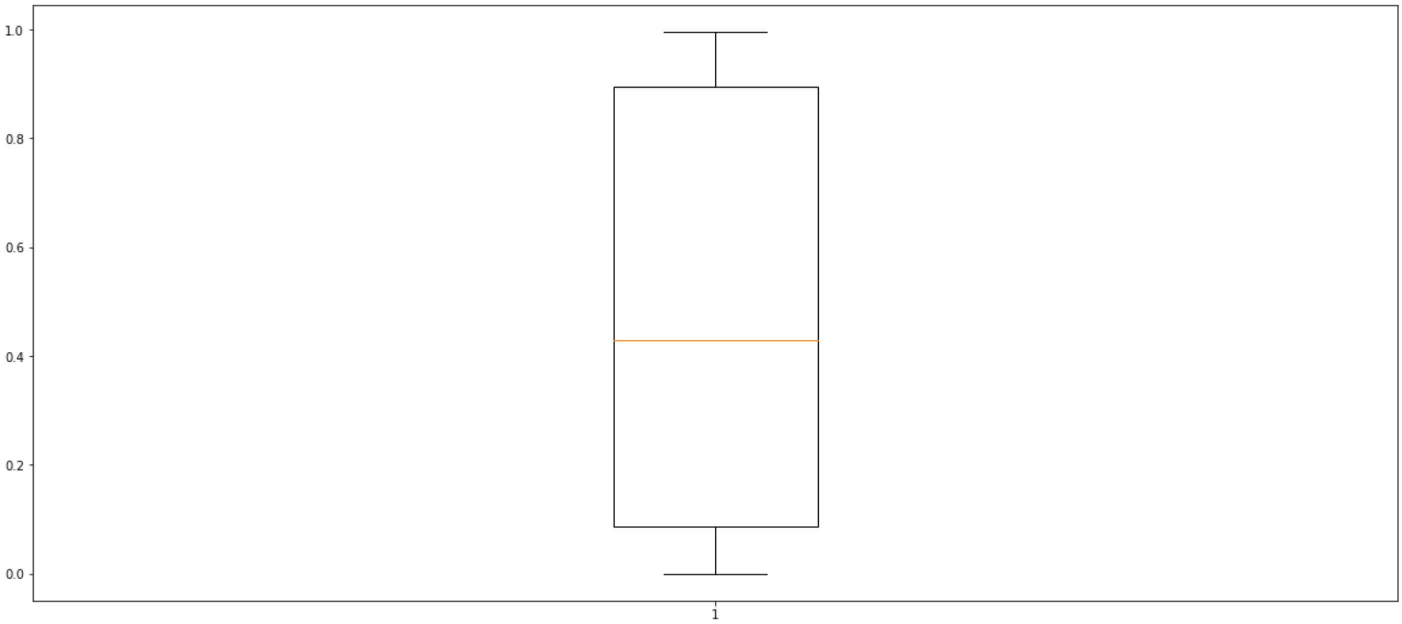
2. 25% of score1 have value between range 0.0 to 0.0

3. 25% of score1 have value between range 0.0 to 0.9
4. 25% of score1 have value between range 0.9 to 1.

The mean score1 is around 0.0

In [ ]:

```
# Perform a box plot on Store_Presence
```



from above box plot graph:

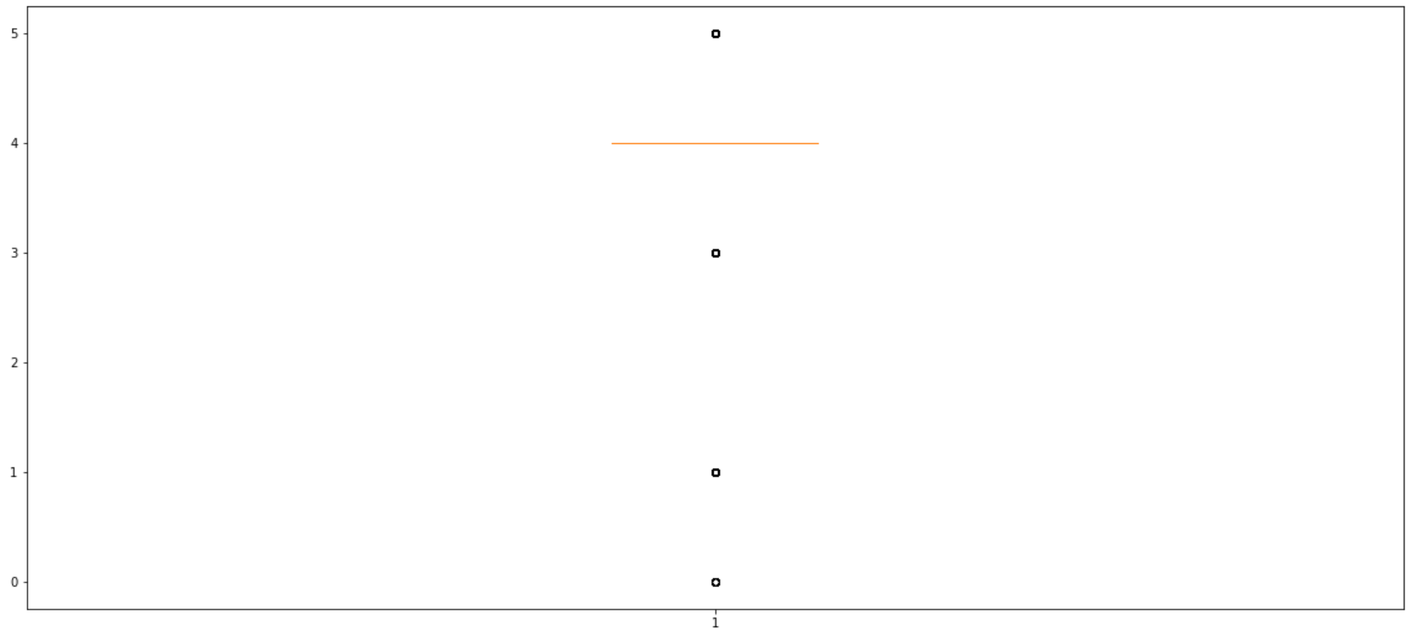
store presence

1. 25% of store presence have value between range 0 to 0.35.
2. 25% of store presence have value between range 0.35 to 0.42
3. 25% of store presence have value between range 0.42 to 0.95.
4. 25% of store presence have value between range 0.95 to 1.

The mean store presence is around 0.42

In [ ]:

```
# Perform a box plot on popularity
```



COUNT PLOT

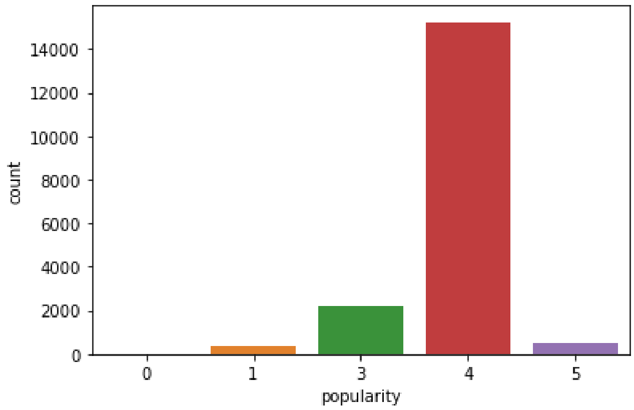
1. A countplot is kind of like a histogram or a bar graph for some categorical area.
2. It simply shows the number of occurrences of an item based on a certain type of category.

In [ ]:

```
# Perform the countplot on the popularity
```

Out[ ]:

<AxesSubplot:xlabel='popularity', ylabel='count'>



From above count plot

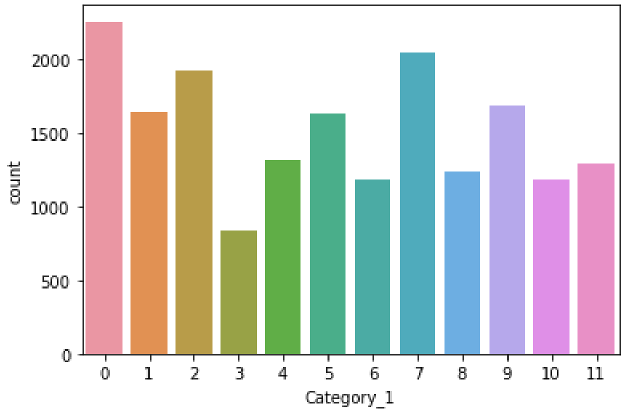
we can observe that the dataset is imbalanced.

In [ ]:

```
# Perform the countplot on the category 1
```

Out[ ]:

<AxesSubplot:xlabel='Category_1', ylabel='count'>



distribution of values over complete dataset is multimodal

point PLOT

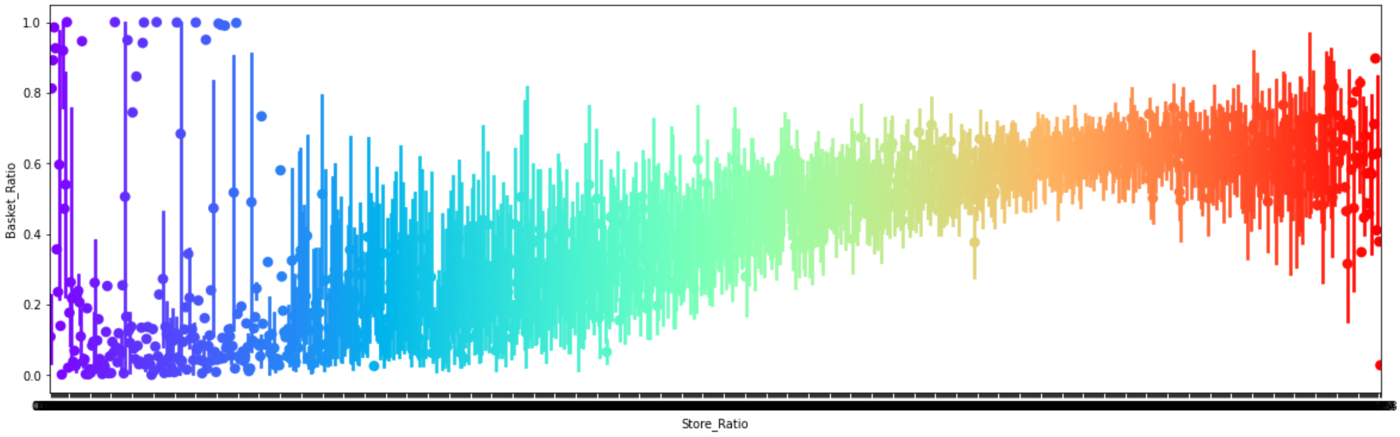
1. A point plot uses scatter plot glyphs to visualize features like point estimates and confidence intervals.
2. A point plot uses scatter plot points to represent the central tendency of numeric data.
3. These plots make use of error bars to indicate any uncertainty around the numeric

In [ ]:

```
# Perform point plot between Store Ratio and Basket Ratio
```

Out[ ]:

<AxesSubplot:xlabel='Store_Ratio', ylabel='Basket_Ratio'>



From above point plot

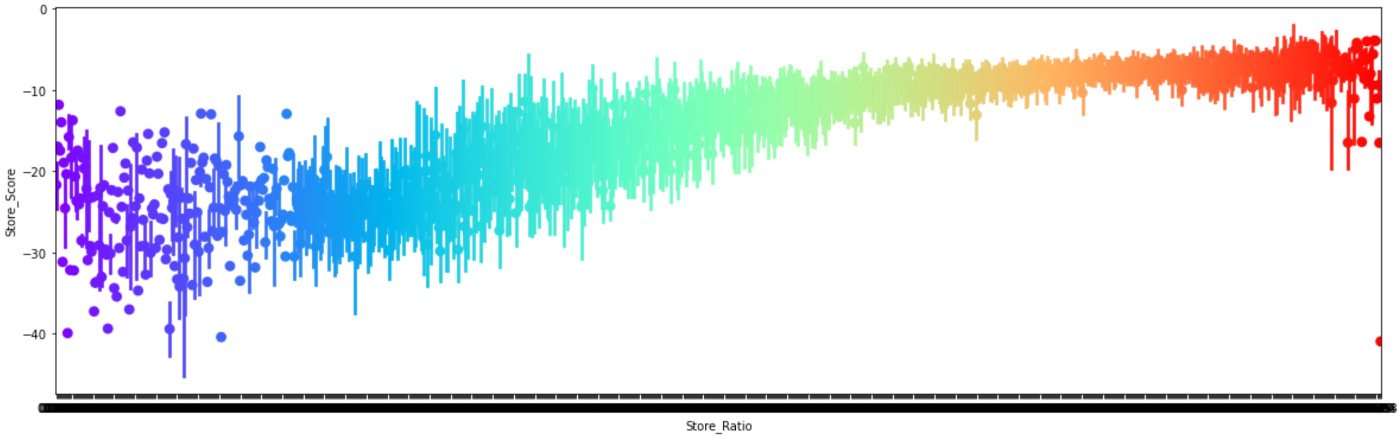
There is a increase in basket ratio when there is a increase in store ratio. That is both are correlated

In [ ]:

```
# Perform point plot between Store Ratio and Store Score
```

Out[ ]:

```
&lt;AxesSubplot:xlabel='Store_Ratio', ylabel='Store_Score'&gt;
```



From above point plot

There is a increase in store score when there is a increase in store ratio. That is both are correlated

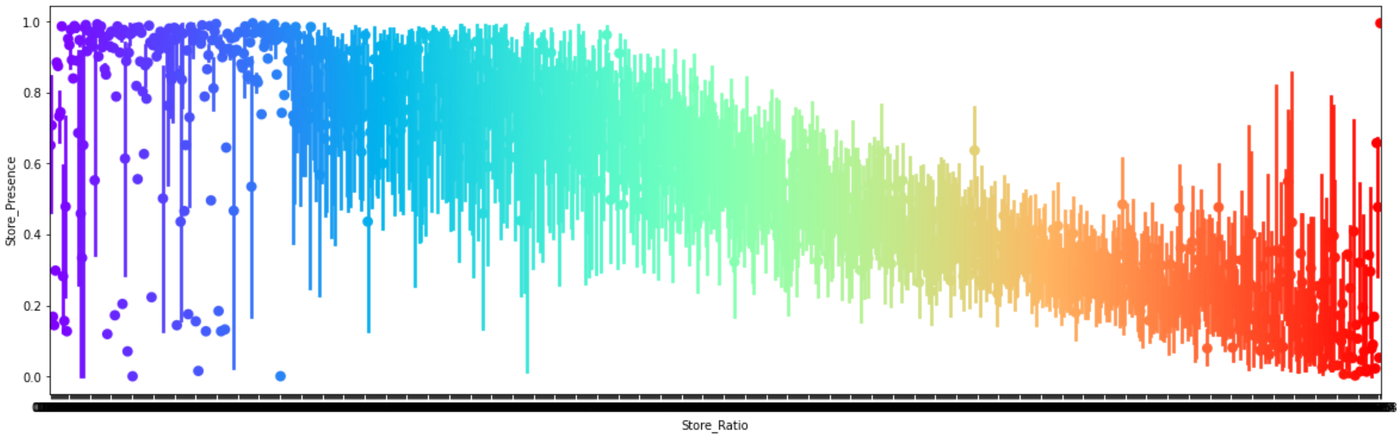
increase decrease increase decrease

In [ ]:

```
# Perform point plot between Store Ratio and Store Presence
```

Out[ ]:

```
&lt;AxesSubplot:xlabel='Store_Ratio', ylabel='Store_Presence'&gt;
```



From above point plot

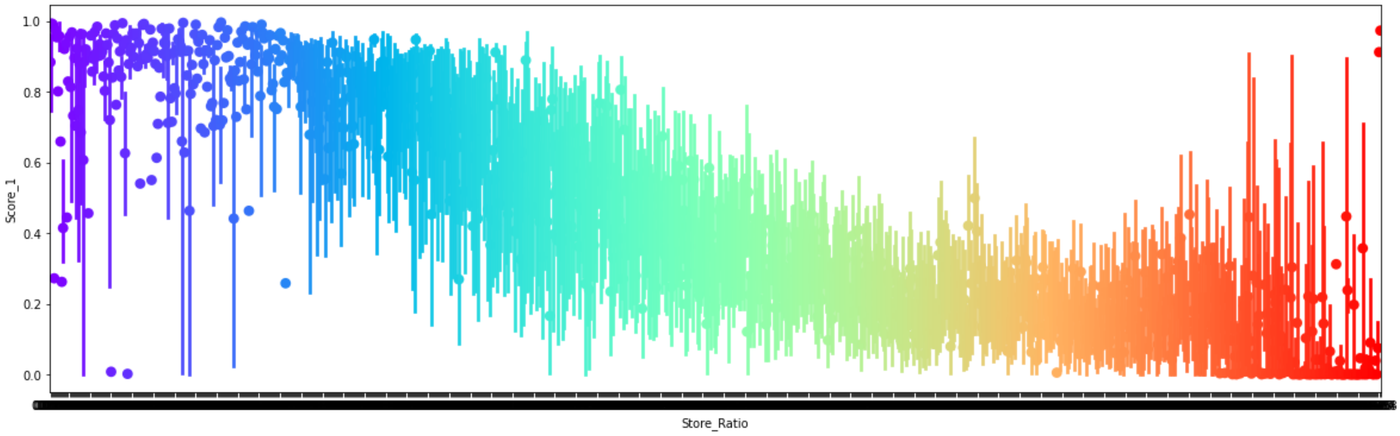
There is a increase in store ratio when there is a decrease in store presence.

In [ ]:

```
# Perform point plot between Store Ratio and Score 1
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Ratio', ylabel='Score_1'>
```



From above point plot

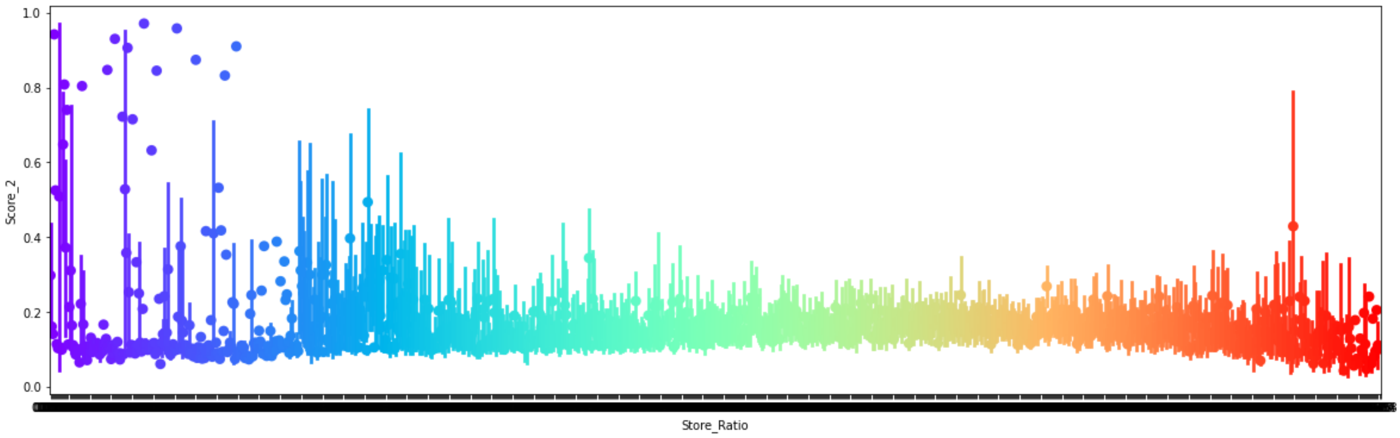
There is a decrease in score3 when there is a increase in store ratio.

In [ ]:

```
# Perform point plot between Store Ratio and Score 2
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Ratio', ylabel='Score_2'>
```

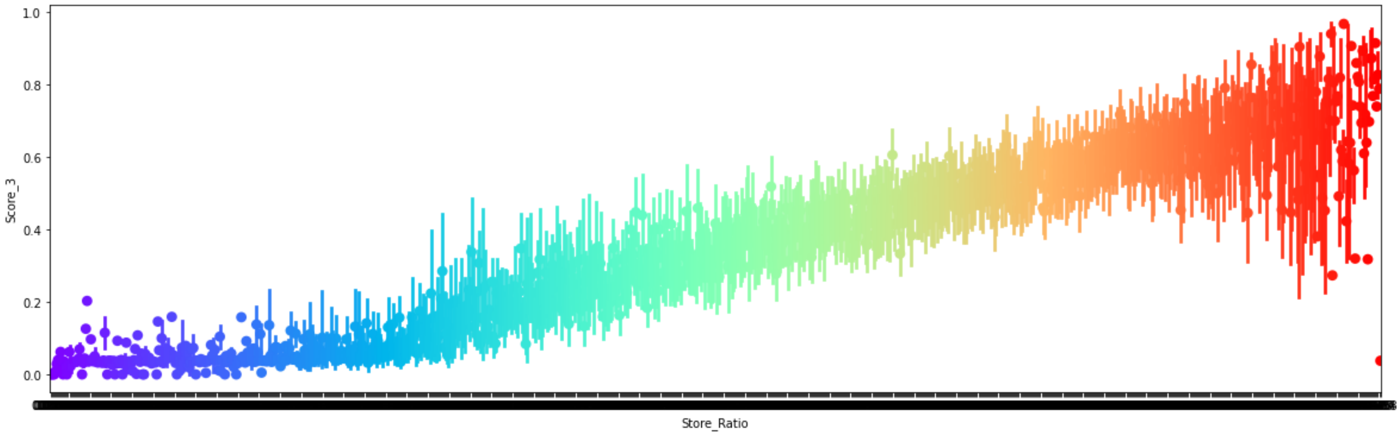


In [ ]:

```
# Perform point plot between Store ratio and Score 3
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Ratio', ylabel='Score_3'>
```



From above point plot

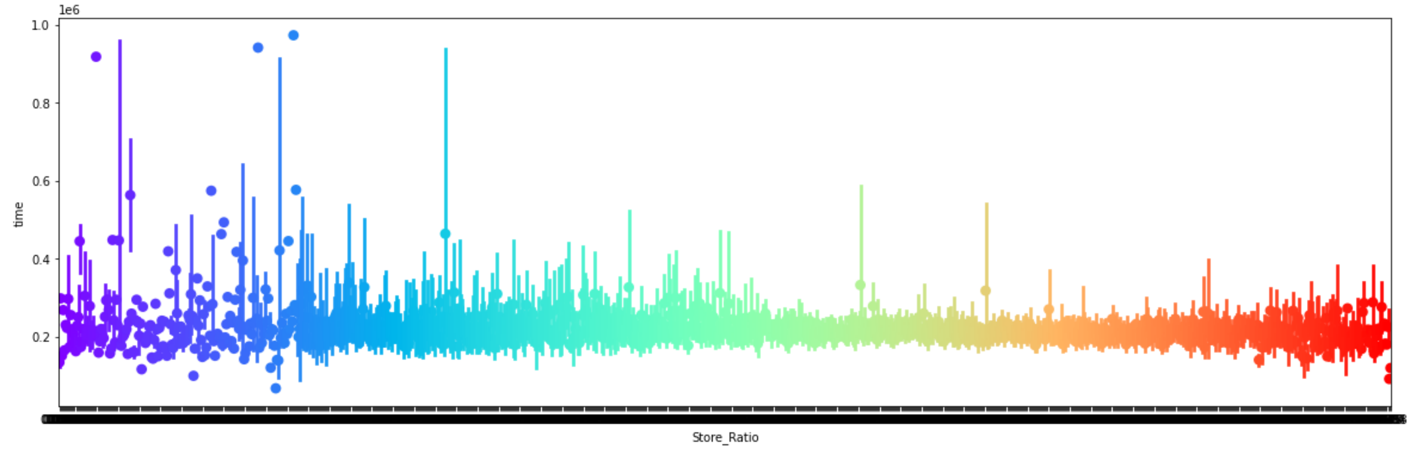
There is a increase in store ratio when there is a increase in score3. That is both are correlated

In [ ]:

```
# Perform point plot between Store Ratio and time
```

Out[ ]:

<AxesSubplot:xlabel='Store_Ratio', ylabel='time'>



From above pointplot

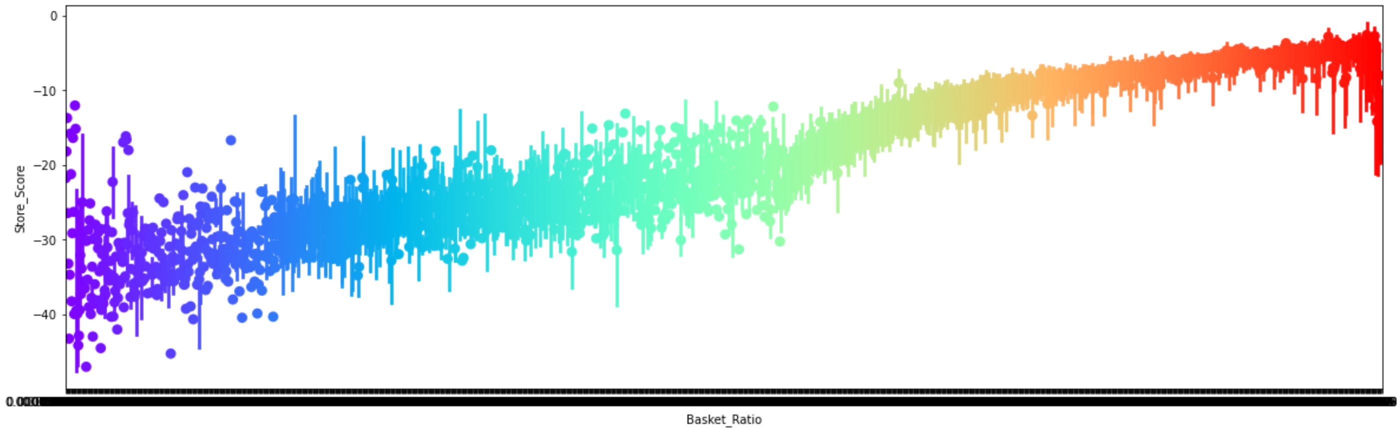
1. Most of the points are between 0.2 to 0.3
2. Very few points above 0.3

In [ ]:

```
# Perform point plot between Basket Ratio and Store Score
```

Out[ ]:

<AxesSubplot:xlabel='Basket_Ratio', ylabel='Store_Score'>



From above point plot

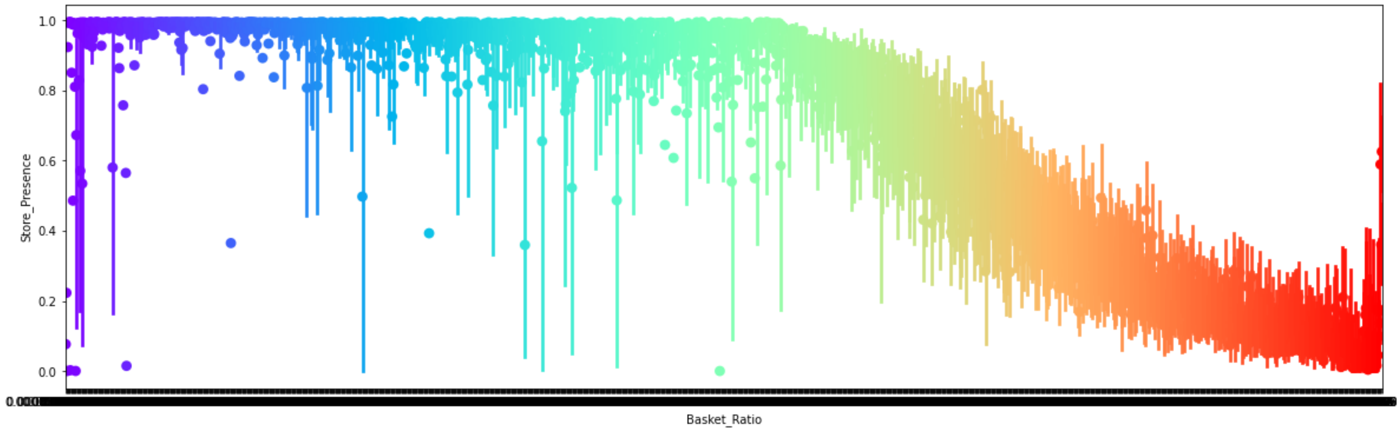
There is a increase in store score when there is a increase in basket ratio.

In [ ]:

```
# Perform point plot between Basket Ratio and Store Presence
```

Out[ ]:

<AxesSubplot:xlabel='Basket_Ratio', ylabel='Store_Presence'>



observation from above point plot

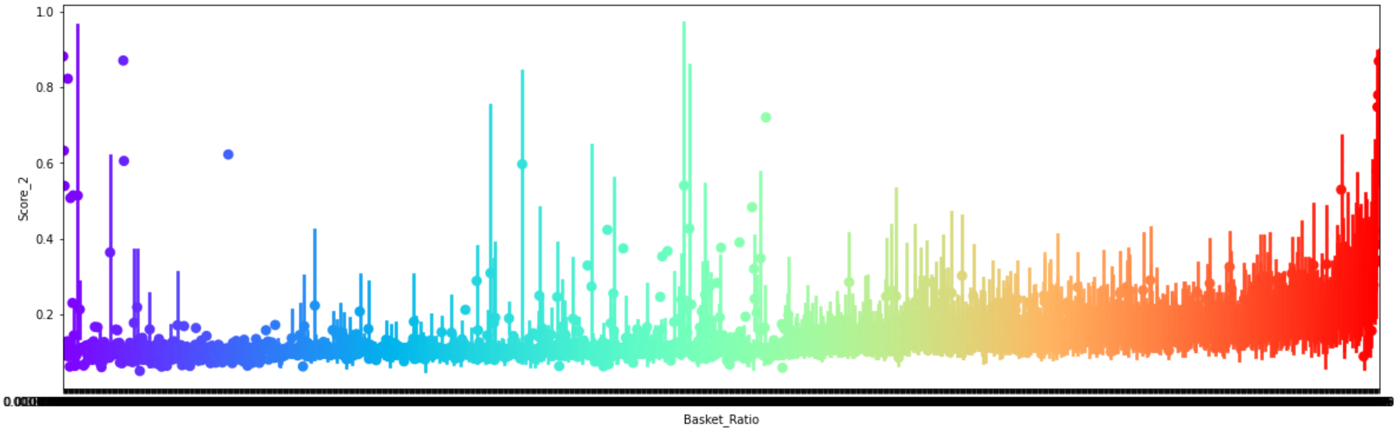
there is decrease in value on store presence as value of basket ratio is increasing

In [ ]:

Perform point plot between Basket Ratio and Score 2

Out[ ]:

<AxesSubplot:xlabel='Basket_Ratio', ylabel='Score_2'>



From above pointplot

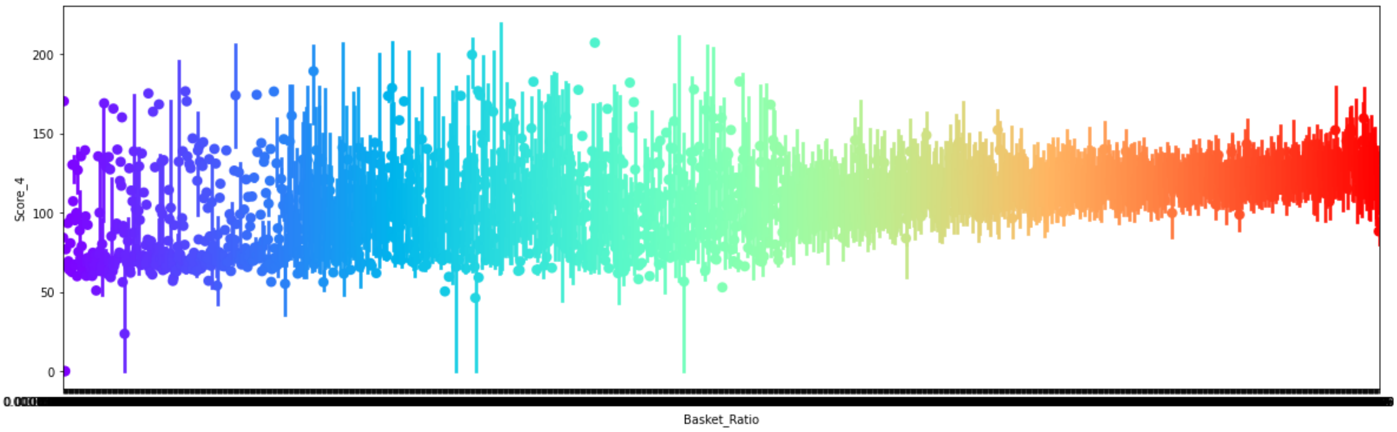
1. **Most of the points are between 0.0 to 0.2**
2. **Very few points above 0.2**

In [ ]:

Perform point plot between Basket Ratio and Score 4

Out[ ]:

<AxesSubplot:xlabel='Basket_Ratio', ylabel='Score_4'>



From above pointplot

Most of the points are between 50 to 150

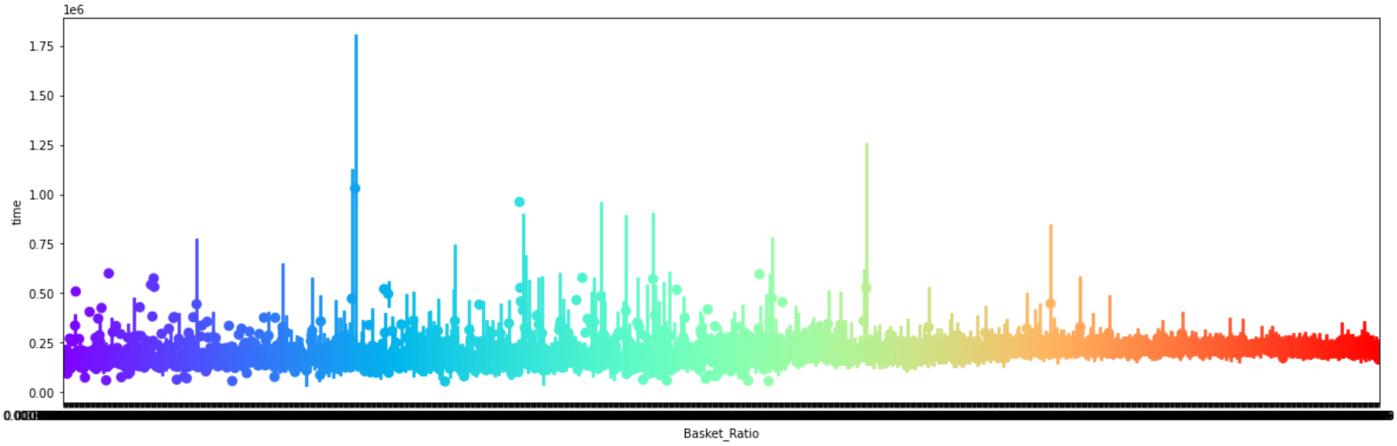
Very few points above 150 and below 50

In [ ]:

Perform point plot between Basket Ratio and Time

Out[ ]:

<AxesSubplot:xlabel='Basket_Ratio', ylabel='time'>



From above pointplot

Most of the points are between 0.0 to 0.50

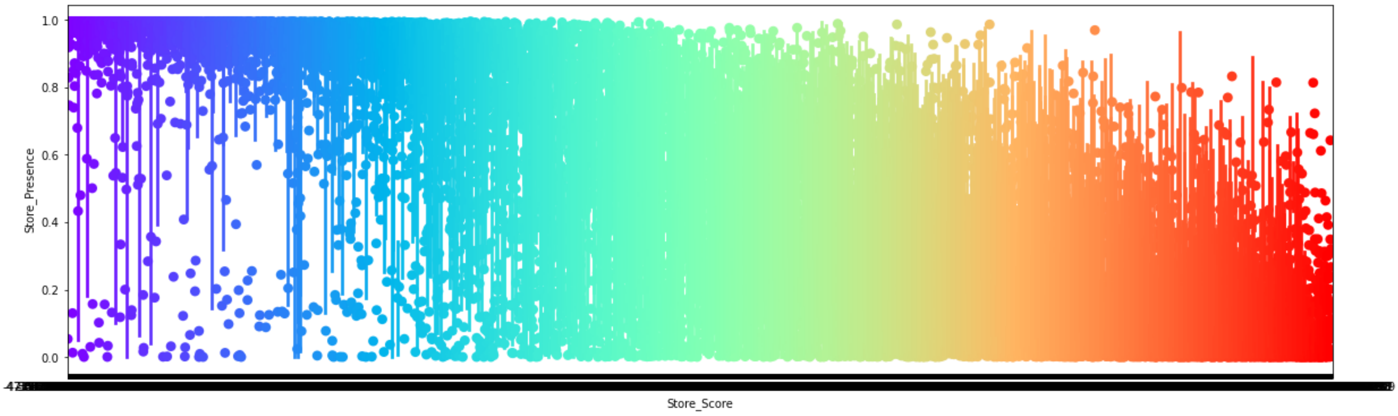
Very few points above 0.50

In [ ]:

```
# Perform point plot between Store Score and Store Presence
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Score', ylabel='Store_Presence'>
```



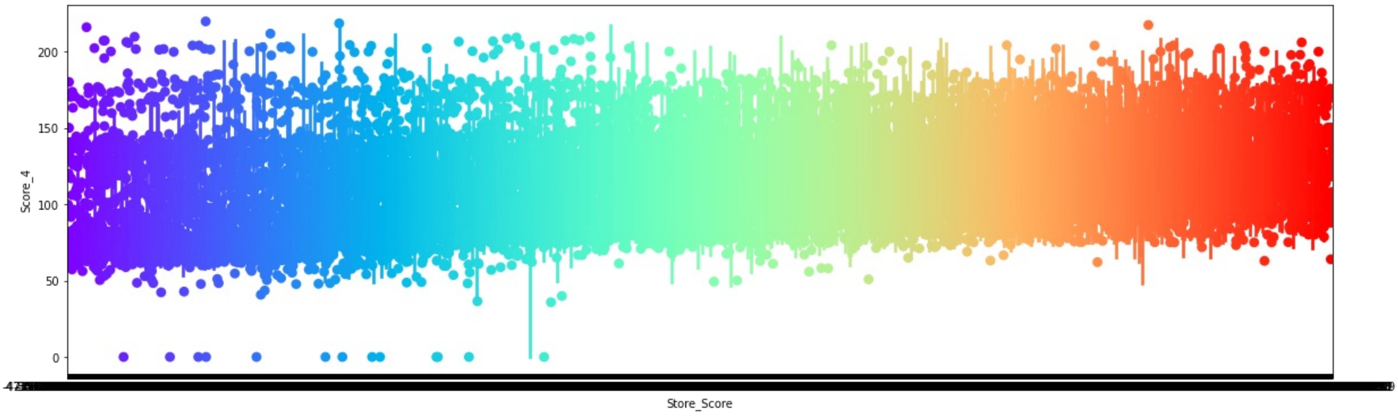
as the value of store score increasing value of store presence decreasing

In [ ]:

```
# Perform point plot between Store Score and Score 4
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Score', ylabel='Score_4'>
```



From above pointplot

Most of the points are between 50 to 200

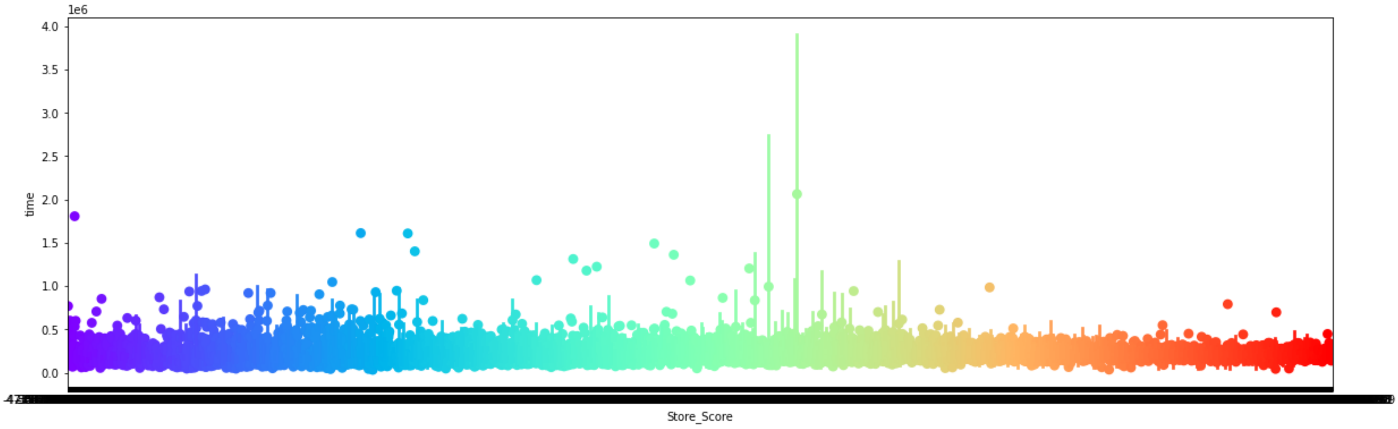
Very few points above 200 and below 50

In [ ]:

```
# Perform point plot between Store Score and time
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Score', ylabel='time'>
```



From above pointplot

Most of the points are between 0.0 and 0.5

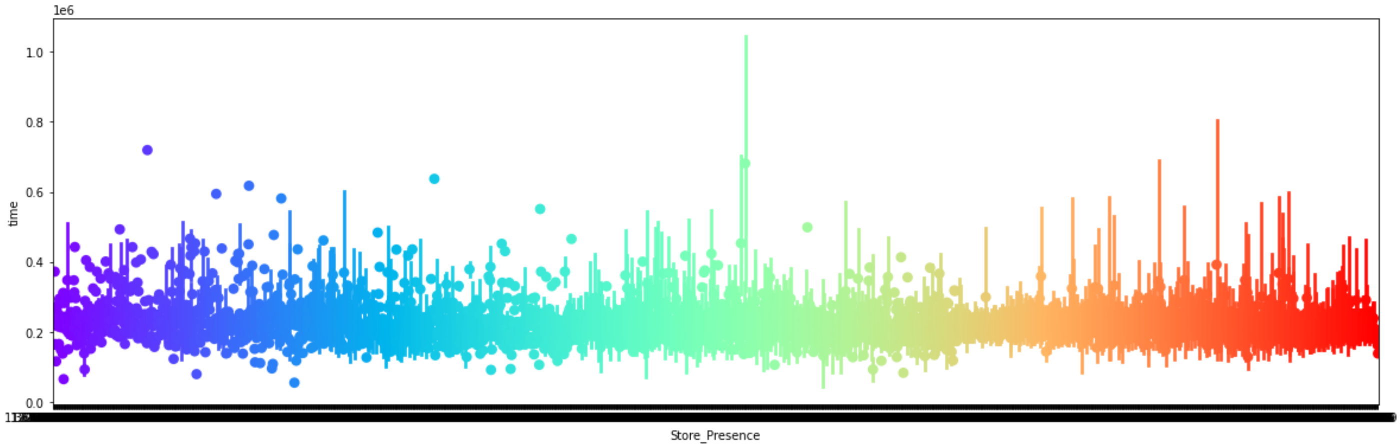
Very few points above 0.5

In [ ]:

```
# Perform point plot between Store Presence and time
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Presence', ylabel='time'>
```



From above pointplot

Most of the points are between 0.1 and 0.4

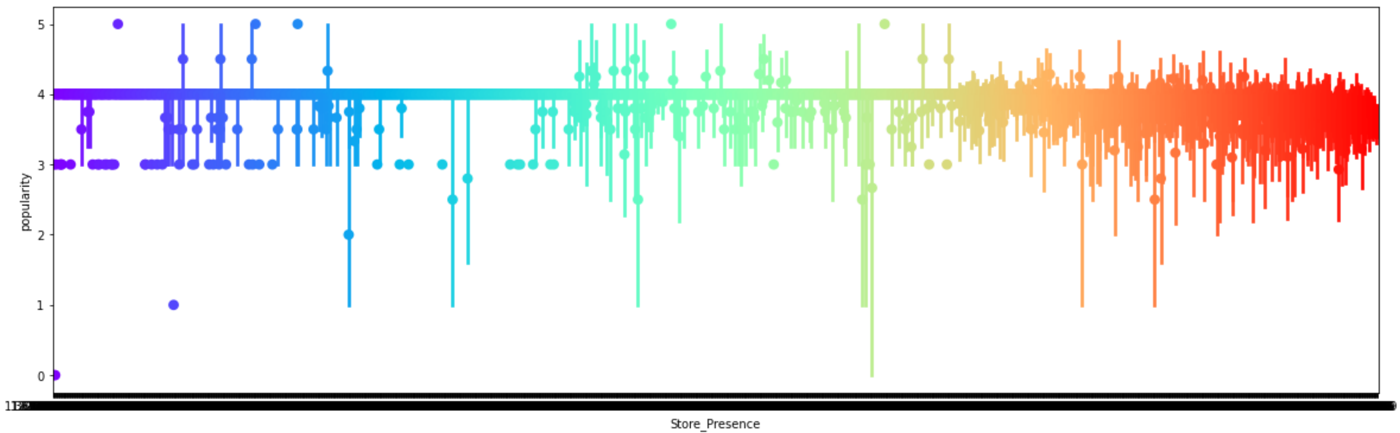
Very few points above 0.4 and below 0,1

In [ ]:

```
# Perform point plot between Store presence and popularity
```

Out[ ]:

```
<AxesSubplot:xlabel='Store_Presence', ylabel='popularity'>
```



From above pointplot

Most of the points are between 3 to 4

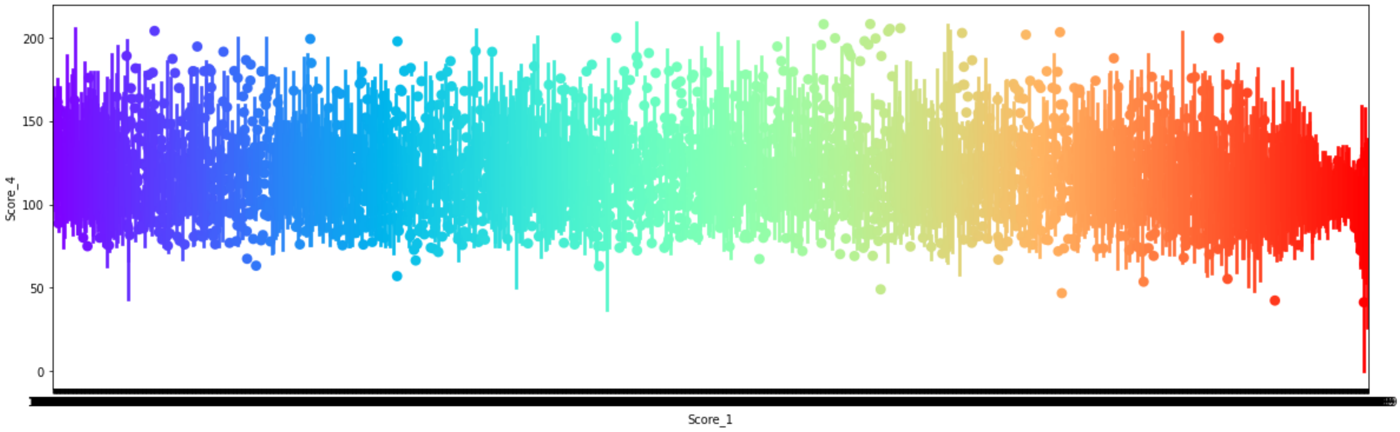
Very few points above 4 and below 3

In [ ]:

```
# Perform point plot between Score 1 and score 4
```

Out[ ]:

```
<AxesSubplot:xlabel='Score_1', ylabel='Score_4'>
```



From above pointplot

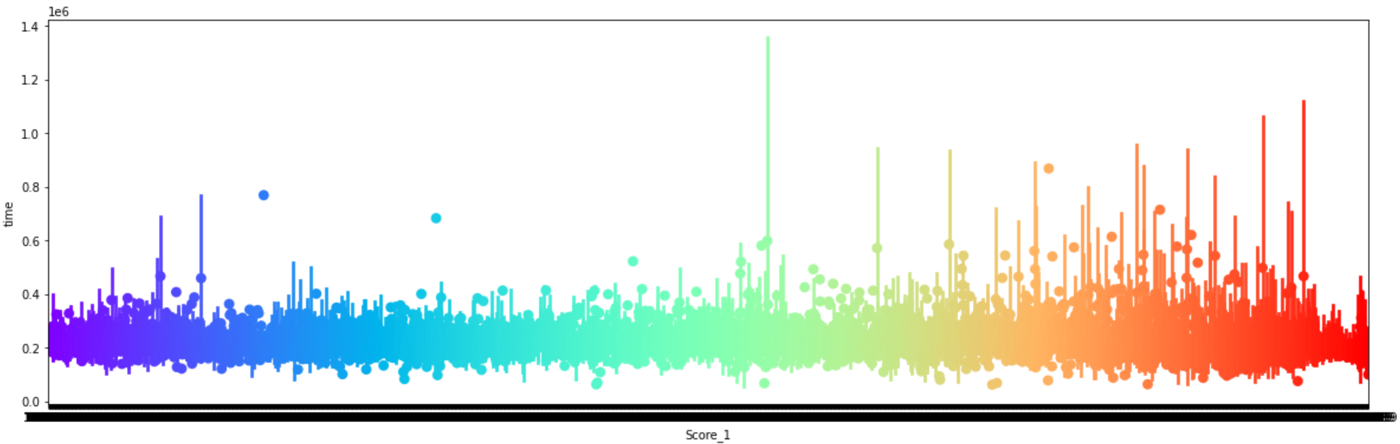
Most of the points are between 75 to 175 Very few points above 175 and below 75

In [ ]:

```
# Perform point plot between Score 1 and time
```

Out[ ]:

```
<AxesSubplot:xlabel='Score_1', ylabel='time'>
```



From above pointplot

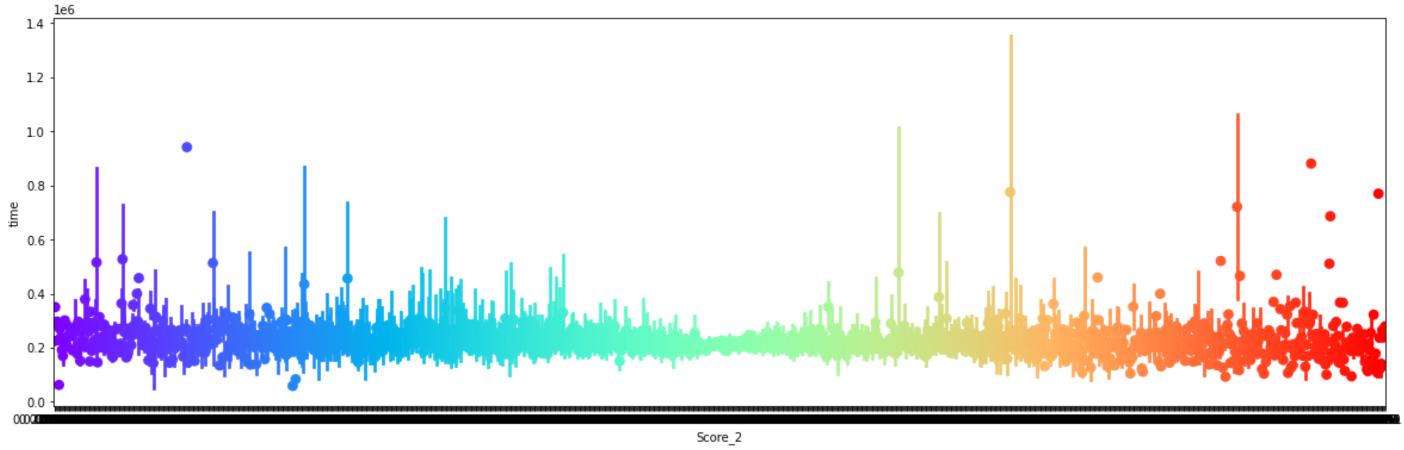
1. Most of the points are between 0.1 to 0.4
2. Very few points above 0.4

In [ ]:

```
# Perform point plot between Score 2 and time
```

Out[ ]:

<AxesSubplot:xlabel='Score_2', ylabel='time'>



From above pointplot

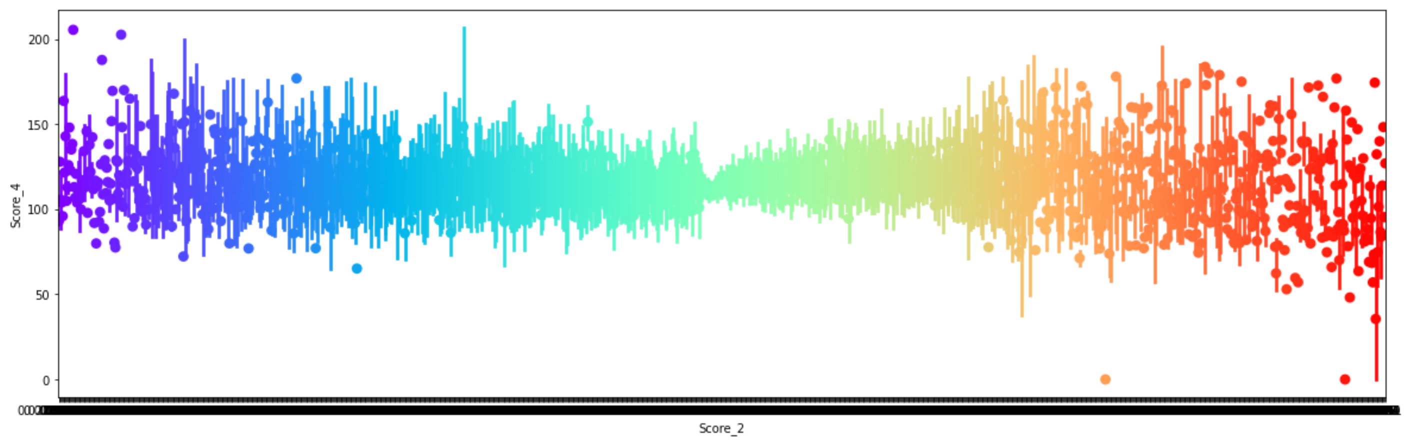
1. Most of the points are between 1 to 3
2. Very few points above 3

In [ ]:

```
# Perform point plot between Score 2 and Score 4
```

Out[ ]:

<AxesSubplot:xlabel='Score_2', ylabel='Score_4'>



From above pointplot

Most of the points are between 75 to 150

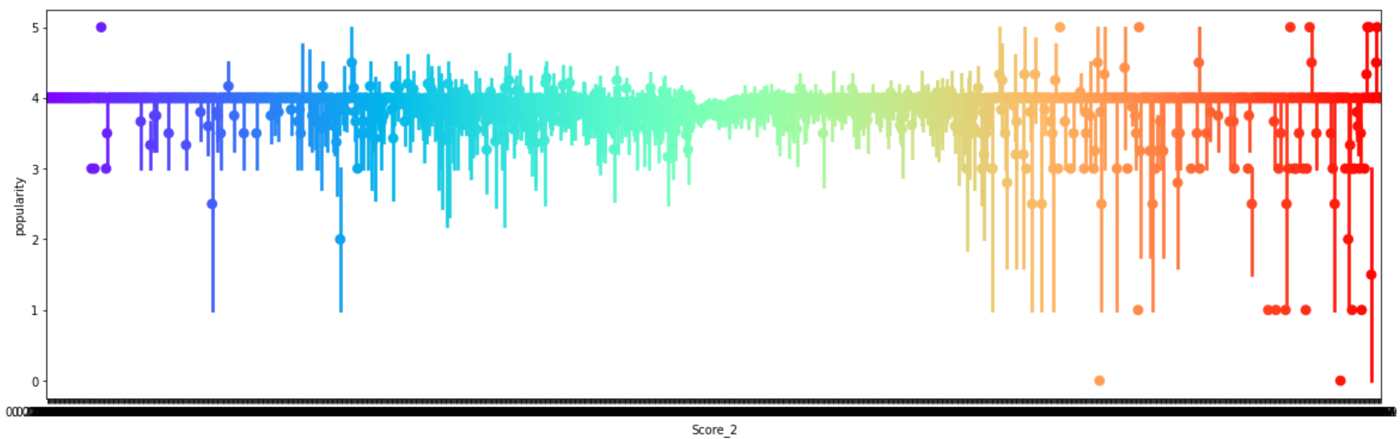
Very few points above 150 and below 75

In [ ]:

```
# Perform point plot between Score 2 and popularity
```

Out[ ]:

<AxesSubplot:xlabel='Score_2', ylabel='popularity'>



From above pointplot

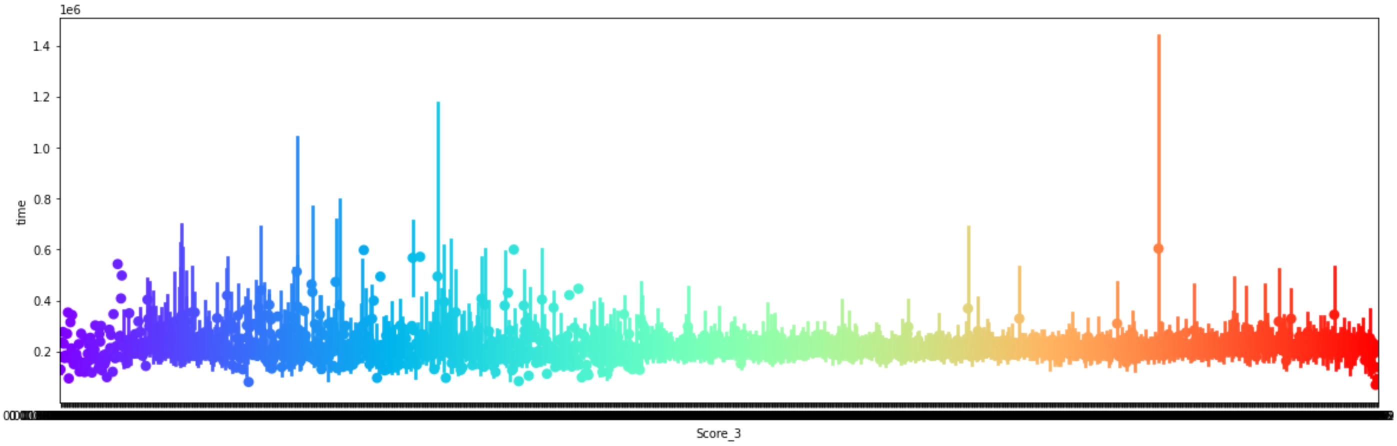
1. Most of the points are around 4
2. few points are between 3 and 4
3. very few are below 3

In [ ]:

```
# Perform point plot between Score 3 and time
```

Out[ ]:

<AxesSubplot:xlabel='Score_3', ylabel='time'>



From above pointplot

Most of the points are between o.0 to 0.4

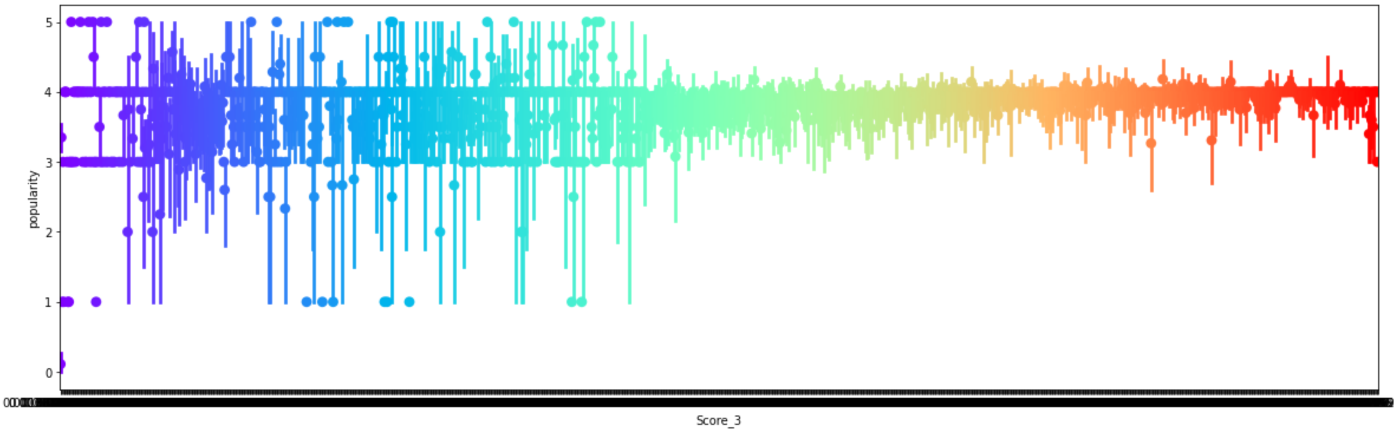
Very few points above 0.4

In [ ]:

```
# Perform point plot between Score 3 and popularity
```

Out[ ]:

<AxesSubplot:xlabel='Score_3', ylabel='popularity'>



From above pointplot

Most of the points are between 3 to 4

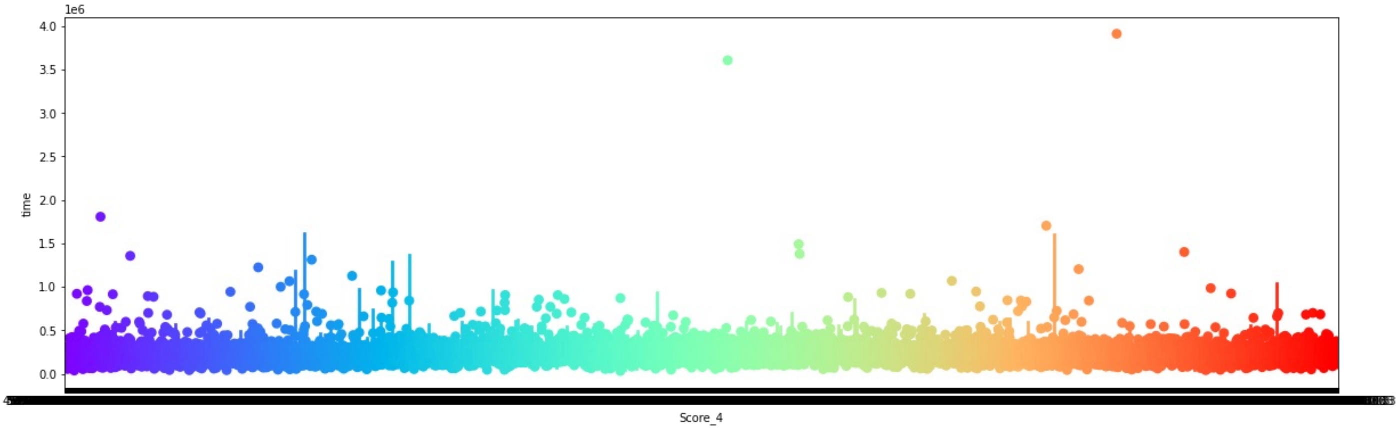
Very few points above 4 and below 3

In [ ]:

```
# Perform point plot between Score 4 and time
```

Out[ ]:

```
<AxesSubplot:xlabel='Score_4', ylabel='time'>
```



From above pointplot

1. Most of the points are between o to 0.5
2. Very few points above 0.5

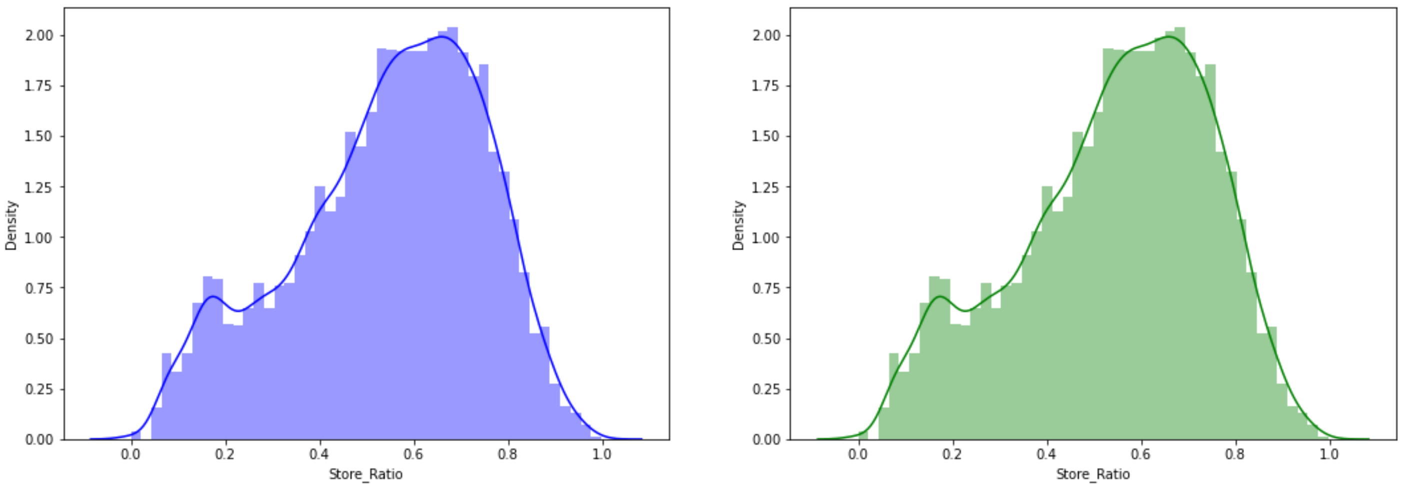
DISTPLOT

The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution

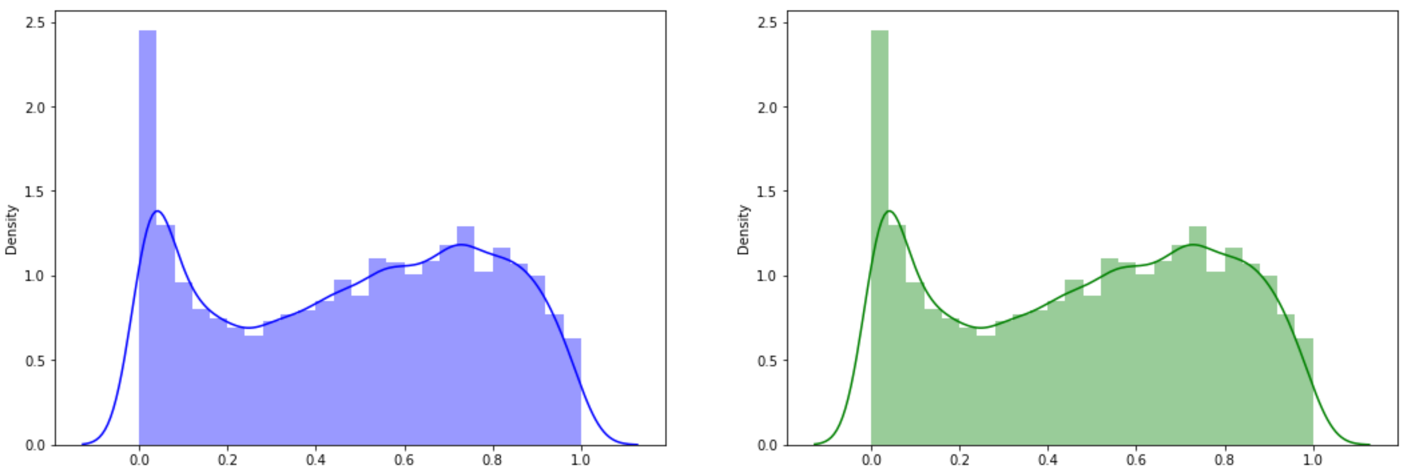
In [ ]:

```
#Perform distplot for all the columns in dataset
for column in df.columns:
    print(column)
    # code below
    fig,ax =
```

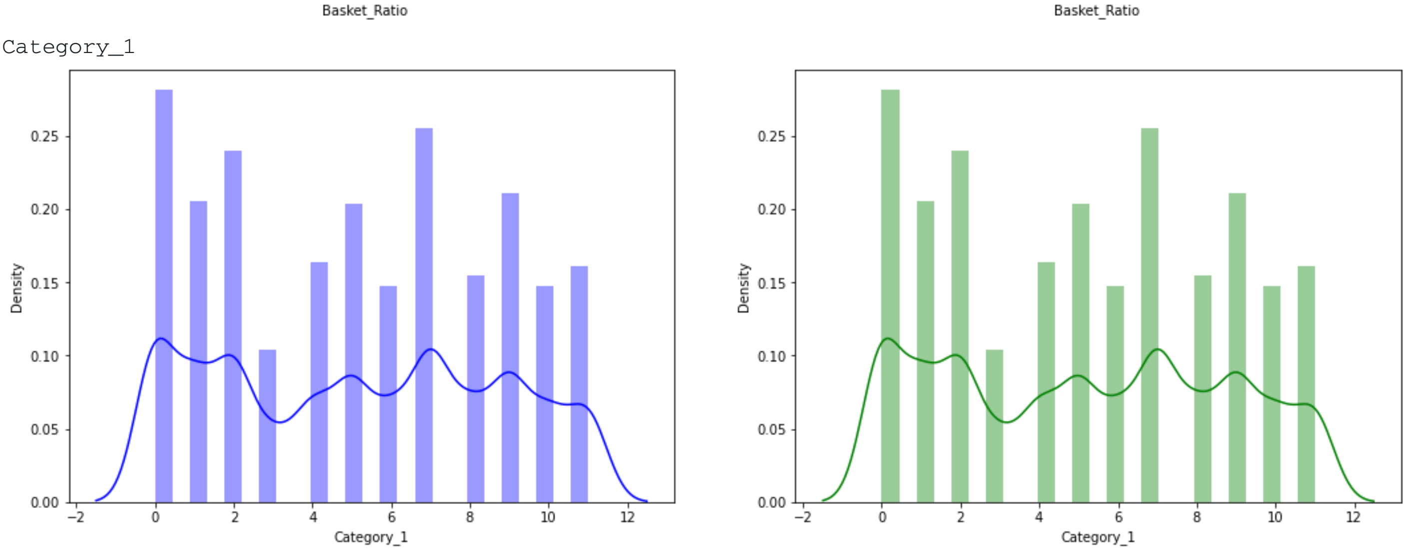
```
Store_Ratio
```



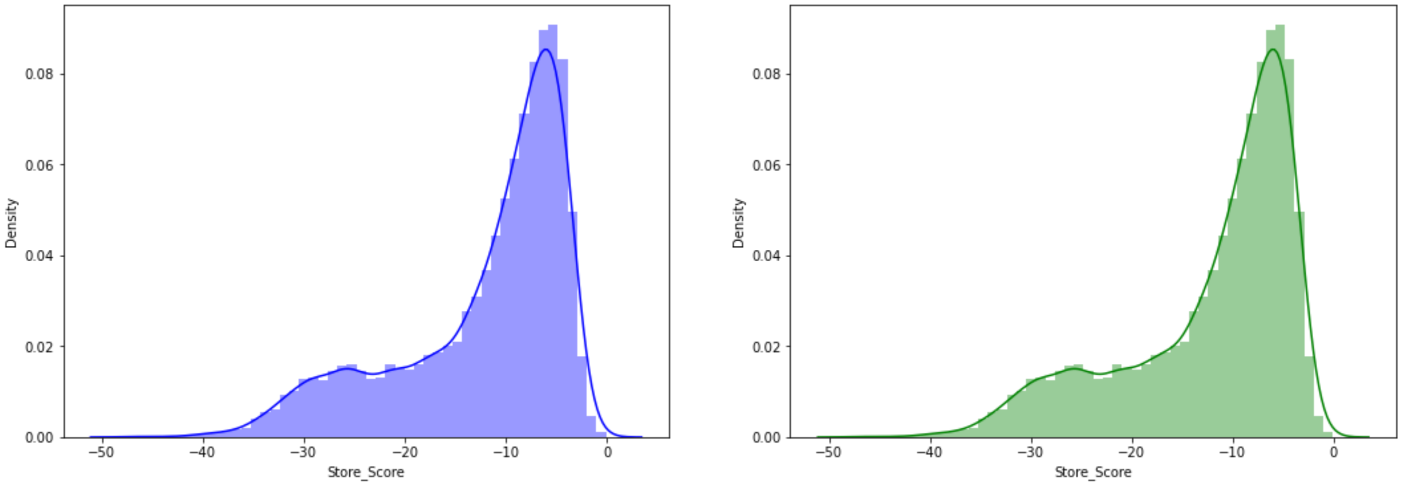

```
Basket_Ratio
```



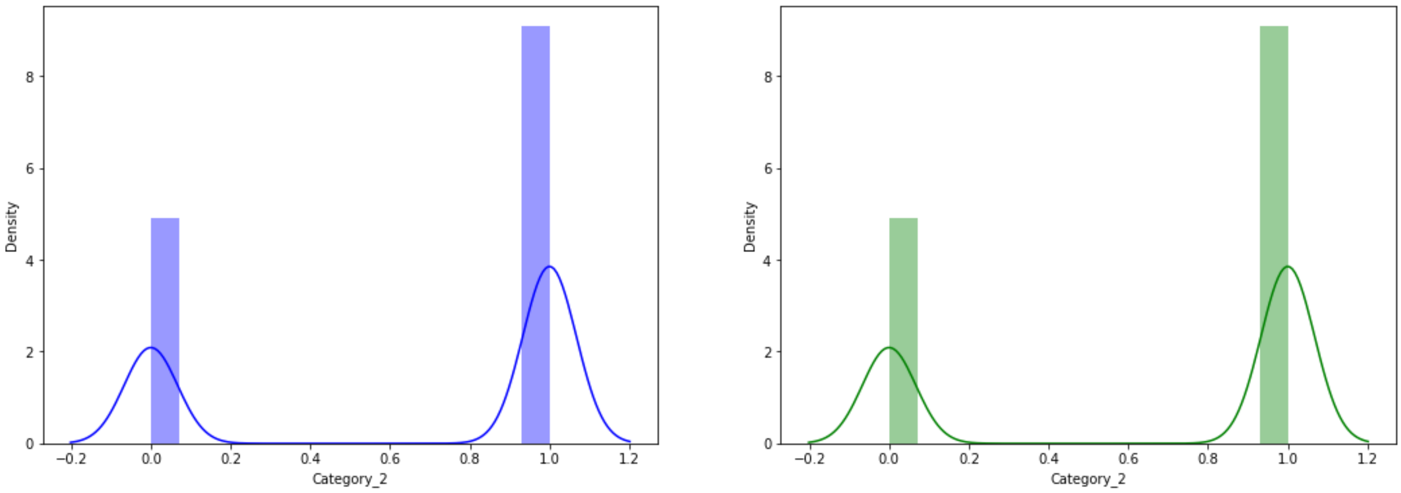
Category_1



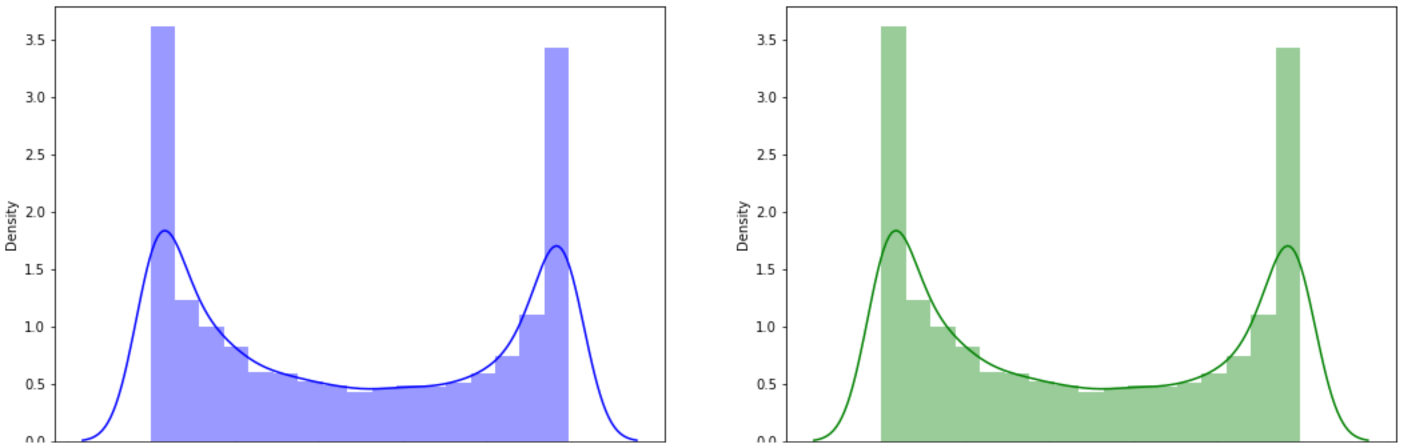
Store_Score



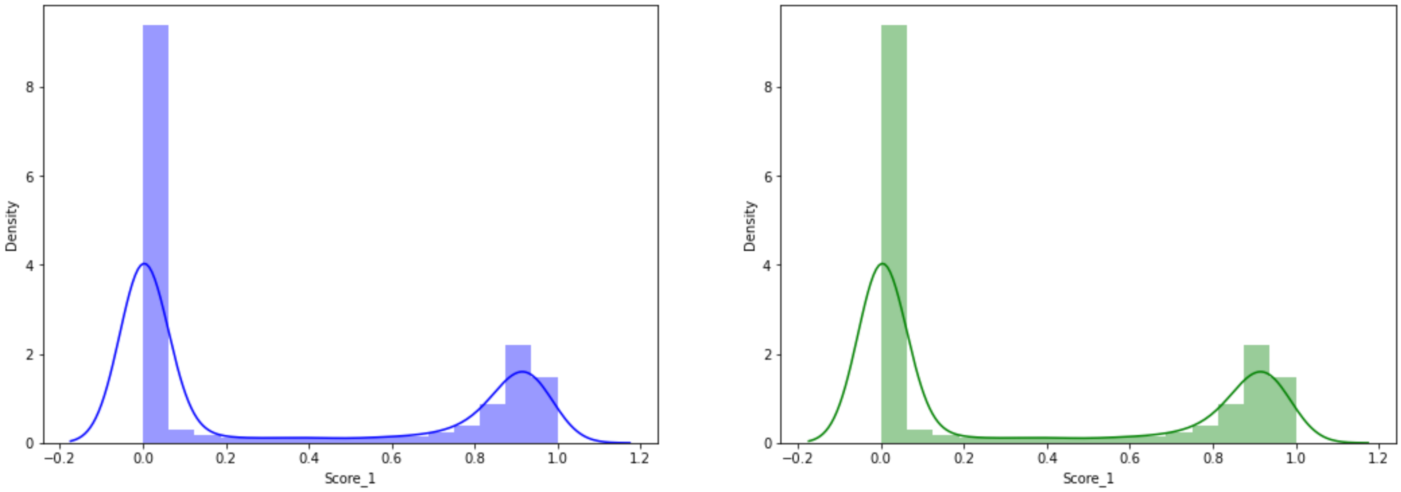
Category_2



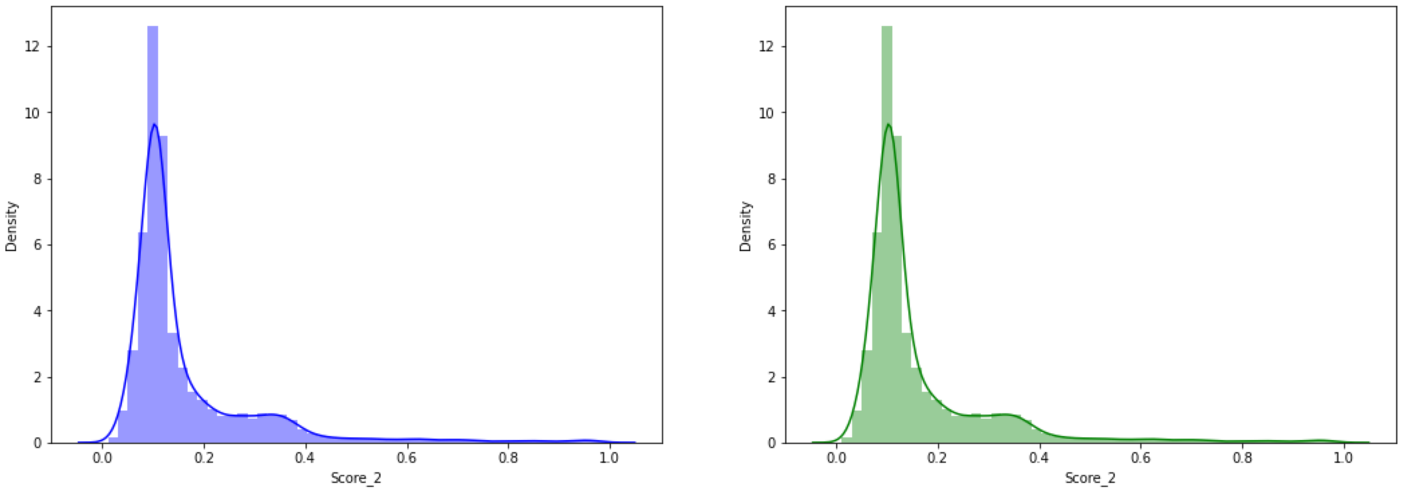
Store_Presence



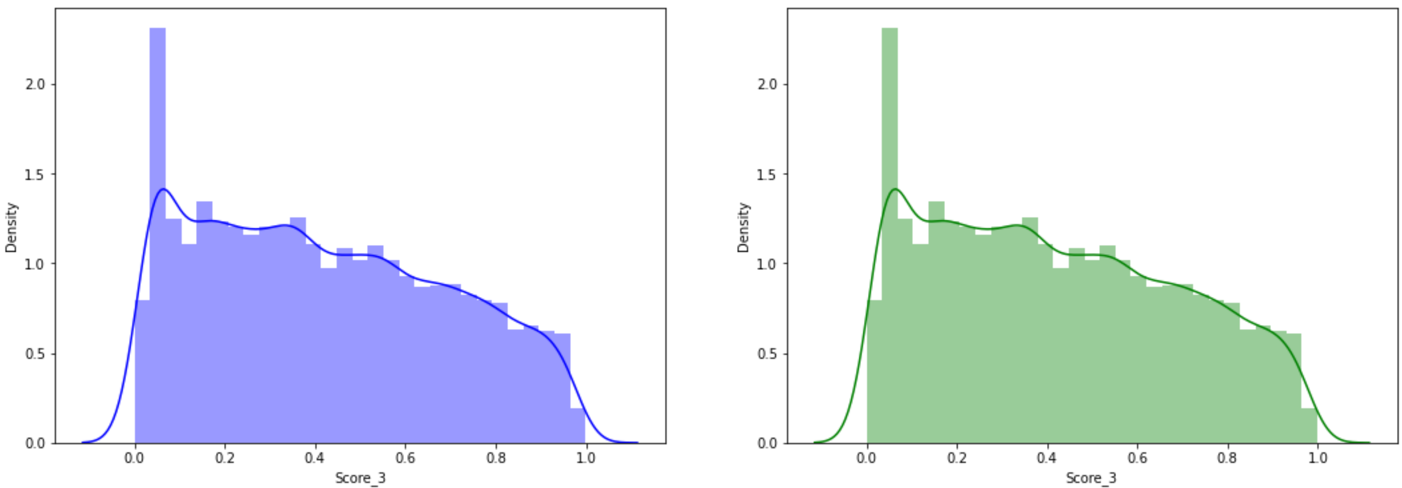
Score_1



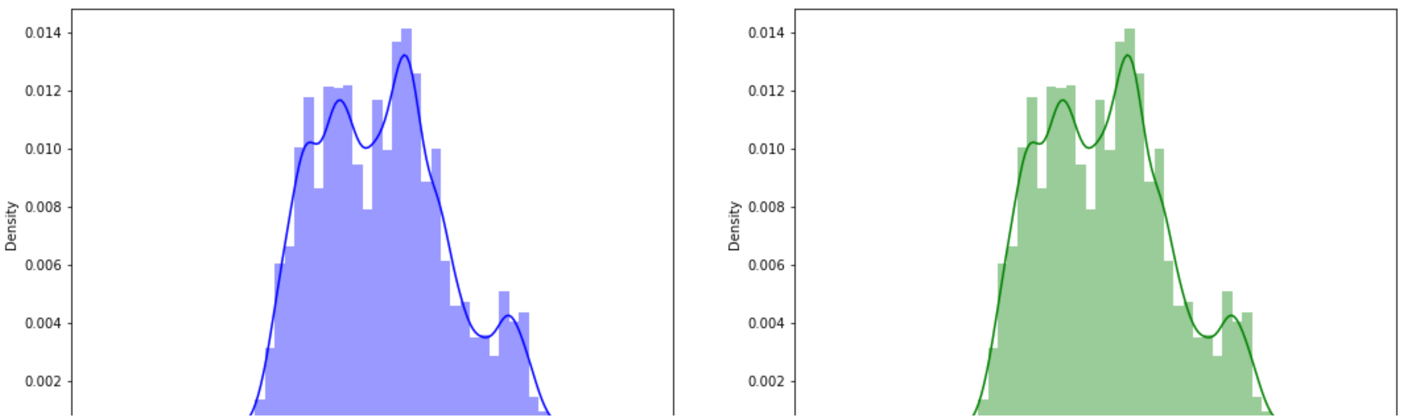
Score_2

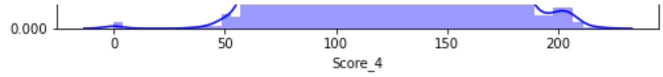


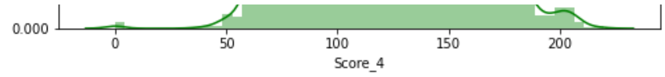
Score_3



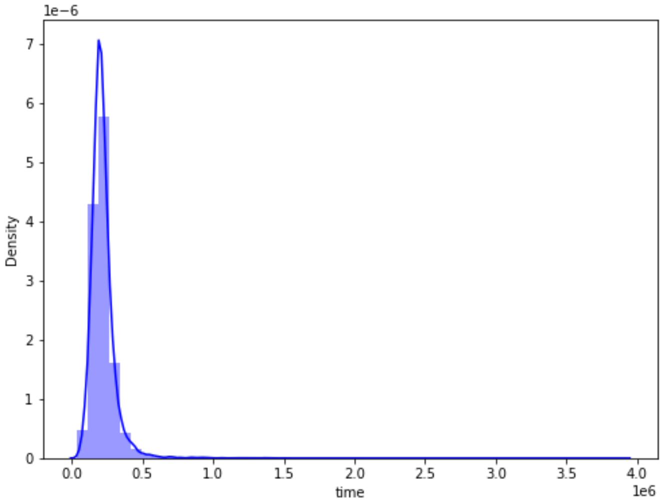
Score_4

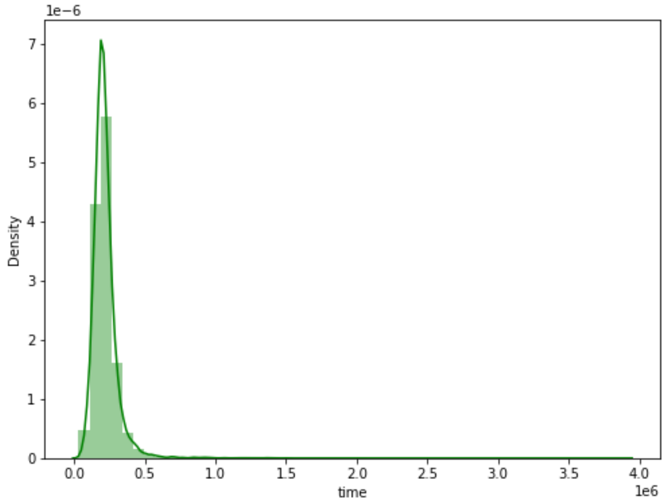




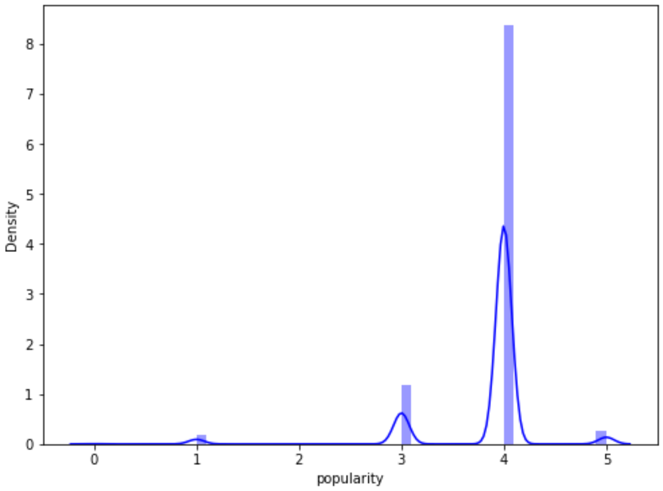


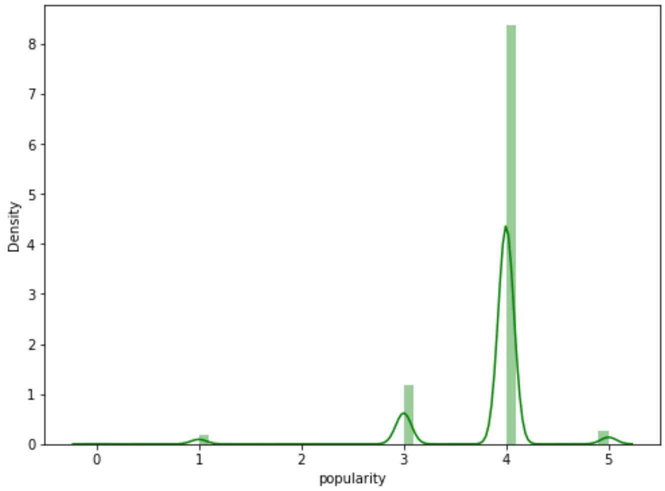
time





popularity





OBSERVATIONS

1. distribution of store ratio, store presence, score3 and popularity are right skewed
2. distribution of basket ratio, score2 and time are skewed left
3. distribution of category 1 is multimodal
4. distribution of category2 and score1 are bimodal

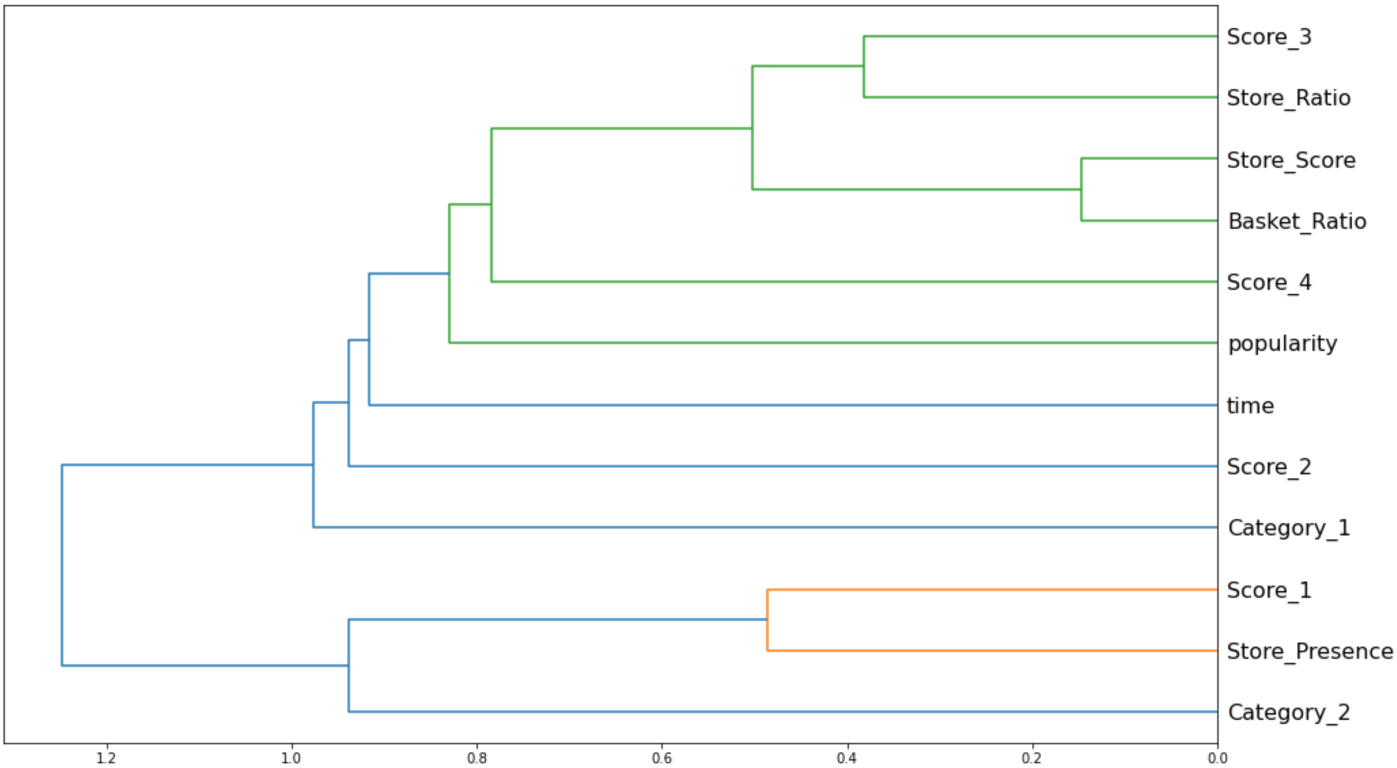
dendrogram

The dendrogram is a visual representation of the compound correlation data. The individual compounds are arranged along the bottom of the dendrogram and referred to as leaf nodes. Compound clusters are formed by joining individual compounds or existing compound clusters with the join point referred to as a node.

In [ ]:

```
# Plot a Dendrogram on the columns of the dataset

# droping the NaN values
X =


# import scipy, hierarchy as hc



corr =

corr_condensed =

z =

fig =
```

```
dendrogram =

plt.show()
```



strongly correlated variables

1. score3 and store ratio
2. store score and basket ratio
3. score1 and store presence

Voilin Plot

1. A violin plot is a method of plotting numeric data.

2. Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.

3. It has:

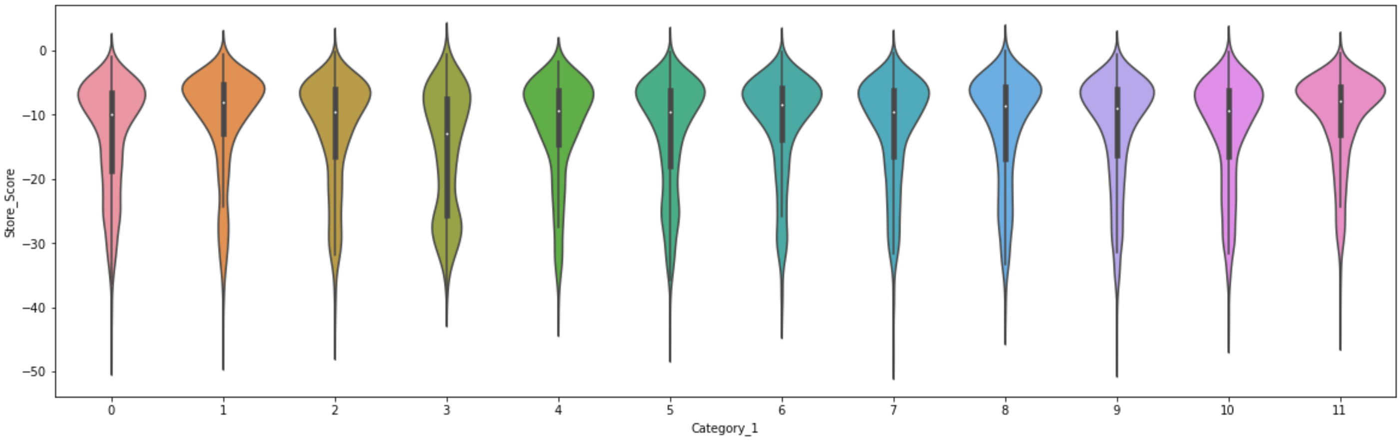
   A. Median (a white dot on the violin plot)
   B. Interquartile range (the black bar in the center of violin)
   C. The lower/upper adjacent values (the black lines stretched from the bar) — defined as first quartile — 1.5 IQR and third quartile + 1.5 IQR respectively.

In [ ]:

```
# perform a violin plot between category1 and store score
```

Out[ ]:

```
<AxesSubplot:xlabel='Category_1', ylabel='Store_Score'>
```



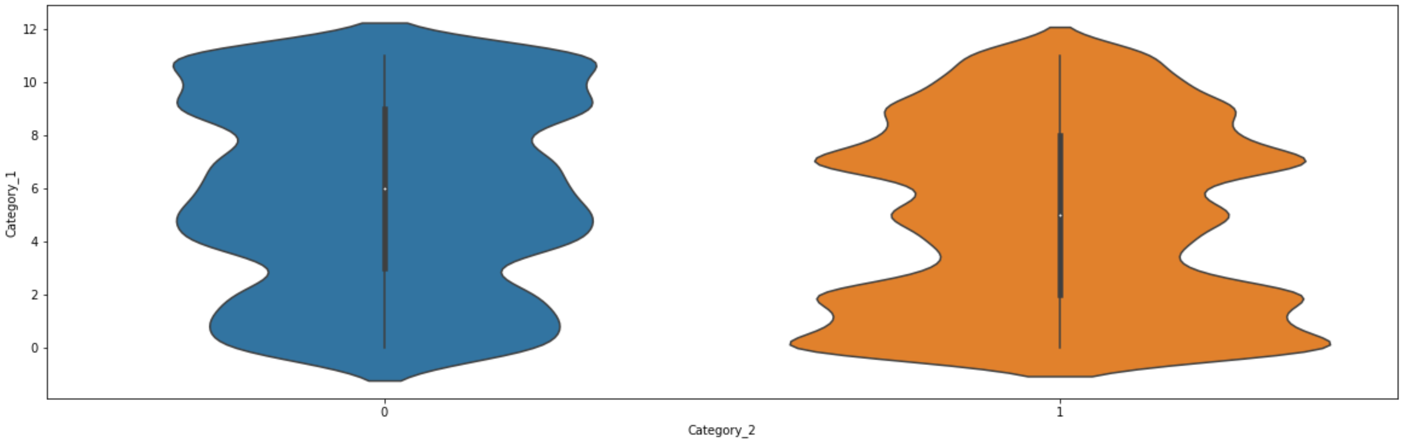
there is no relation between store score and category1

In [ ]:

```
# perform a violin plot between category2 and category1
```

Out[ ]:

```
<AxesSubplot:xlabel='Category_2', ylabel='Category_1'>
```



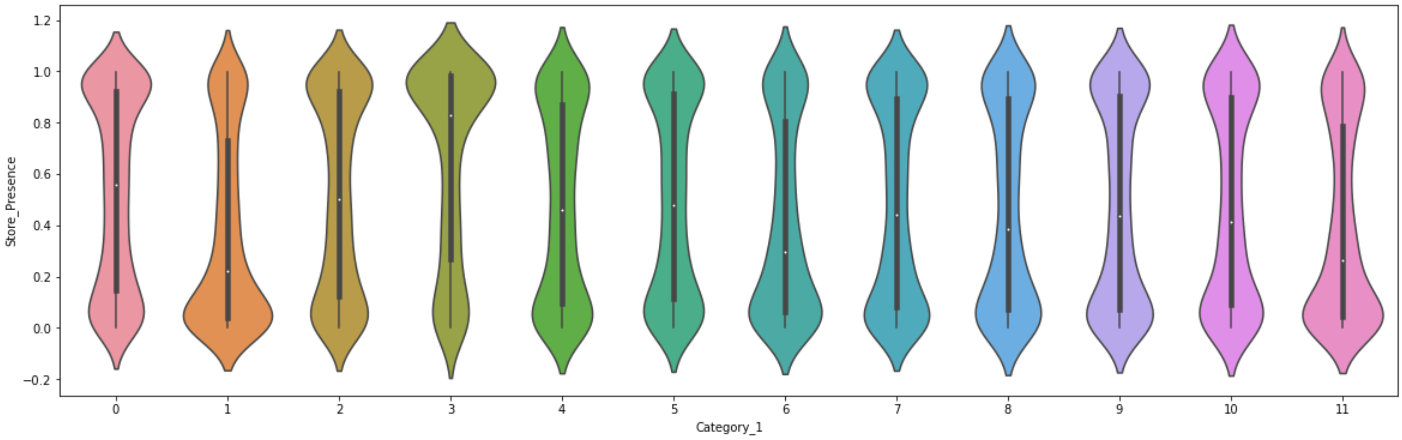
there is no relation between category2 and category1

In [ ]:

```
# perform a violin plot between categort1 and store prsence
```

Out[ ]:

```
<AxesSubplot:xlabel='Category_1', ylabel='Store_Presence'>
```



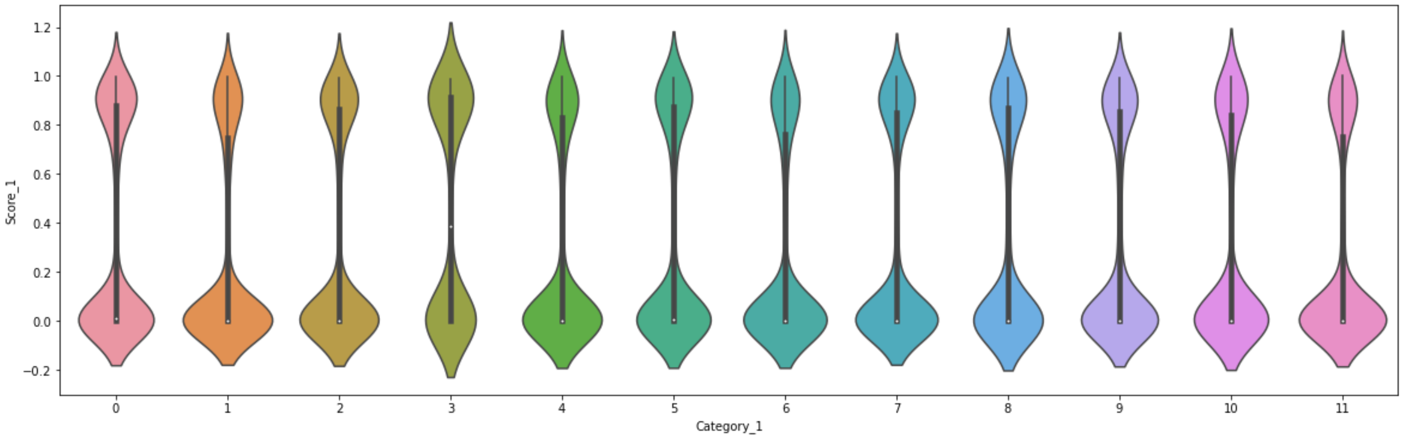
there is no relation between store presence and category1

In [ ]:

```
# perform a violin plot between category1 and score1
```

Out[ ]:

```
<AxesSubplot:xlabel='Category_1', ylabel='Score_1'>
```



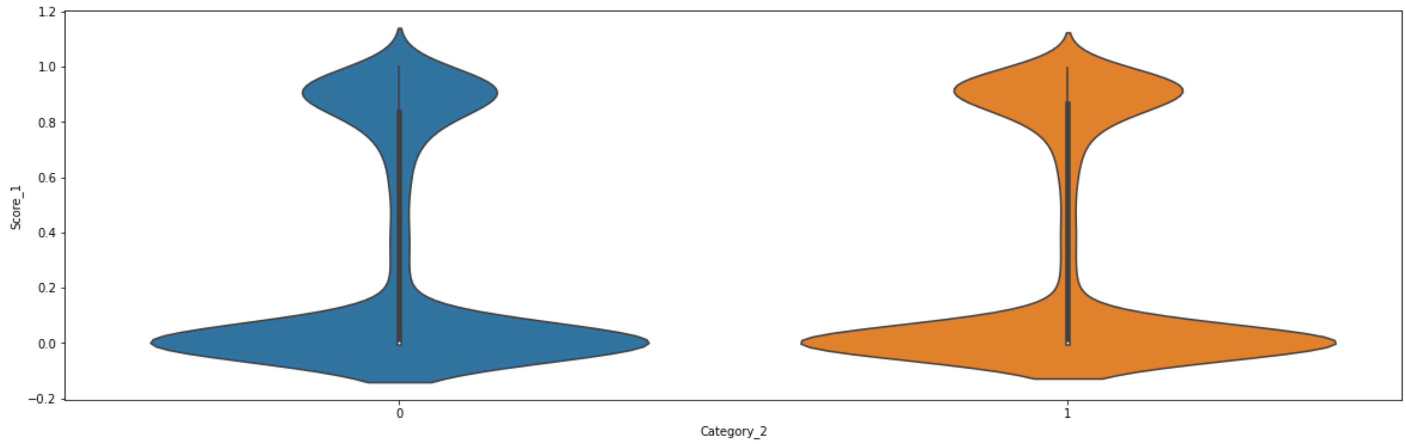
there is no relation between score1 and category1

In [ ]:

```
# perform a violin plot between category2 and score2
```

Out[ ]:

<AxesSubplot:xlabel='Category_2', ylabel='Score_1'>



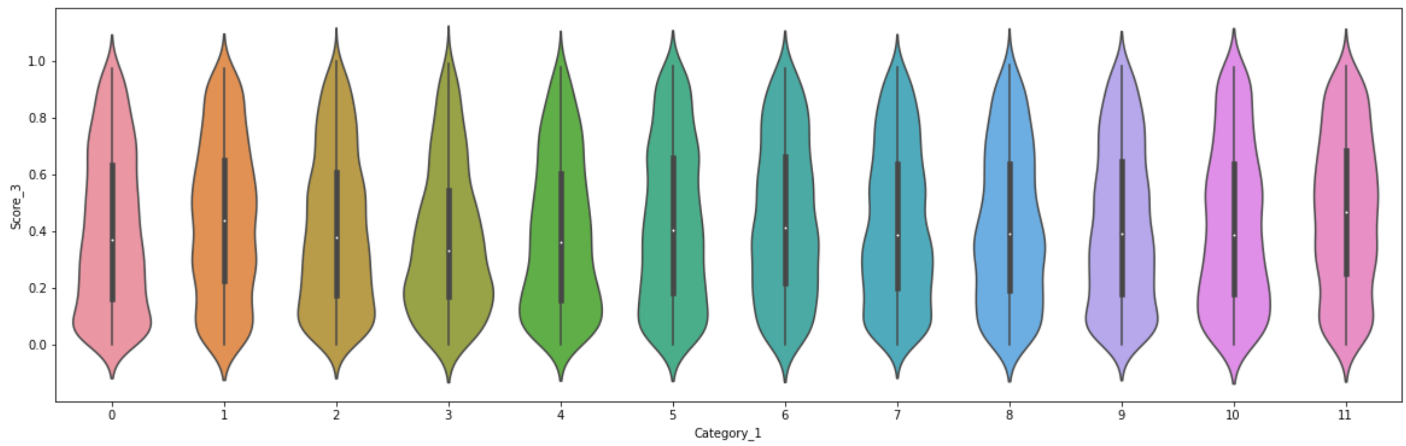
there is no relation between score2 and category1

In [ ]:

```
# perform a violin plot between category1 and score3
```

Out[ ]:

<AxesSubplot:xlabel='Category_1', ylabel='Score_3'>



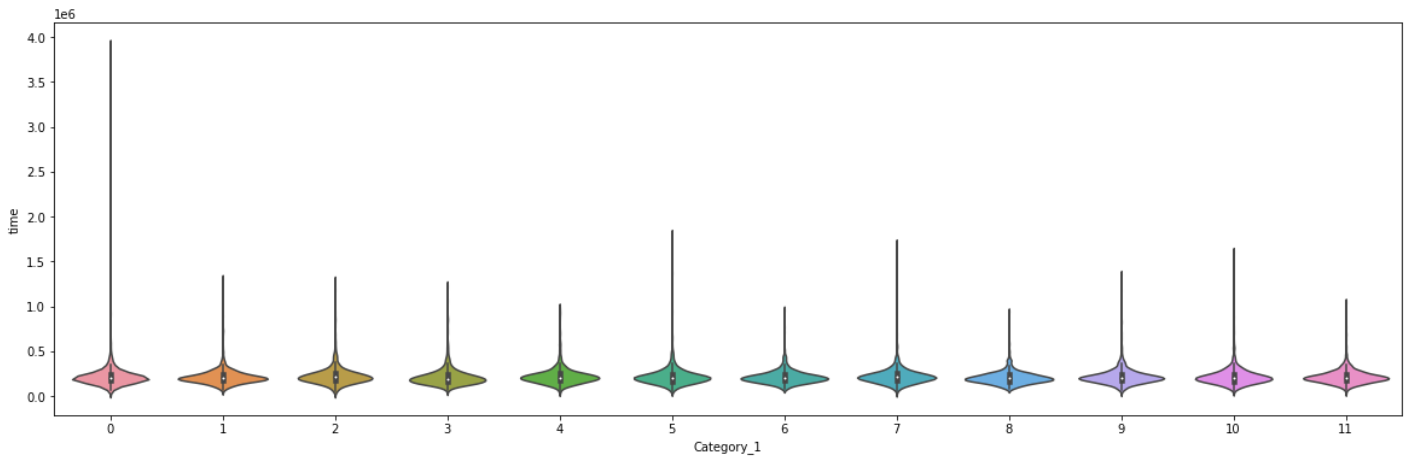
there is no relation between score3 and category1

In [ ]:

```
# perform a violin plot between category1 and time
```

Out[ ]:

<AxesSubplot:xlabel='Category_1', ylabel='time'>



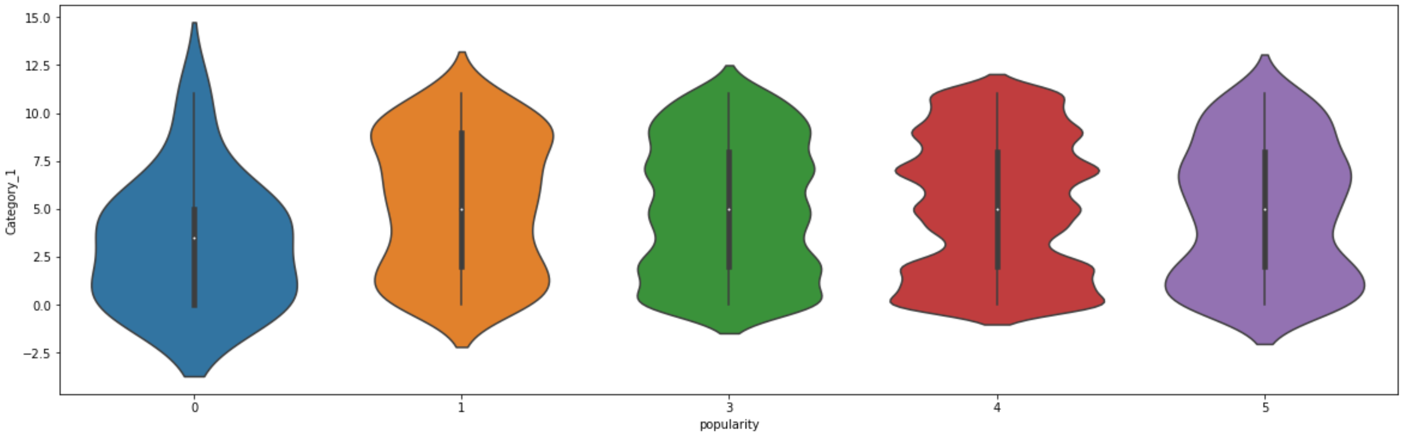
there is no relation between time and category1

In [ ]:

```
# perform a violin plot between popularity and category1
```

Out[ ]:

<AxesSubplot:xlabel='popularity', ylabel='Category_1'>



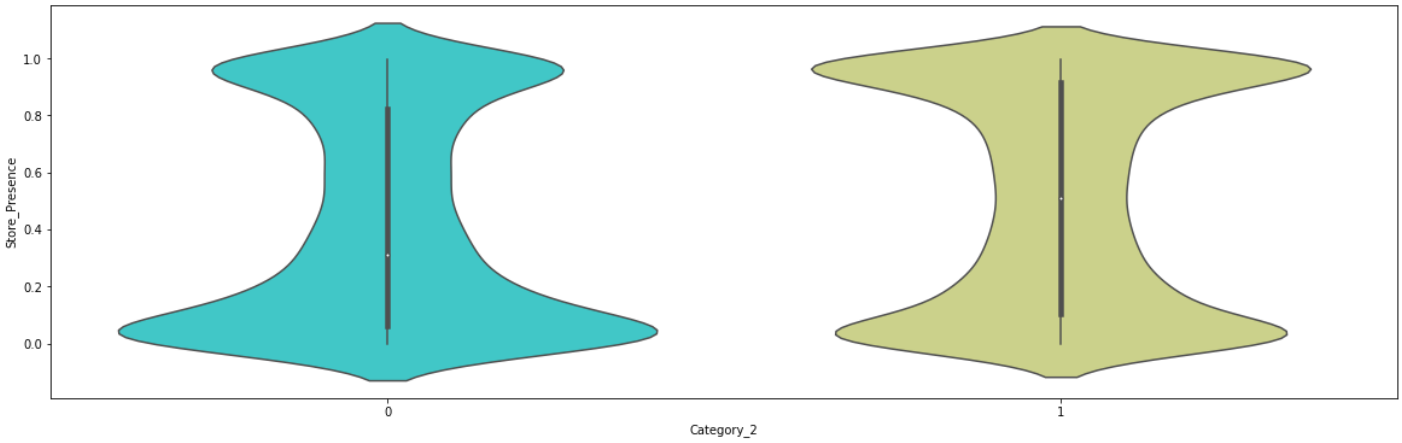
there is no relation between popularity and category1

In [ ]:

```
# perform a violin plot between category2 and store presence
```

Out[ ]:

<AxesSubplot:xlabel='Category_2', ylabel='Store_Presence'>



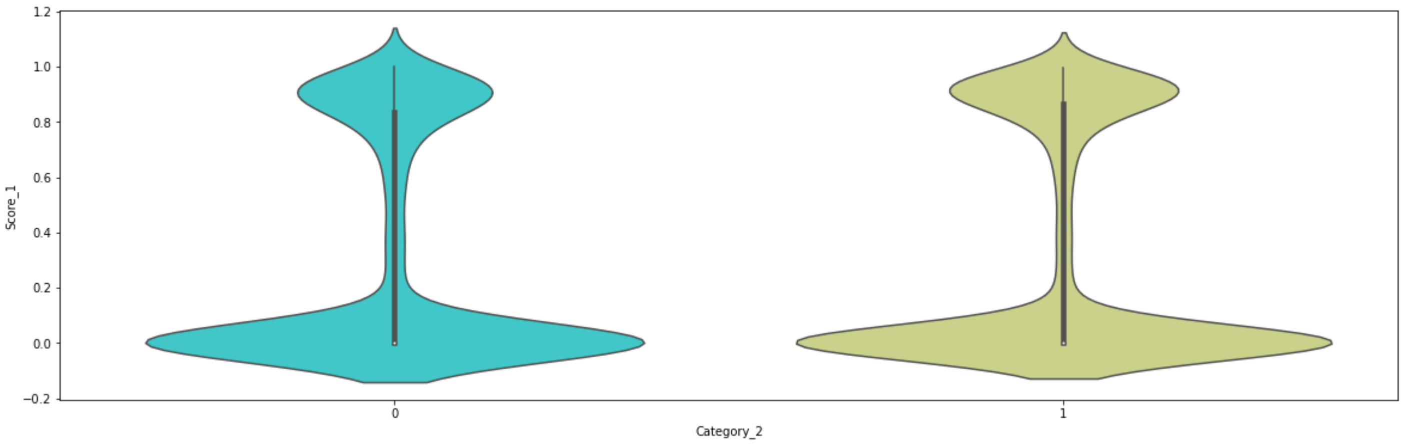
there is no relation between category2 and store presence

In [ ]:

```
# perform a violin plot between category2 and score1
```

Out[ ]:

<AxesSubplot:xlabel='Category_2', ylabel='Score_1'>



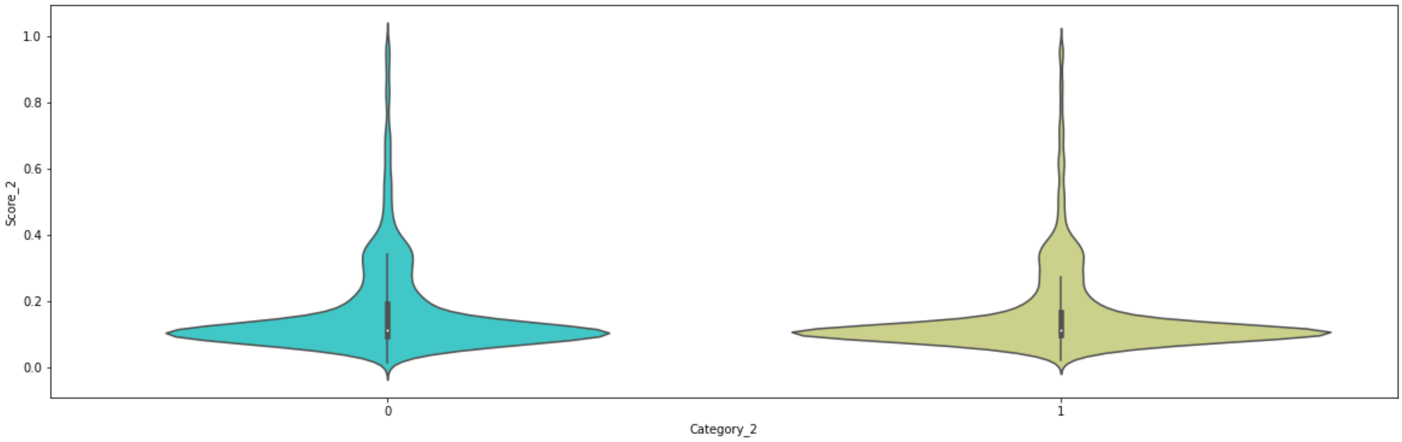
there is no relation between category2 and score1

In [ ]:

```
# perform a violin plot between category2 and score2
```

Out[ ]:

<AxesSubplot:xlabel='Category_2', ylabel='Score_2'>



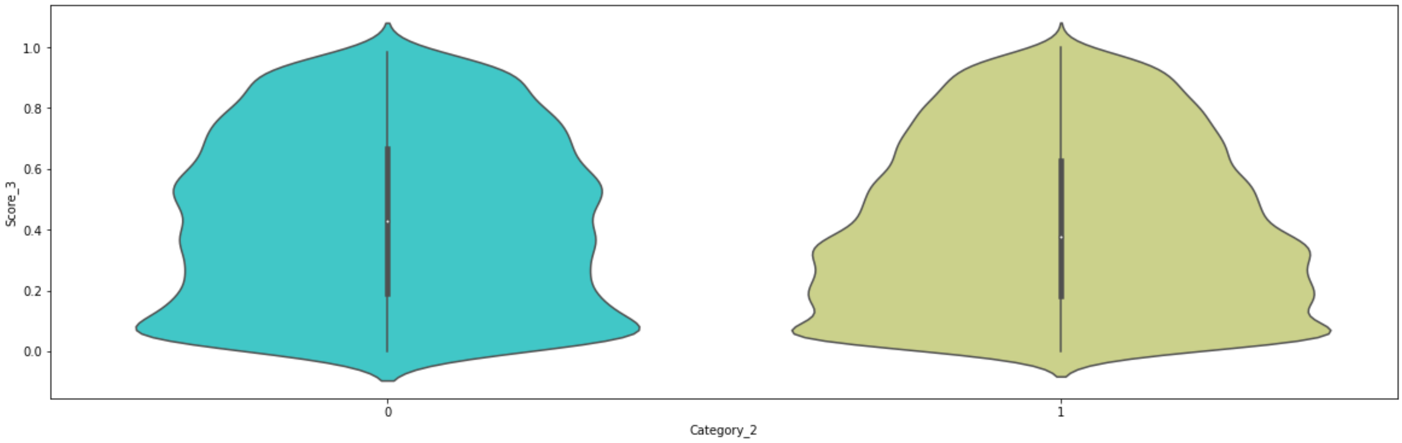
there is no relation between category2 and score2

In [ ]:

```
# perform a violin plot between category2 and score3
```

Out[ ]:

<AxesSubplot:xlabel='Category_2', ylabel='Score_3'>



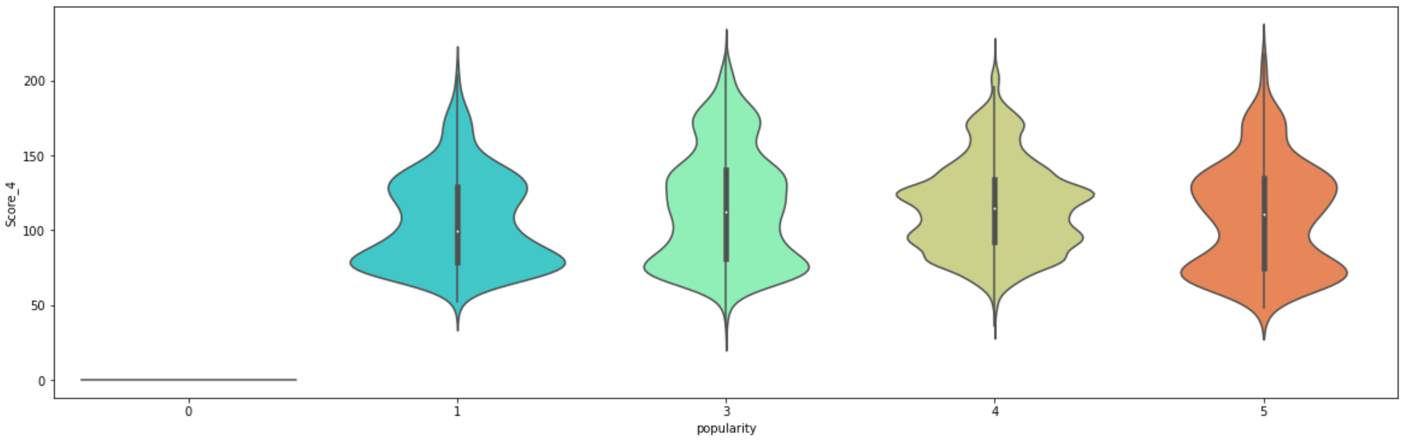
there is no relation between category2 and score3

In [ ]:

```
# perform a violin plot between popularity and score4
```

Out[ ]:

<AxesSubplot:xlabel='popularity', ylabel='Score_4'>



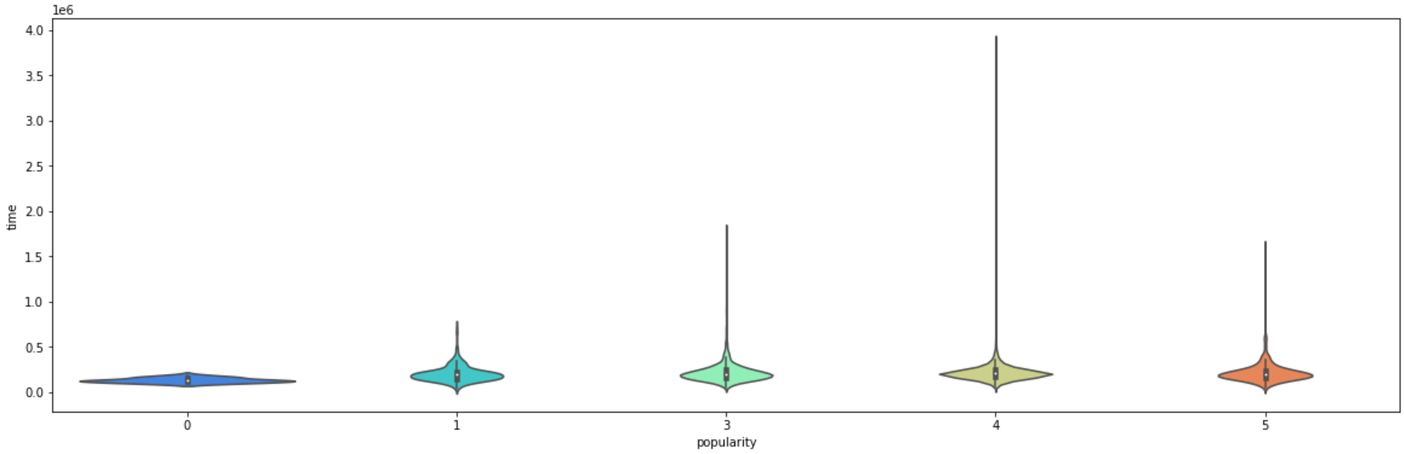
there is no relation between popularity and score 4

In [ ]:

```
# perform a violin plot between popularity and time
```

Out[ ]:

```
<AxesSubplot:xlabel='popularity', ylabel='time'>
```



there is on relation between time and popularity

Preprocessing

In [ ]:

```
#convert the time column into more columns like hour, month, day, year , minute
train_df['hour'] =

train_df['month'] =

train_df['day'] =

train_df['year'] =

train_df['minute'] =
```

In [ ]:

```
# drop popularity from train_df and name that variable as y

#drop popularity n time from train dataset
y =

train_df =
```

Scaling

Why scaling is necessary?

1. Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidean distance between two data points in their computations, this is a problem.
2. If left alone, these algorithms only take in the magnitude of features neglecting the units.
3. The results would vary greatly between different units, 5kg and 5000gms.
4. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.
5. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

min max scaling

Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias. Thus, to deal with this potential problem feature-wise normalization such as MinMax Scaling is usually used prior to model fitting.

In [ ]:

```
# Helper function for scaling all the numerical data using MinMaxScalar
# import asarray
#  import MinMaxScaler
def scale_data(df,col):

  scaler =
```

```
  df[col] =

  return df
```

In [ ]:

```
# Making a list of the column names to be scaled

# passing data and column name for scaling
col_X = ['Store_Ratio', 'Basket_Ratio', 'Store_Score', 'Store_Presence', 'Score_1', 'Score_2', 'Score_3',
X =
X =
```

Splitting the data into train and test set

In [ ]:

```
# split the dataset  into test and train
# 90% train , 10% test and random state 42
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
```

In [ ]:

```
# print X_train shape, y_train shape
```

Out[ ]:

```
((14566, 15), (14566,))
```

In [ ]:

```
# check for nan value in X_train
np.any(np.isnan(X_train))
```

Out[ ]:

```
False
```

Modelling

In [ ]:

```
# importing necessary libraries for geting metrics of models
import math
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Function for calculating all the relevant metrics
def print_score(m):
    res =

    print("Classification Report \n",res)
```

In [ ]:

```
# Visualize importance of all the features in the dataset for the prediction

def visualize_importance(feature_importances, feat_train_df):

    # creating dataframe for feature name and feature importance
    feature_importance_df =

    _df = pd.DataFrame()
    _df['feature_importance'] =
    _df['column'] =
    feature_importance_df =


    # grouping all data and sorting in descending order
    order =


    # ploting feature importance data using boxenplot
    fig, ax =
```

```
    return fig, ax
```

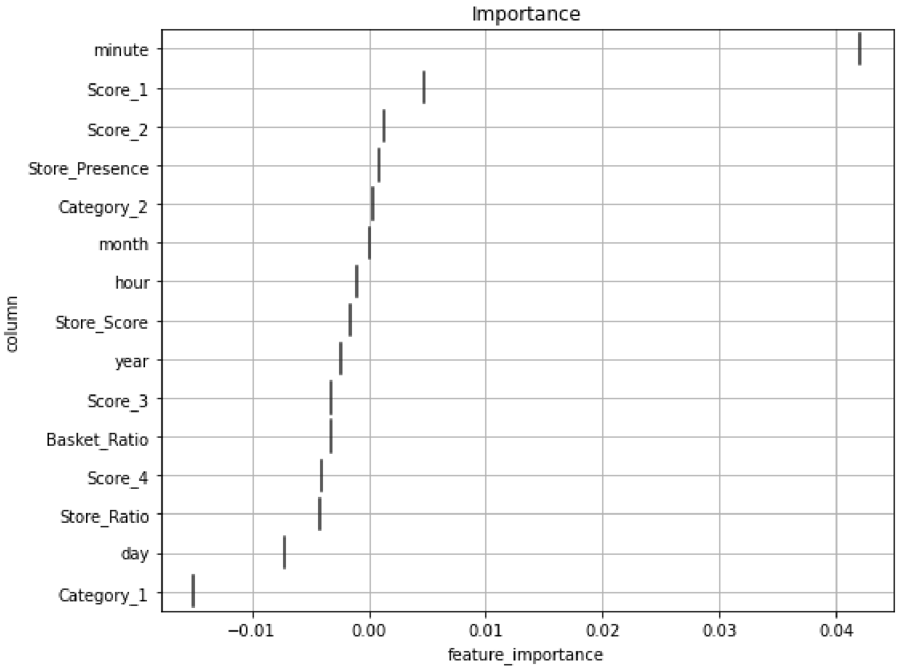
LOGISTIC REGRESSION

1. Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
2. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

In [ ]:

```
%%time
# Fit a logistic Regression model to the train dataset

# Import logisticRegression



# Instantiate the model
log_class =


# fitting the model on train data



# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00        74
           3       0.22      0.01      0.02       444
           4       0.83      1.00      0.91      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.83      3642
   macro avg       0.21      0.20      0.19      3642
weighted avg       0.72      0.83      0.76      3642

Wall time: 597 ms
```



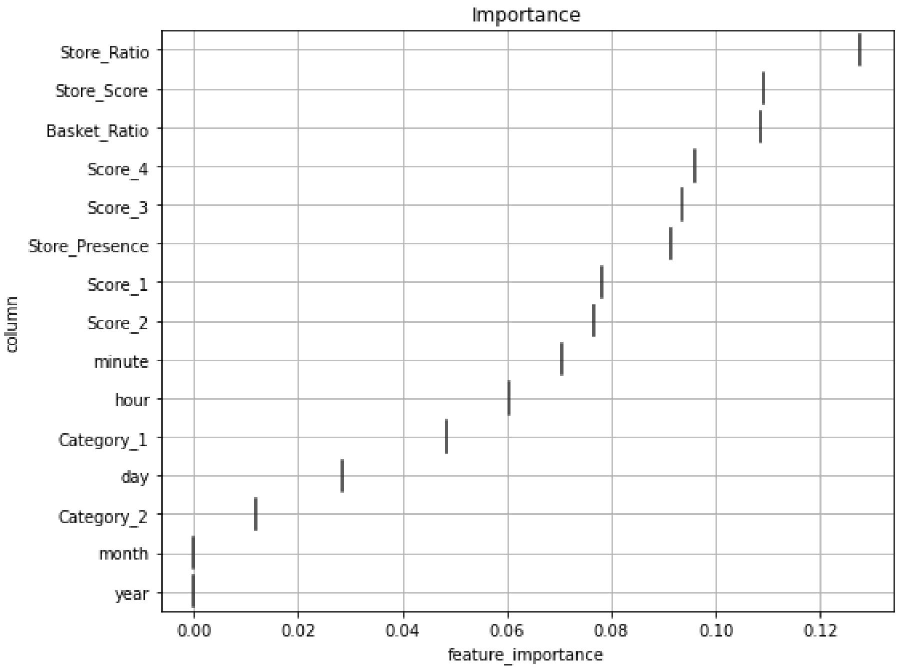
RANDOM FOREST CLASSIFIER

random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

In [ ]:

```
%%time
# Fit a RandomForestClassifier model to the train dataset

#import RandomForestClassifier



# Instantiate the model
rf_clf =


# fitting the model on train data



# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      0.35      0.52        74
           3       0.91      0.31      0.46       444
           4       0.88      1.00      0.93      3030
           5       1.00      0.32      0.48        91

    accuracy                           0.88      3642
   macro avg       0.96      0.60      0.68      3642
weighted avg       0.89      0.88      0.86      3642

Wall time: 4.4 s
```



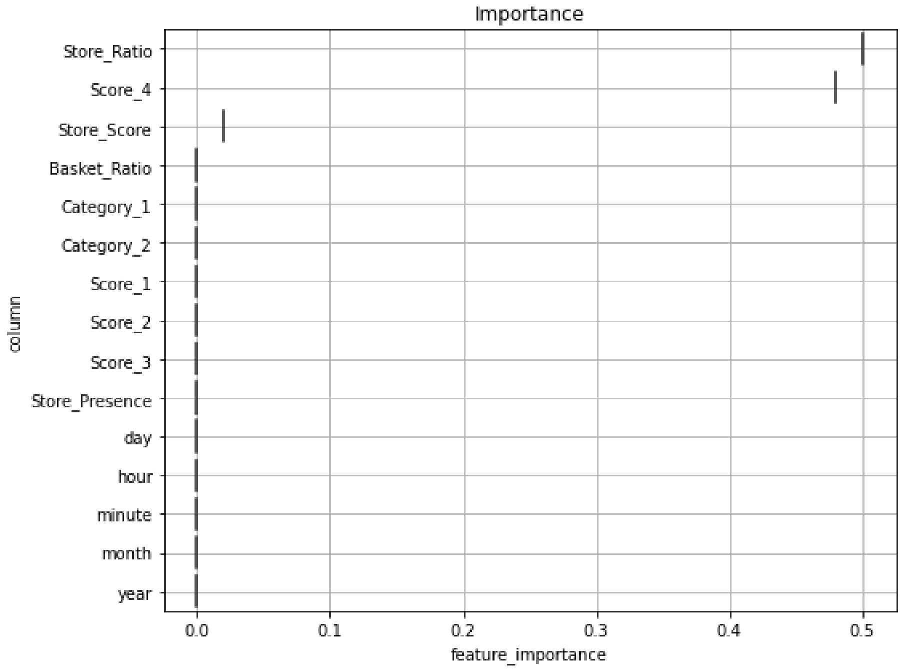
ADA BOOST CLASSIFIER

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

In [ ]:

```
%%time
# Fit a AdaBoost classifier model to the train dataset

# Import AdaBoostClassifier



# Instantiate the model
Ada_clf =


# fitting the model on train data




# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.01      0.18      0.01        74
           3       0.00      0.00      0.00       444
           4       0.69      0.34      0.45      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.29      3642
   macro avg       0.34      0.30      0.29      3642
weighted avg       0.57      0.29      0.38      3642

Wall time: 840 ms
```



SUPORT VECTOR CLASSIFIER

1. A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

In [ ]:

```
%%time
# Fit a support vector classifier model to the train dataset

#import SVC




# Instantiate the model
svc =



#fit the model on train data
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00        74
           3       0.00      0.00      0.00       444
           4       0.83      1.00      0.91      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.83      3642
   macro avg       0.17      0.20      0.18      3642
weighted avg       0.69      0.83      0.76      3642

Wall time: 4.95 s
```

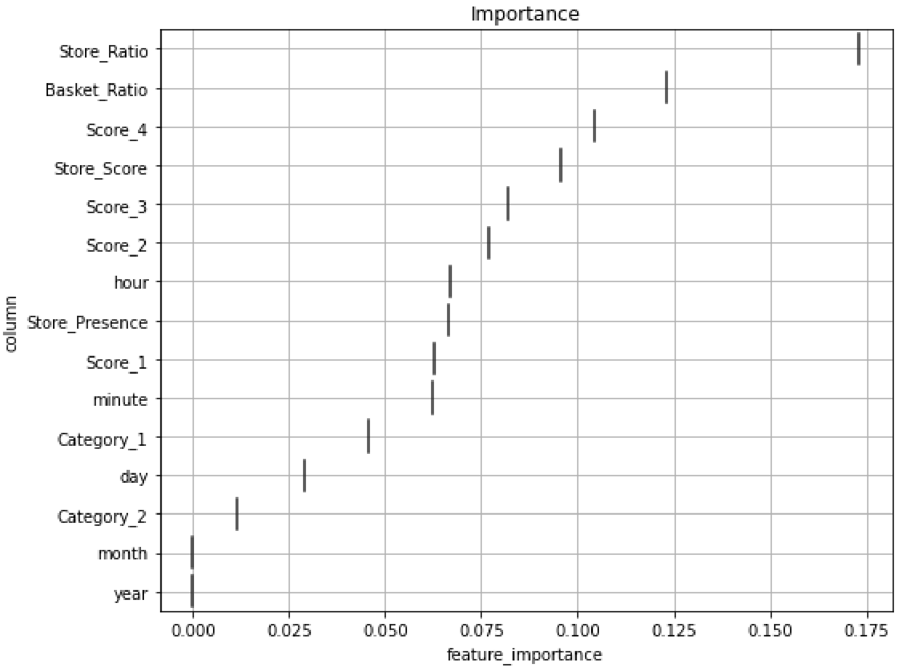
DESISION TREE CLASSICIFIER

Decision Tree Classifier is a simple and widely used classification technique. It applies a straitforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time time it receive an answer, a follow-up question is asked until a conclusion about the calss label of the record is reached.

In [ ]:

```
%%time
# Fit a DecisionTreeClassifier model to the train dataset

#import DecisionTreeClassifier



# Instantiate the model
dt_clf =


# fitting the model on train data



# print score of the model




# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.31      0.38      0.34        74
           3       0.46      0.48      0.47       444
           4       0.91      0.89      0.90      3030
           5       0.29      0.35      0.32        91

    accuracy                           0.82      3642
   macro avg       0.59      0.62      0.61      3642
weighted avg       0.83      0.82      0.82      3642

Wall time: 247 ms
```



K NEIGHBOUR CLASSIFIER

In [ ]:

```
%%time
# Fit a K-Neighbour classifier model to the train dataset

# Import KNeighborsClassifier



# Instantiate the model
knn_clf =


# fitting the model on train data



# print score of the model
```

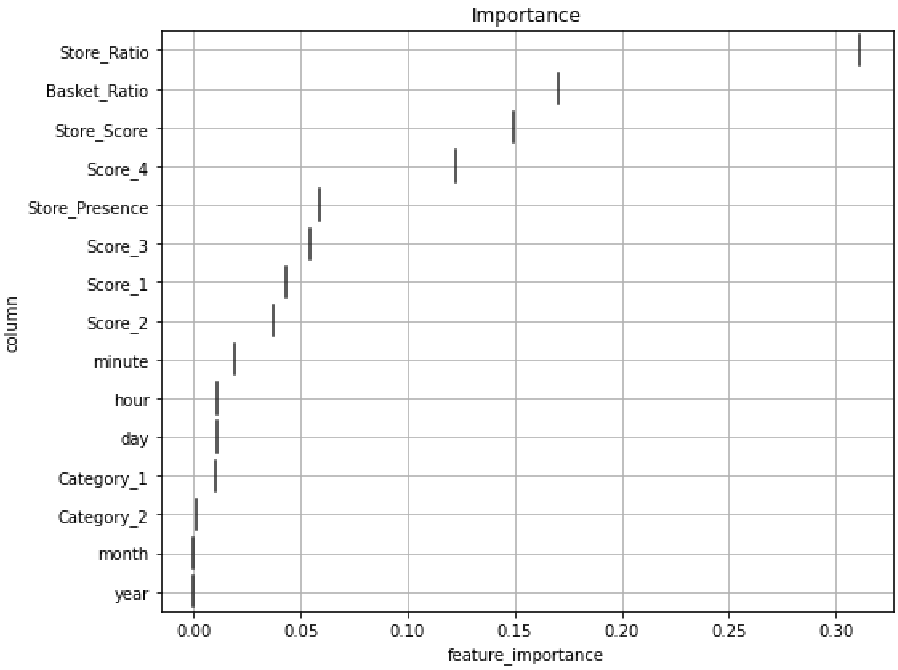
```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.23      0.04      0.07        74
           3       0.37      0.15      0.21       444
           4       0.85      0.96      0.90      3030
           5       0.43      0.10      0.16        91

    accuracy                           0.82      3642
   macro avg       0.38      0.25      0.27      3642
weighted avg       0.77      0.82      0.78      3642

Wall time: 226 ms
```

GRADIENT BOOSTING CLASSIFIER

In [ ]:

```
%%time
# Fit a Gradient Boosting Classifier model to the train dataset

# Import GradientBoostingClassifier



# Instantiate the model
GBR_clf =


# fitting the model on train data



# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.67      0.16      0.26        74
           3       0.55      0.11      0.18       444
           4       0.85      0.99      0.91      3030
           5       0.75      0.13      0.22        91

    accuracy                           0.84      3642
   macro avg       0.76      0.48      0.52      3642
weighted avg       0.81      0.84      0.79      3642

Wall time: 15.7 s
```



BAGGING CLASSIFIER

In [ ]:

```
%%time
# Fit a Bagging Classifier model to the train dataset
```

```
# Import BaggingClassifier


# Instantiate the model
bg_clf =


# fitting the model on train data



# print score of the model
```

```
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.76      0.35      0.48        74
           3       0.56      0.41      0.47       444
           4       0.89      0.96      0.92      3030
           5       0.81      0.29      0.42        91

    accuracy                           0.86      3642
   macro avg       0.81      0.60      0.66      3642
weighted avg       0.85      0.86      0.85      3642

Wall time: 947 ms
```

VOTING CLASSIFIER

In [ ]:

```
%%time
# Fit a VotingClassifier model to the train dataset

# Import VotingClassifier



# list of classifier objects
classifiers = [("knn", knn_clf), ("svc", svc), ("dt", dt_clf), ("rf_clf", rf_clf)]

# Instantiate the model
voting_clf =


# fitting the model on train data



# print score of the model
```

```
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.96      0.35      0.51        74
           3       0.85      0.31      0.45       444
           4       0.87      0.99      0.93      3030
           5       1.00      0.11      0.20        91

    accuracy                           0.87      3642
   macro avg       0.94      0.55      0.62      3642
weighted avg       0.88      0.87      0.84      3642

Wall time: 12.3 s
```

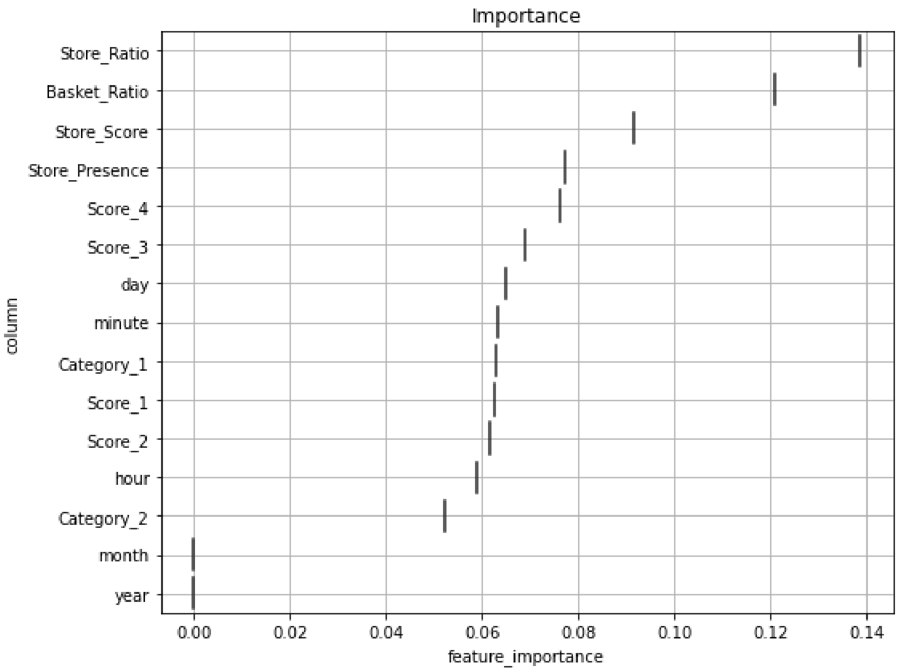
XGB CLASSIFIER

In [ ]:

```
%%time
# Fit a XGBClassifier model to the train dataset
```

```
# Import XGBClassifier



# Instantiate the model
xgb_clf =


# fitting the model on train data



# print score of the model



# visualizing importance of features
fig, ax =
```

```
[00:15:28] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095:
Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.90      0.35      0.50        74
           3       0.68      0.31      0.43       444
           4       0.88      0.98      0.93      3030
           5       0.90      0.31      0.46        91

    accuracy                           0.87      3642
   macro avg       0.87      0.59      0.66      3642
weighted avg       0.86      0.87      0.85      3642

Wall time: 4.93 s
```



Comparing all the model based on metric

In [ ]:

```
# import metrics, train_test_split


def compare_models(models,names,X_train,y_train,X_test,y_test):
    # the libraries we need


    for (model,name) in zip(models,names):
      print(name)
```

```
    # then predict on the test set
    y_pred =
    res =
    print("Classification Report \n",res)
    print("------------------------------------------------------------------------------------")
```

In [ ]:

```
# list of model objects
models= [log_class,rf_clf,Ada_clf,svc, dt_clf, knn_clf, GBR_clf,bg_clf, voting_clf, xgb_clf ]
# list of model names
names = ['logistic', 'rf','AdaBoost', 'svc', 'Dtree','KNN','GBR','bagging','voting','XGB']
# print the comparison of models
```

```
logistic
Classification Report
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00        74
           3       0.22      0.01      0.02       444
           4       0.83      1.00      0.91      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.83      3642
   macro avg       0.21      0.20      0.19      3642
weighted avg       0.72      0.83      0.76      3642

------------------------------------------------------------------------------------
rf
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      0.35      0.52        74
           3       0.91      0.31      0.46       444
           4       0.88      1.00      0.93      3030
           5       1.00      0.32      0.48        91

    accuracy                           0.88      3642
   macro avg       0.96      0.60      0.68      3642
weighted avg       0.89      0.88      0.86      3642

------------------------------------------------------------------------------------
AdaBoost
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.01      0.18      0.01        74
           3       0.00      0.00      0.00       444
           4       0.69      0.34      0.45      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.29      3642
   macro avg       0.34      0.30      0.29      3642
weighted avg       0.57      0.29      0.38      3642

------------------------------------------------------------------------------------
svc
Classification Report
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00        74
           3       0.00      0.00      0.00       444
           4       0.83      1.00      0.91      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.83      3642
   macro avg       0.17      0.20      0.18      3642
weighted avg       0.69      0.83      0.76      3642

------------------------------------------------------------------------------------
Dtree
Classification Report
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.31      0.38      0.34        74
           3       0.46      0.48      0.47       444
           4       0.91      0.89      0.90      3030
           5       0.29      0.35      0.32        91

    accuracy                           0.82      3642
   macro avg       0.59      0.62      0.61      3642
weighted avg       0.83      0.82      0.82      3642

-------------------------------------------------------------------------------------
KNN
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.23      0.04      0.07        74
           3       0.37      0.15      0.21       444
           4       0.85      0.96      0.90      3030
           5       0.43      0.10      0.16        91

    accuracy                           0.82      3642
   macro avg       0.38      0.25      0.27      3642
weighted avg       0.77      0.82      0.78      3642

-------------------------------------------------------------------------------------
GBR
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.67      0.16      0.26        74
           3       0.55      0.11      0.18       444
           4       0.85      0.99      0.91      3030
           5       0.75      0.13      0.22        91

    accuracy                           0.84      3642
   macro avg       0.76      0.48      0.52      3642
weighted avg       0.81      0.84      0.79      3642

-------------------------------------------------------------------------------------
bagging
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.76      0.35      0.48        74
           3       0.56      0.41      0.47       444
           4       0.89      0.96      0.92      3030
           5       0.81      0.29      0.42        91

    accuracy                           0.86      3642
   macro avg       0.81      0.60      0.66      3642
weighted avg       0.85      0.86      0.85      3642

-------------------------------------------------------------------------------------
voting
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.96      0.35      0.51        74
           3       0.85      0.31      0.45       444
           4       0.87      0.99      0.93      3030
           5       1.00      0.11      0.20        91

    accuracy                           0.87      3642
   macro avg       0.94      0.55      0.62      3642
weighted avg       0.88      0.87      0.84      3642

-------------------------------------------------------------------------------------
XGB
Classification Report
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00         3
           1       0.90      0.35      0.50        74
           3       0.68      0.31      0.43       444
           4       0.88      0.98      0.93      3030
           5       0.90      0.31      0.46        91

    accuracy                           0.87      3642
   macro avg       0.87      0.59      0.66      3642
weighted avg       0.86      0.87      0.85      3642

--------------------------------------------------------------------------------------------
```

Hyperparameter tuning

A hyperparameter is a parameter whose value is set before the learning process begins.

Hyperparameters tuning is crucial as they control the overall behavior of a machine learning model.

Every machine learning models will have different hyperparameters that can be set.

grid search

One traditional and popular way to perform hyperparameter tuning is by using an Exhaustive Grid Search from Scikit learn.

This method tries every possible combination of each set of hyper-parameters.

Using this method, we can find the best set of values in the parameter search space.

This usually uses more computational power and takes a long time to run since this method needs to try every combination in the grid size.

In [ ]:

```
%%time
# Helper function to perform hyper parameter tunning with GridSearchCV
def grid_search(model,grid):
  from sklearn.model_selection import GridSearchCV, train_test_split
  from sklearn.model_selection import KFold
  from sklearn.model_selection import GridSearchCV


  cv =

  clf =

  # print clf.score and best_params_
```

```
Wall time: 0 ns
```

In [ ]:

```
%%time
# create  parameters dict in list for tunning
log_para_grid = {
    'C':10.0 **np.arange(-2,3),
    'penalty':['l1','l2']
    }

# passing data for hyper parameter tunning with Gridsearchcv
```

```
0.18232898718571372
{'C': 10.0, 'penalty': 'l2'}
Wall time: 9.23 s
```

NOTE : you can use any one of RandomizedSearchCv or GridSearchCV, both works fine.

RamdomizedSearchCV

In [ ]:

```python
# Helper function to perform hyper parameter tunning with RandomizedSearchCV
def random_Search(model,X_train, Y_train,param_grid):
  from sklearn.model_selection import RandomizedSearchCV

  # Random search of parameters, using 3 fold cross validation,
  # search across 100 different combinations, and use all available cores
  # Fit the random search model
  random =

  #fit on train

  # print best_params_
```

In [ ]:

```python
%%time
# create parameters dict for tunning
rf_para_grid = {'n_estimators': list(range(150, 301, 50)),
               'max_features': ['auto', 'sqrt'],
               'max_depth': [int(x) for x in np.linspace(3, 10, num = 3)],
               'min_samples_split': [2, 5],
               'min_samples_leaf': [1, 2],
               'bootstrap': [True, False]}

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth':
10, 'bootstrap': False}
Wall time: 19.5 s
```

In [ ]:

```python
%%time
# create parameters dict for tunning
GBR_para_grid = {
'n_estimators':  [x for x in range(200,351, 50)],
 'learning_rate' : [0.01, 0.1, 0.2],
 'max_depth': [x for x in range(5,7)],
 'min_samples_split': [x for x in range(2,6)]
}

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_estimators': 300, 'min_samples_split': 5, 'max_depth': 6, 'learning_rate': 0.2}
Wall time: 8min 9s
```

In [ ]:

```python
%%time
# create parameters dict for tunning
knn_para_grid = {
                  'leaf_size' :list(range(3,15,2)),
                  'n_neighbors' : list(range(10,30))
              }

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_neighbors': 25, 'leaf_size': 7}
Wall time: 4.41 s
```

In [ ]:

```python
%%time
# create  parameters dict for tunning
DTR_para_grid = {
                  "splitter":["best","random"],
            "max_depth" : [3,5,7,9],
           "min_samples_leaf":[1,2,3,4],
           "max_features":["auto","log2","sqrt"]
              }

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'splitter': 'best', 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 3}
Wall time: 350 ms
```

In [ ]:

```
%%time
# create  parameters dict for tunning
Ada_para_grid = {
                  'n_estimators' :[100, 200, 300],
                  'learning_rate' : [0.001, 0.01, 0.1, 1.0]
              }

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_estimators': 300, 'learning_rate': 0.01}
Wall time: 24.1 s
```

In [ ]:

```
%%time
# create  parameters dict for tunning
XGB_para_grid = {"learning_rate"    : [0.05, 0.10] ,
 "max_depth"        : [ 3, 4, 5],
 "min_child_weight" : [ 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1],
 "colsample_bytree" : [ 0.3, 0.4] }

# passing data for hyper parameter tunning with Randomized search cv
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[00:48:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095:
Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was ch
anged from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
{'min_child_weight': 5, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.1, 'colsample_bytree': 0.3}
Wall time: 25.4 s
```

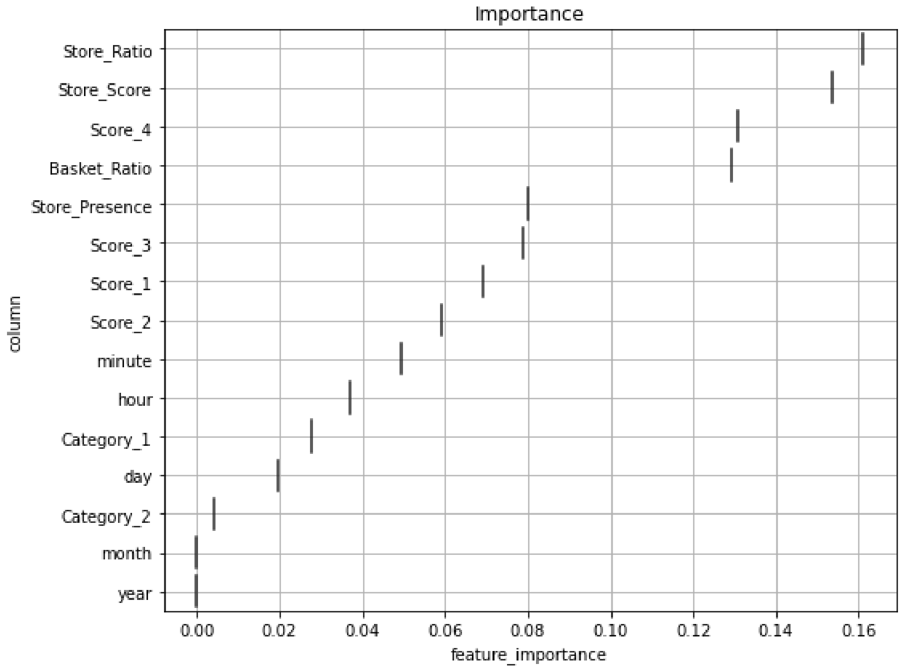
Using the tuned parameters and training the models

Gradient Boosting Classifier

In [ ]:

```
%%time

# Instantiate the model
GBR_clf =


# fitting the model on train data



# print score of the model


# visualizing importance of features
fig, ax =
```

```
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.68      0.35      0.46        74
           3       0.69      0.36      0.47       444
           4       0.89      0.98      0.93      3030
           5       0.77      0.33      0.46        91

    accuracy                           0.87      3642
   macro avg       0.81      0.60      0.67      3642
weighted avg       0.86      0.87      0.85      3642

Wall time: 1min 24s
```

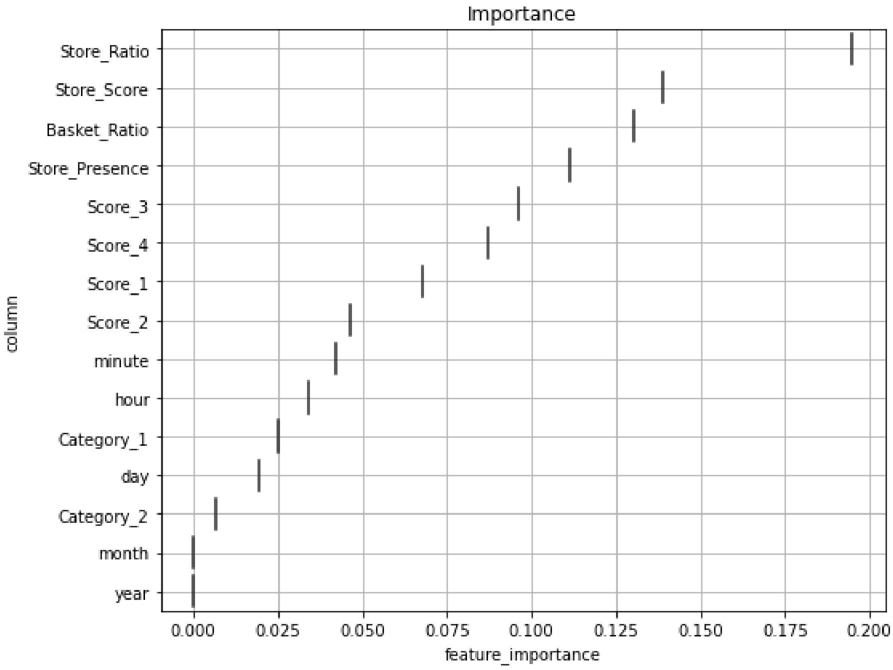


Random Forest Classifier

In [ ]:

```
%%time

# Instantiate the model
rf_clf =


# fitting the model on train data



# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      0.07      0.13        74
           3       0.93      0.09      0.16       444
           4       0.85      1.00      0.92      3030
           5       1.00      0.10      0.18        91

    accuracy                           0.85      3642
   macro avg       0.96      0.45      0.48      3642
weighted avg       0.86      0.85      0.79      3642

Wall time: 4.25 s
```

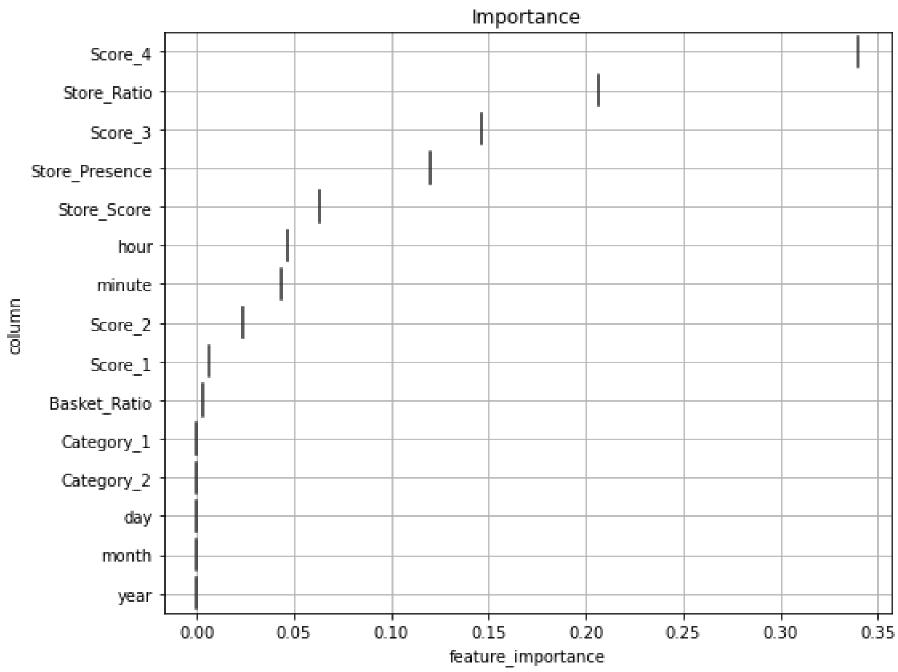


Adaboost Classifier

In [ ]:

```
%%time

# Instantiate the model
Ada_clf =


# fitting the model on train data




# print score of the model



# visualizing importance of features
fig, ax =
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       0.00      0.00      0.00        74
           3       0.00      0.00      0.00       444
           4       0.83      1.00      0.91      3030
           5       0.00      0.00      0.00        91

    accuracy                           0.83      3642
   macro avg       0.37      0.40      0.38      3642
weighted avg       0.69      0.83      0.76      3642

Wall time: 4.54 s
```



Now working with the test dataset provided

In [ ]:

```
# preparing test data as similarly as done for train data before.
test_df['hour'] =

test_df['month'] =

test_df['day'] =

test_df['year'] =

test_df['minute'] =

test_df =
```

In [ ]:

```
X_test = test_df
```

In [ ]:

```
# check columns of test data
```

Out[ ]:

```
Index(['Store_Ratio', 'Basket_Ratio', 'Category_1', 'Store_Score',
       'Category_2', 'Store_Presence', 'Score_1', 'Score_2', 'Score_3',
       'Score_4', 'hour', 'month', 'day', 'year', 'minute'],
      dtype='object')
```

In [ ]:

```
# passing test data for scaling
col_X_test = ['Store_Ratio', 'Basket_Ratio', 'Category_1', 'Store_Score',
       'Category_2', 'Store_Presence', 'Score_1', 'Score_2', 'Score_3',
       'Score_4']
X_test =
```

In [ ]:

```
# Perforn the prediction on the test dataset
y_predicted =
```

Out[ ]:

```
array([4, 4, 4, ..., 4, 4, 4], dtype=int64)
```

In [ ]:

```
# creating a dataframe of predicted results
predictions =
```

In [ ]:

```
# predicted values in dataframe
```

Out[ ]:

	0
0	4
1	4
2	4
3	4
4	4

CONCLUSION

We have performed EDA, preprocessing, build different models, visualized feature importance, did hyper parameter tunning of each model and did prediction. store ratio is most important data in the dataset. we used voting classifier for prediction

Congratulation for completing the assignment.

You have learned a lot while doing this assignment.