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

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
```

Importing the dataset

In [ ]:
```
# Load the dataset using pandas
```

In [ ]:
```
# Make a copy of the dataset
```

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 valu
es 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 valu
es 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	1
count	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	18208.000000	
mean	0.544283	0.483585	5.155536	-12.198086	0.648506	0.477702	
std	0.202709	0.302010	3.535068	8.370566	0.477450	0.380634	
min	0.000000	0.000216	0.000000	-47.576000	0.000000	0.000000	
25%	0.411000	0.200000	2.000000	-16.496250	0.000000	0.086175	
50%	0.573000	0.517000	5.000000	-9.166500	1.000000	0.430000	
75%	0.699000	0.742000	8.000000	-5.943750	1.000000	0.895000	
max	0.998000	1.000000	11.000000	-0.079000	1.000000	0.996000	

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

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence	
count	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12140.000000	12
mean	0.543776	0.488879	5.121417	-12.062847	0.642916	0.474675	
std	0.200109	0.301217	3.528765	8.300385	0.479160	0.377582	
min	0.000000	0.000000	0.000000	-46.847000	0.000000	0.000000	
25%	0.414750	0.213000	2.000000	-16.066000	0.000000	0.087775	
50%	0.570000	0.521000	5.000000	-9.046500	1.000000	0.430500	
75%	0.696000	0.745000	8.000000	-5.891750	1.000000	0.882000	
max	0.978000	1.000000	11.000000	0.662000	1.000000	0.996000	

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
```

In [ ]:
```
# get unique values in test dataset
```

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
```

Out[ ]:

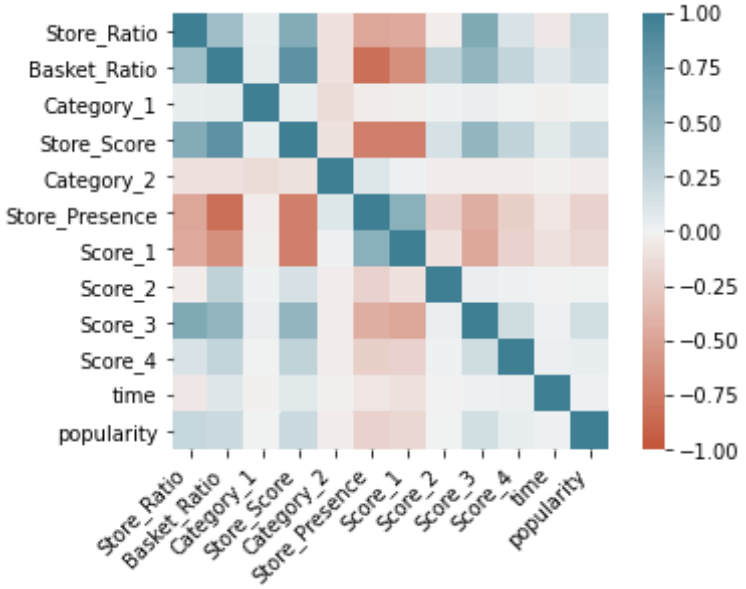
	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence
Store_Ratio	1.00	0.47	0.04	0.60	-0.10	-0.47
Basket_Ratio	0.47	1.00	0.05	0.83	-0.11	-0.82
Category_1	0.04	0.05	1.00	0.04	-0.14	-0.05
Store_Score	0.60	0.83	0.04	1.00	-0.09	-0.72
Category_2	-0.10	-0.11	-0.14	-0.09	1.00	0.10
Store_Presence	-0.47	-0.82	-0.05	-0.72	0.10	1.00
Score_1	-0.45	-0.61	-0.03	-0.73	0.02	0.57
Score_2	-0.04	0.28	0.02	0.15	-0.04	-0.19

	Store_Ratio	Basket_Ratio	Category_1	Store_Score	Category_2	Store_Presence
Score_3	0.62	0.53	0.04	0.52	-0.04	-0.43
Score_4	0.12	0.25	0.00	0.26	-0.05	-0.22
time	-0.06	0.10	-0.01	0.08	-0.01	-0.07
popularity	0.23	0.20	0.00	0.20	-0.03	-0.20

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

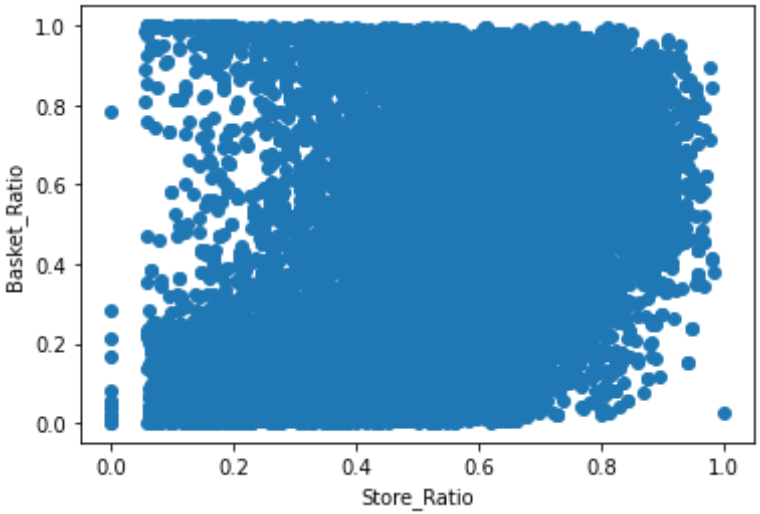
In [ ]:
```
#perfom correlation matrix Using seaborn
```

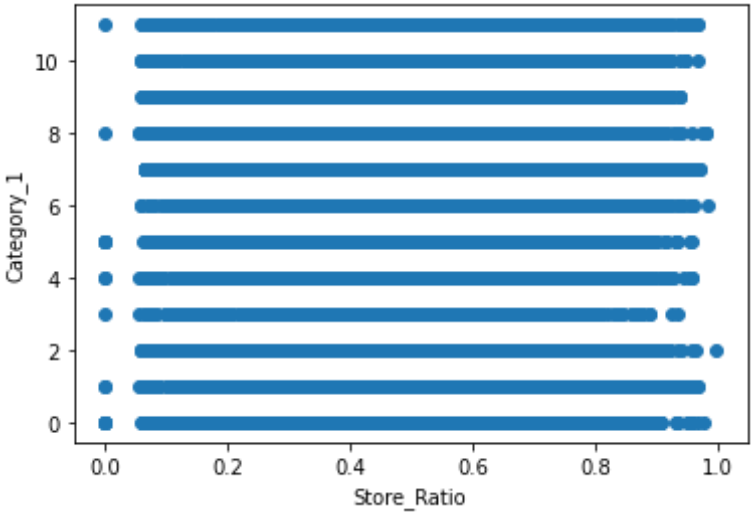


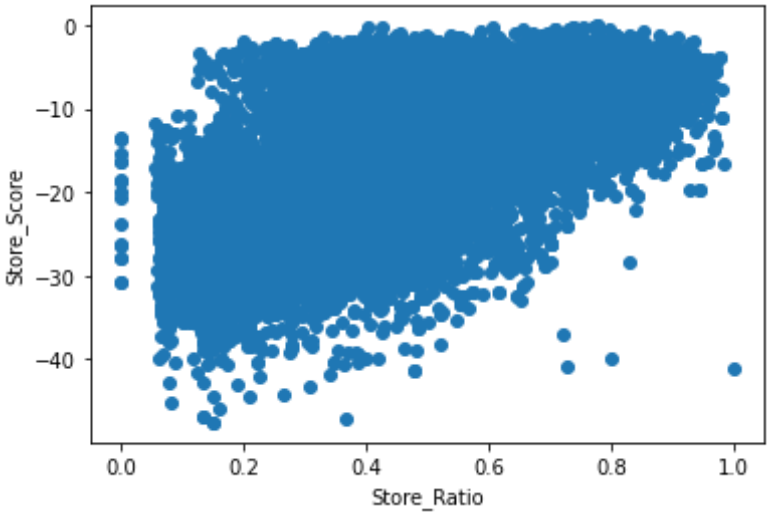
SCATTER PLOT

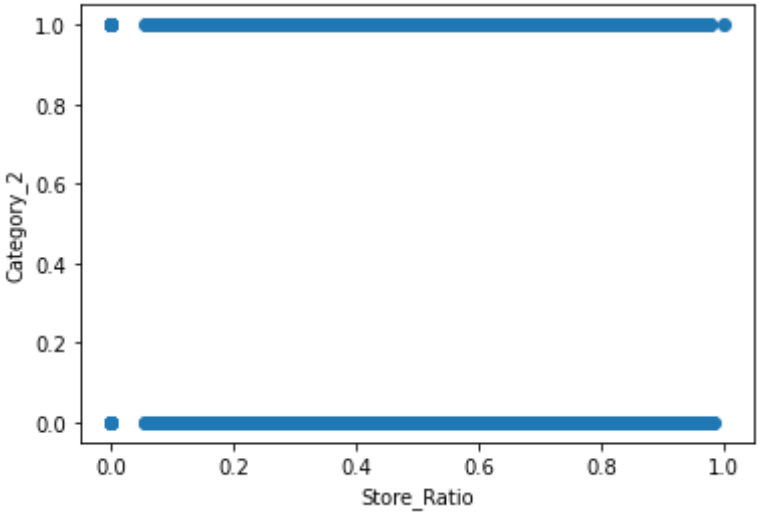
1. A scatter plot is a type of plot using Cartesian coordinates to display values for typically two variables for a set of data.
2. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable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

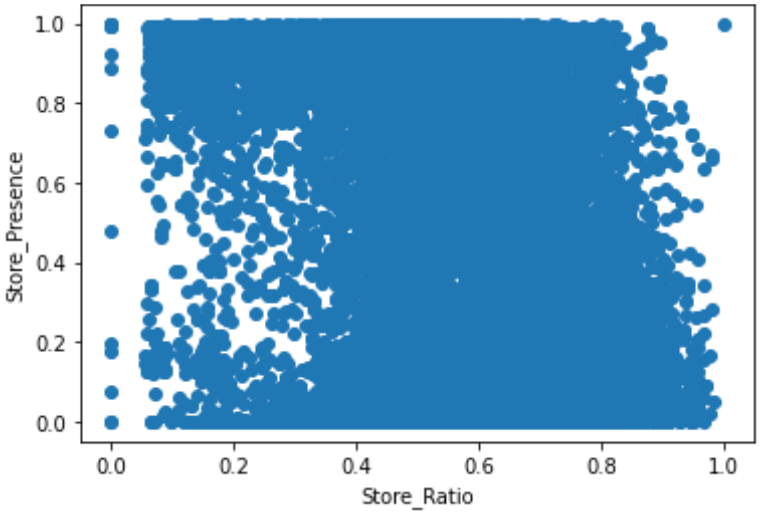
In [ ]:
```
# perform scatterplot
# Make a list of all the columns of train dataset

# Loop through the different columns
```

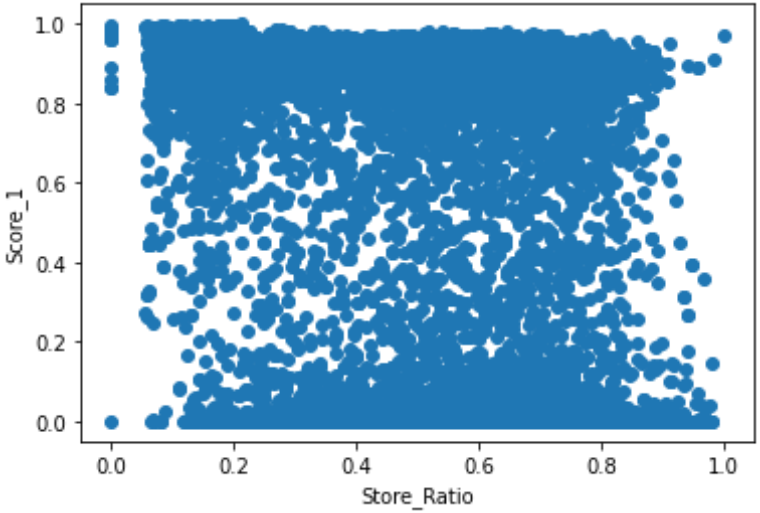


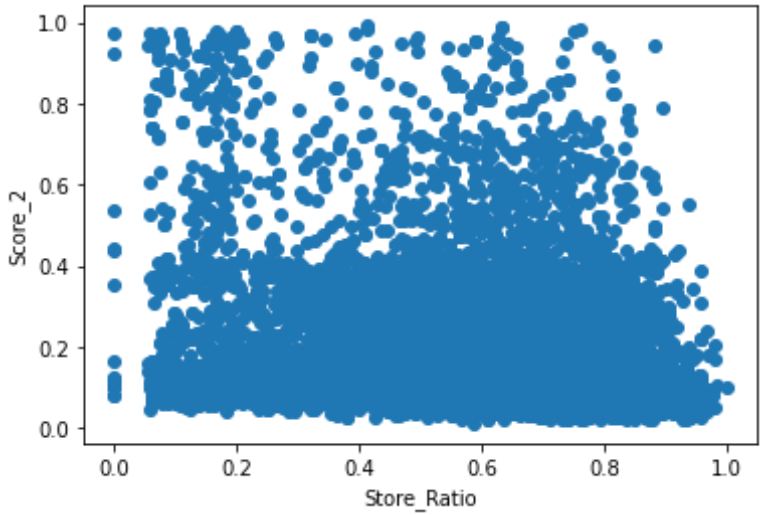


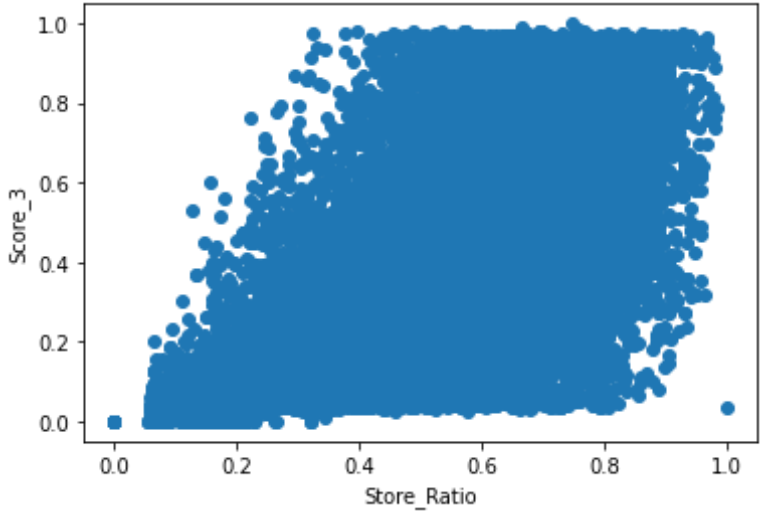


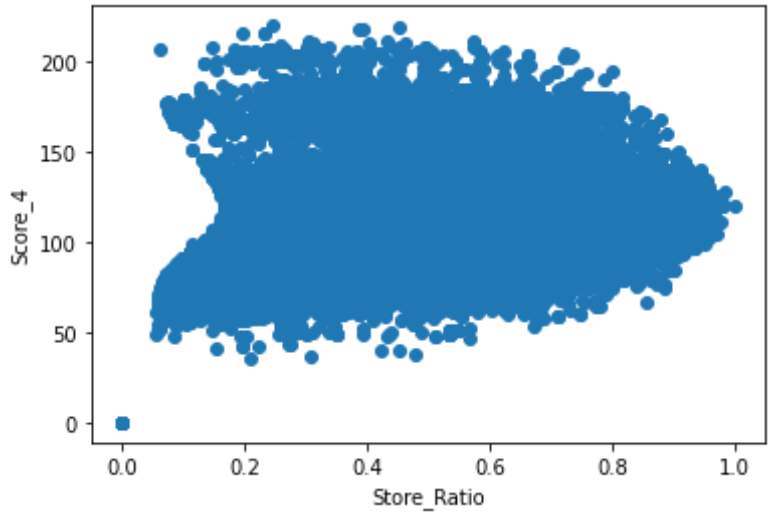


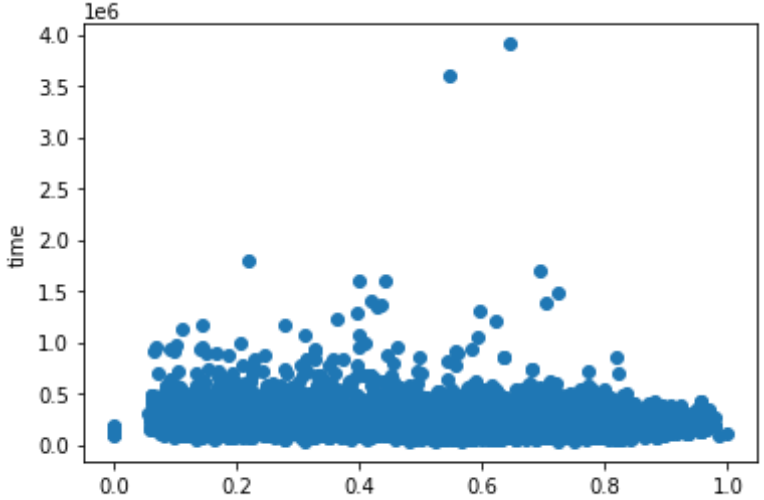




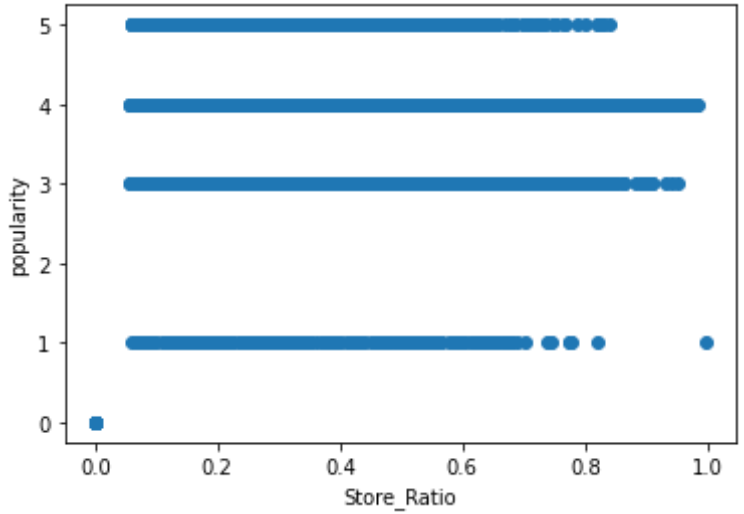


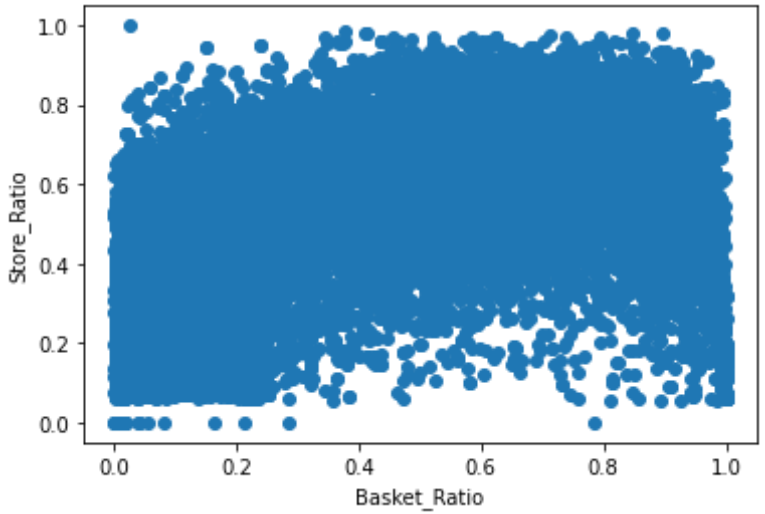


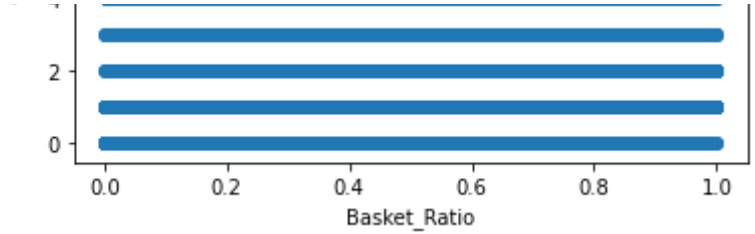


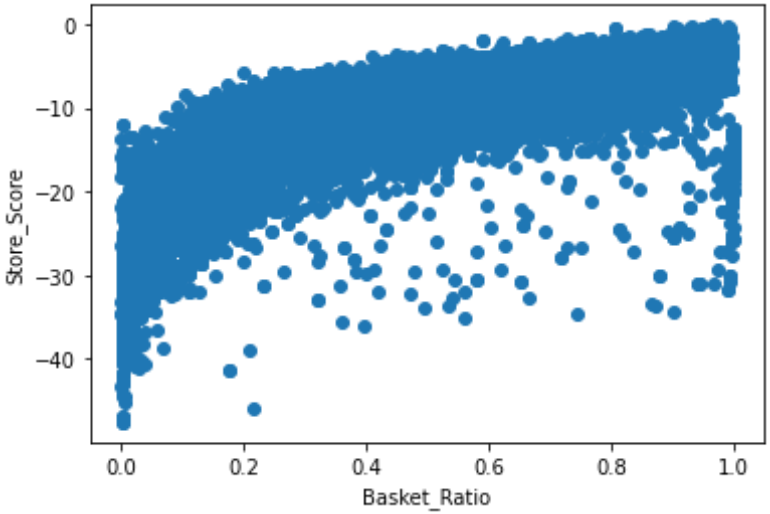


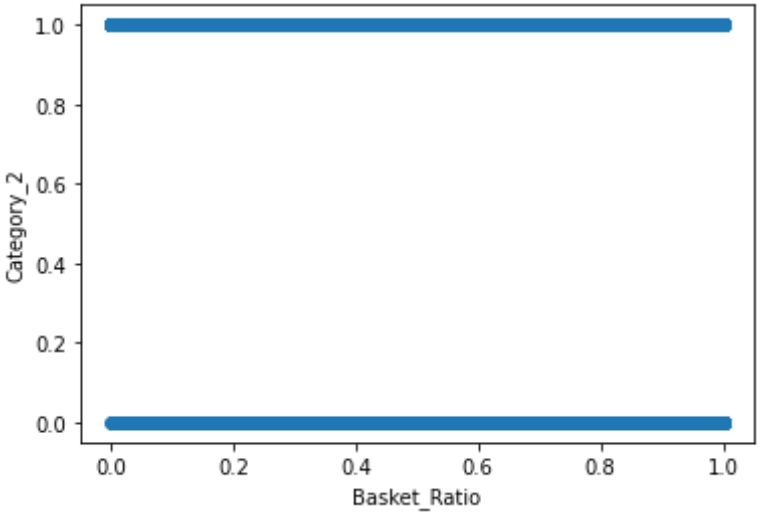
Store_Ratio

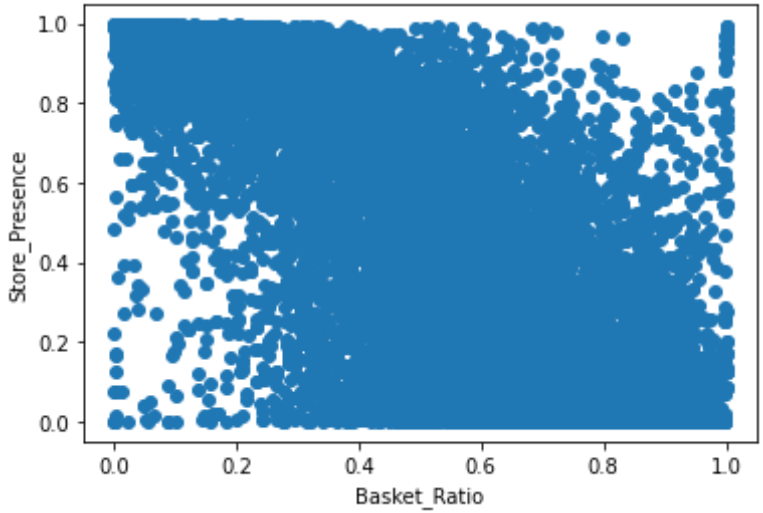


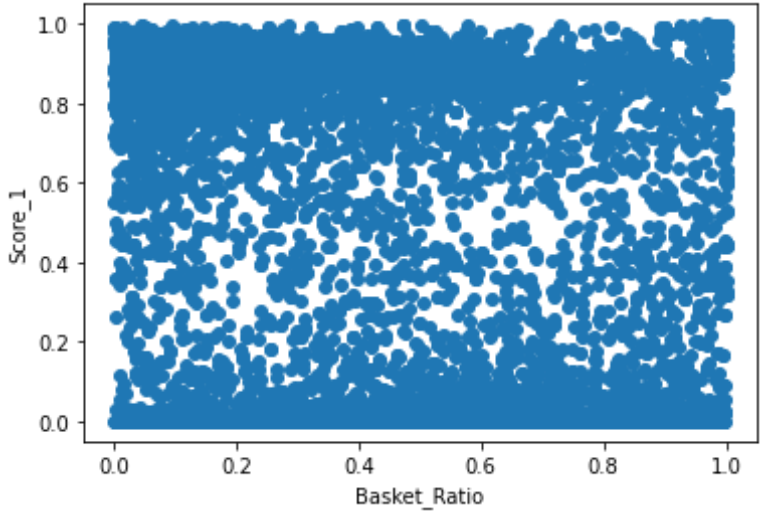


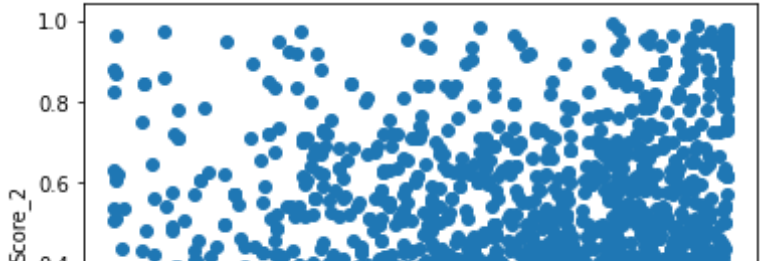


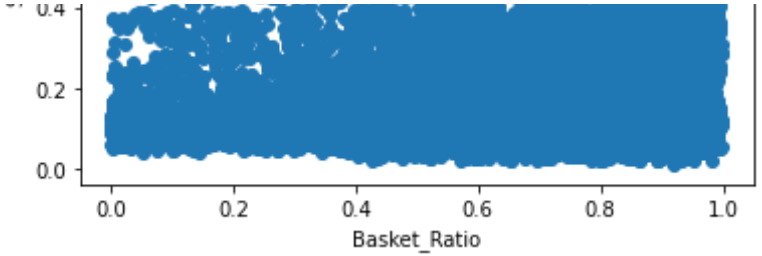


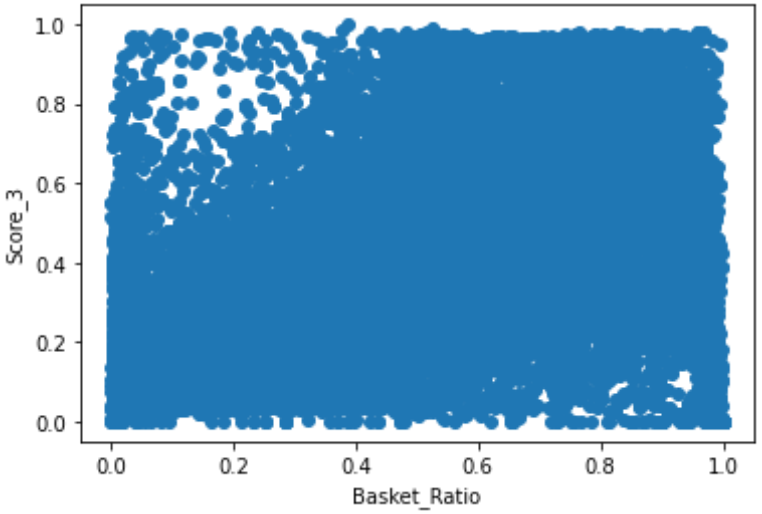


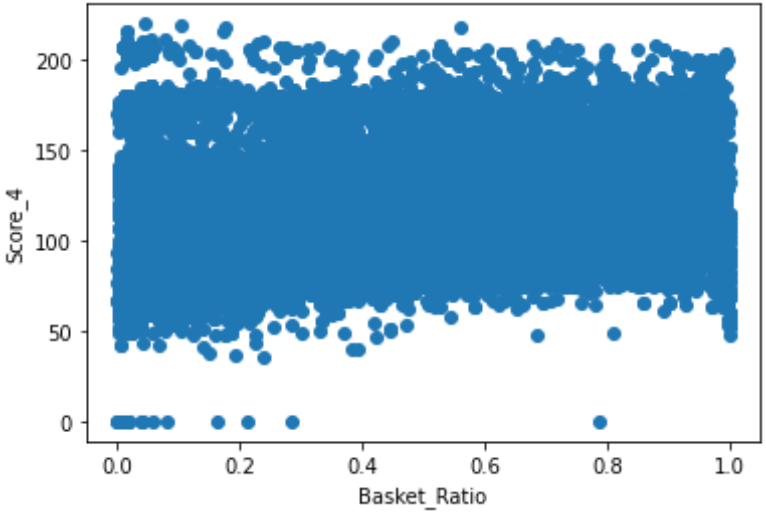


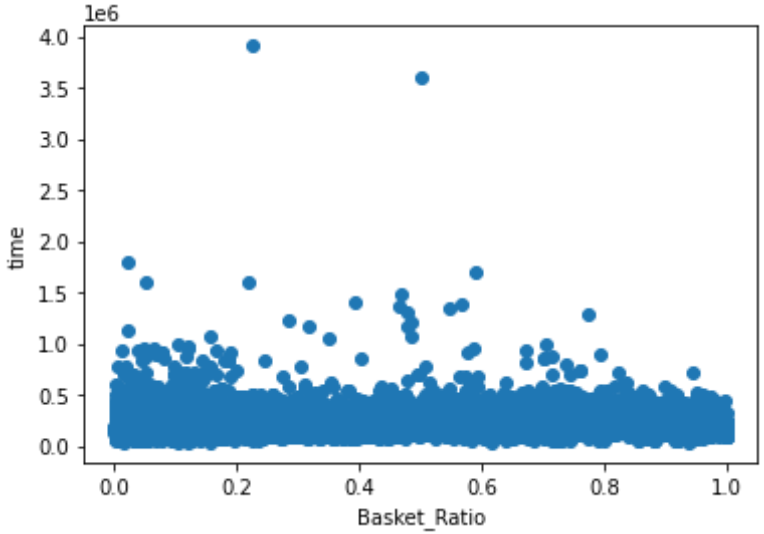


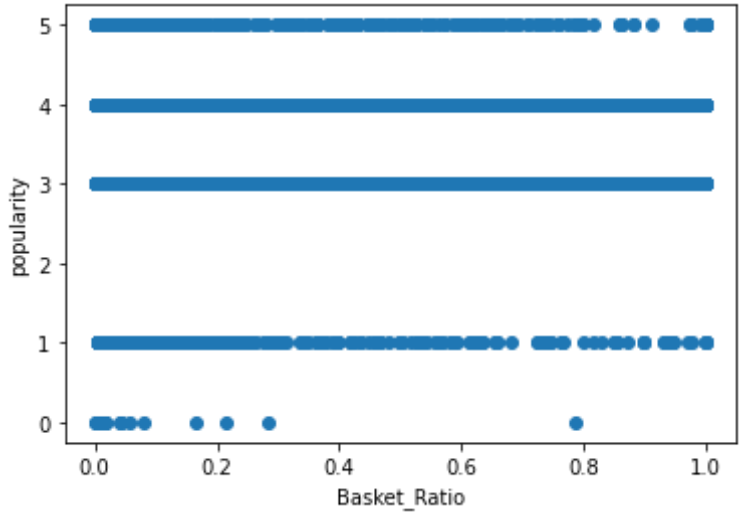


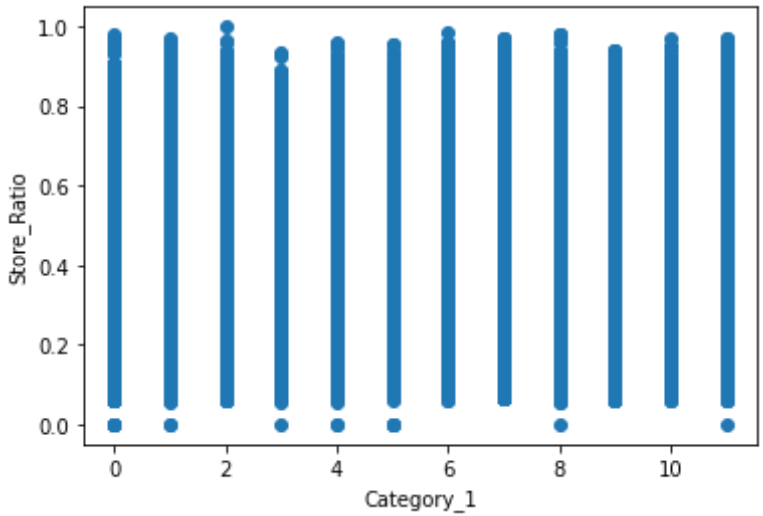


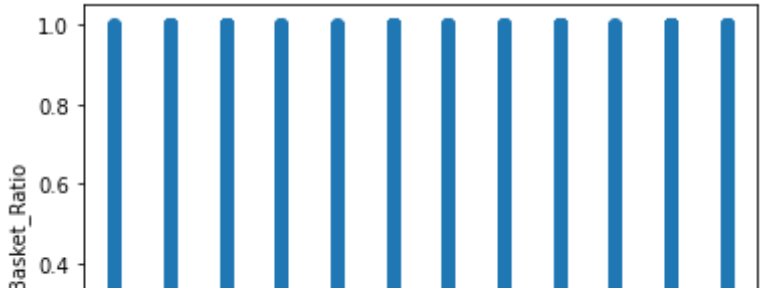


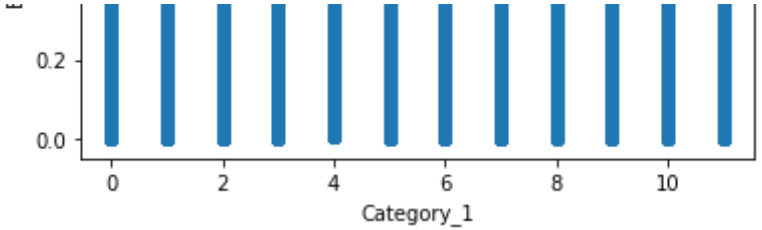


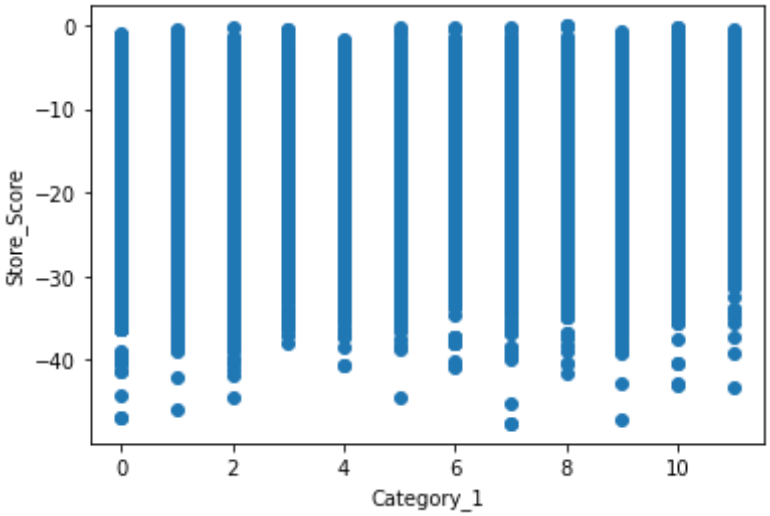


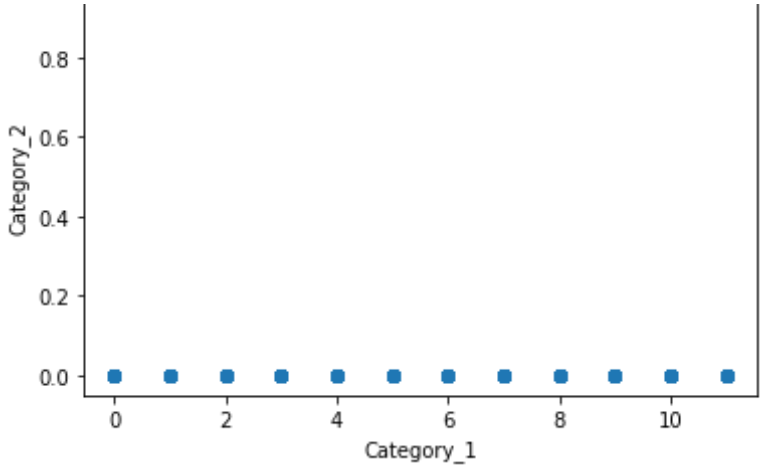


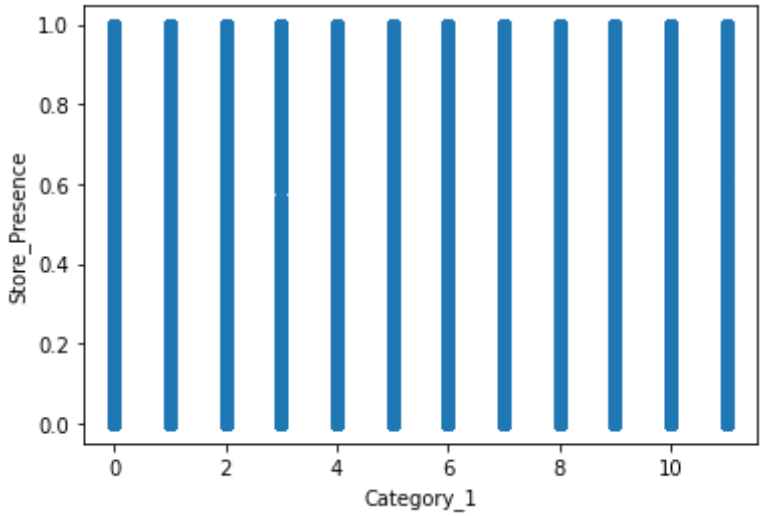


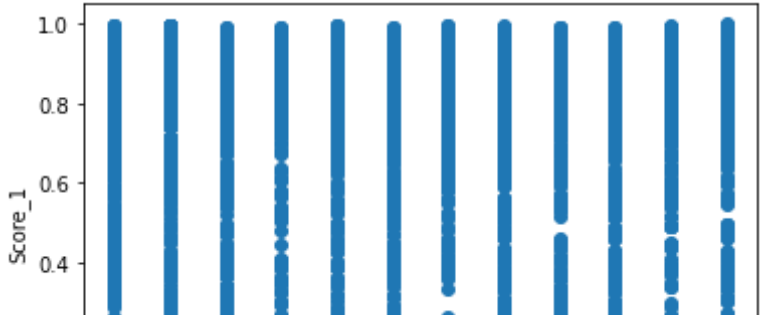


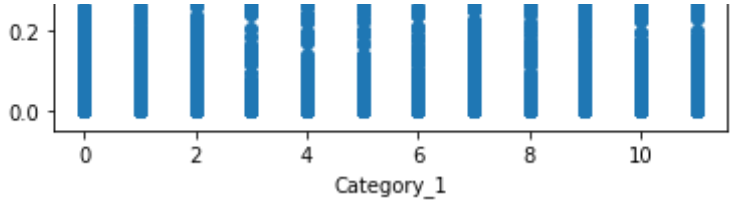


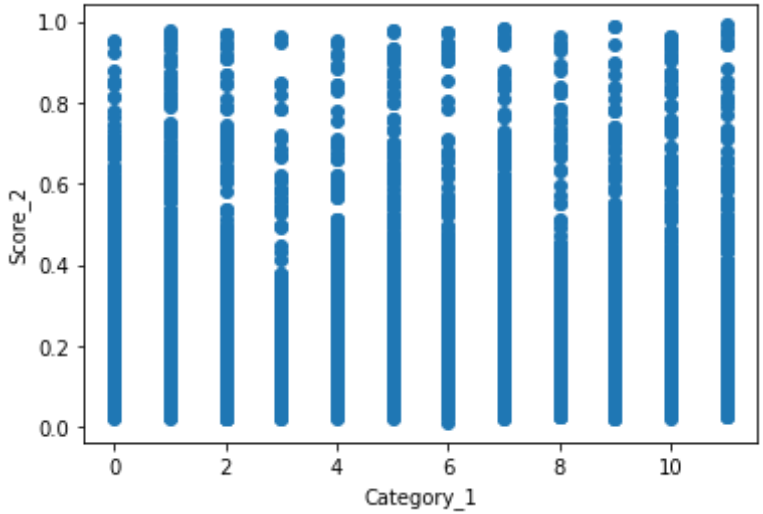


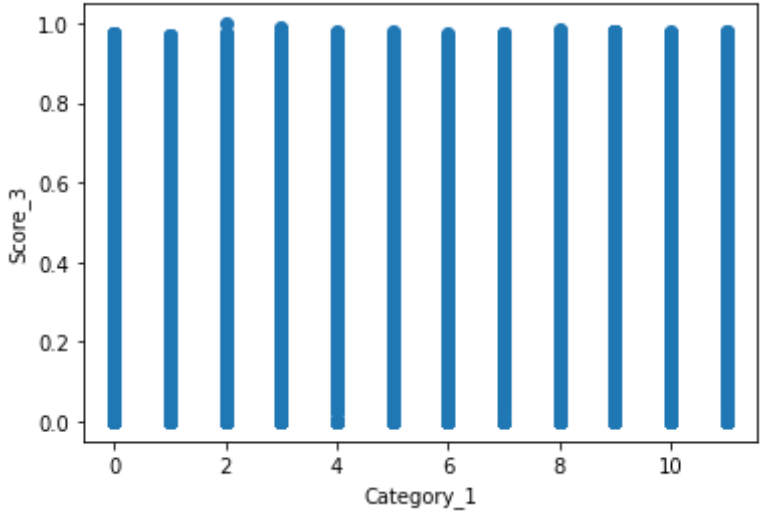


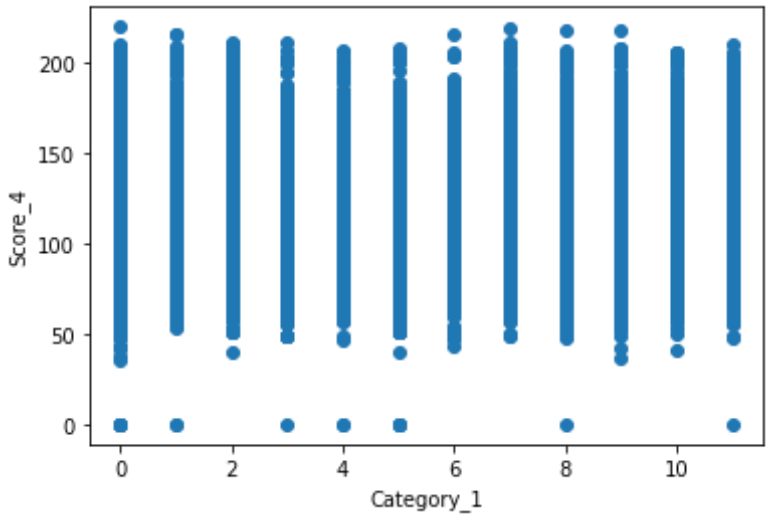


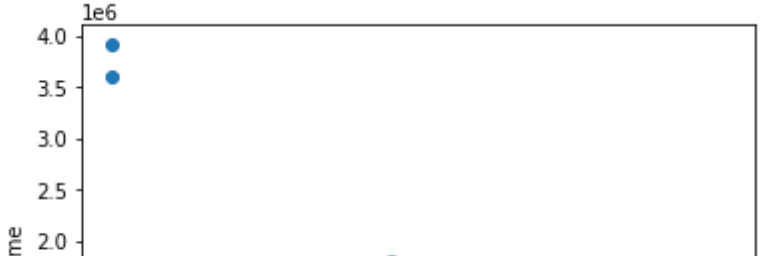


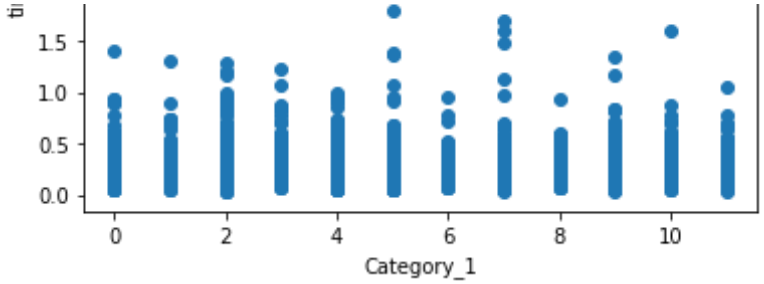


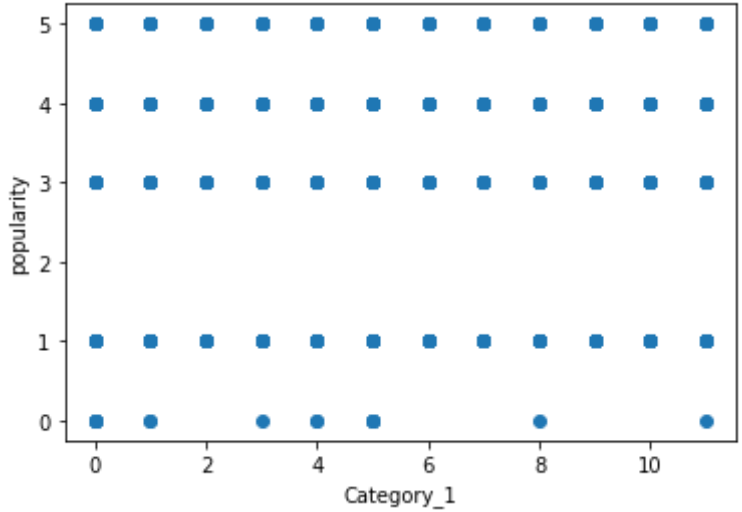


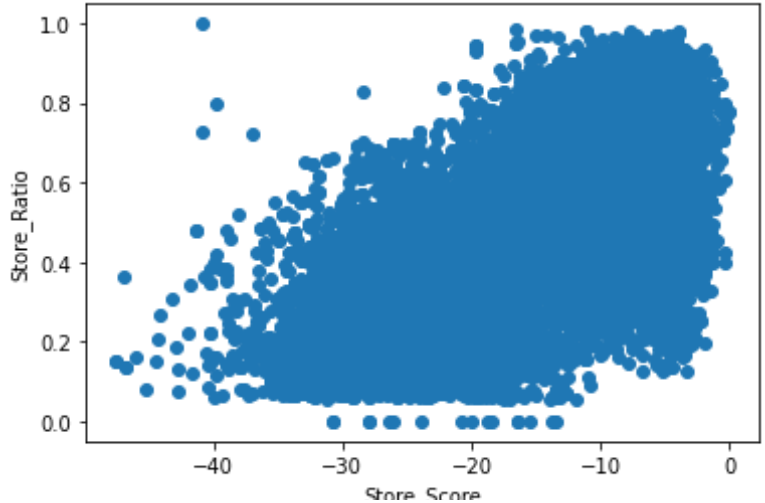


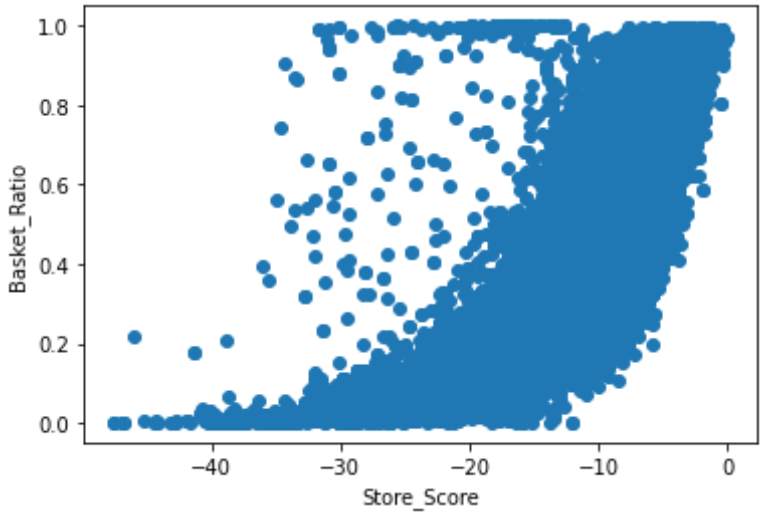


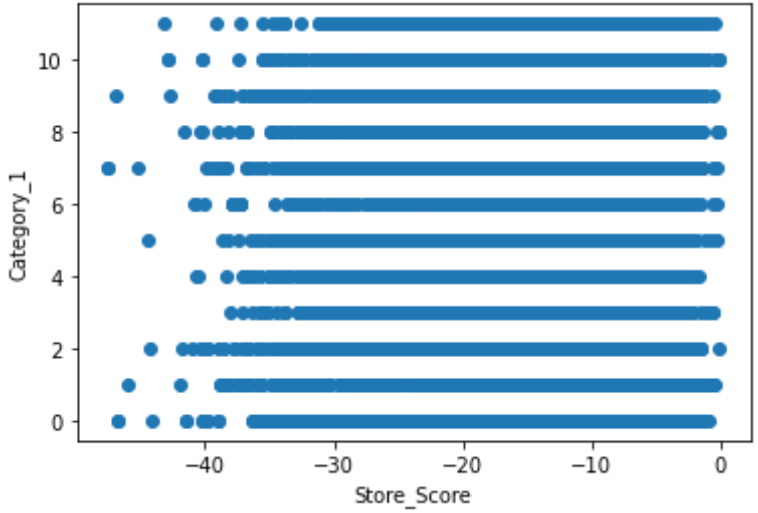


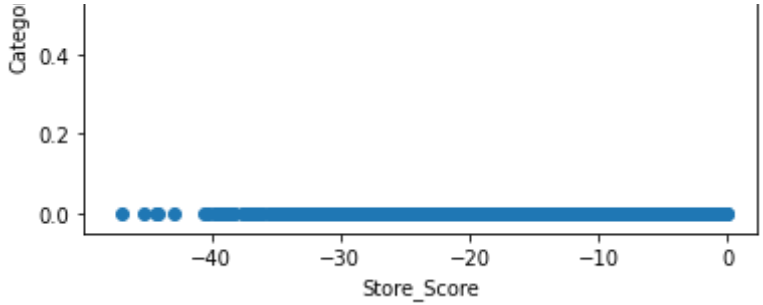


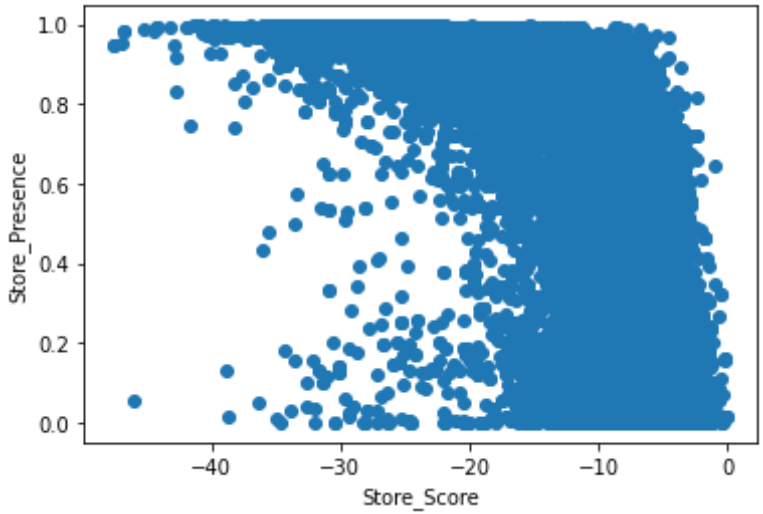


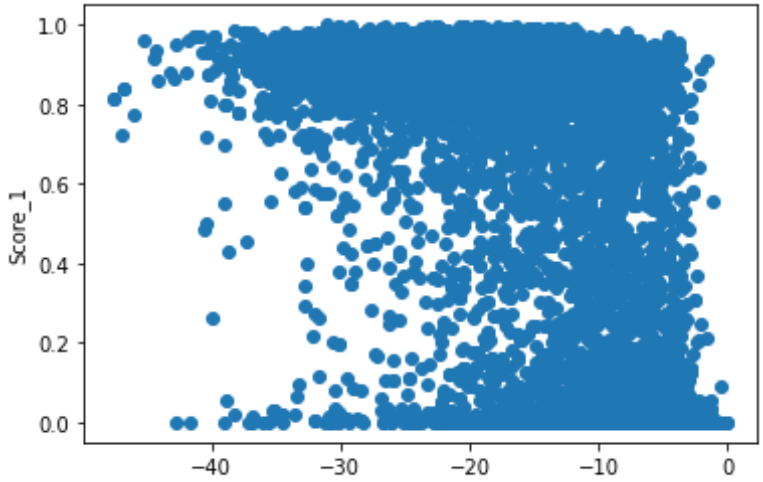




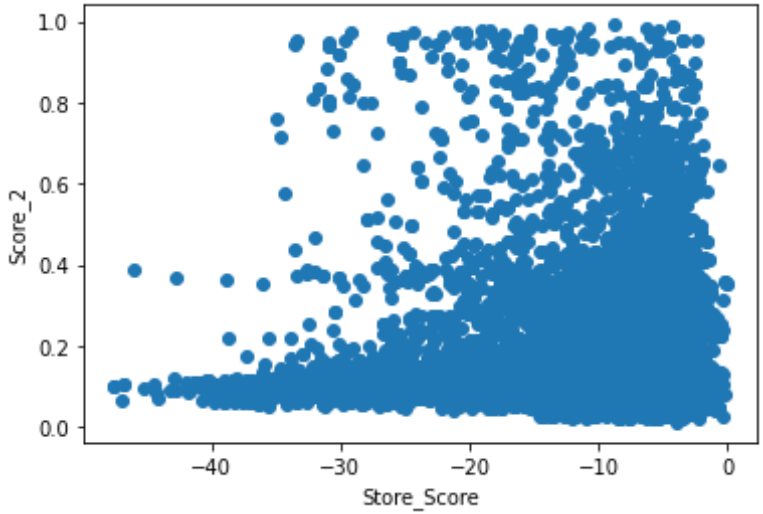


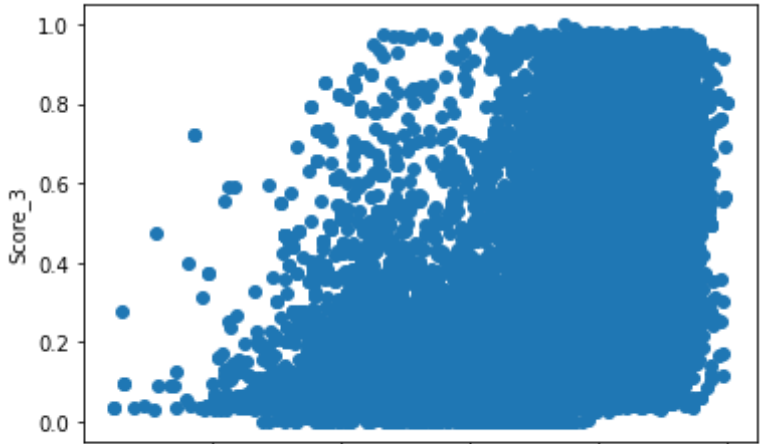


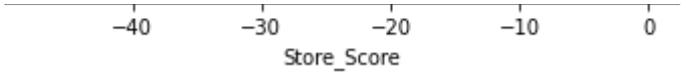


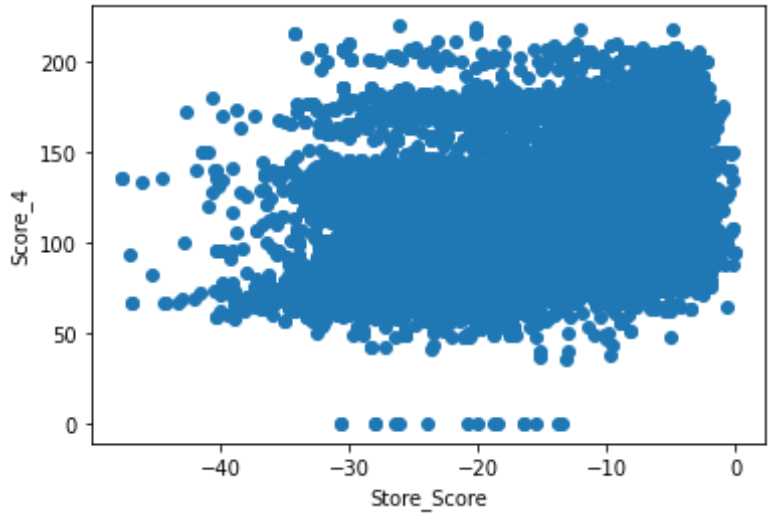


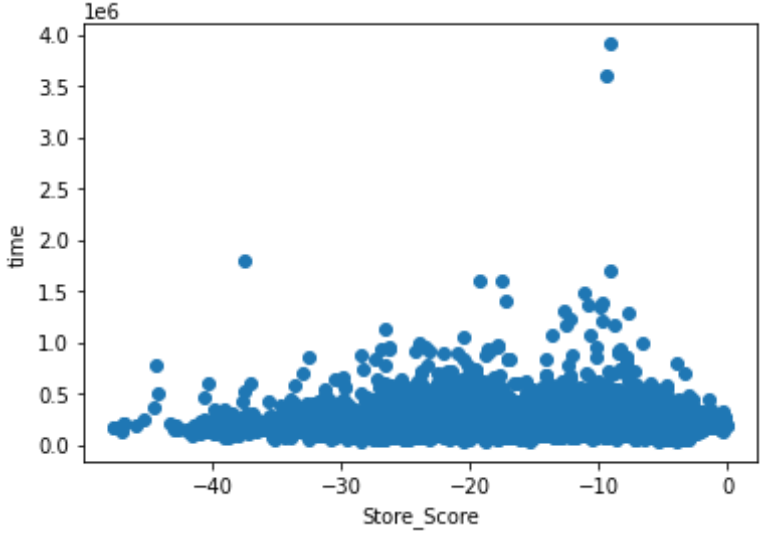
Store_Score

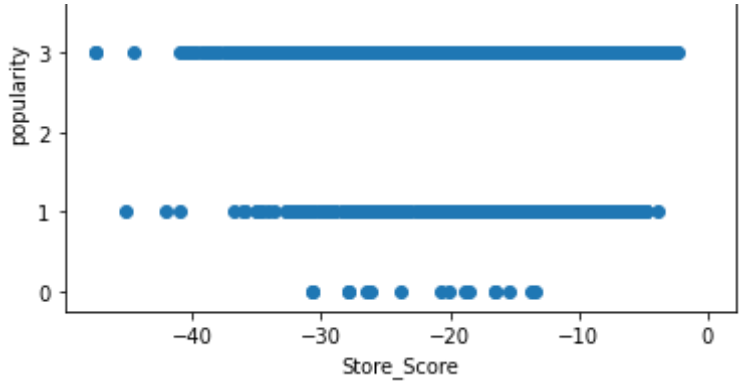


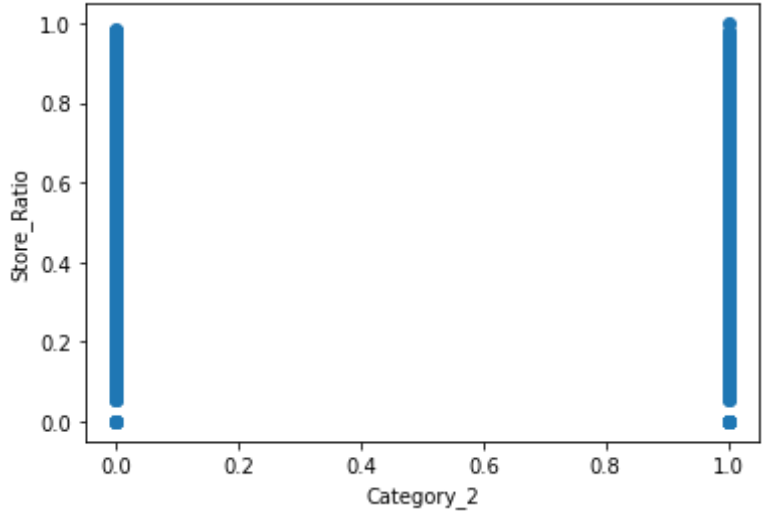


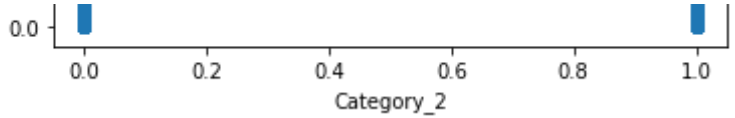


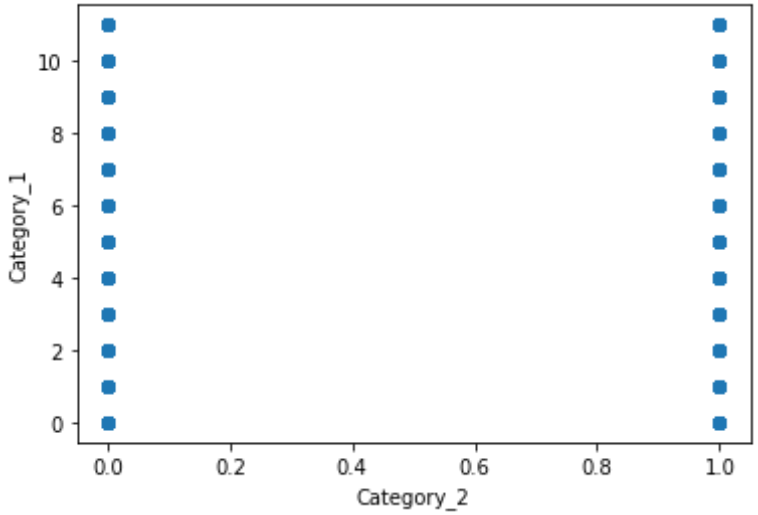


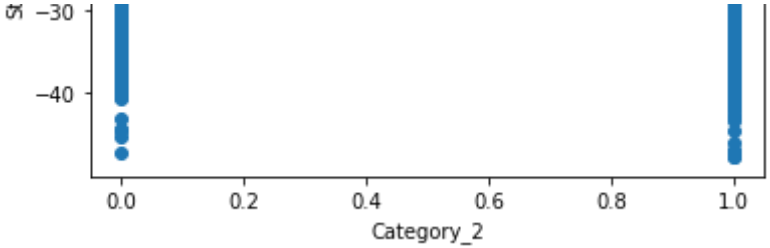


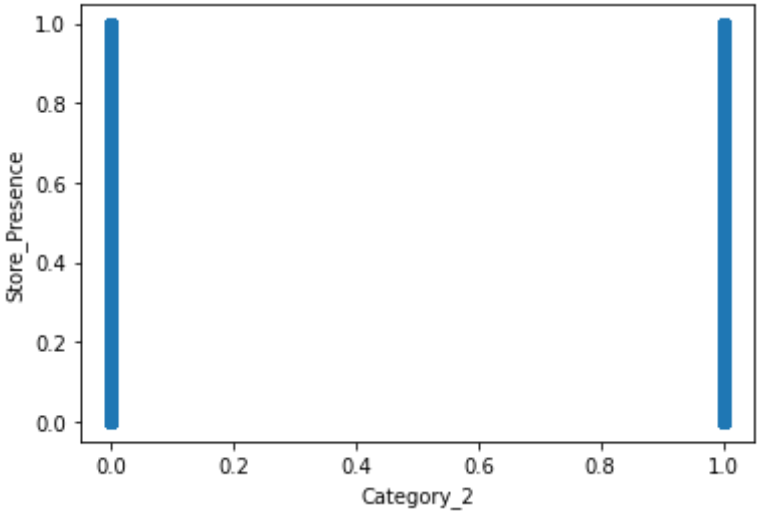


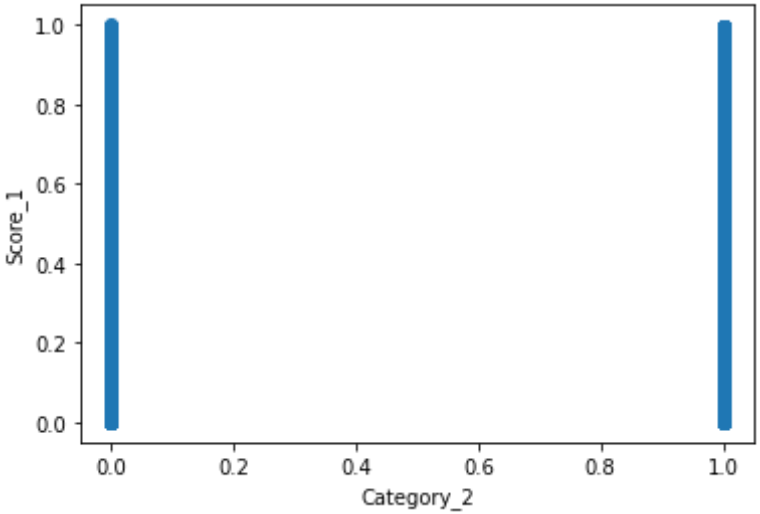


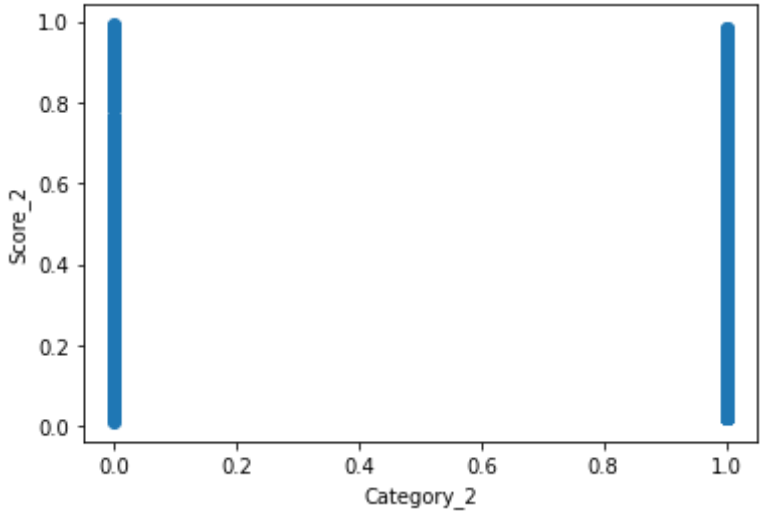


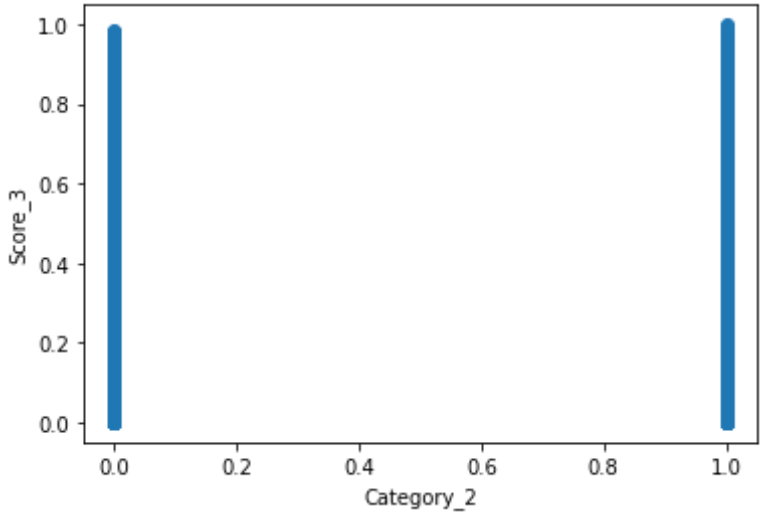


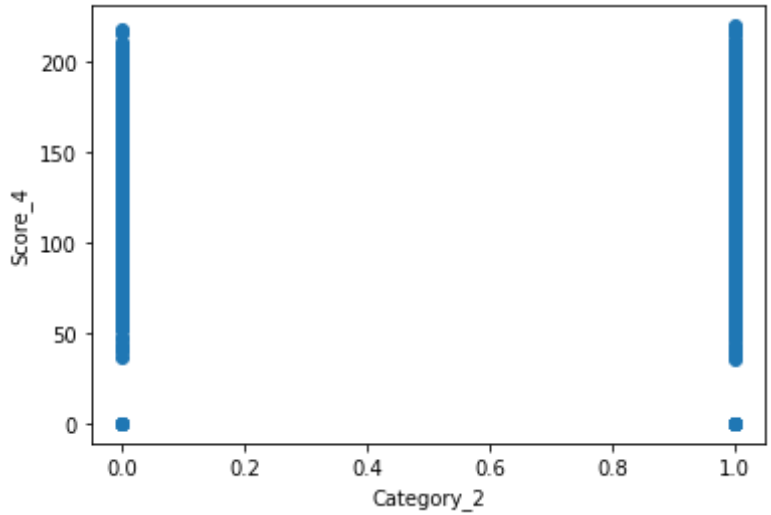


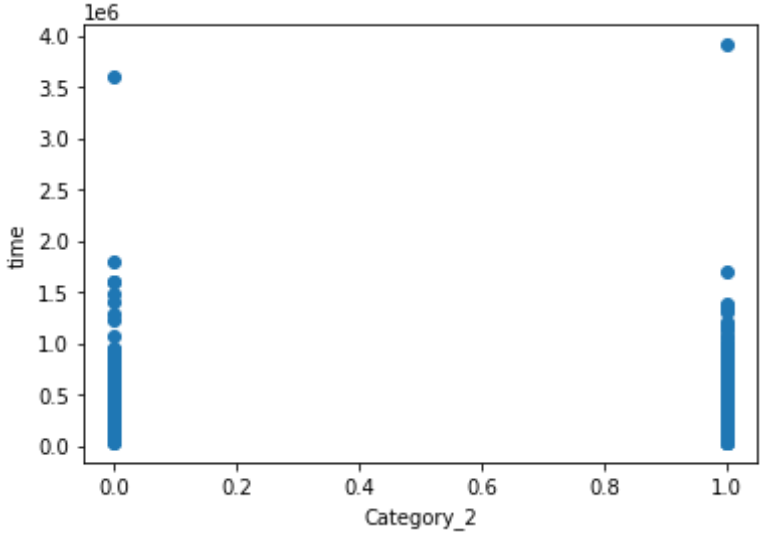


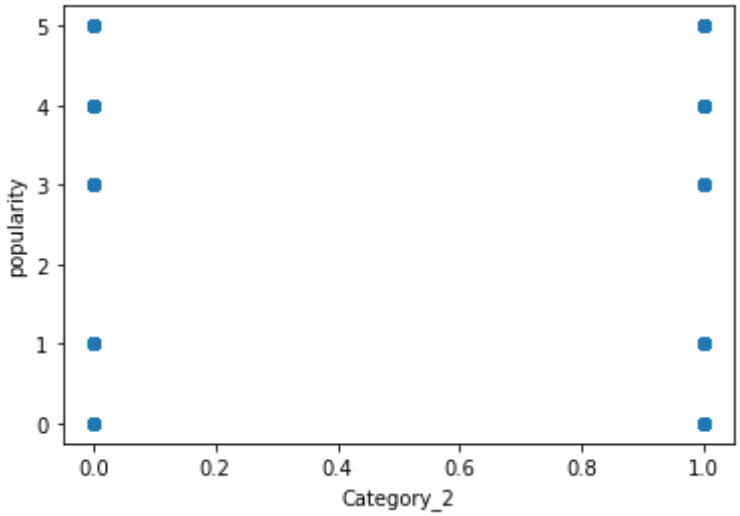


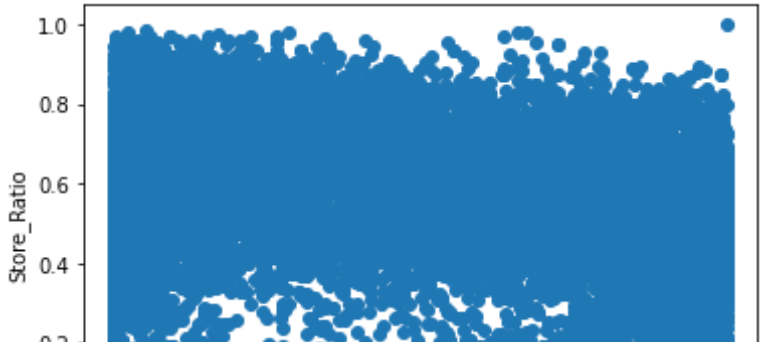


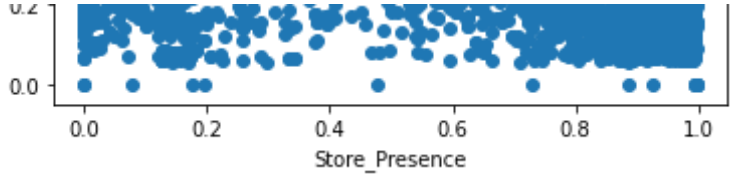


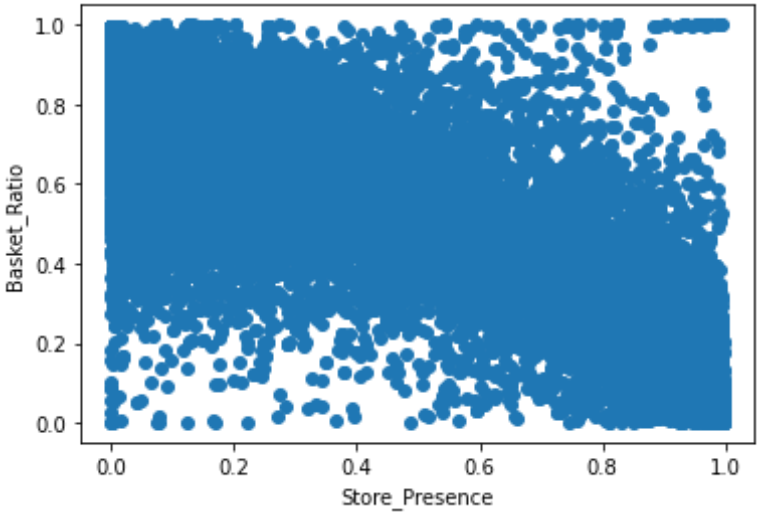


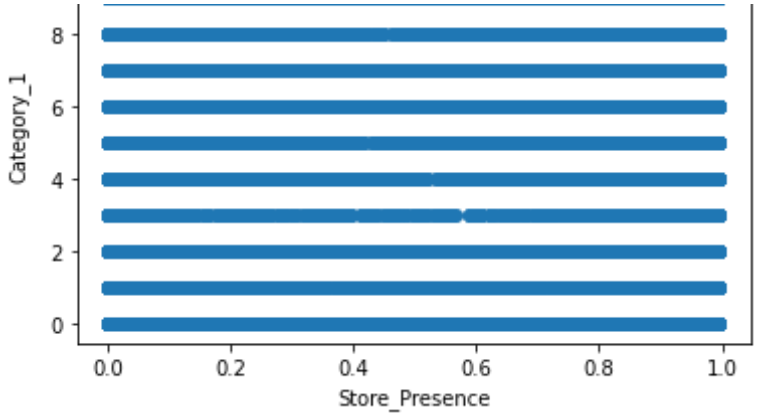


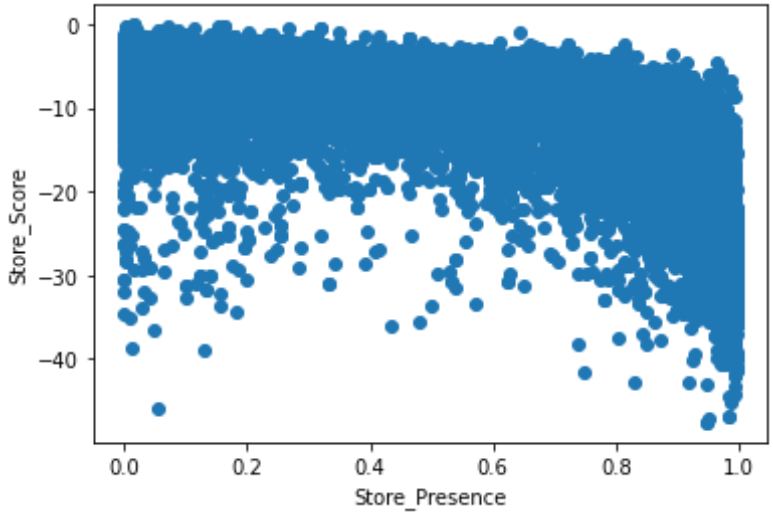


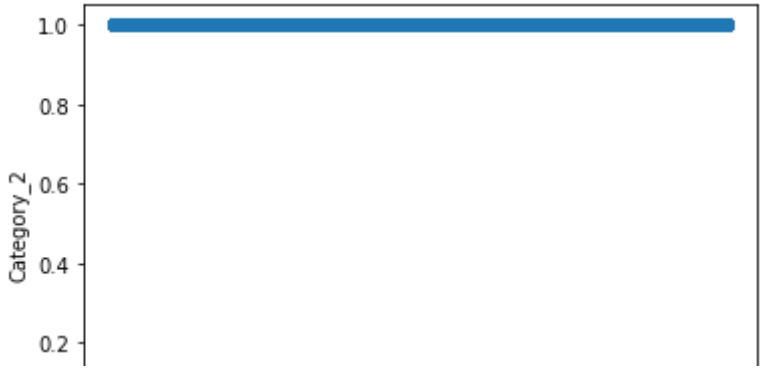


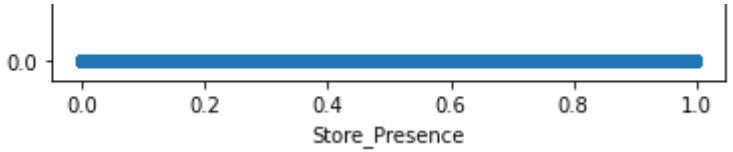


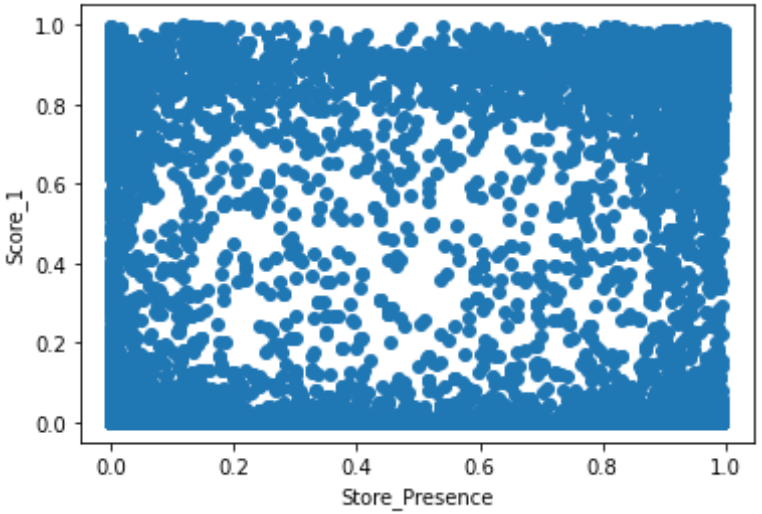


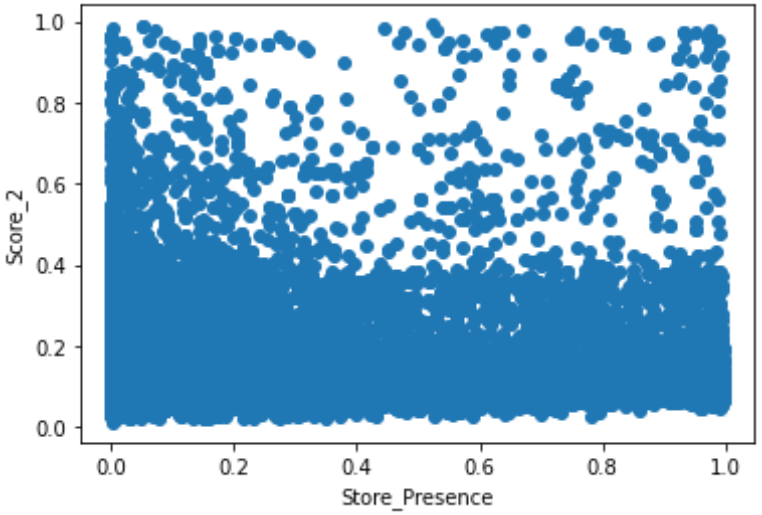


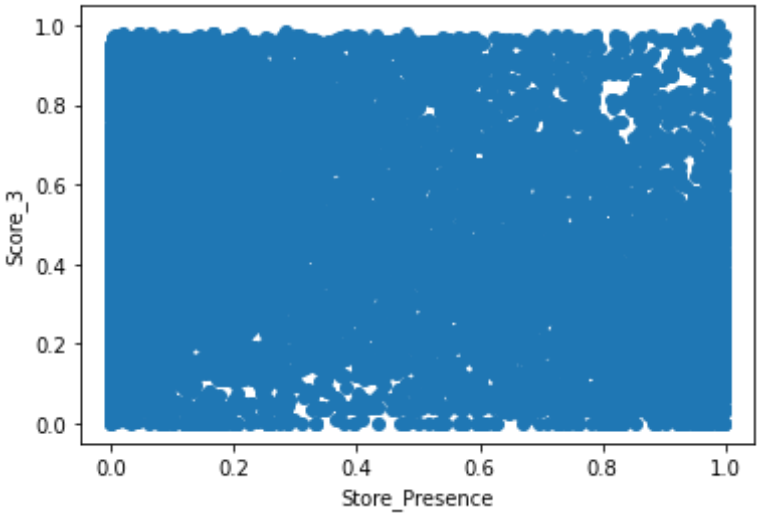


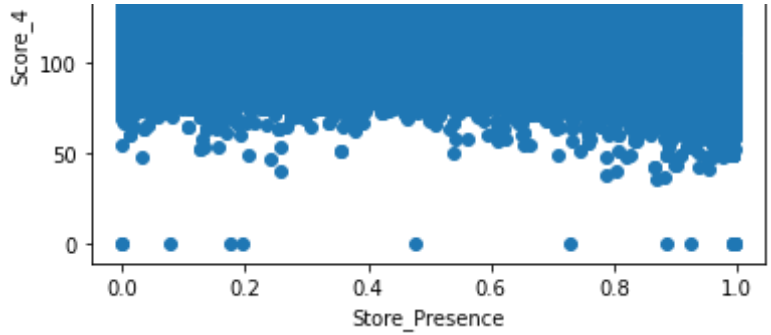


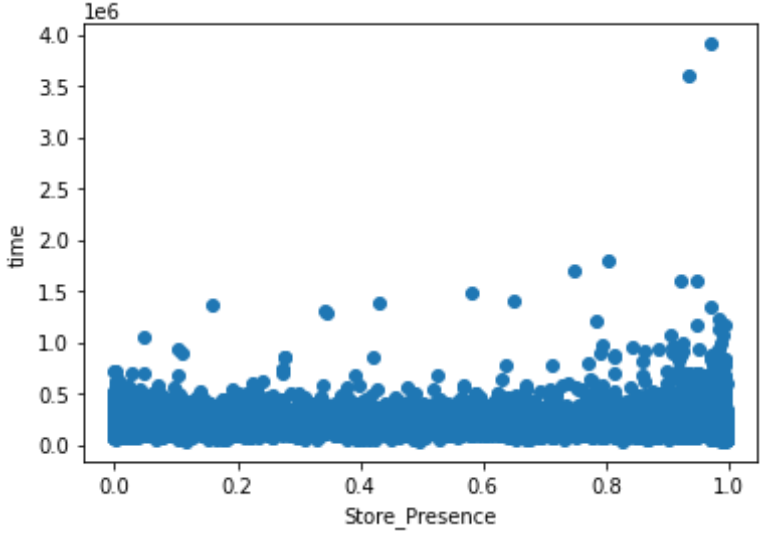


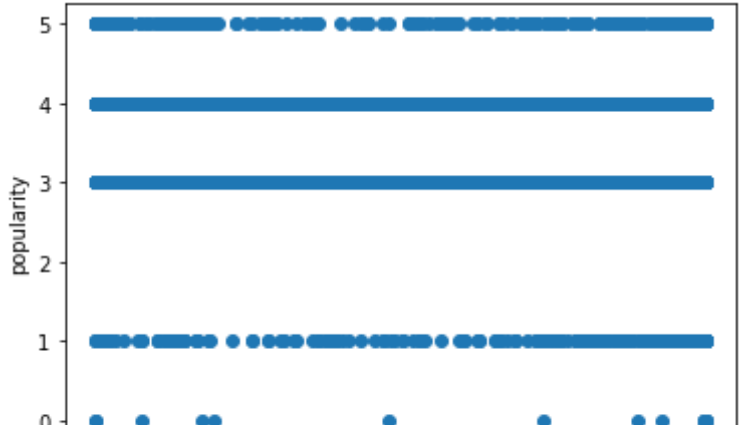


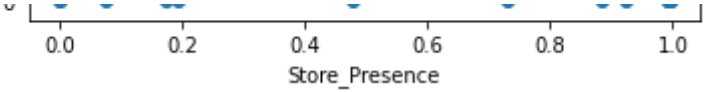


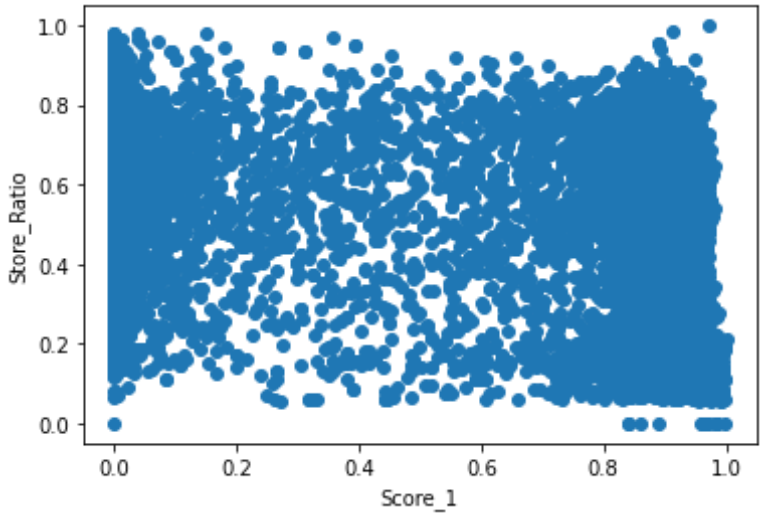


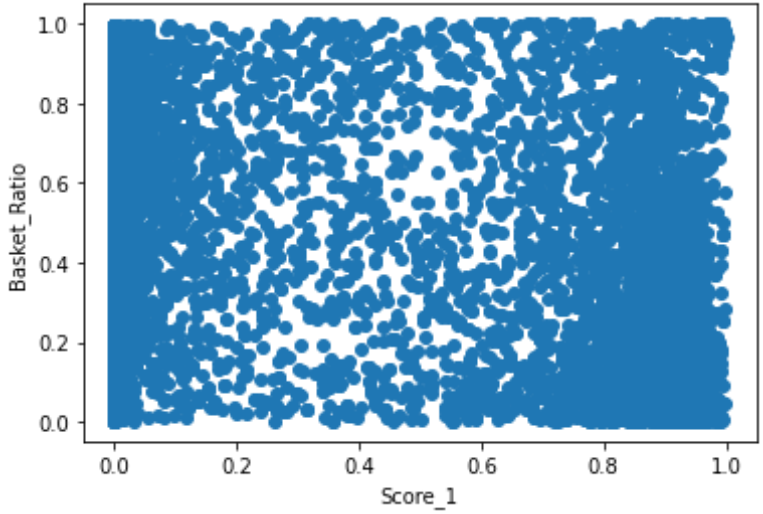


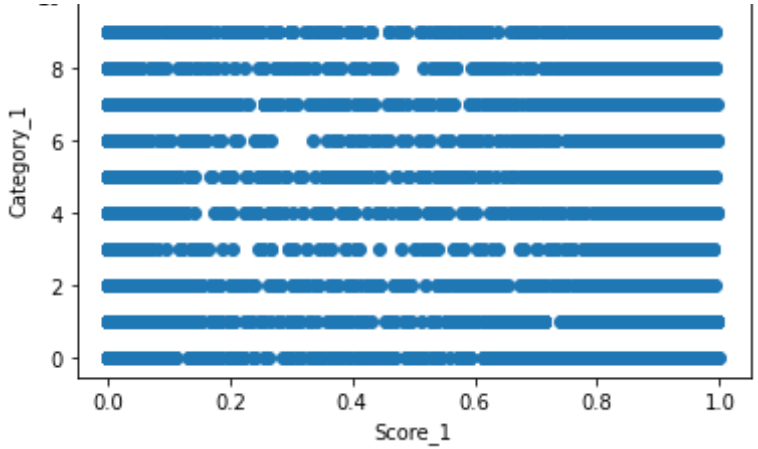


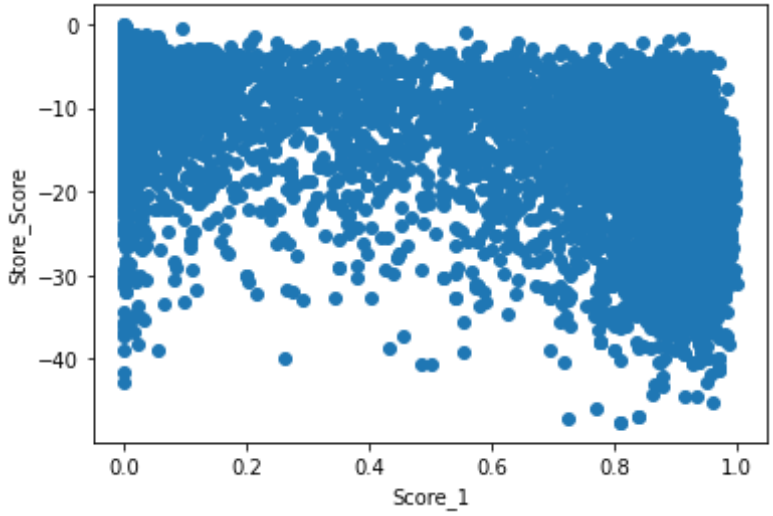


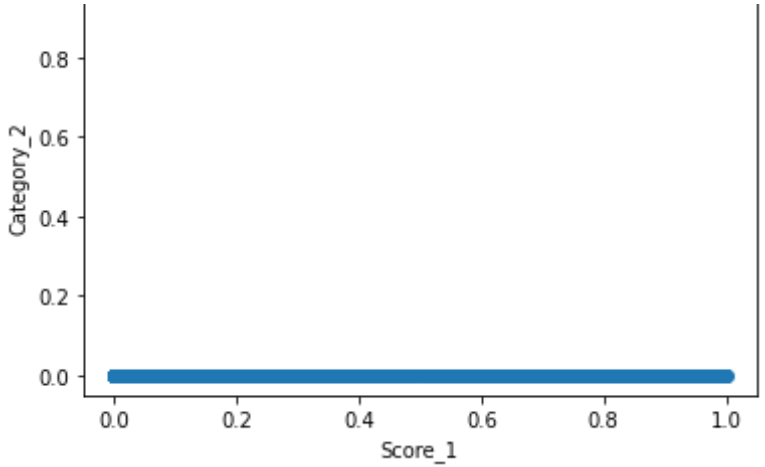


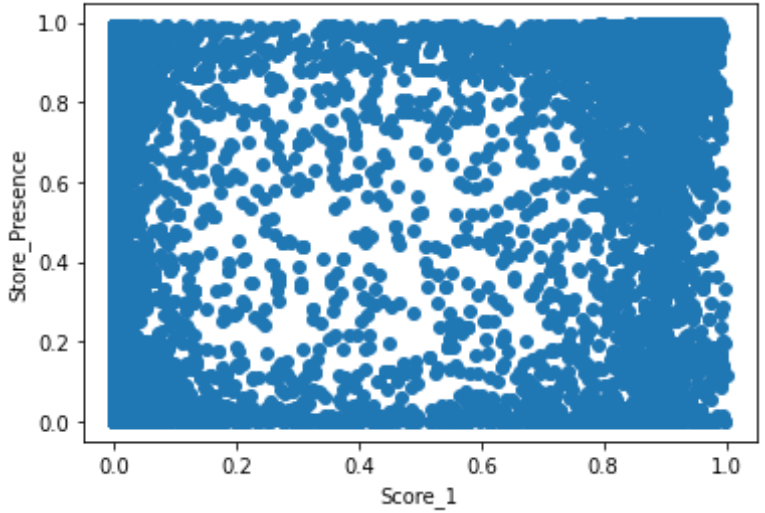


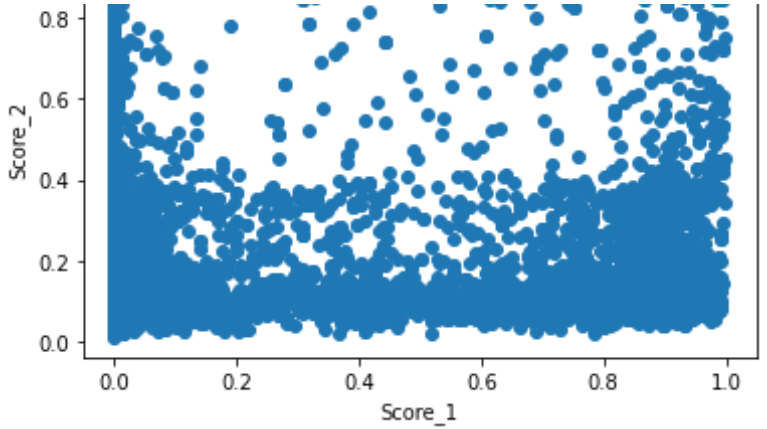


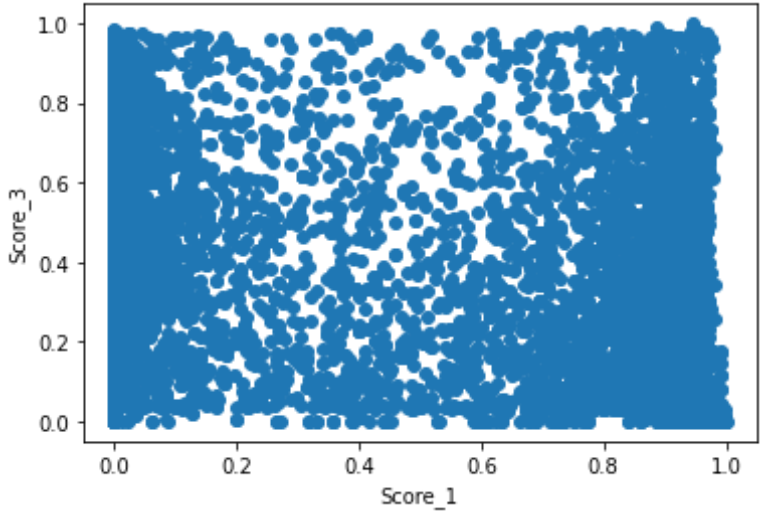


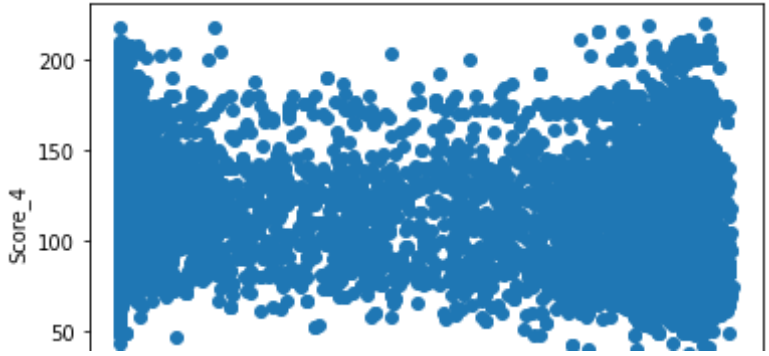


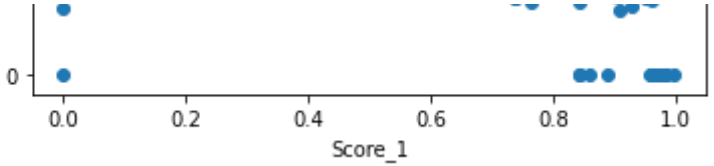


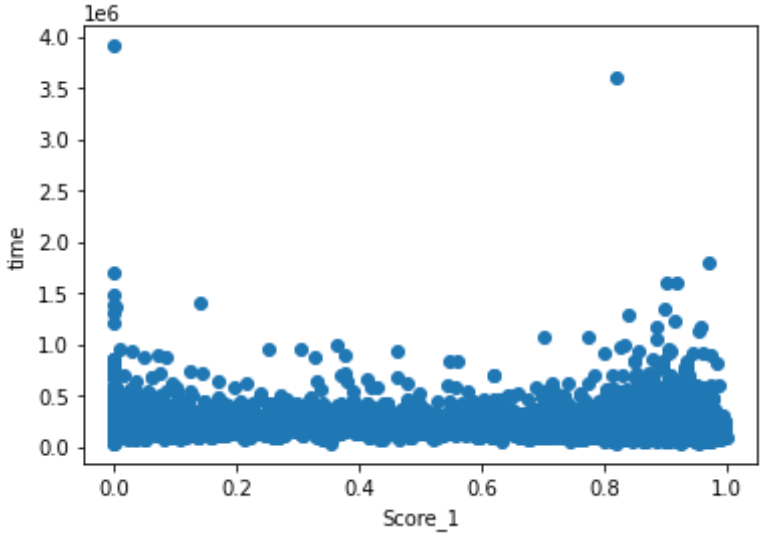


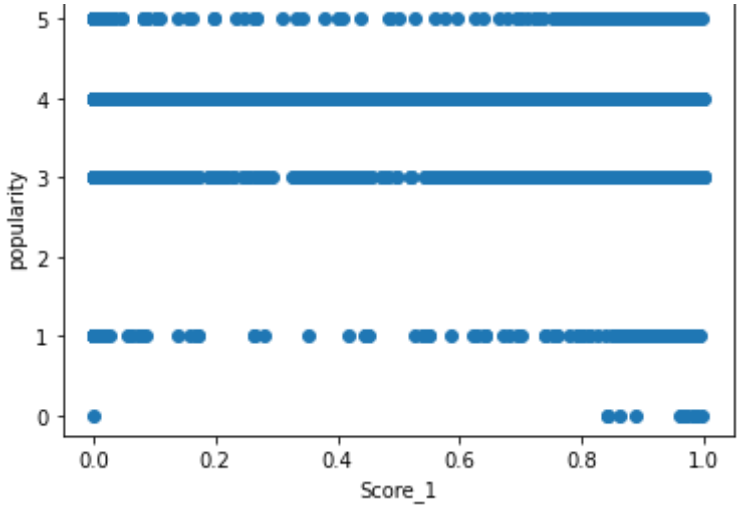


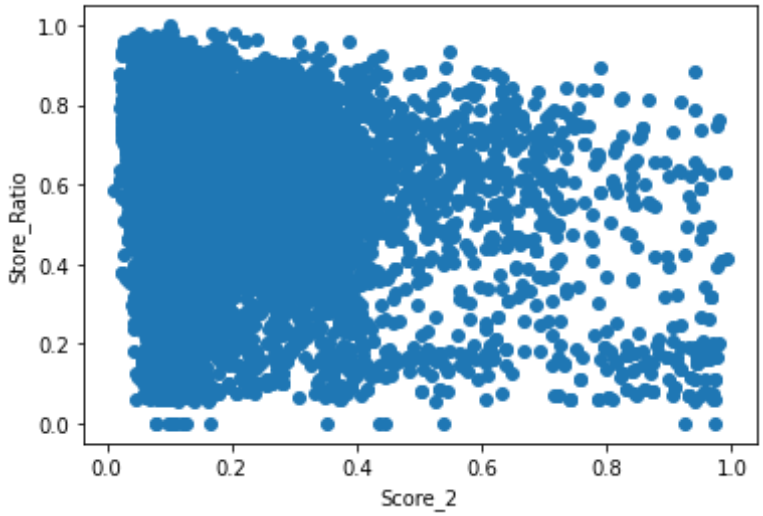


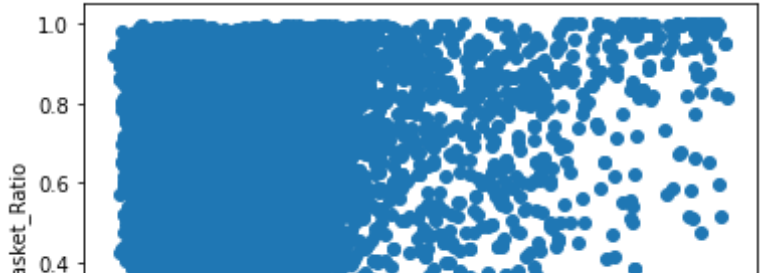


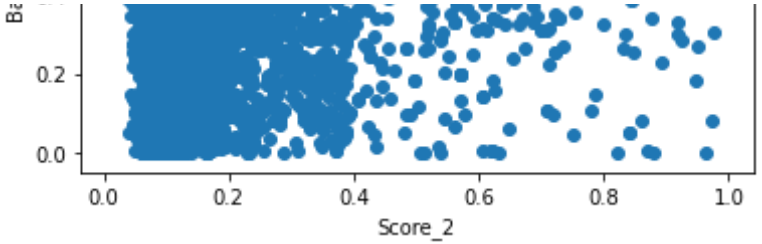


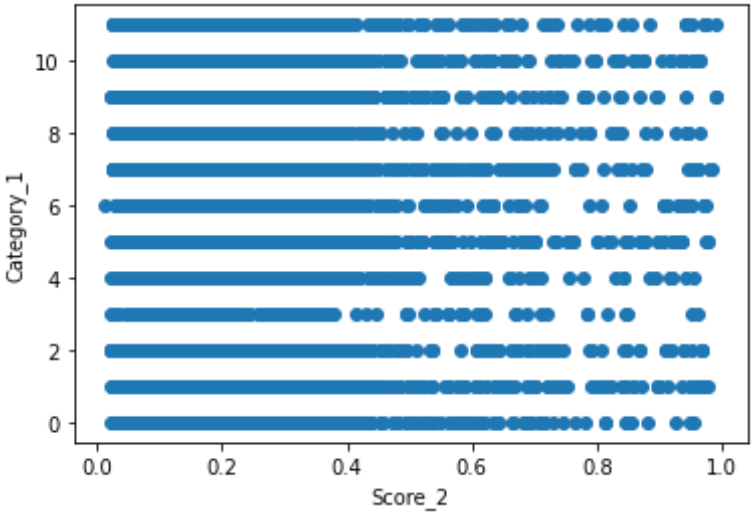


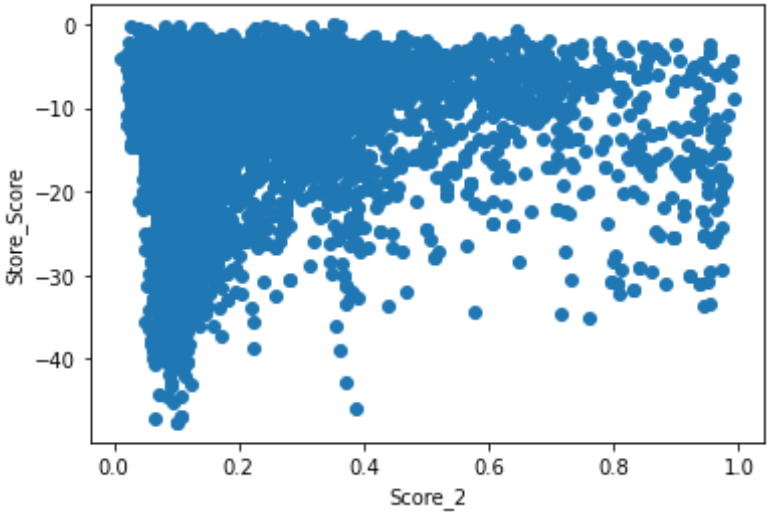


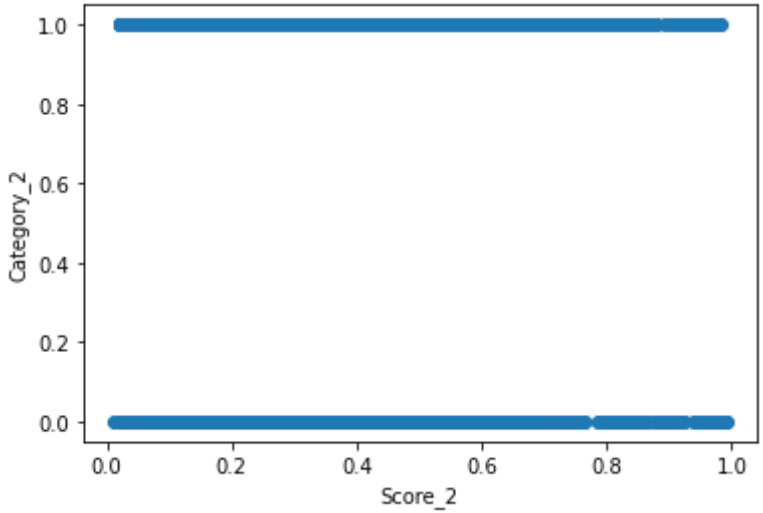


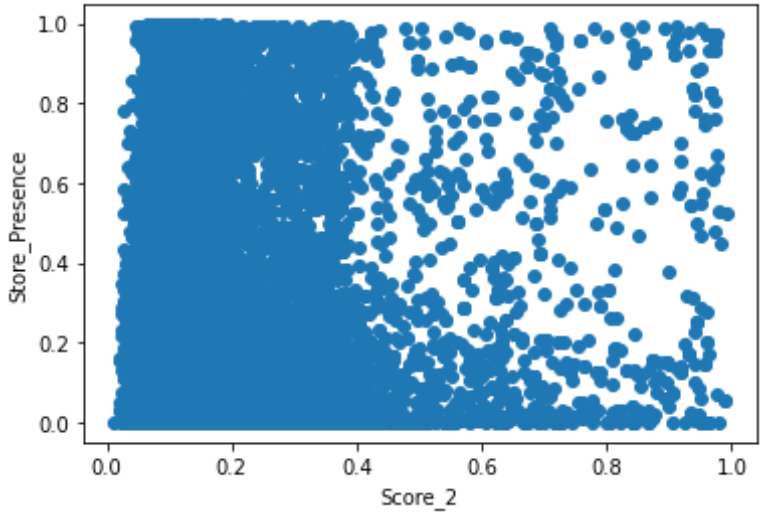


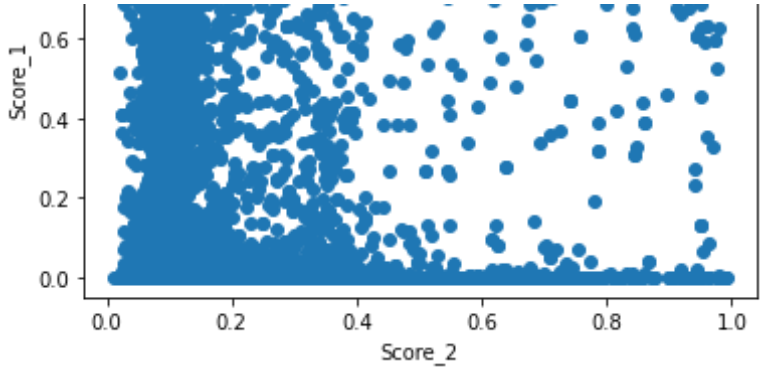


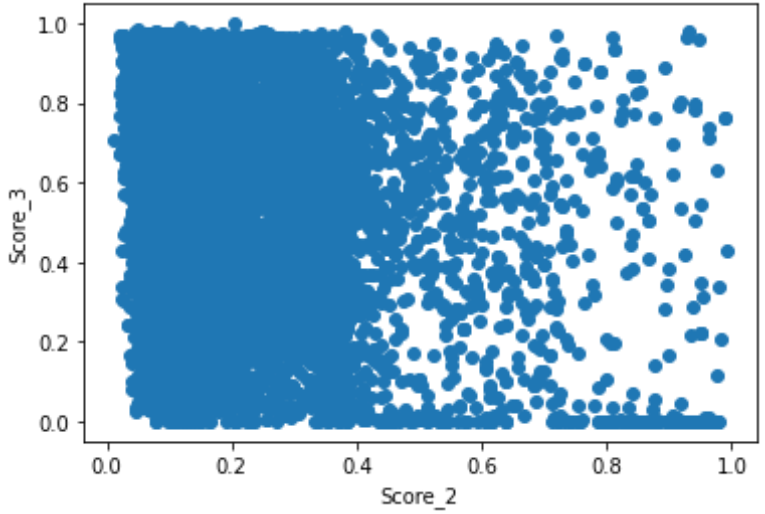


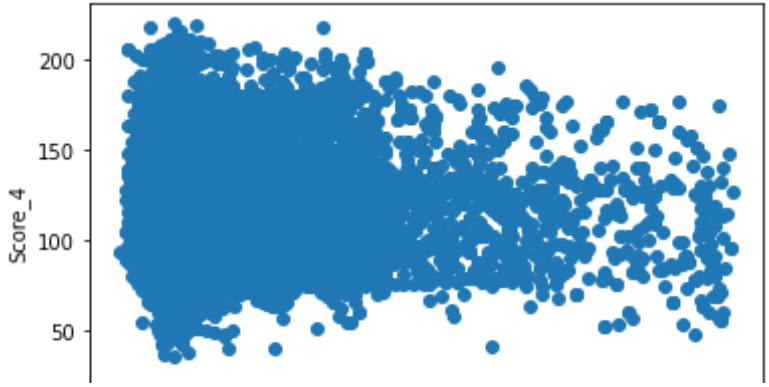


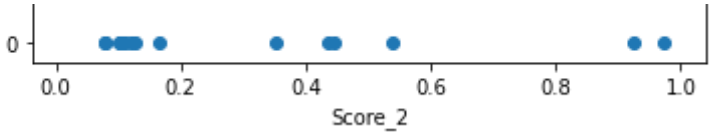


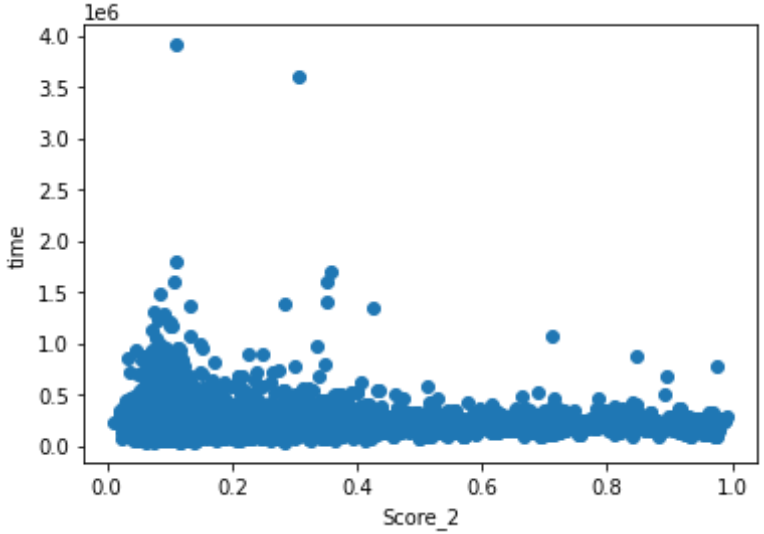


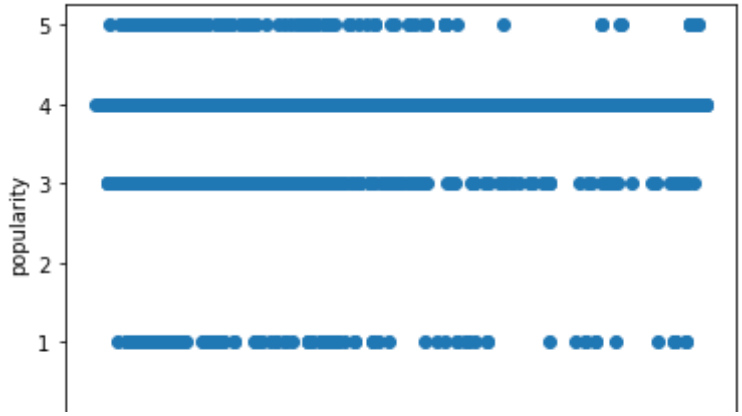


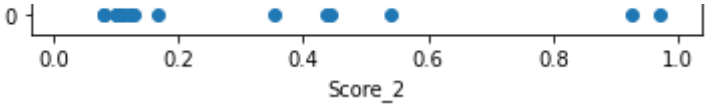


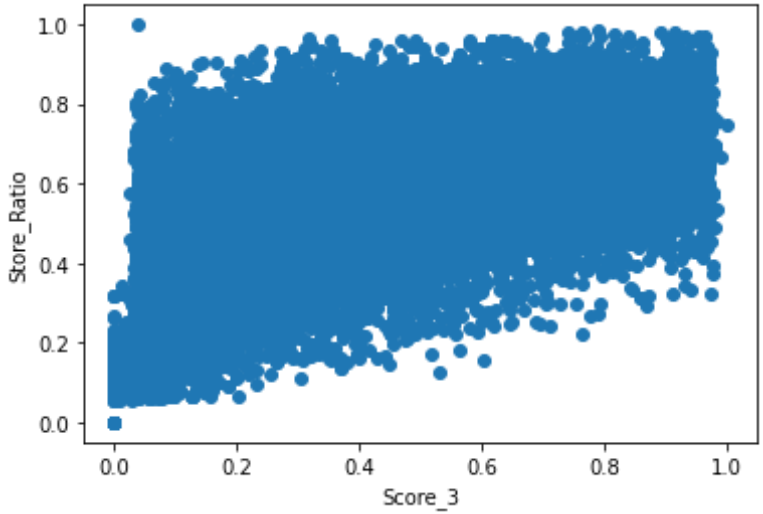


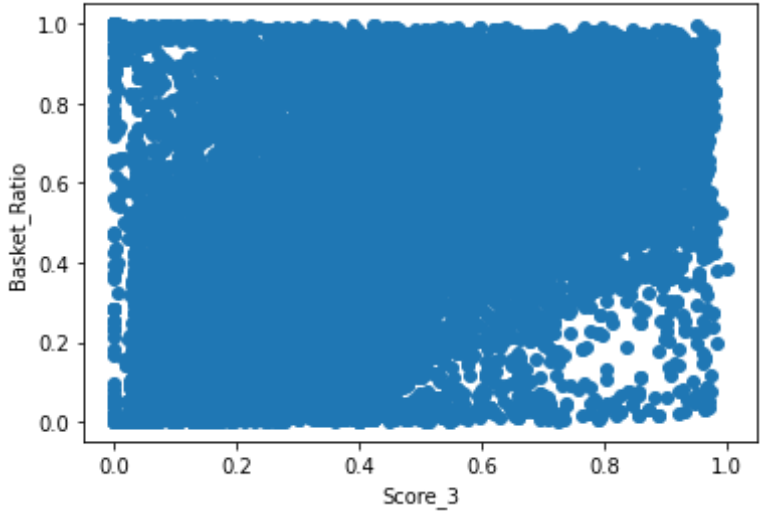


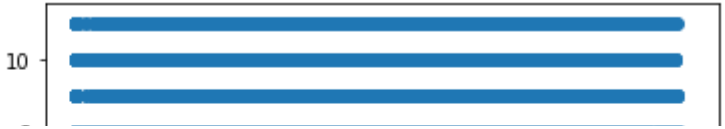


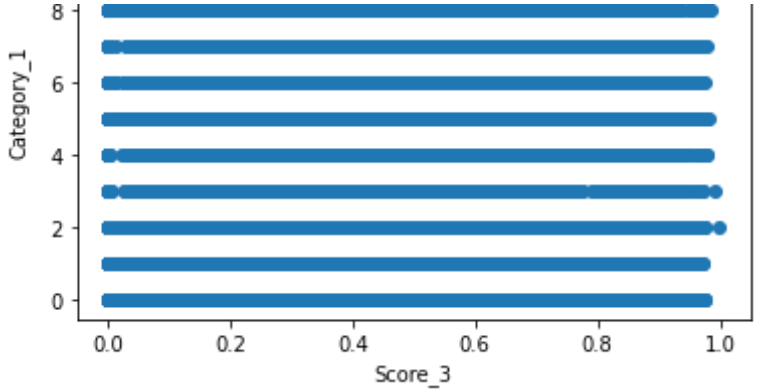


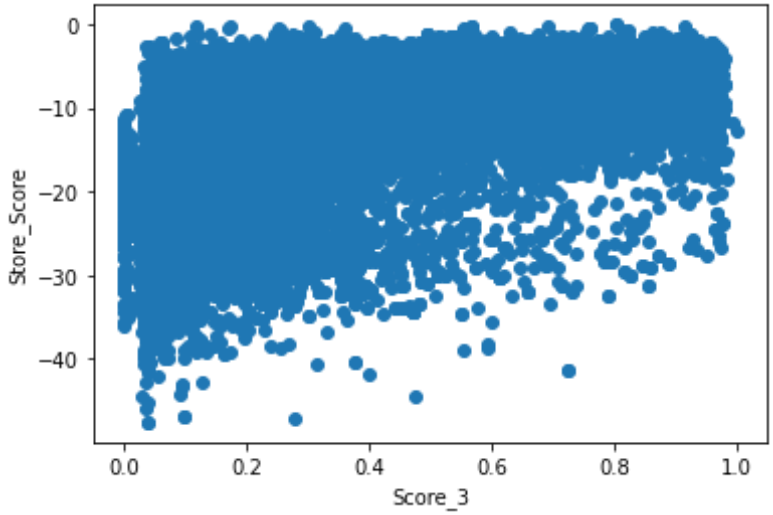


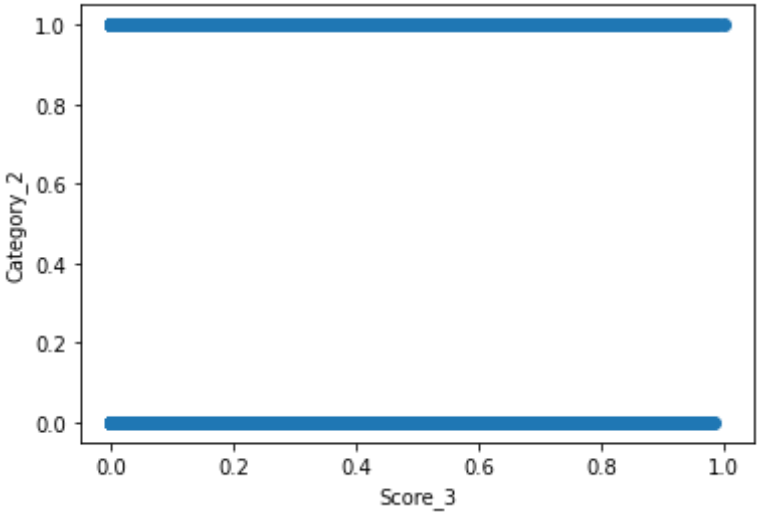


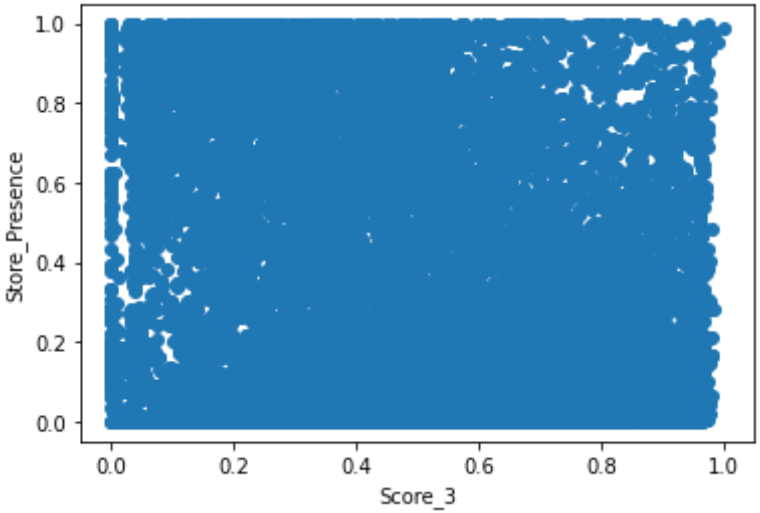


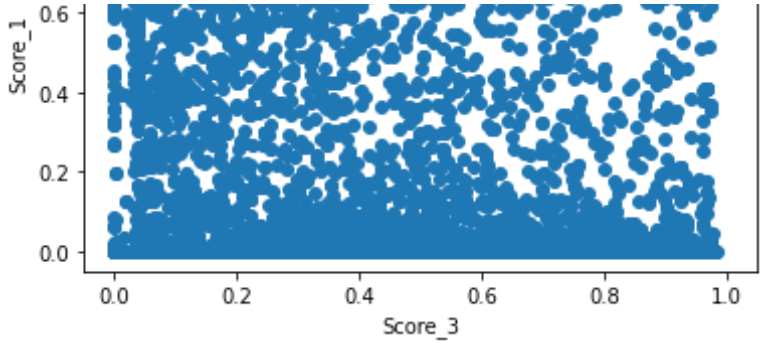


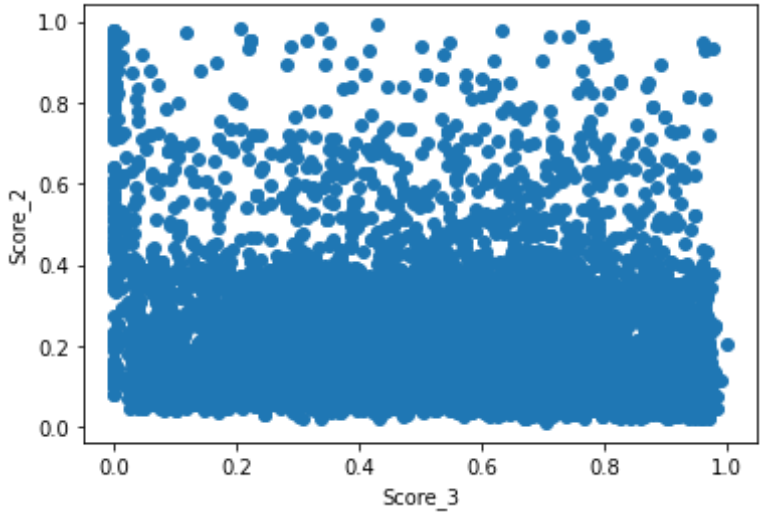


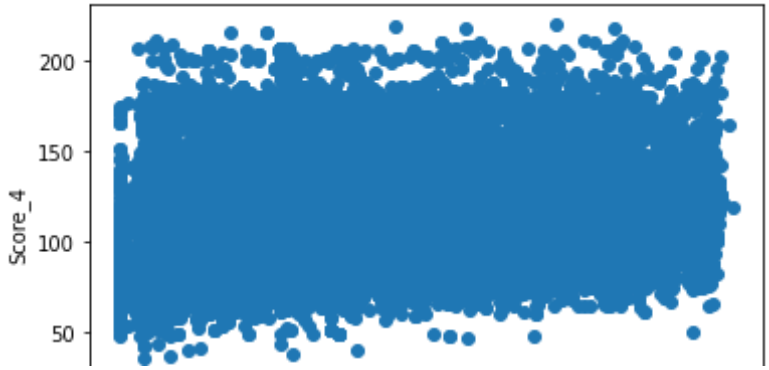


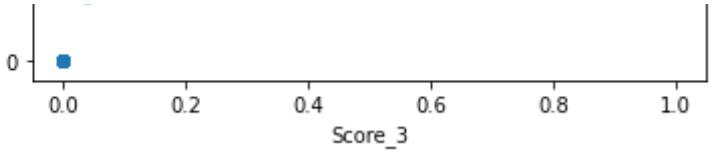


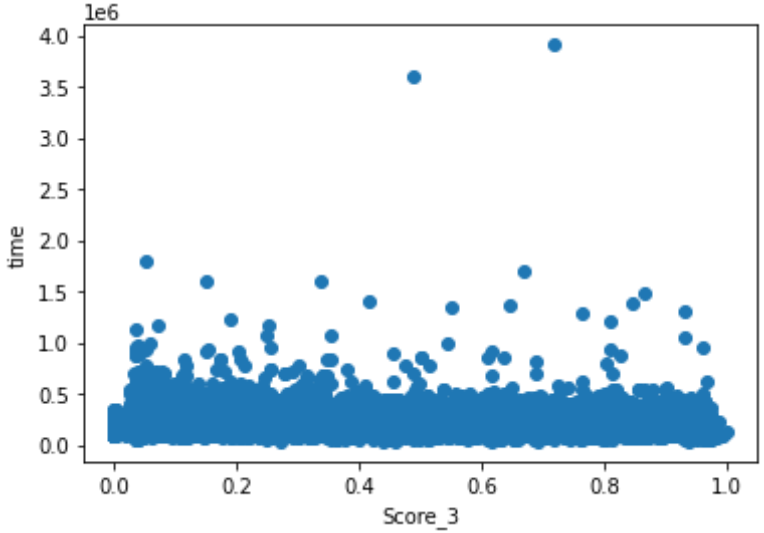


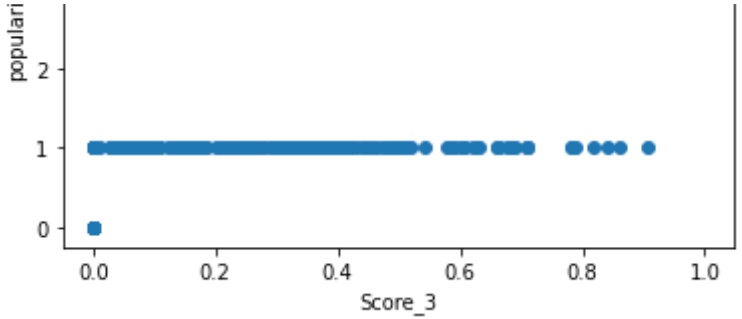


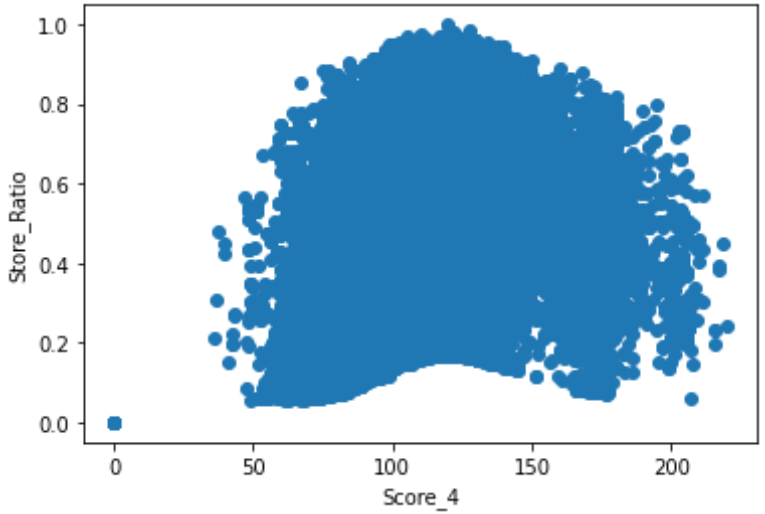


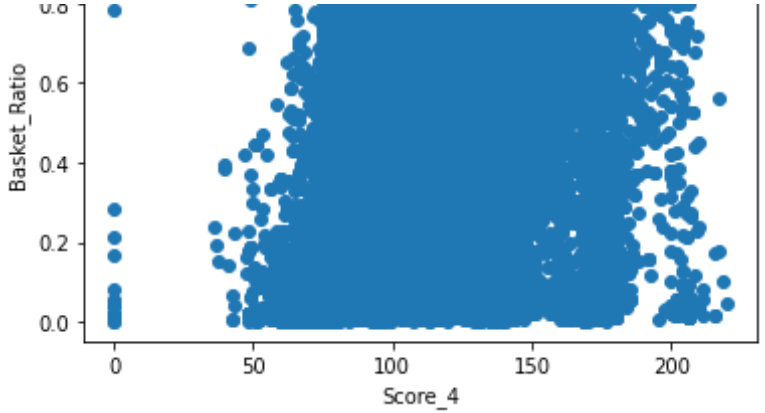


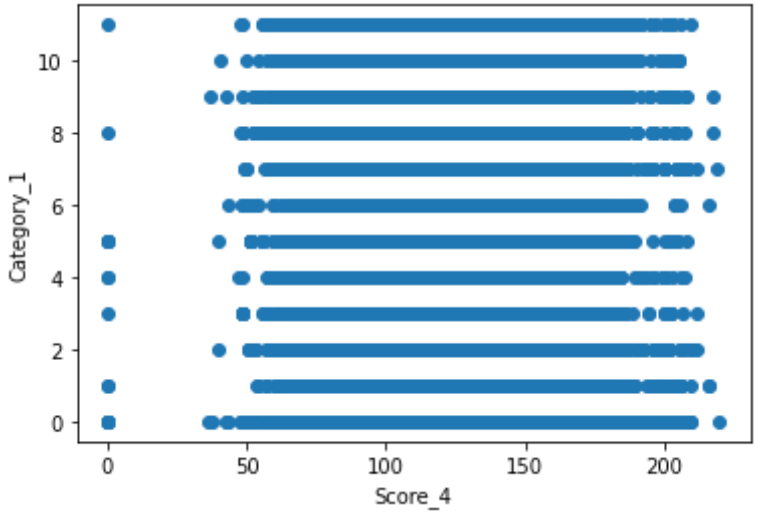


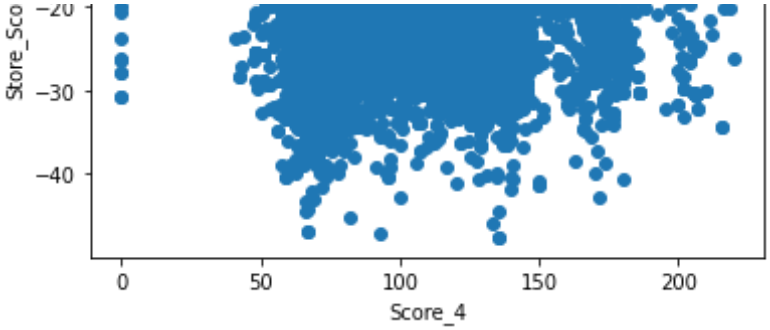


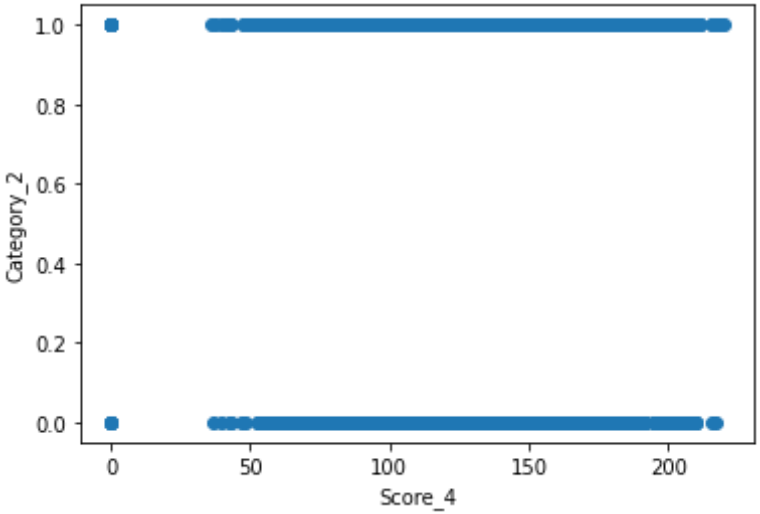


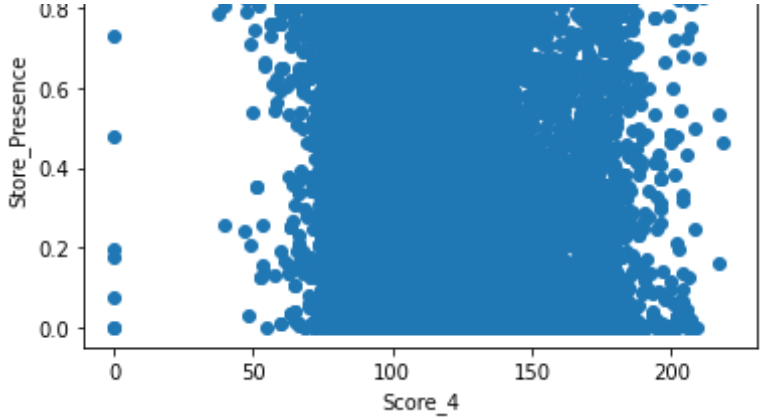


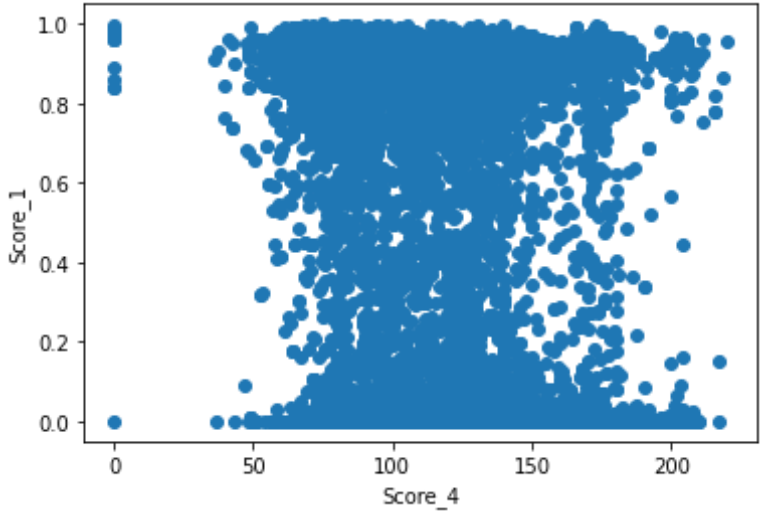


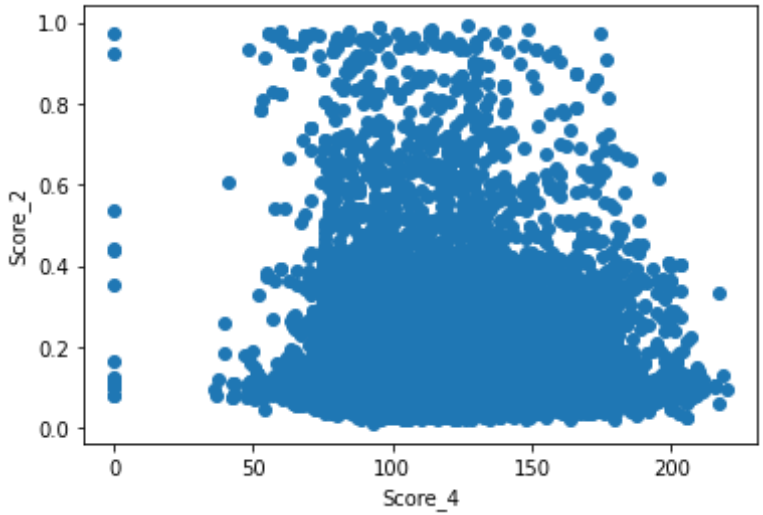


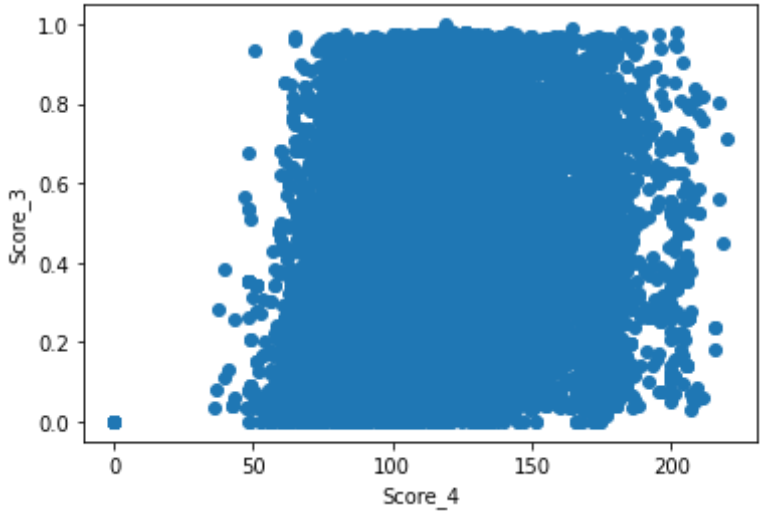


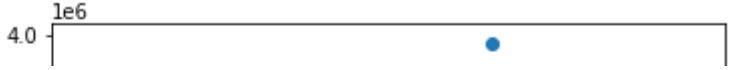


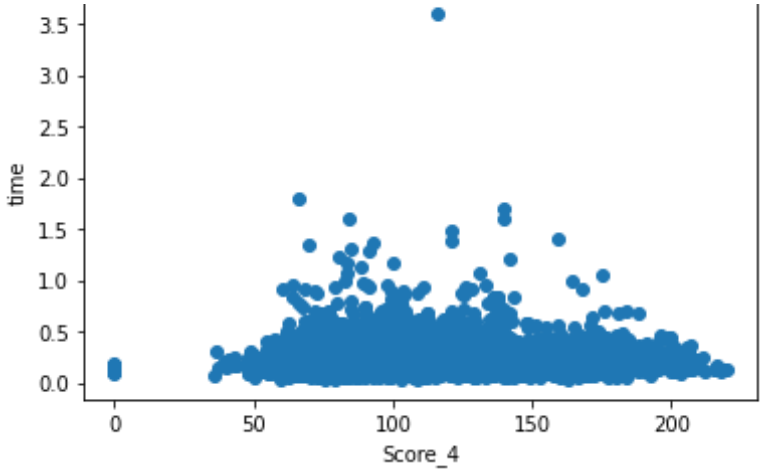


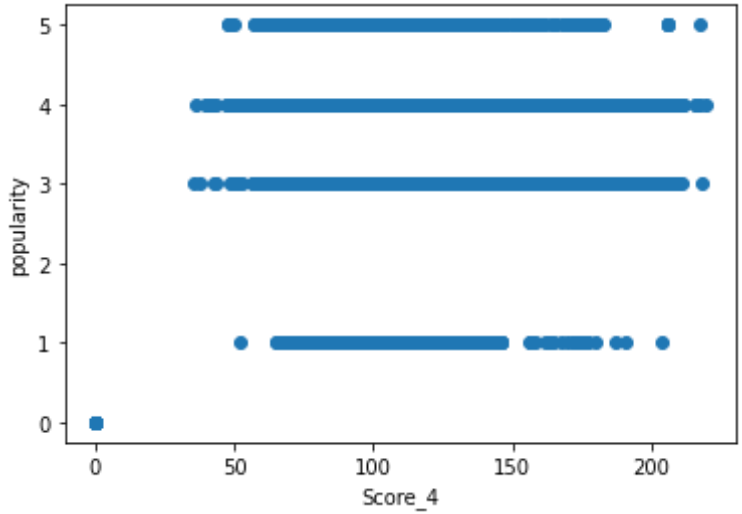


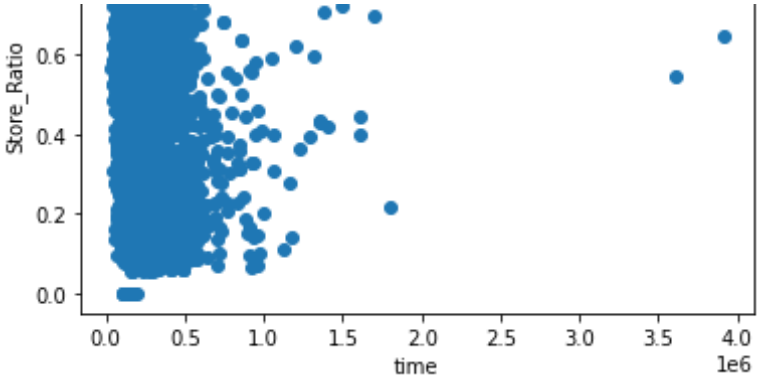


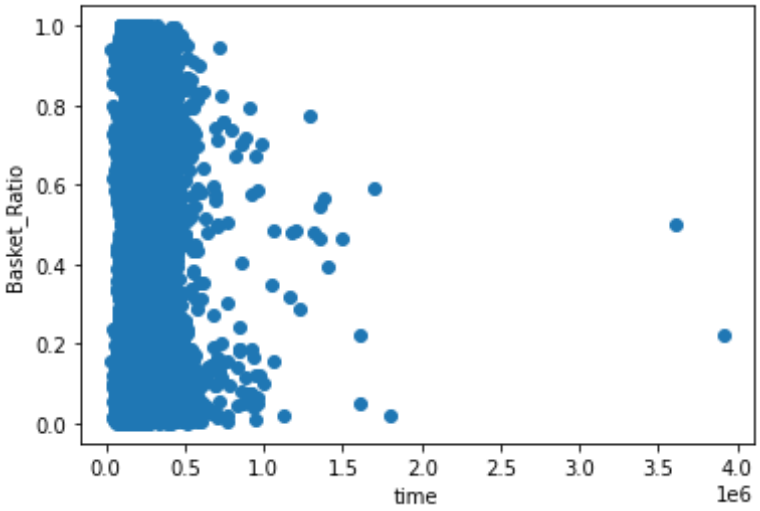


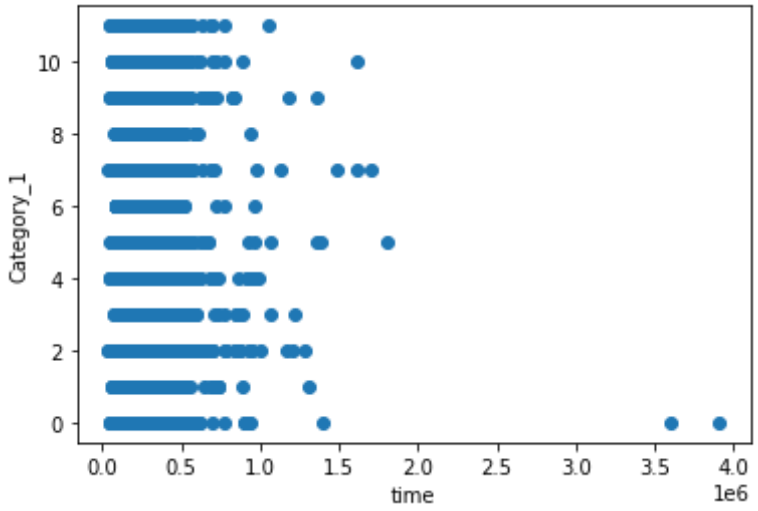


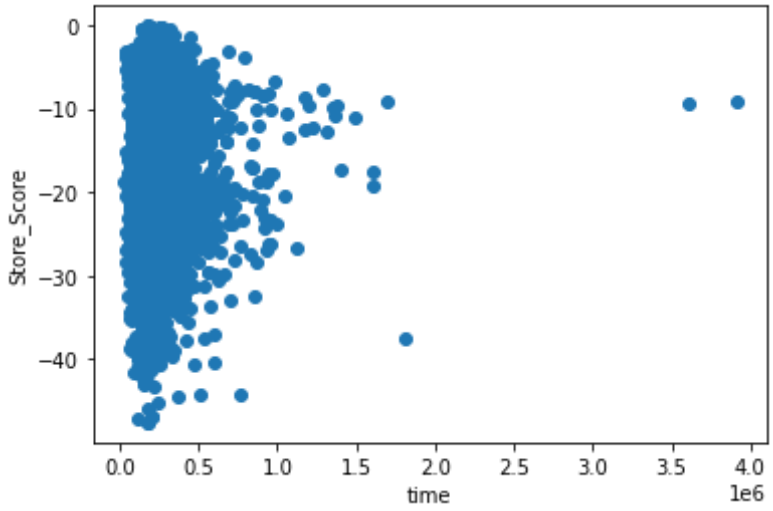


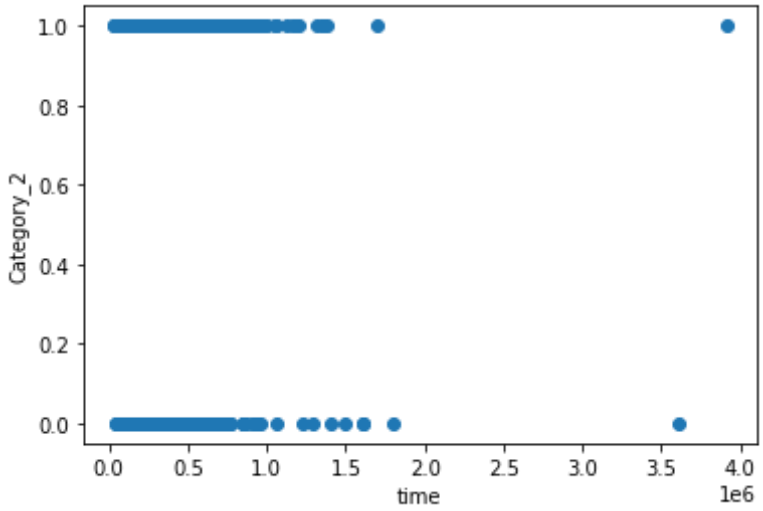


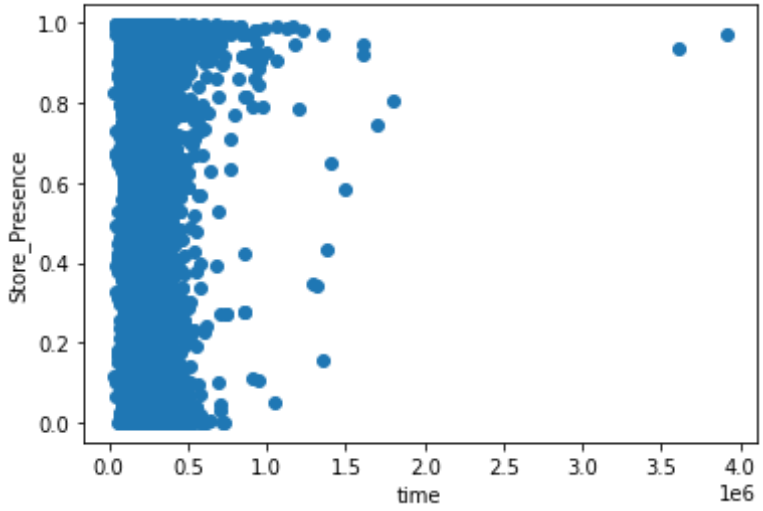


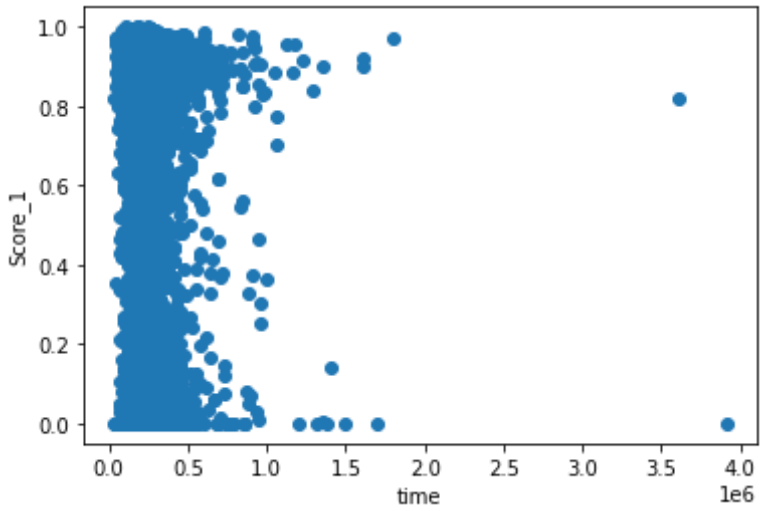


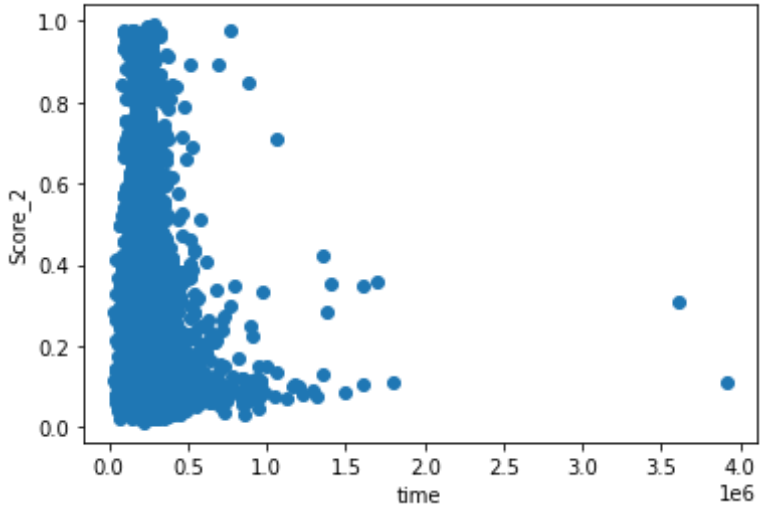


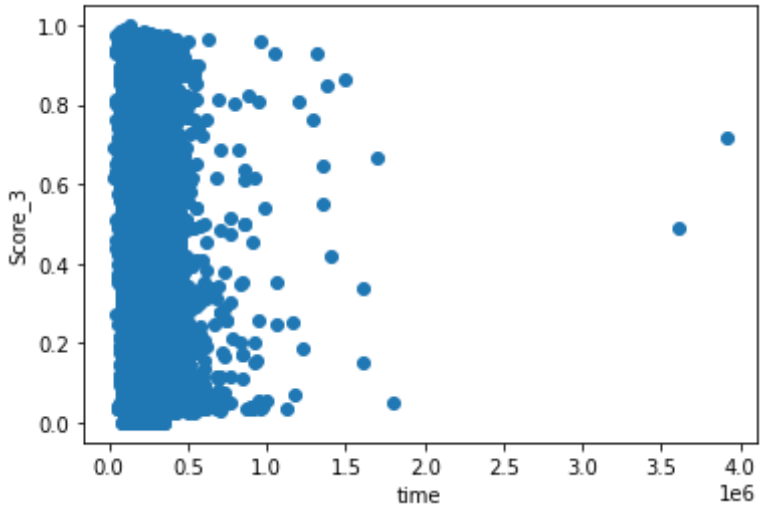


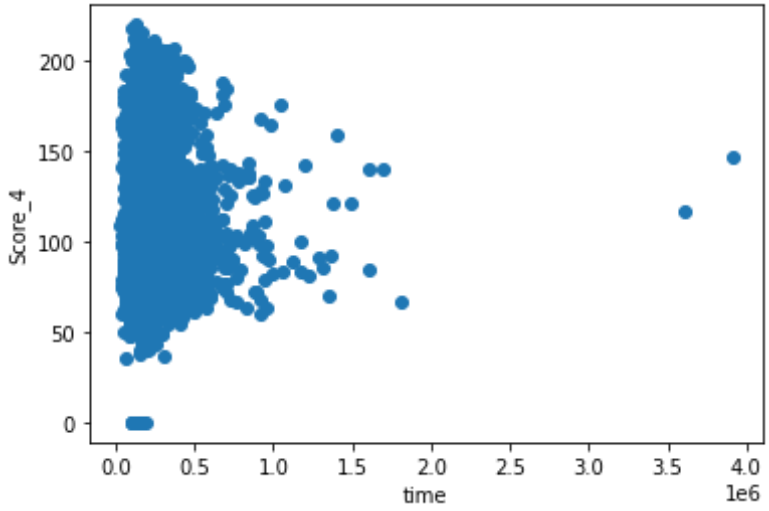


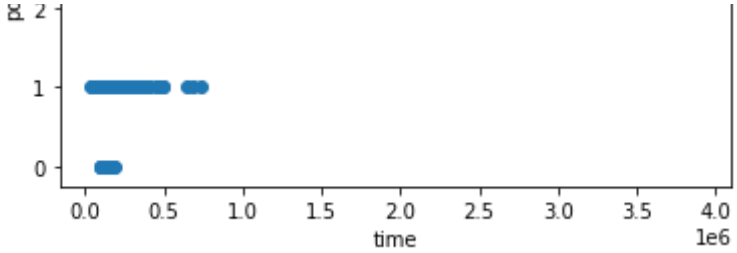


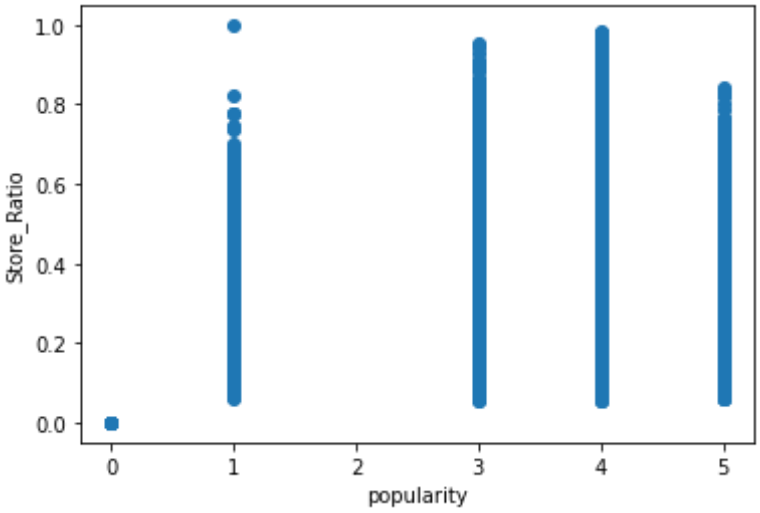


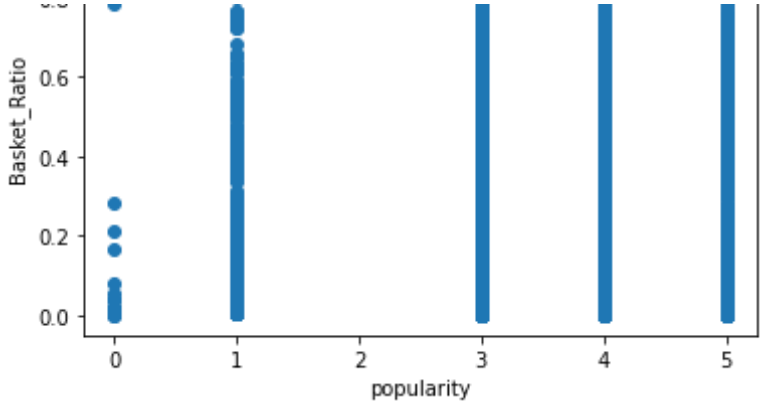


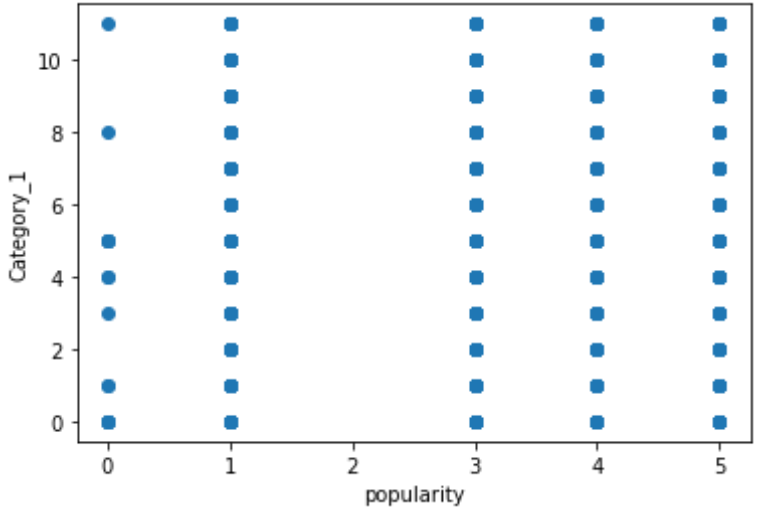


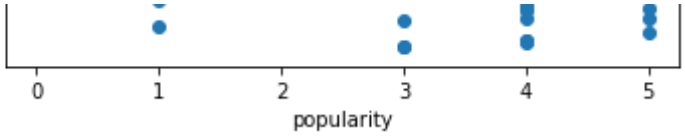


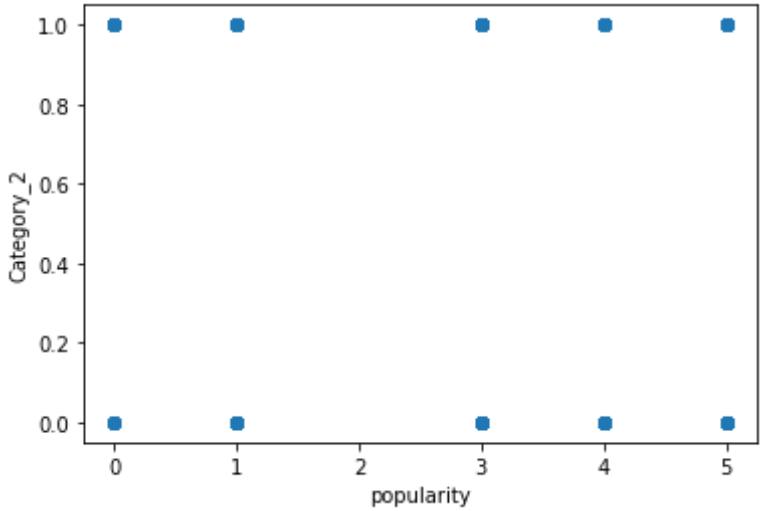


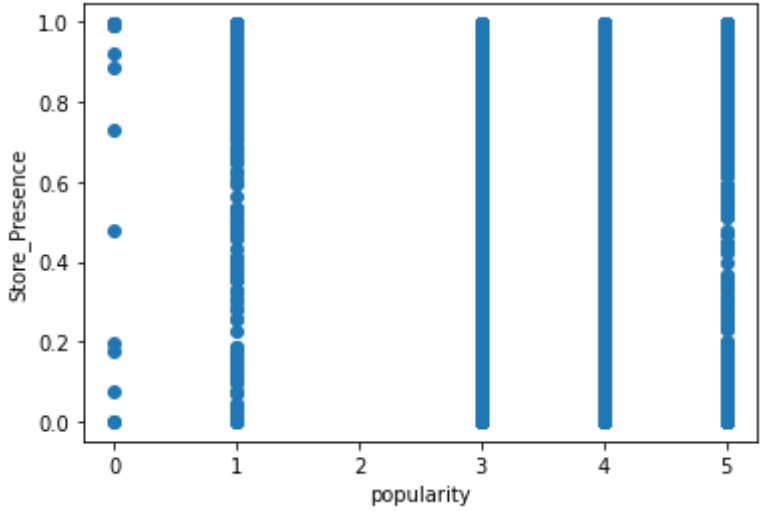


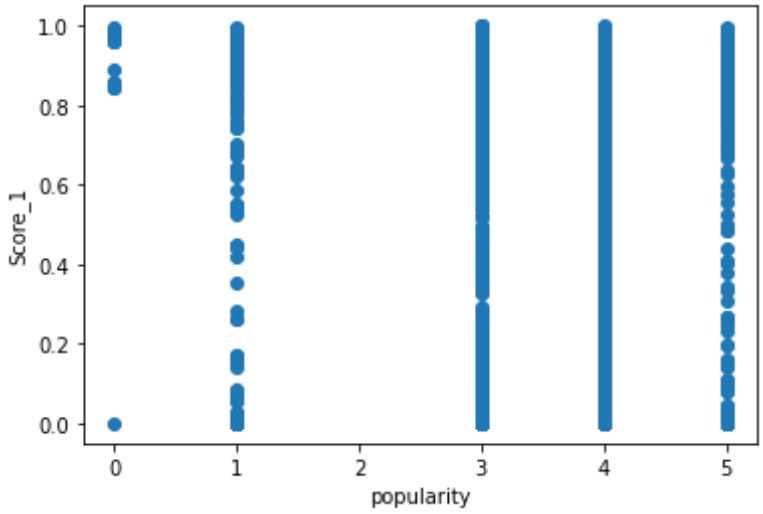


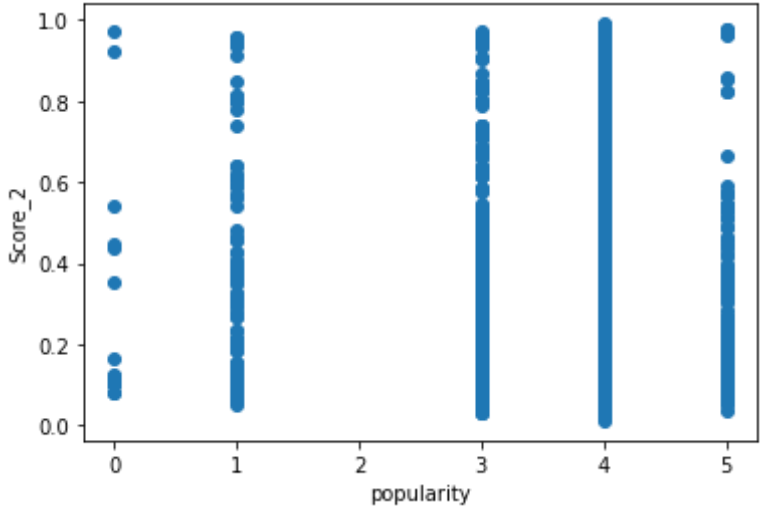


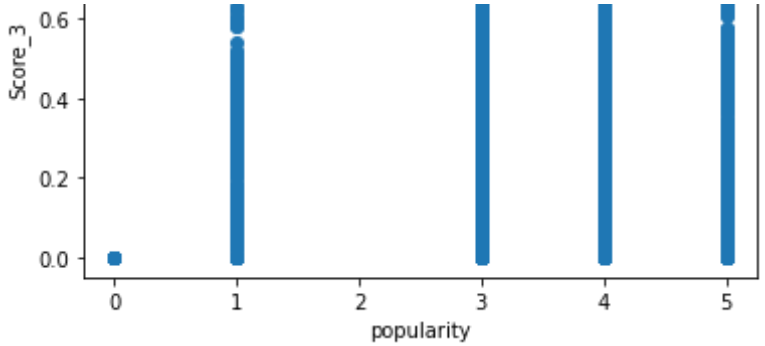


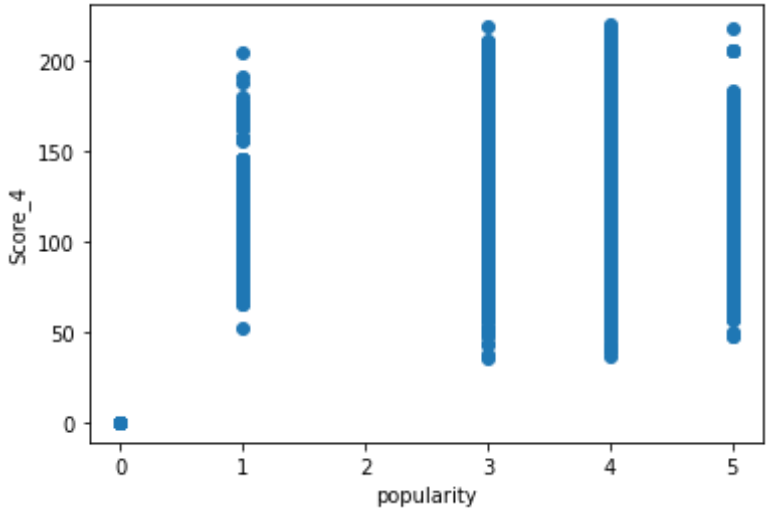


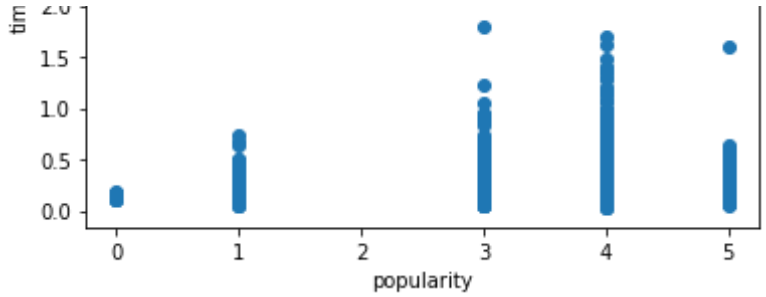












Observations from above scatter plot

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

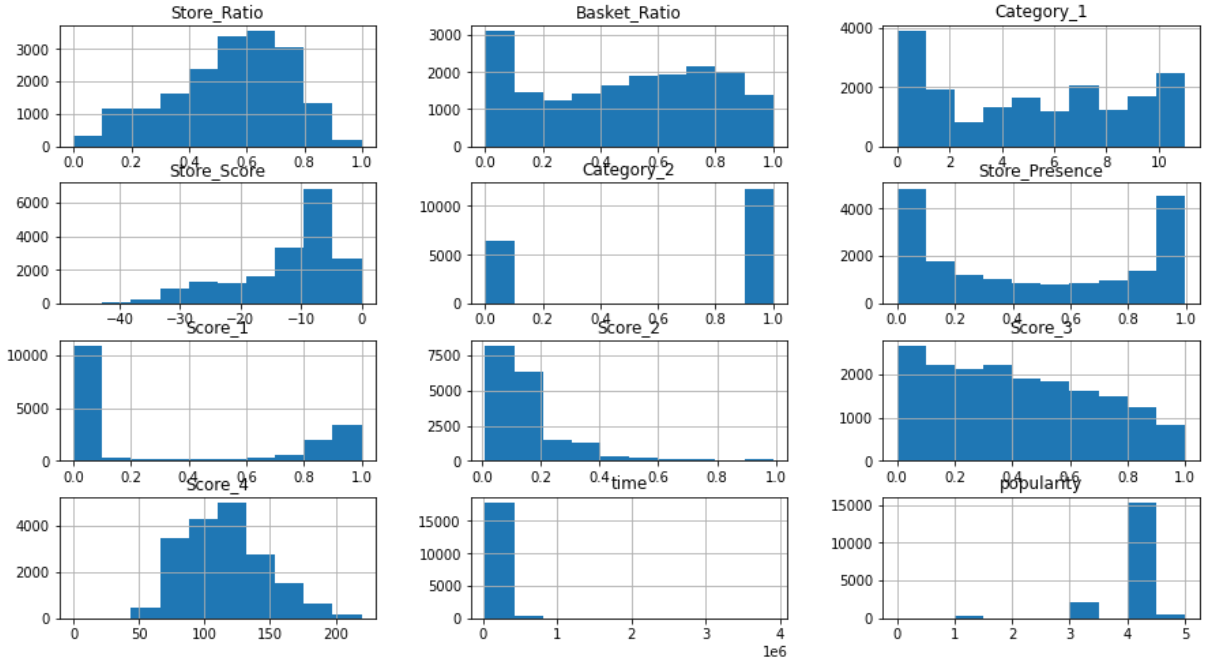
HISTOGRAM

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
```

Out[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c095c10
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04bd9f4d0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c128b50
>],
```

```
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc04be7a210
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04bd44890
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04bd9b650
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc04bfe4210
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c335390
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c335710
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c2e2ed0
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc054d7d310
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc04c3ed190
>]],
      dtype=object)
```



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


# target values


# numerical values column names


#print names
```

Out[ ]:
```
['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,  True,  True,  True,  True,  True,  True,  True,
        True,  True])
```

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
```

```
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 column
s 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 colu
mns 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 fea
tures
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.

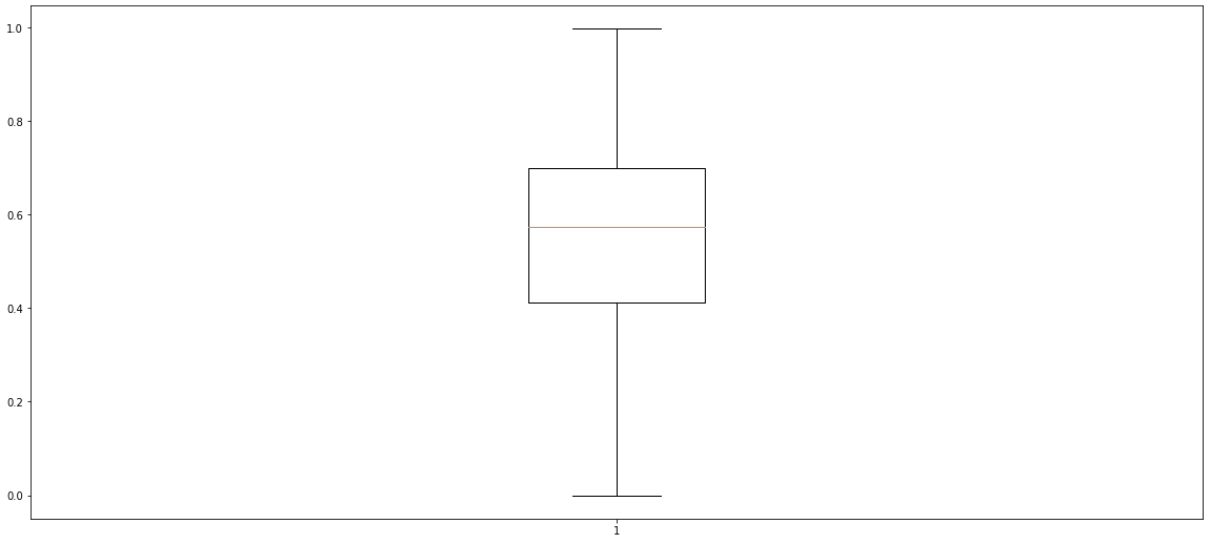
3. Median (Q2 or 50th percentile): the middle value of the dataset.

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f002137c9d0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0021381950>,
  <matplotlib.lines.Line2D at 0x7f0021381e90>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0021387990>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0021387450>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f00213c8150>,
  <matplotlib.lines.Line2D at 0x7f0021381410>]}
```



from above box plot graph:

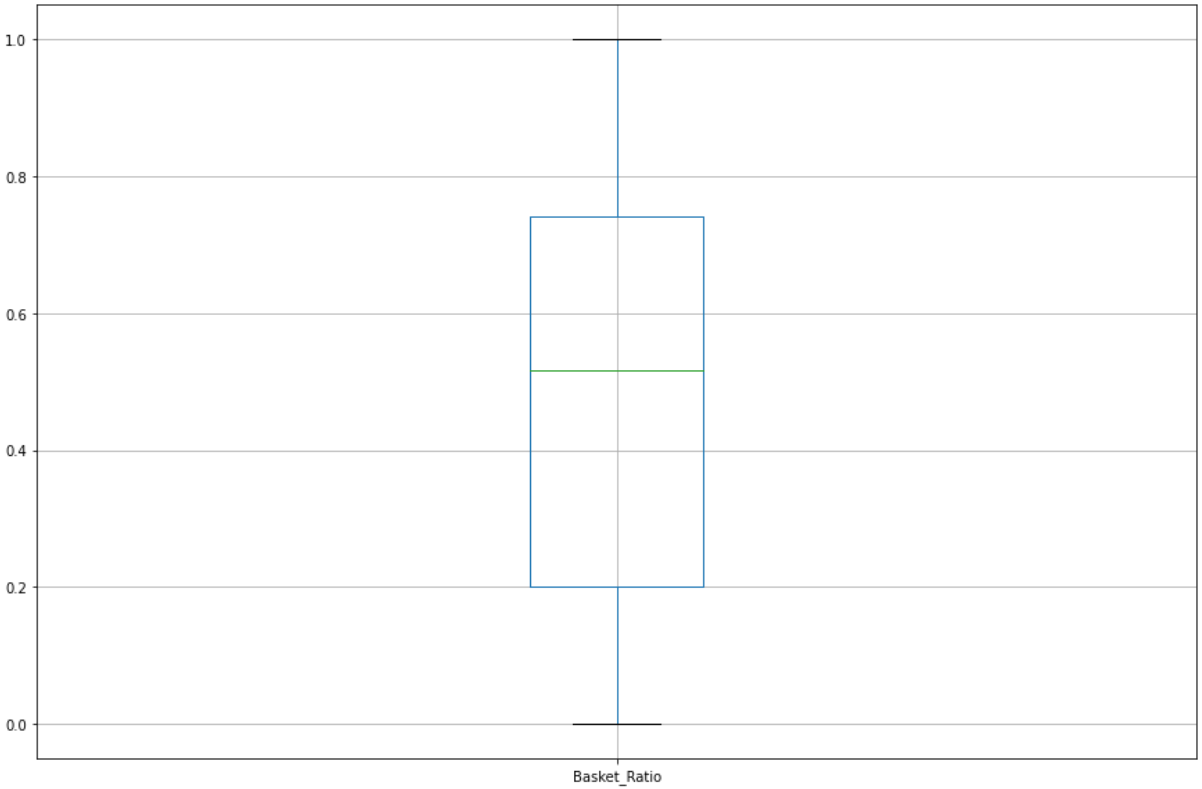
Store ratio

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
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f002129c810>
```



from above box plot graph:

basket ratio

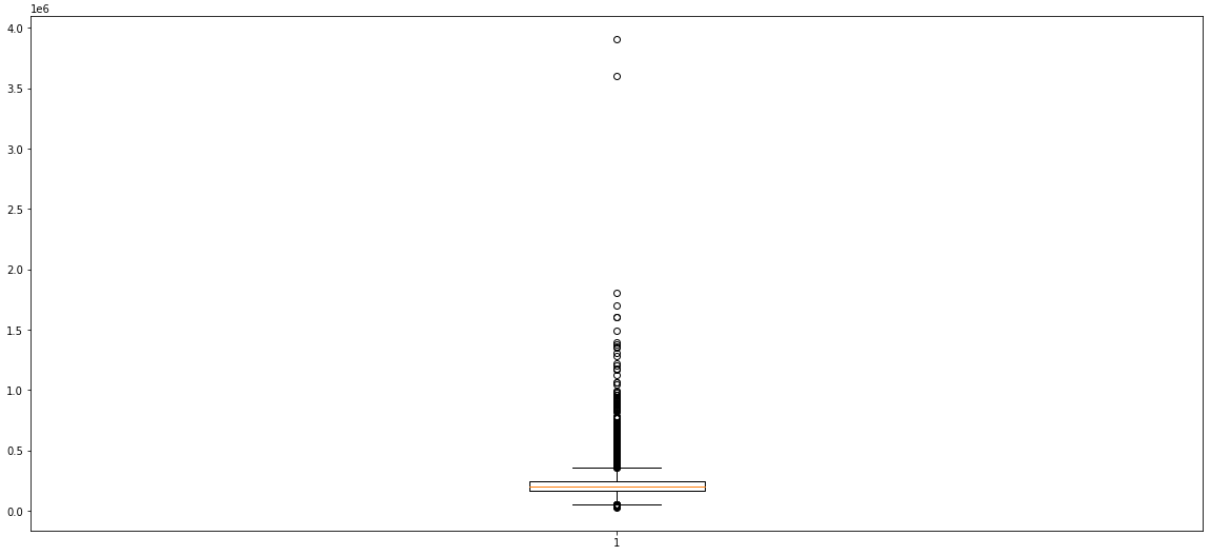
1. 25% of basket ratio have value between range 0 to 0.2.
2. 25% of basket ratio have value between range 0.4 to 0.52
3. 25% of basket ratio have value between range 0.52 to 0.78.
4. 25% of basket ratio have value between range 0.78 to 1.

The mean basket ratio is around 0.52

In [ ]:
```
# Perform a box plot on time
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba52aec50>],
```

```
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba52c2c90>,
  <matplotlib.lines.Line2D at 0x7f0ba52ca210>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba52cad10>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba52ca790>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba52e9710>,
  <matplotlib.lines.Line2D at 0x7f0ba52c2750>]}
```



from above box plot graph:

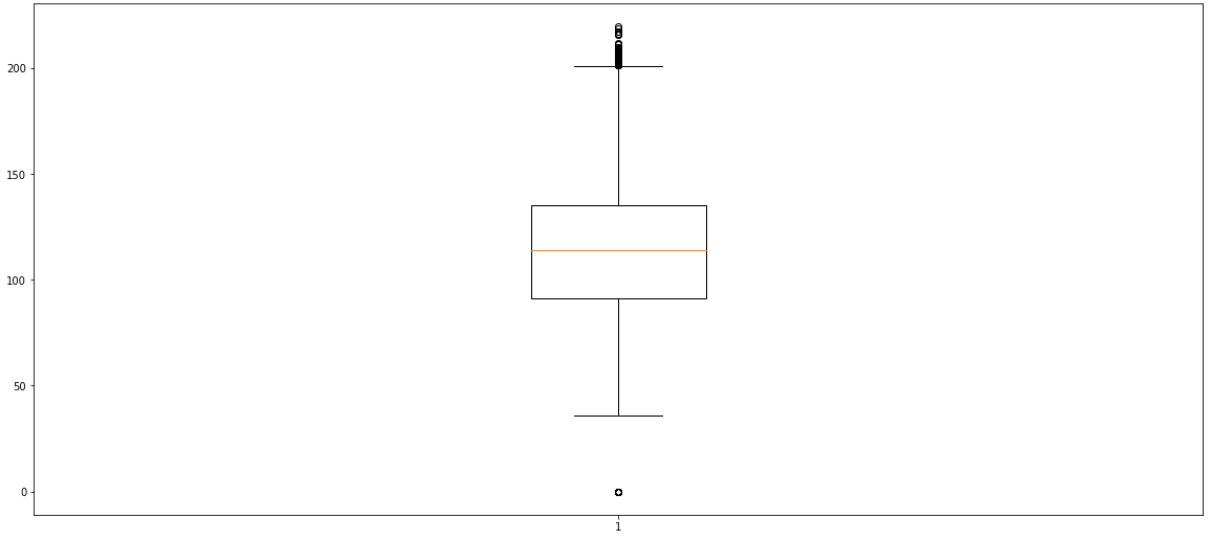
time

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4da0910>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4da49d0>,
  <matplotlib.lines.Line2D at 0x7f0ba4da4f10>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4daca10>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4dac4d0>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4da0f10>,
  <matplotlib.lines.Line2D at 0x7f0ba4da4490>]}
```



from above box plot graph:

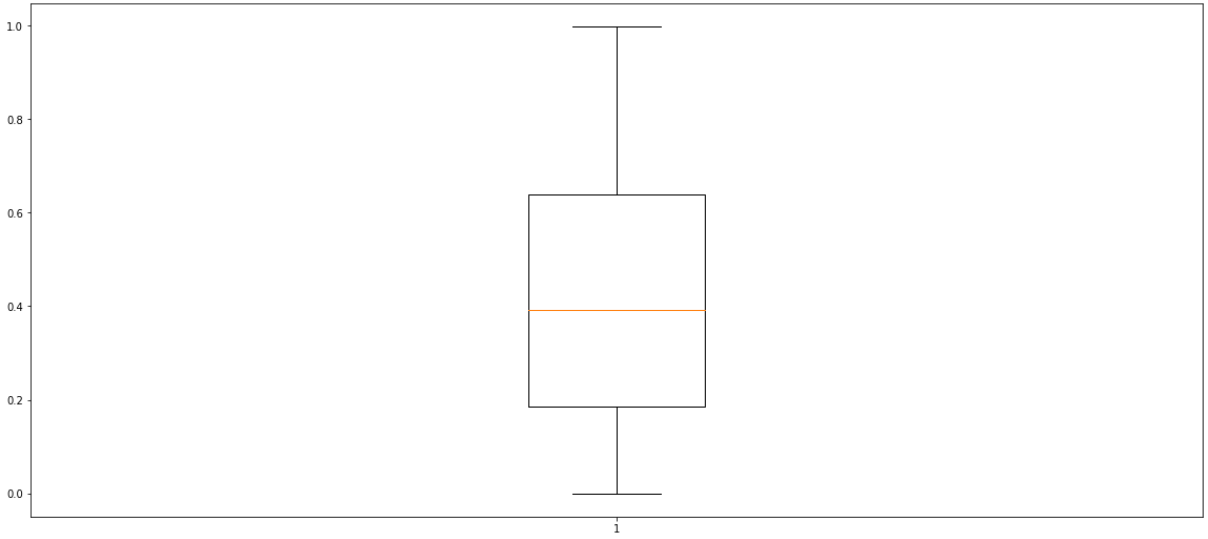
score 4

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4d8be90>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4d92f50>,
  <matplotlib.lines.Line2D at 0x7f0ba4d1a4d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4d1af90>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4d1aa50>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4d924d0>,
  <matplotlib.lines.Line2D at 0x7f0ba4d92a10>]}
```



from above box plot graph:

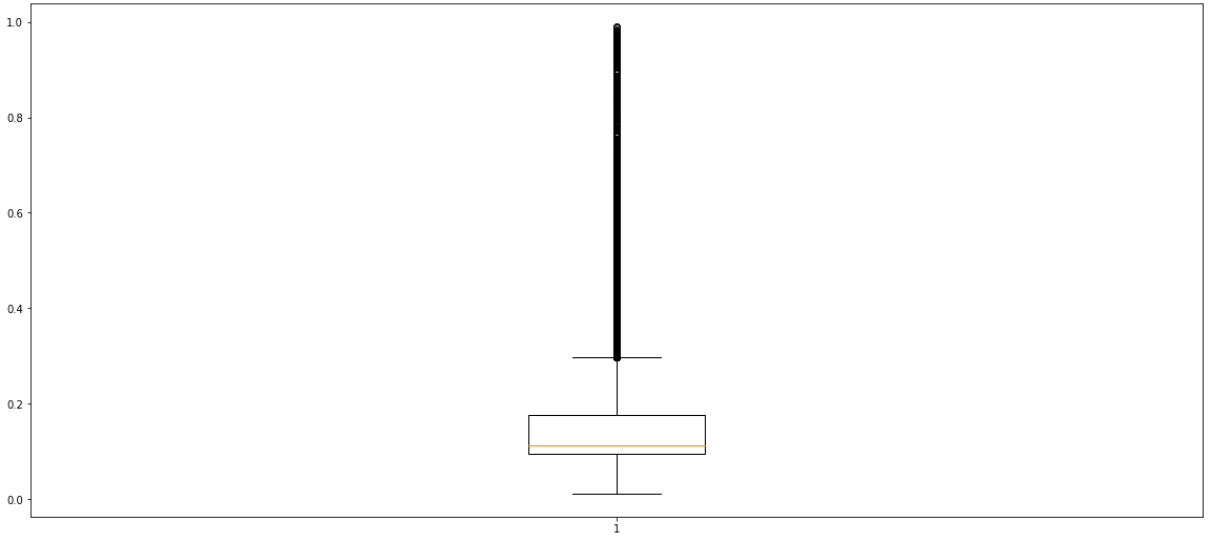
score3

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4cffcd0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4d06d90>,
  <matplotlib.lines.Line2D at 0x7f0ba4d0b310>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4d0bdd0>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4d0b890>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4d06310>,
  <matplotlib.lines.Line2D at 0x7f0ba4d06850>]}
```



from above box plot graph:

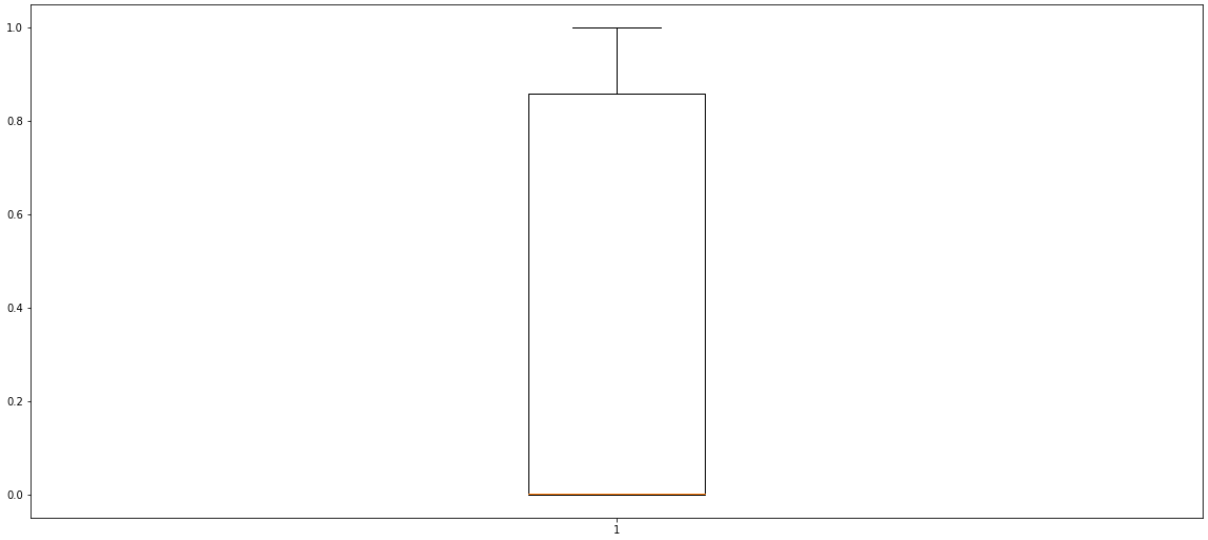
score2

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4d5da50>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4dd0750>,
  <matplotlib.lines.Line2D at 0x7f0ba4dd0c90>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4d554d0>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4d8bd10>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4d64110>,
  <matplotlib.lines.Line2D at 0x7f0ba4d64750>]}
```



from above box plot graph:

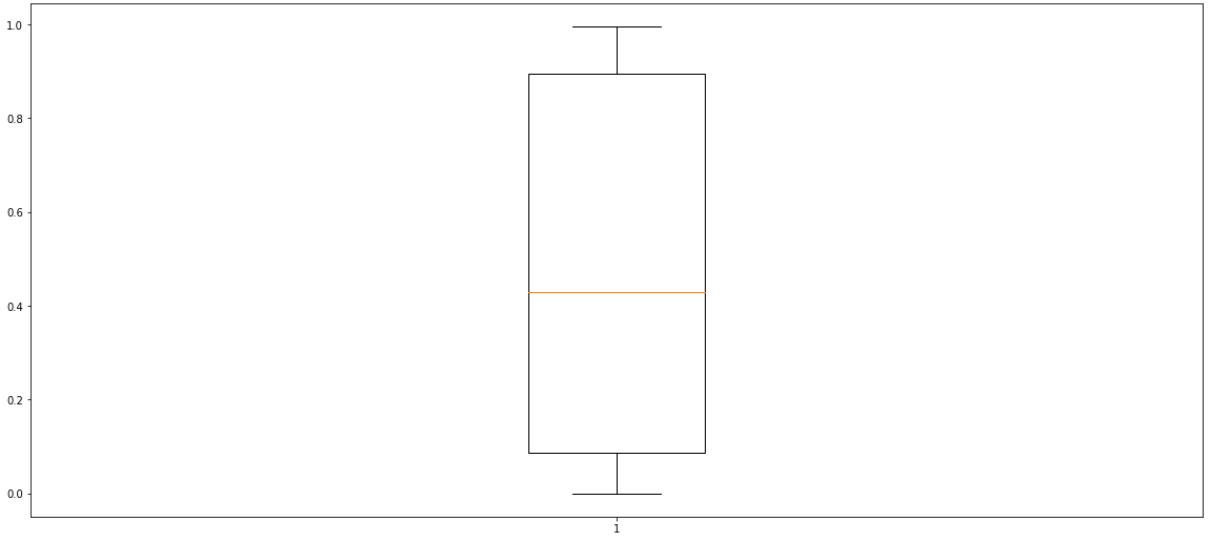
score1

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
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4c54250>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4c59310>,
  <matplotlib.lines.Line2D at 0x7f0ba4c59850>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4be1350>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4c59dd0>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4c54850>,
  <matplotlib.lines.Line2D at 0x7f0ba4c54d90>]}
```



from above box plot graph:

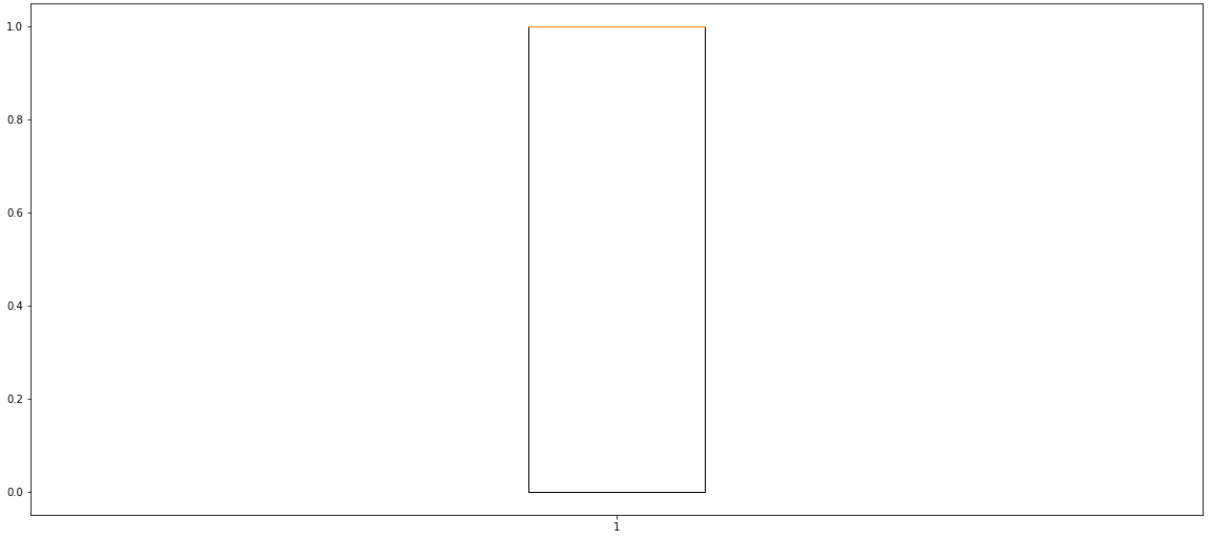
store presence

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 Category_2
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4b2b690>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4b30750>,
  <matplotlib.lines.Line2D at 0x7f0ba4b30c90>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4b37790>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4b37250>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4b2bc90>,
  <matplotlib.lines.Line2D at 0x7f0ba4b30210>]}
```



from above box plot graph:

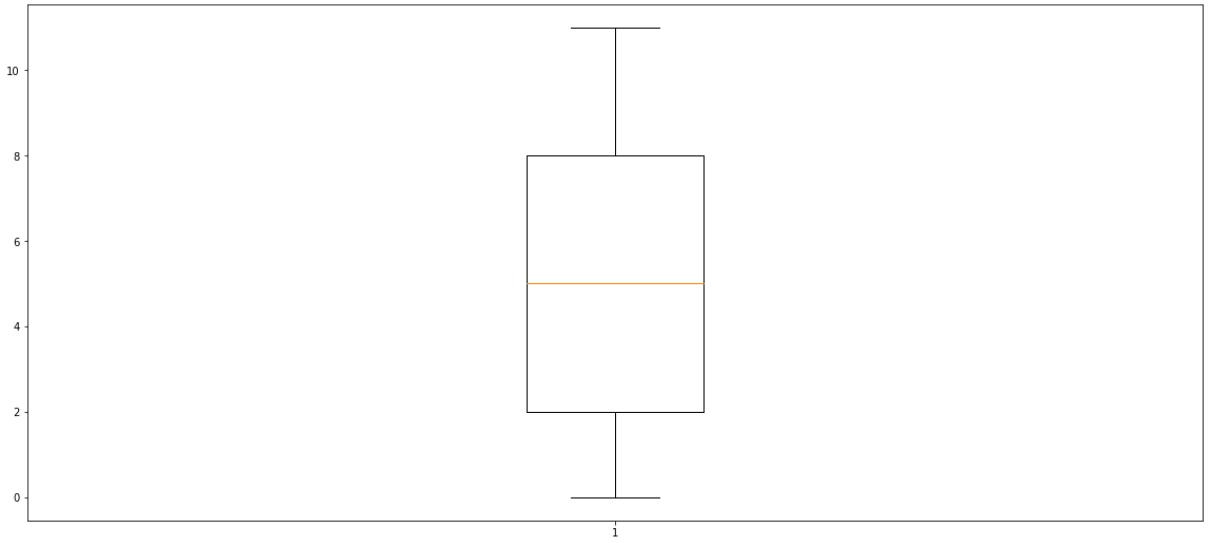
category 2

1. 25% of category2 have value between range 0 to 0.0.
2. 25% of category2 have value between range 0.0 to 0.1
3. 25% of category2 have value between range 0.1 to 0.1
4. 25% of category2 have value between range 0.1 to 1.

The mean category2 is around 0.1

In [ ]:
```
# Perform a box plot on Category_1
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4a9b090>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4aa0150>,
  <matplotlib.lines.Line2D at 0x7f0ba4aa0690>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4aa8190>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4aa0c10>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4a9b690>,
  <matplotlib.lines.Line2D at 0x7f0ba4a9bbd0>]}
```



from above box plot graph:

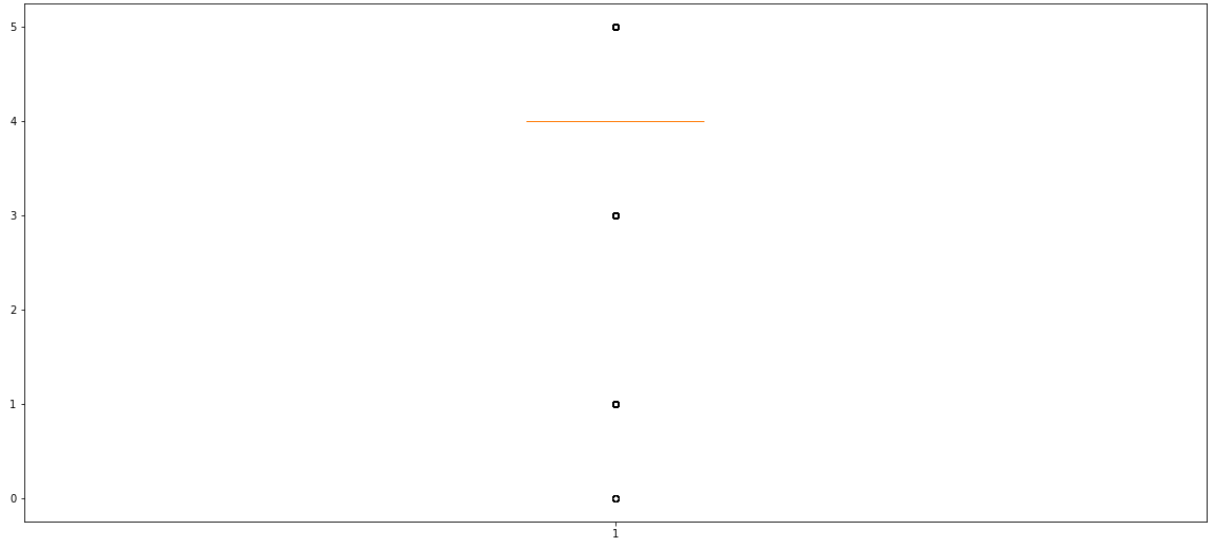
category1

1. 25% of category1 have value between range 0 to 0.2.
2. 25% of category1 have value between range 0.2 to 0.5
3. 25% of category1 have value between range 0.5 to 0.8.
4. 25% of category1 have value between range 0.8 to 1.

The mean category1 is around 0.5

In [ ]:
```
# Perform a box plot on popularity
```

Out[ ]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x7f0ba4a86910>],
 'caps': [<matplotlib.lines.Line2D at 0x7f0ba4a8b9d0>,
  <matplotlib.lines.Line2D at 0x7f0ba4a8bf10>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f0ba4a90a10>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7f0ba4a904d0>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f0ba4a86f10>,
  <matplotlib.lines.Line2D at 0x7f0ba4a8b490>]}
```



COUNT PLOT

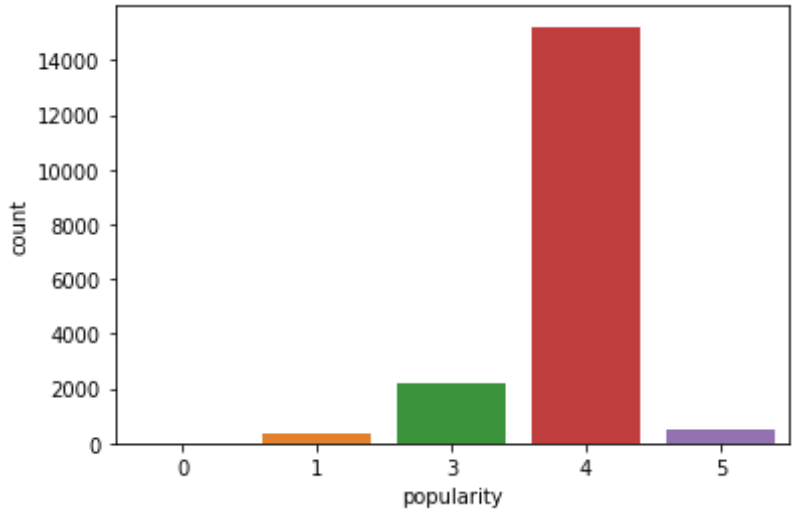
1. A countplot is kind of like a histogram or a bar graph for some categorical area.
2. It simply shows the number of occurrences of an item based on a certain type of category.

In [ ]:
```
# Perform the countplot on the popularity
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
```

```
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f03a59a0b10>
```



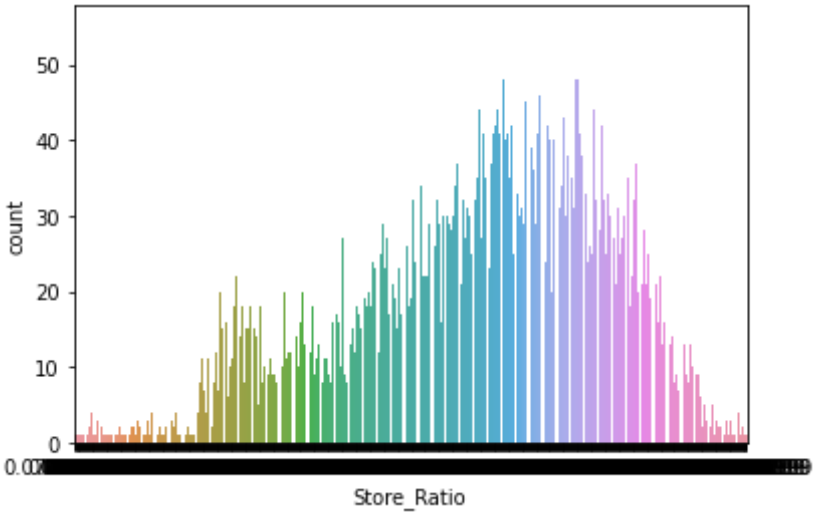
From above count plot

distribution of values over complete dataset are skewed right

In [ ]:
```
# Perform the countplot on the Store Ratio
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ba46cf610>
```



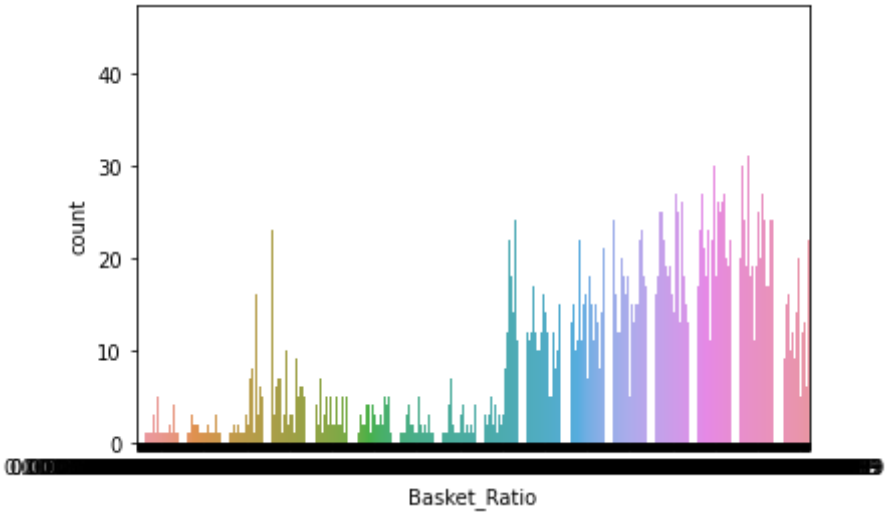
From above count plot

distribution of values over complete dataset are skewed right.

In [ ]:
```
# Perform the countplot on the Basket ratio
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ba2c1f890>
```



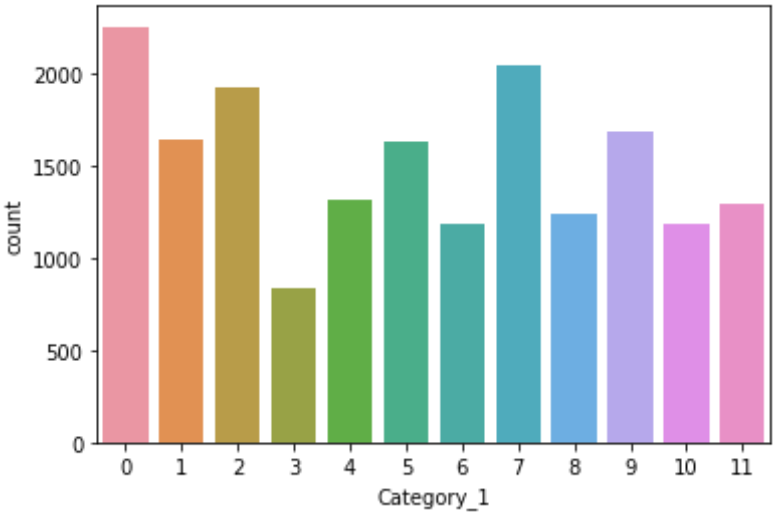
From above count plot

distribution of values over complete dataset are skewed right

In [ ]:
```
# Perform the countplot on the category 1
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ba18f3110>
```

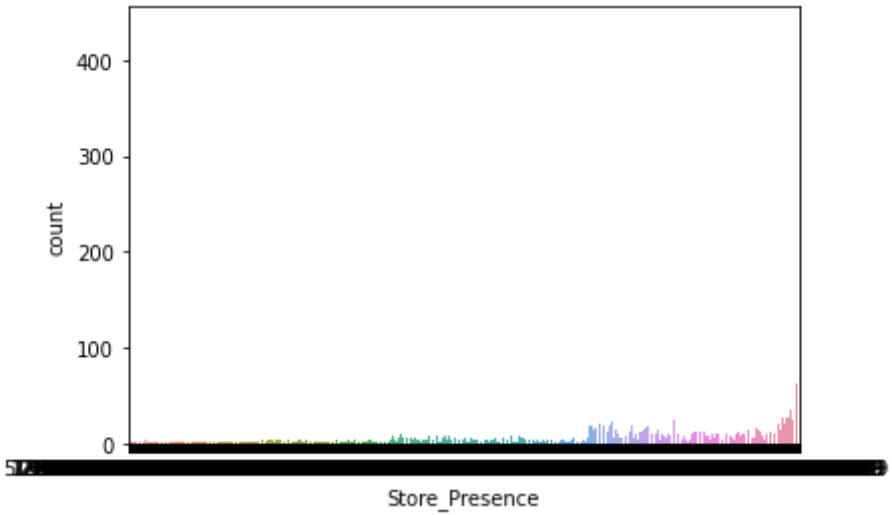


distribution of values over complete dataset is multimodal

In [ ]:
```
# Perform the countplot on the store presence
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9fb2ced0>
```



distribution of values over complete dataset is skewed to right

In [ ]:
```
# Perform the countplot on the score 1
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

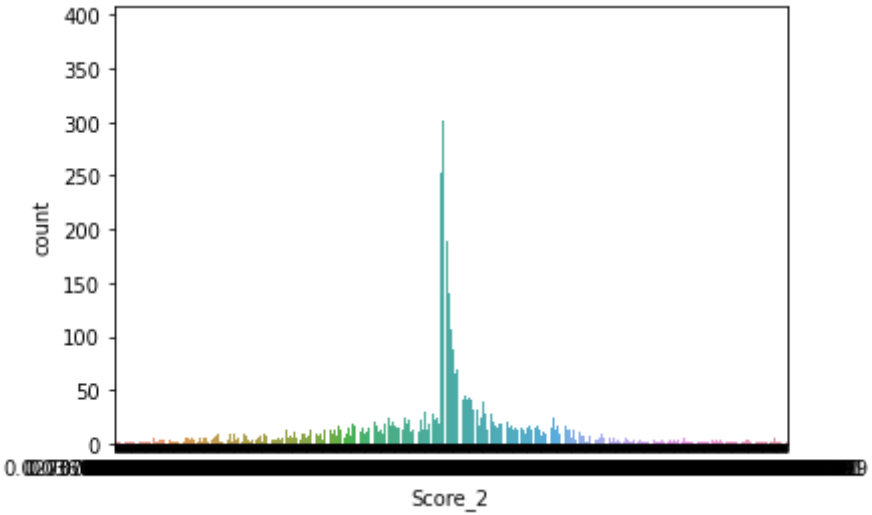
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9b3f3410>
```

All the values are round zero very few are above zero

In [ ]:
```
# Perform the countplot on the Score 2
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b9bafa250>
```

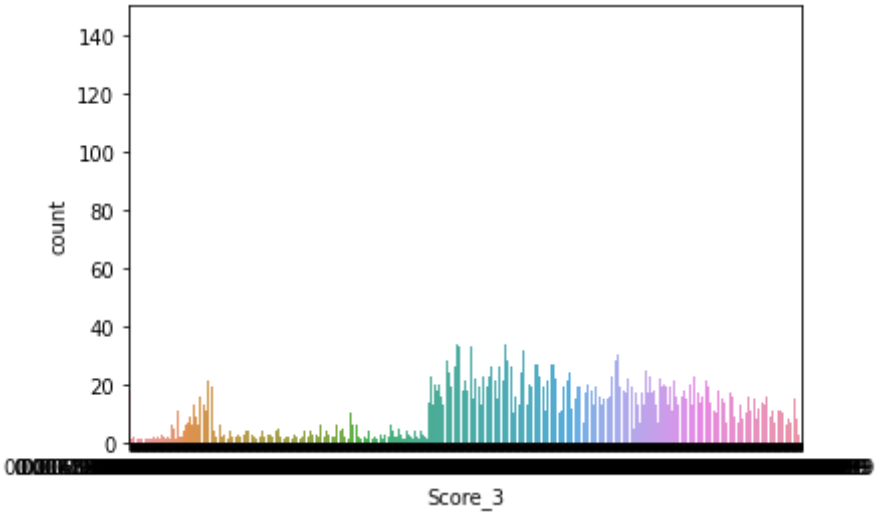


distribution of values over complete dataset is unimodal.

In [ ]:
```
# Perform the countplot on the Score 3
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

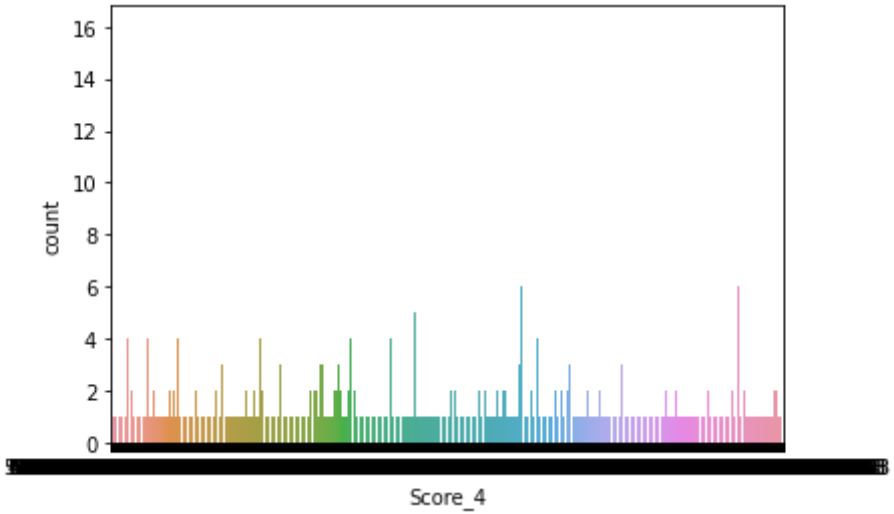
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b8c3f2510>
```



In [ ]:
```
# Perform the countplot on the Score 4
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b8dbd2b10>
```



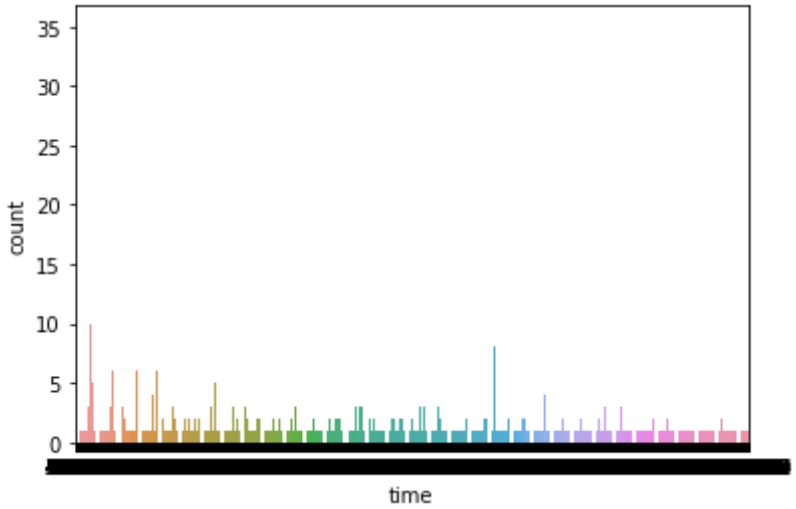
From above count plot

distribution of values over complete dataset are multi model that is more than one peak.

In [ ]:
```
# Perform the countplot on time
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
eWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing ot
her arguments without an explicit keyword will result in an error or mi
sinterpretation.
  FutureWarning
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b74e47450>
```



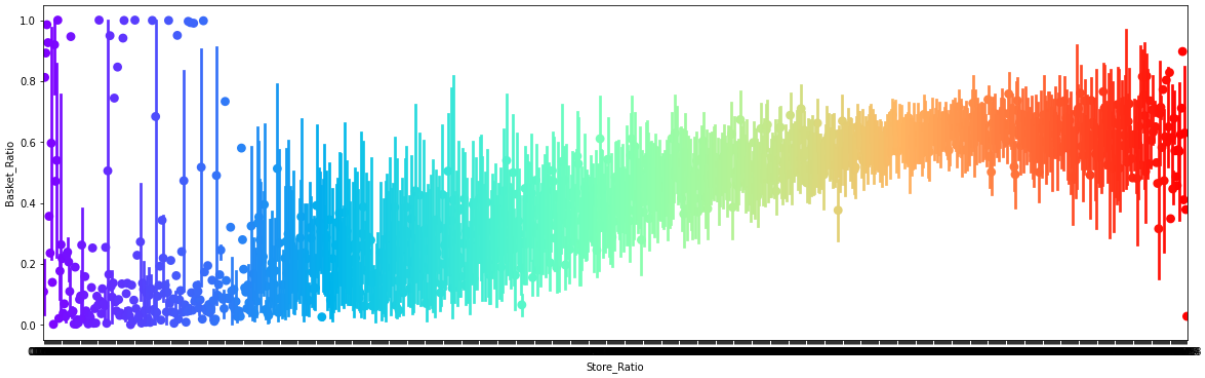
distribution of values over complete dataset are multi model

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b89ab7d50>
```

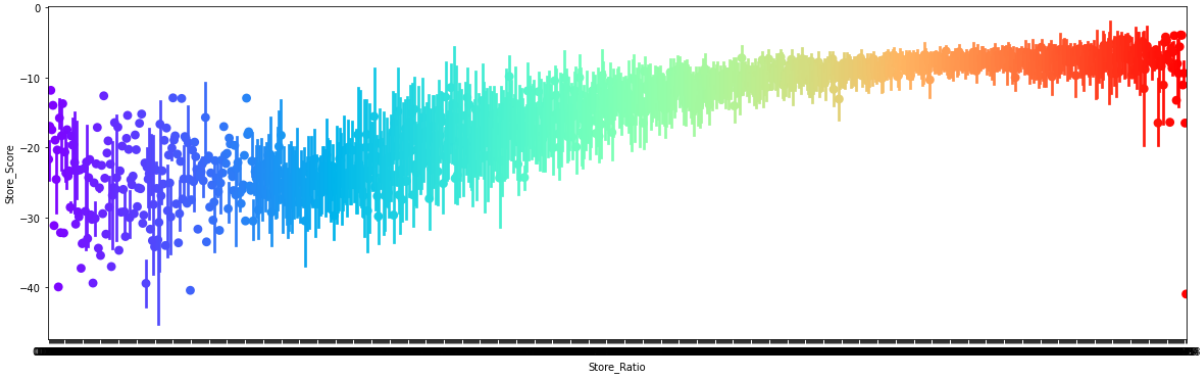


From above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b5f7ebd90>
```



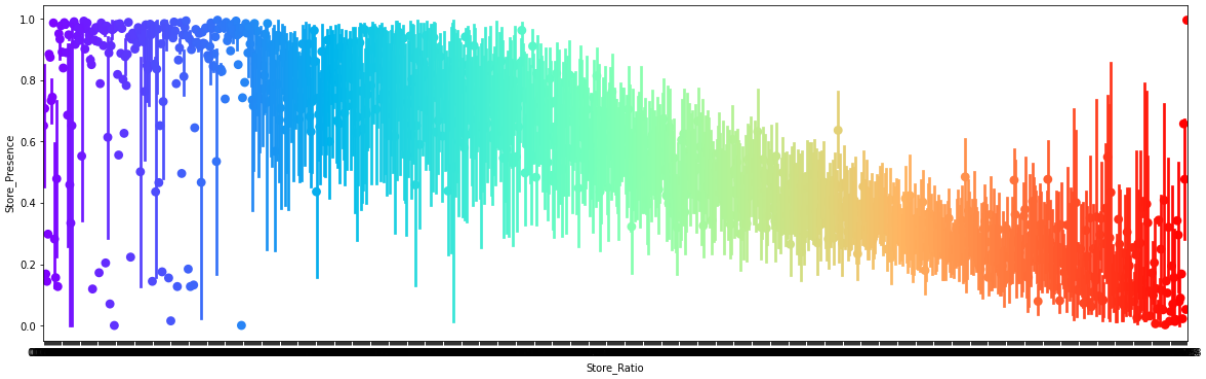
From above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b5cf93ed0>
```

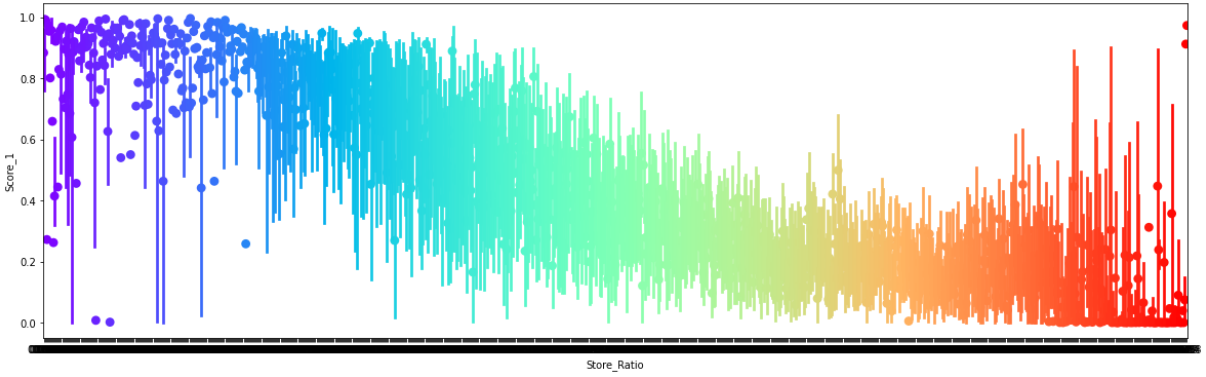


From above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b62a3fd10>
```

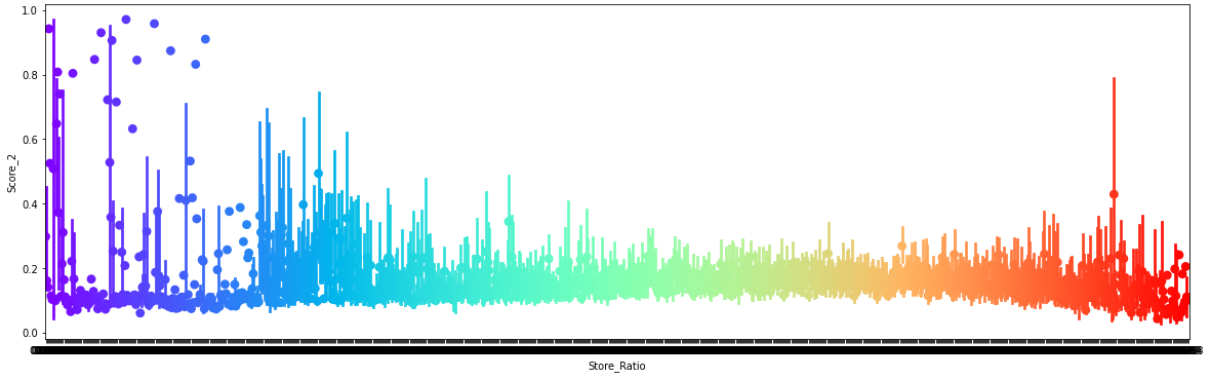


From above point plot

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

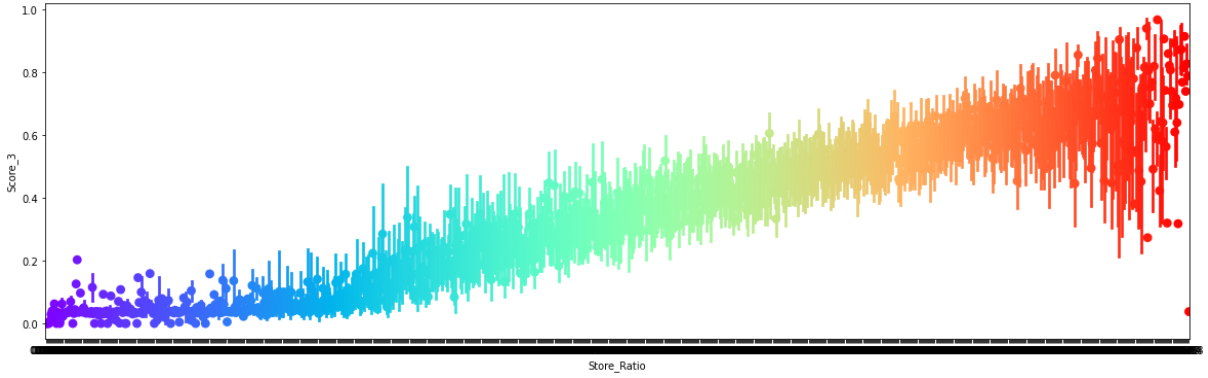
In [ ]:
```
# Perform point plot between Store Ratio and Score 2
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b5917b250>
```



In [ ]:
```
# Perform point plot between Store ratio and Score 3
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b591083d0>
```

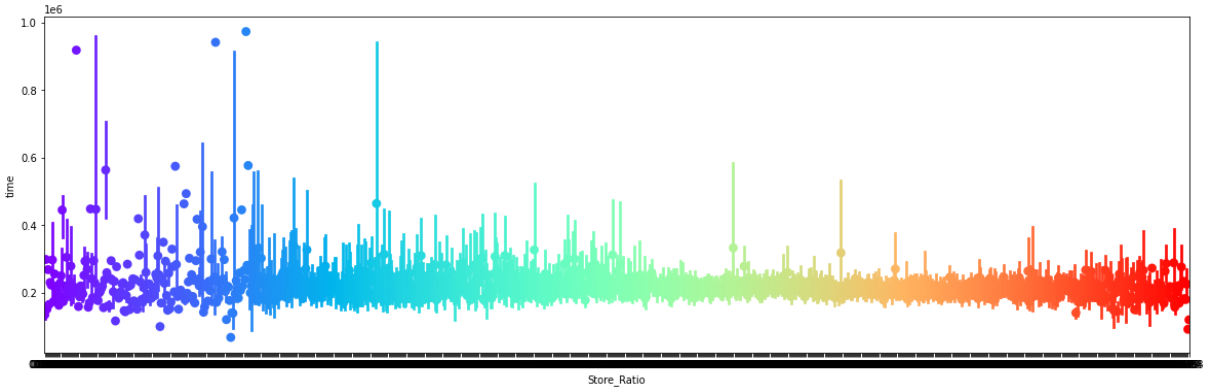


From above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b54f57250>
```

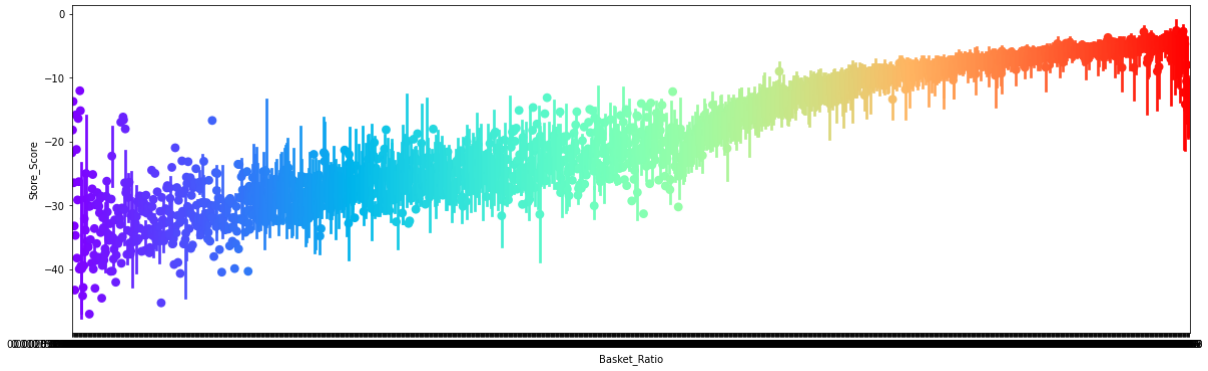


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b4c0fc850>
```

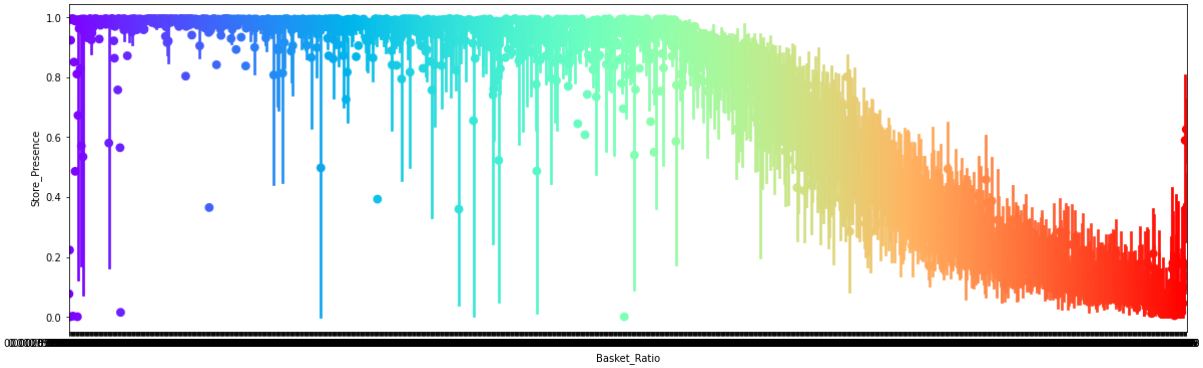


From above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b45ba4810>
```

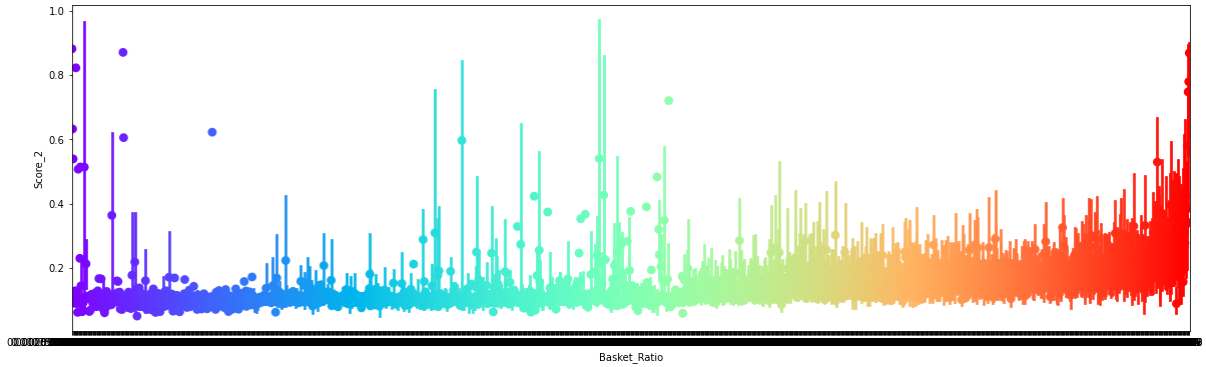


observation from above point plot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b3fcaabd0>
```

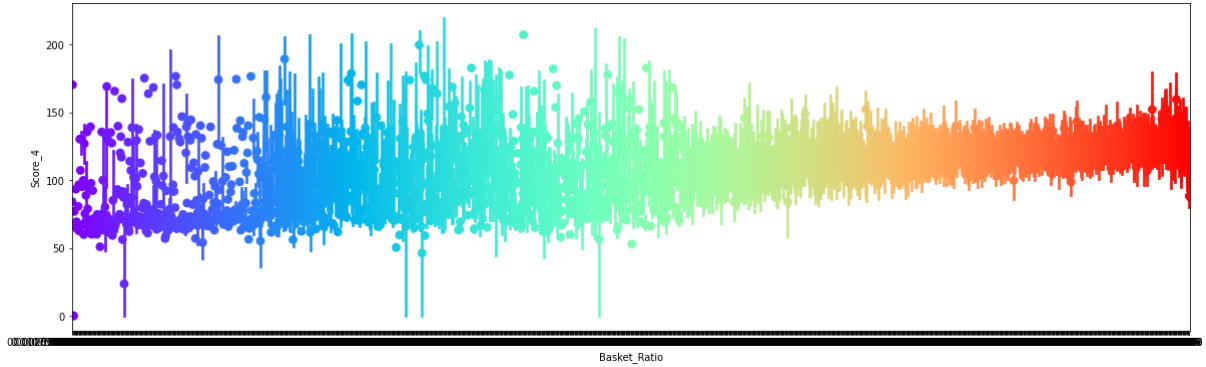


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b3cef53d0>
```



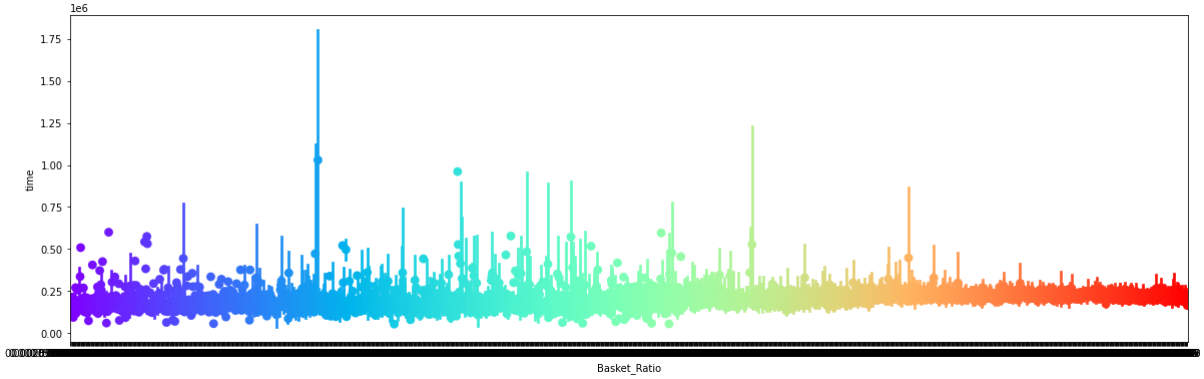
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b39ebc3d0>
```



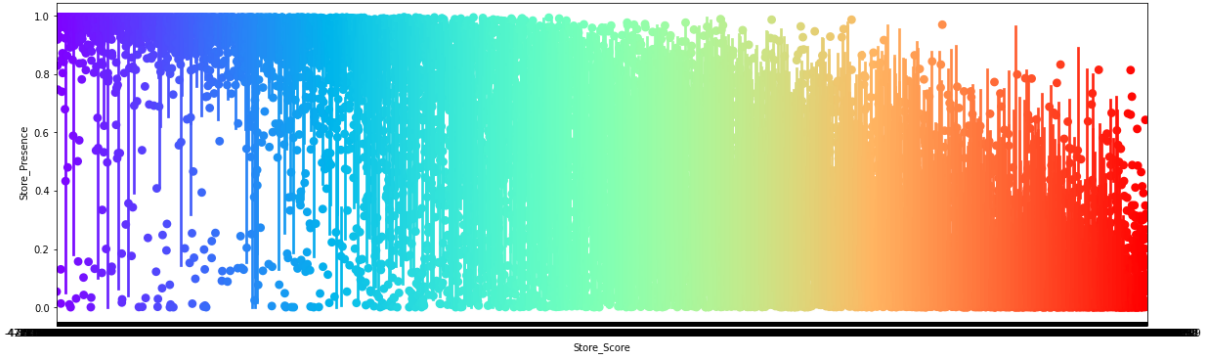
From above pointplot

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
```

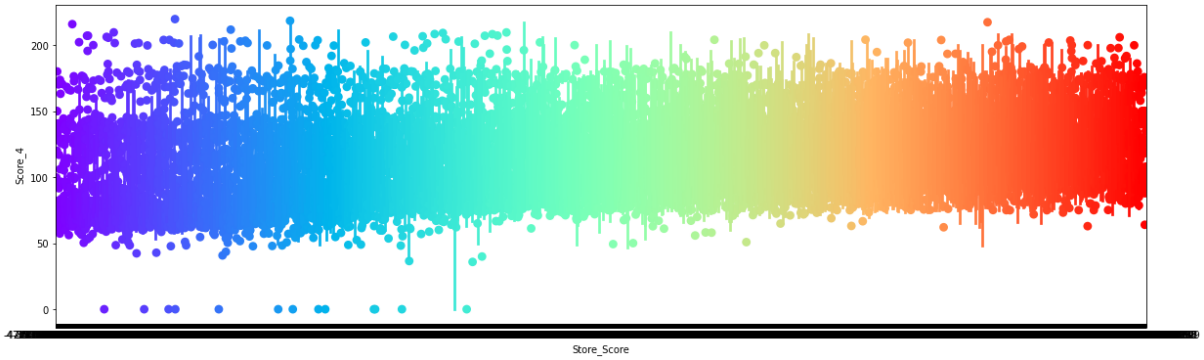
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b2fb78910>
```



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

In [ ]:
```
# Perform point plot between Store Score and Score 4
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0af04e2750>
```



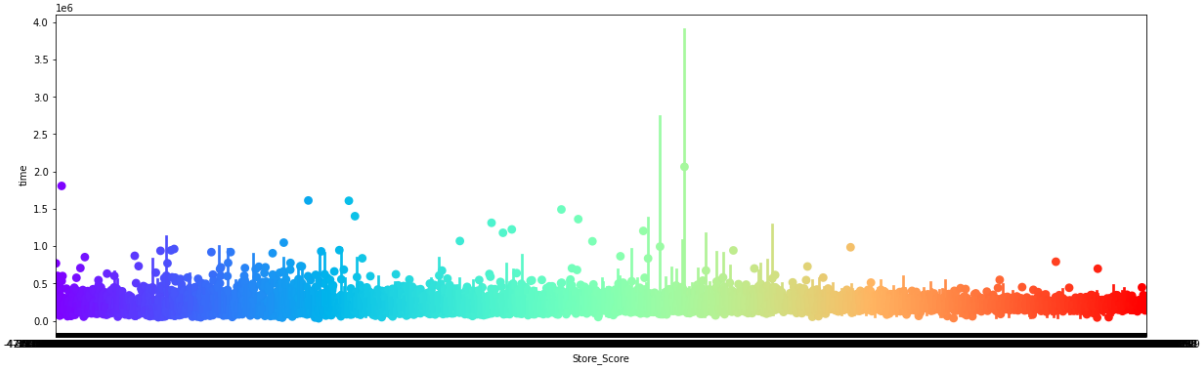
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ac3715150>
```



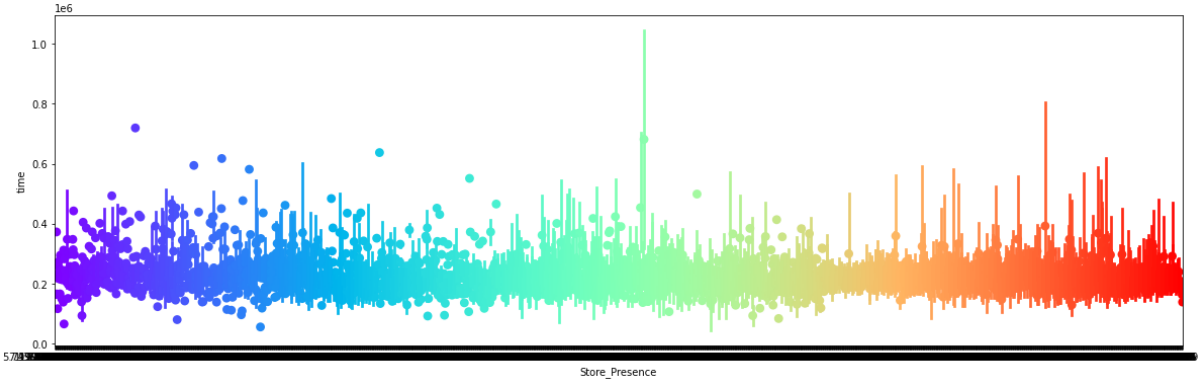
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a1cf37f10>
```



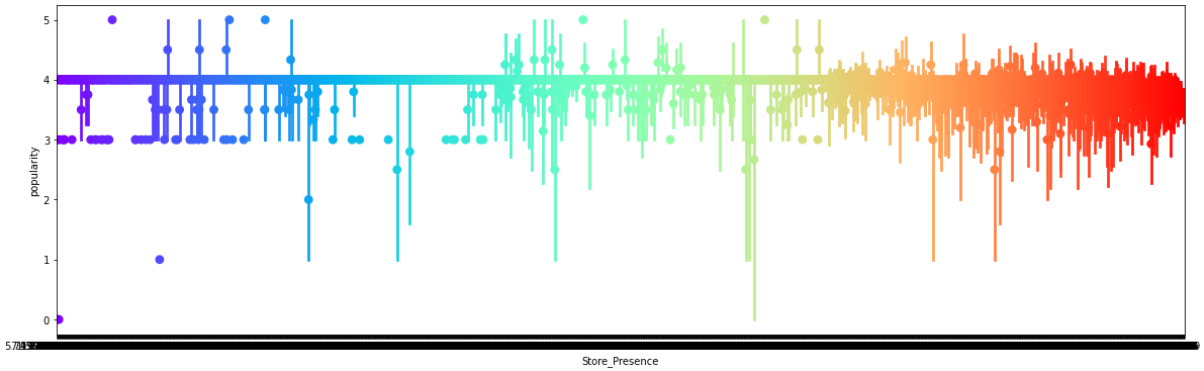
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a16d26a10>
```



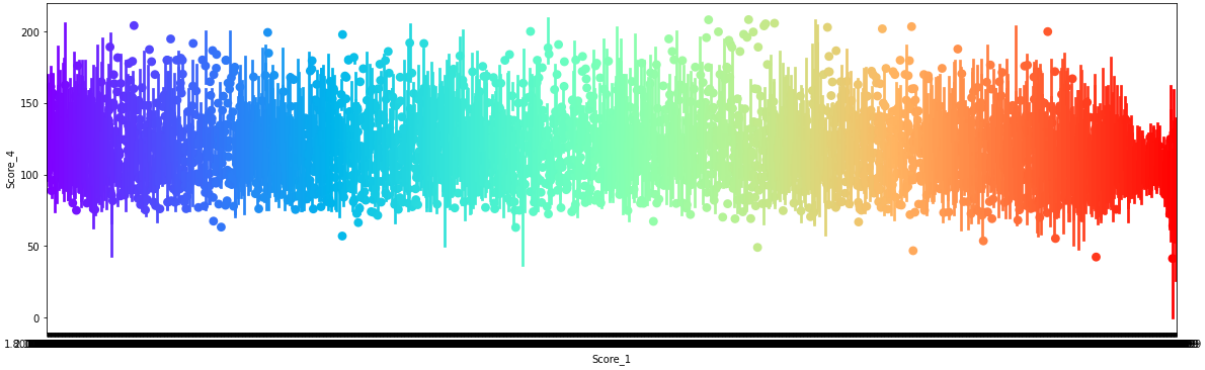
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a085b7650>
```

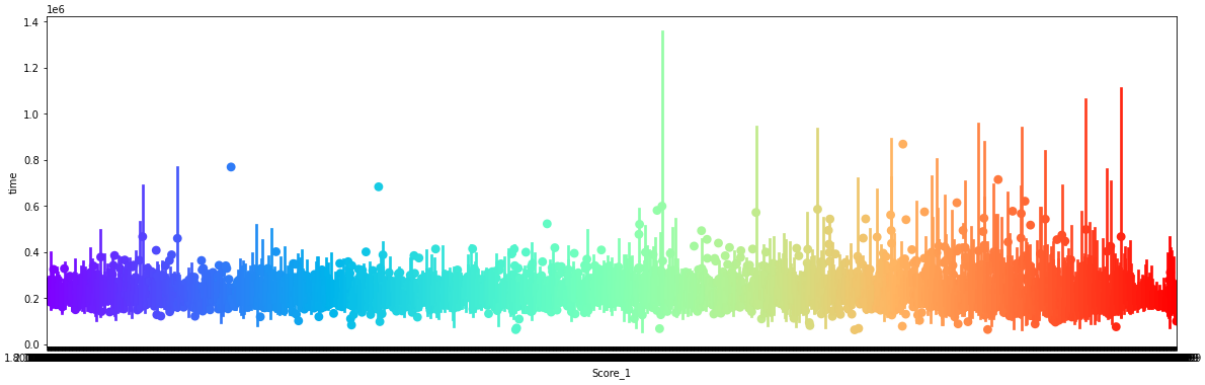


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a7df19510>
```

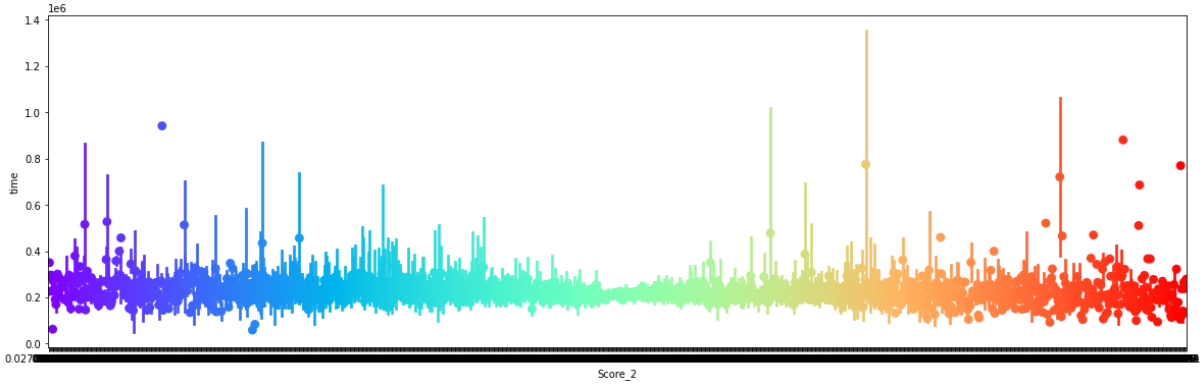


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a00029b90>
```

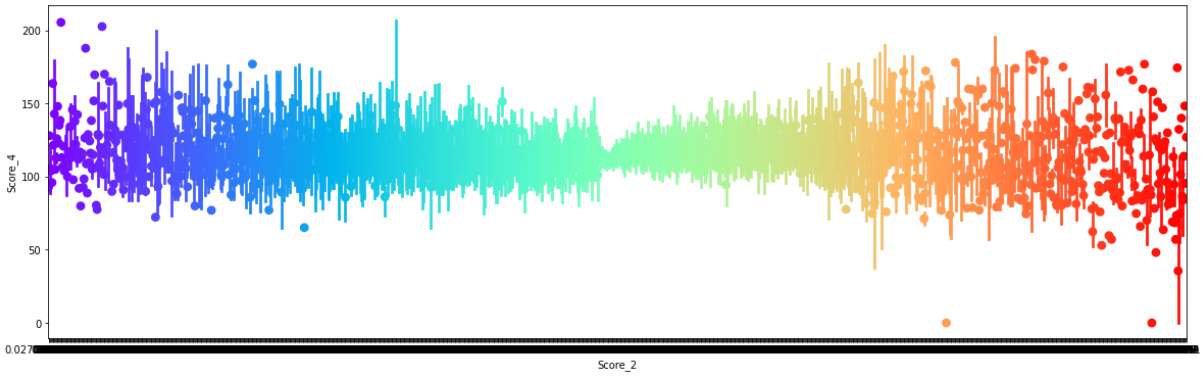


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09faf4a2d0>
```



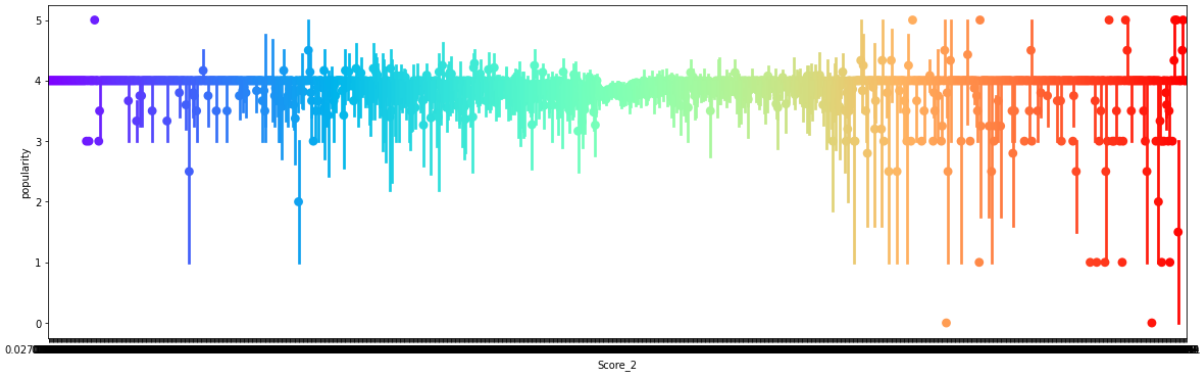
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09f4dde690>
```

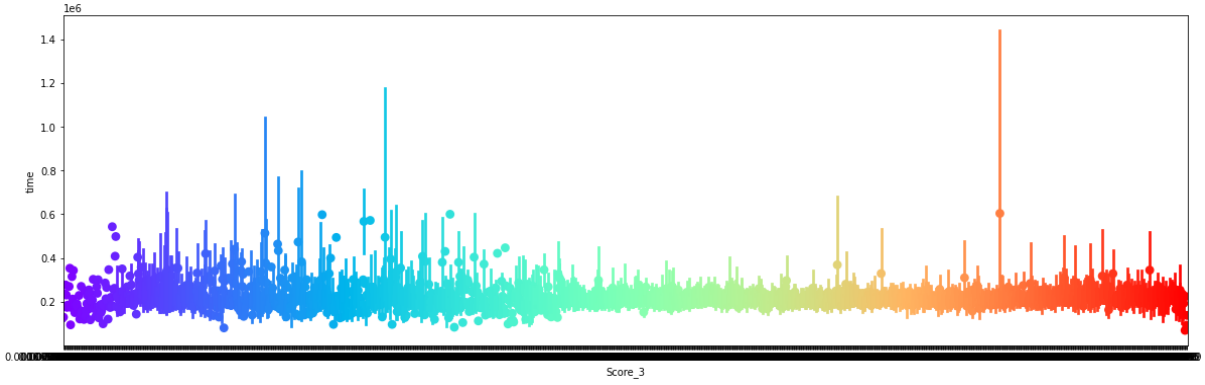


From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09f29db290>
```



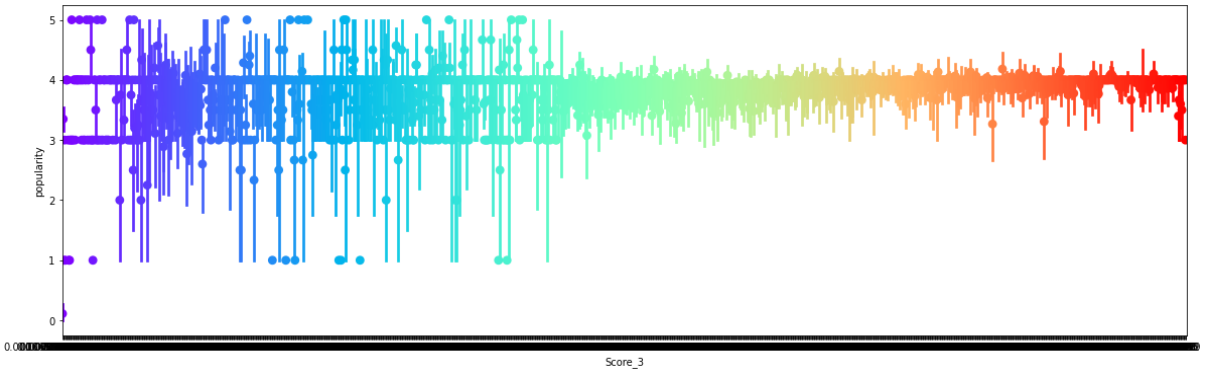
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a632d0e90>
```



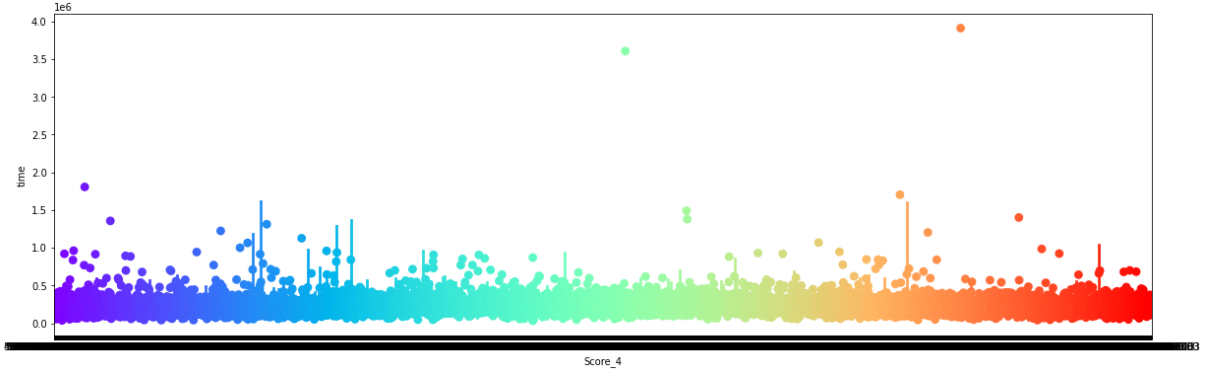
From above pointplot

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[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a632d08d0>
```



From above pointplot

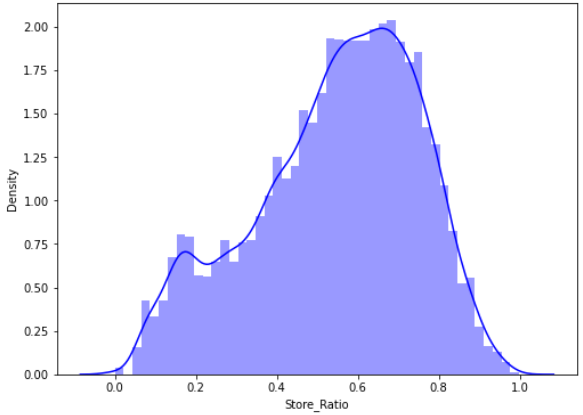
1. Most of the points are between o to 0.5
2. Very few points above 0.5

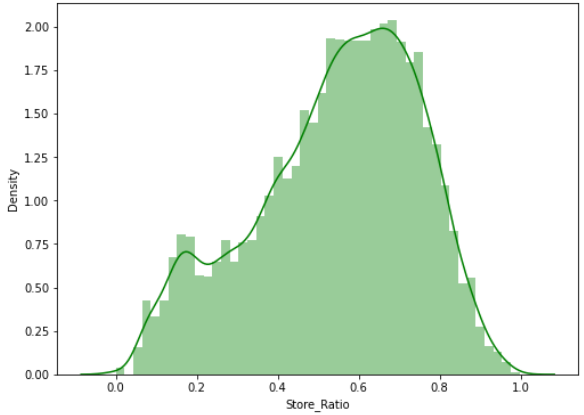
DISTPLOT

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

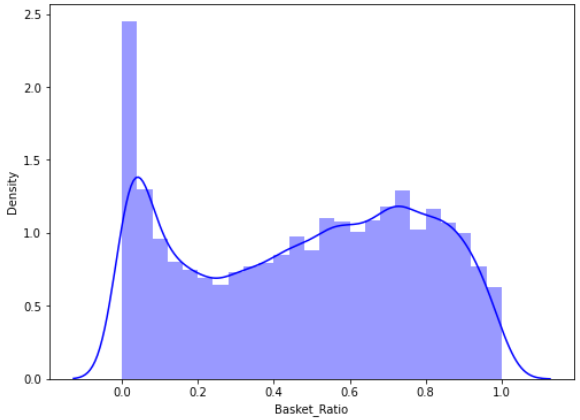
In [ ]:
```
#Perform distplot for all the columns in dataset
```

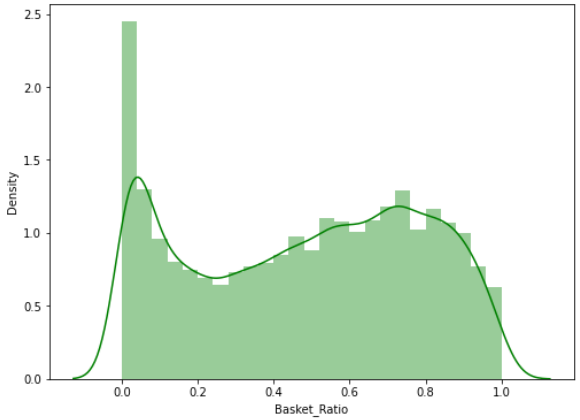
Store_Ratio



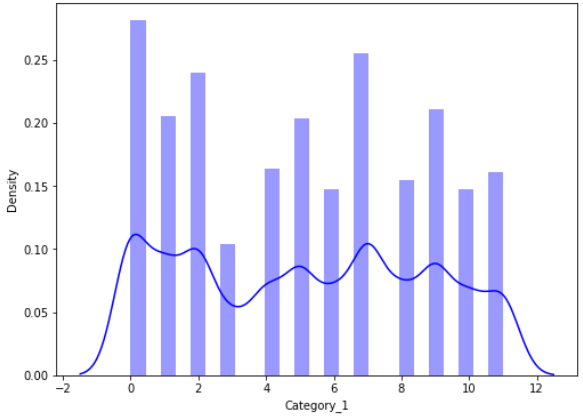


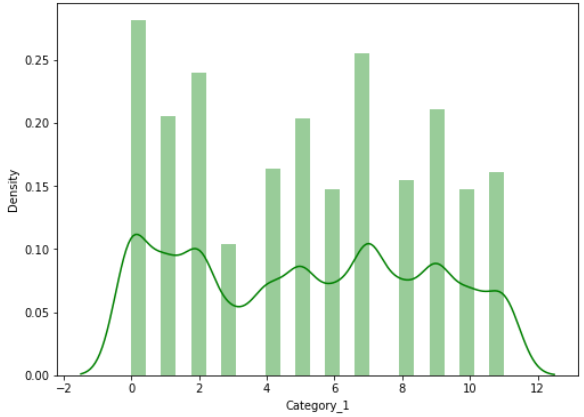
Basket_Ratio



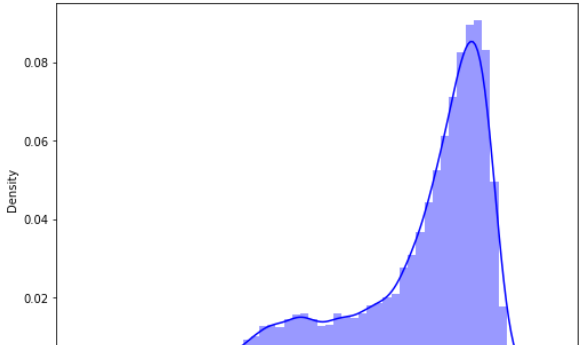


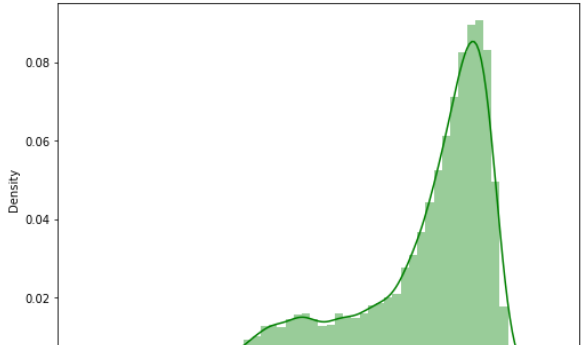
Category_1



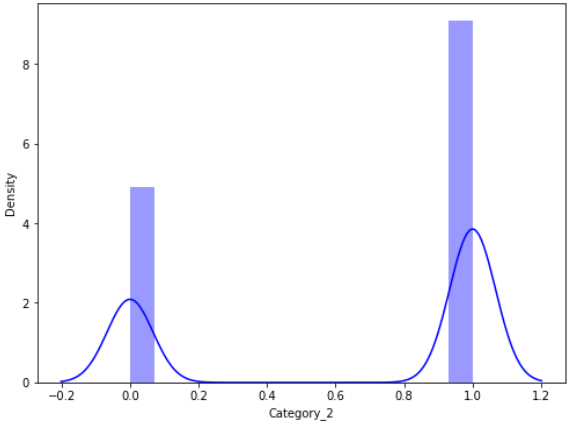


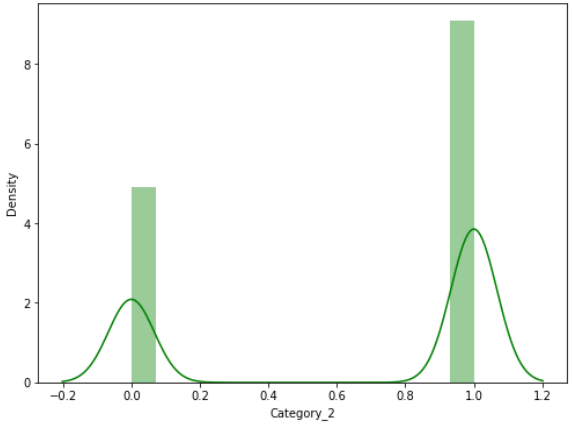
Store_Score



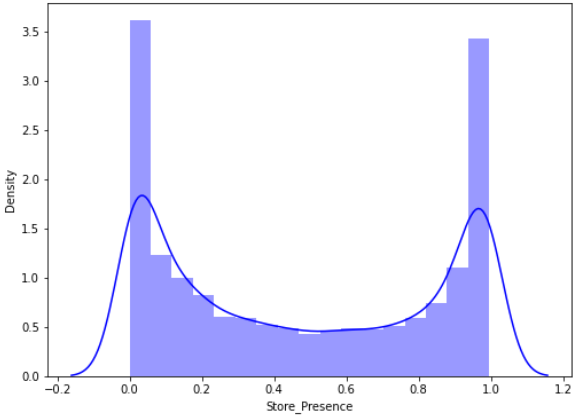


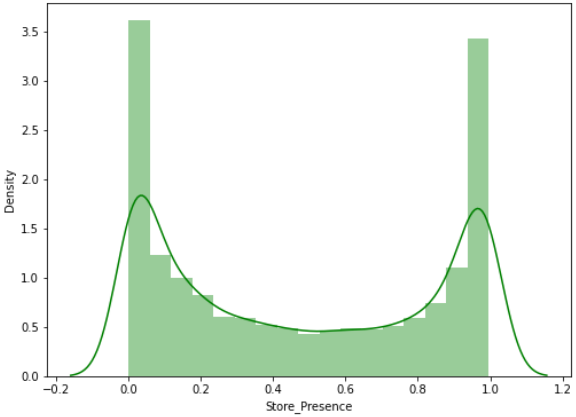
Category_2



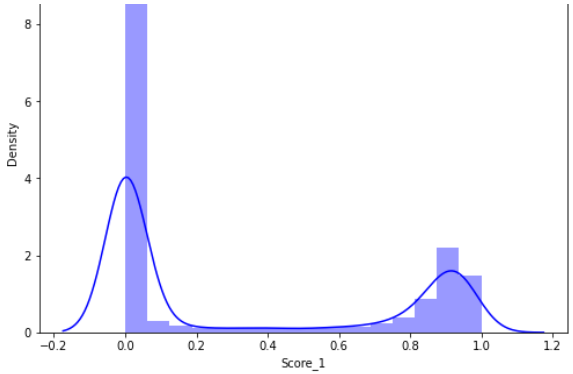


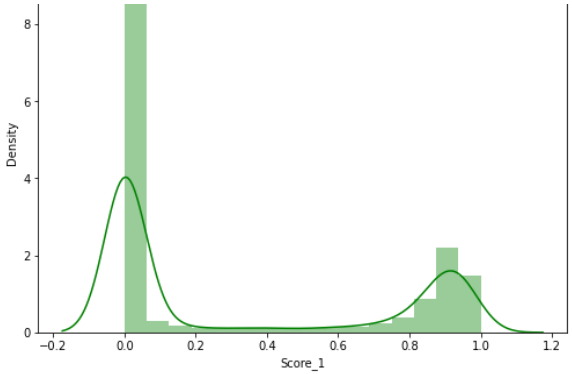
Store_Presence



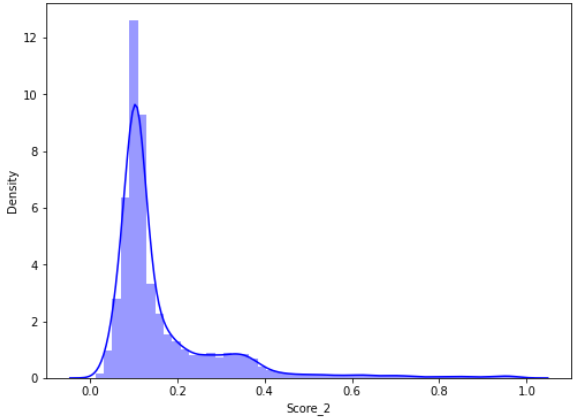


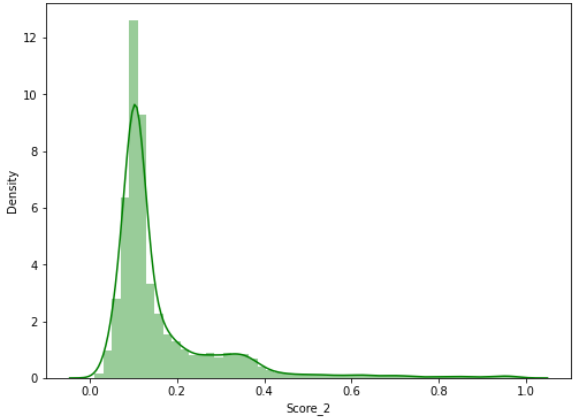
Score_1



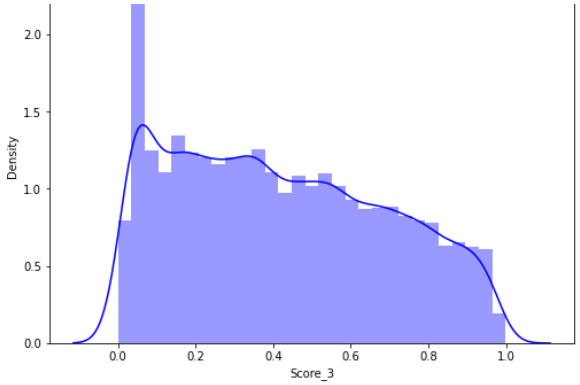


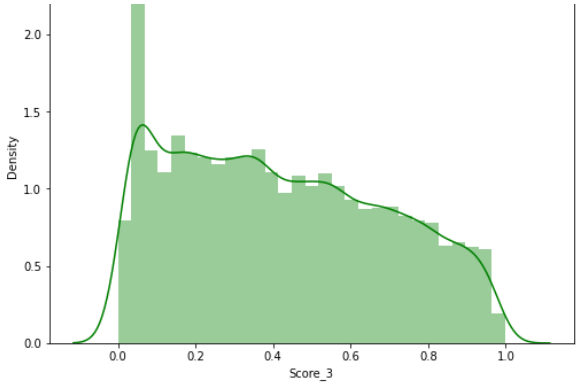
Score_2



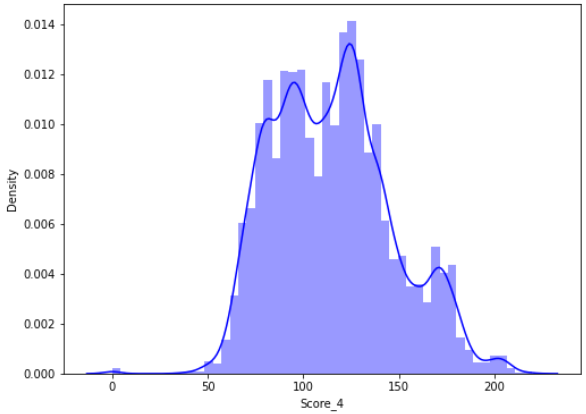


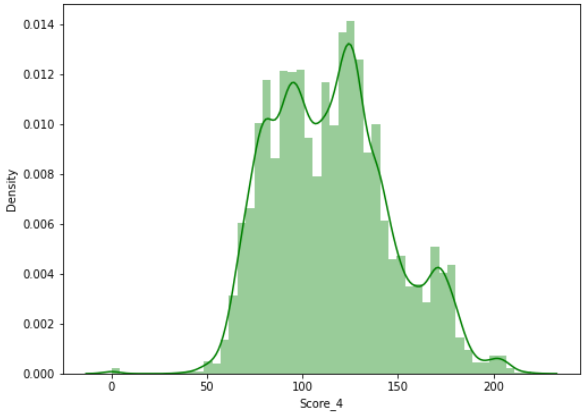
Score_3



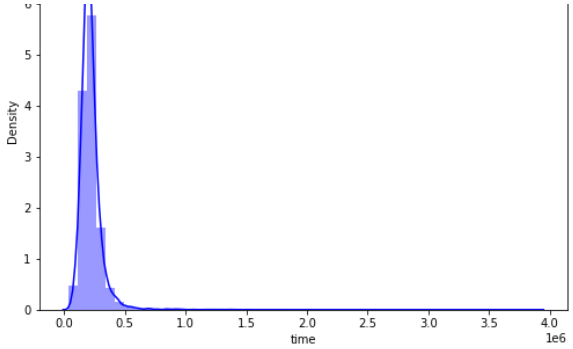


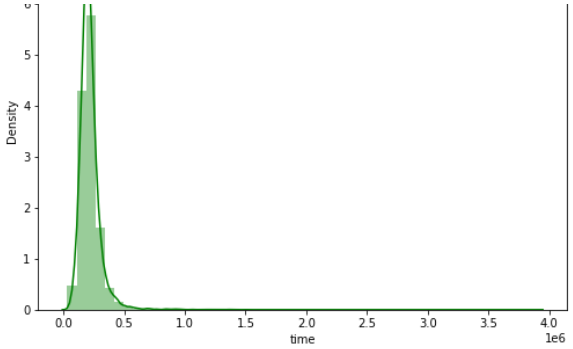
Score_4



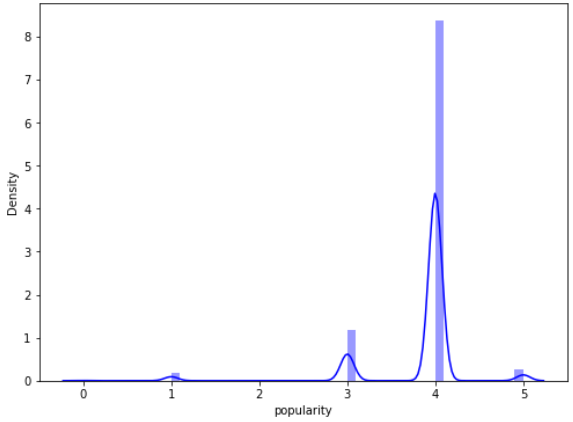


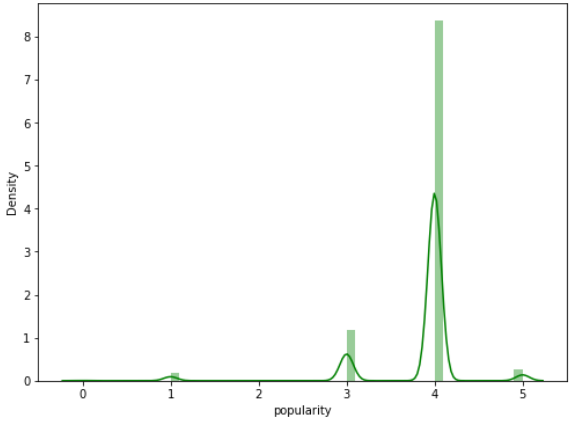
time





popularity





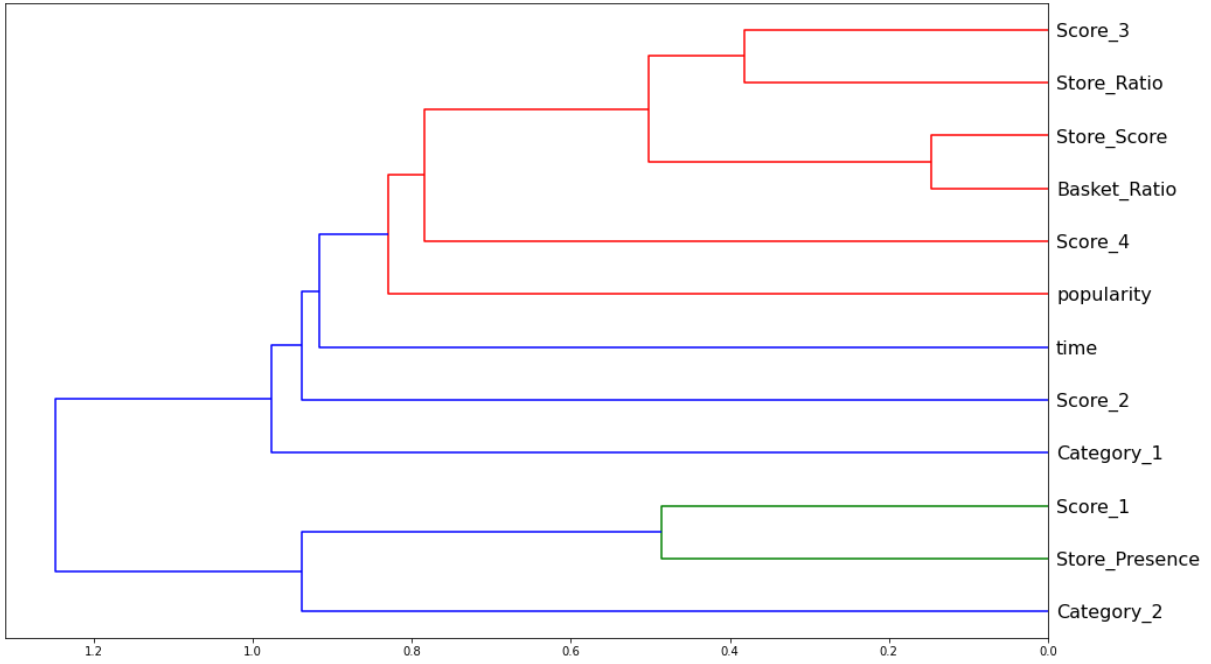
OBSERVATIONS

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
```



strongly correlated variables

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

Boxen plot

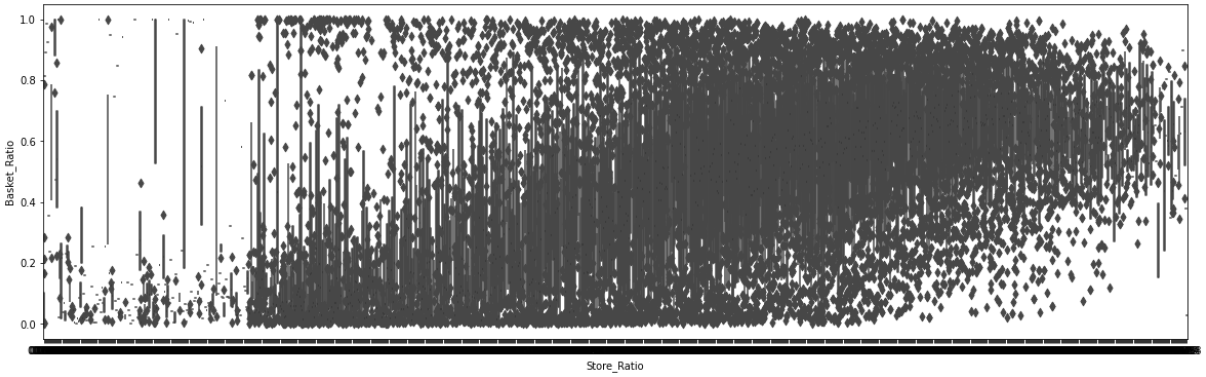
The boxen plot, otherwise known as a Letter-value plot, is a box plot meant for large data sets (n > 10,000).

The Boxen plot is very similar to box plot, except for the fact that it plots different quartile values.

By plotting different quartile values, we are able to understand the shape of the distribution particularly in the head end and tail end.

In [ ]:
```
#Perform boxen plot between Store ratio and Basket ratio
```

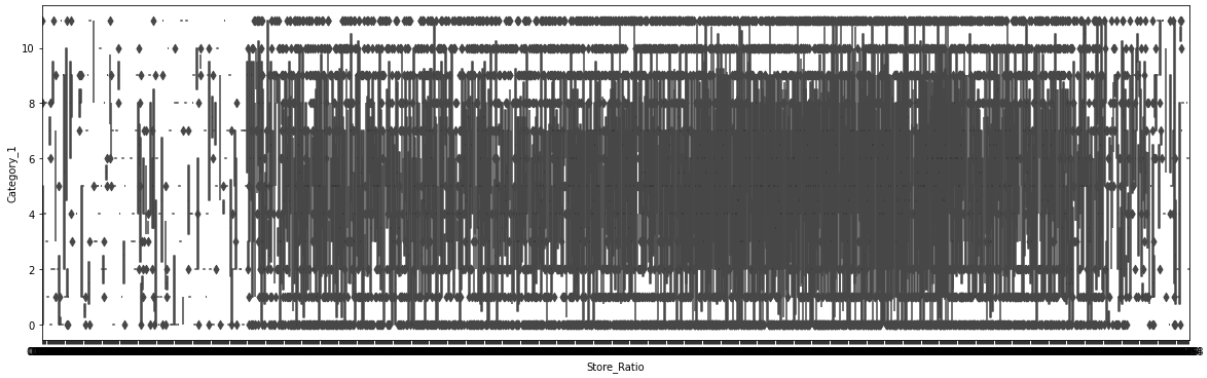
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09ee5b6290>
```



there is positive correlation betwwen store ratio and basket ratio.

In [ ]:
```
#Perform boxen plot between Store ratio and Category 1
```

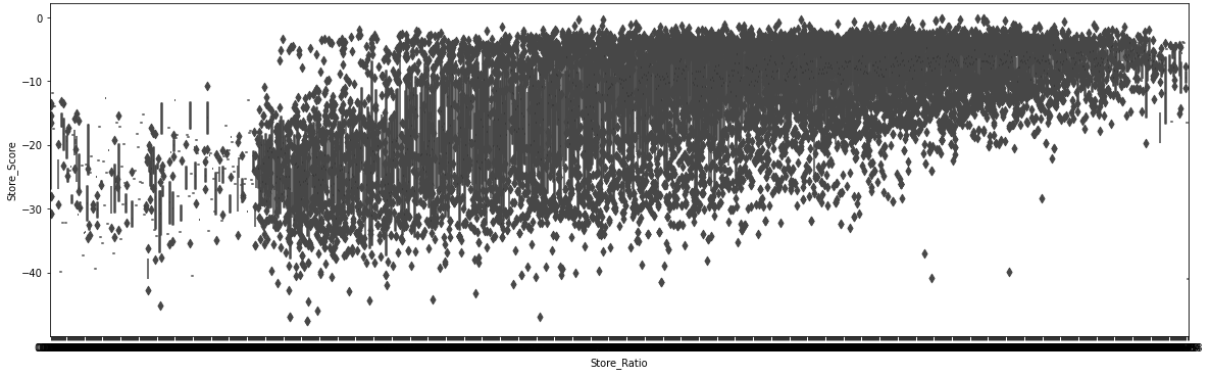
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09ee123f10>
```



there is no relation between store ratio and category 1

In [ ]:
```
#Perform boxen plot between Store ratio and Store Score
```

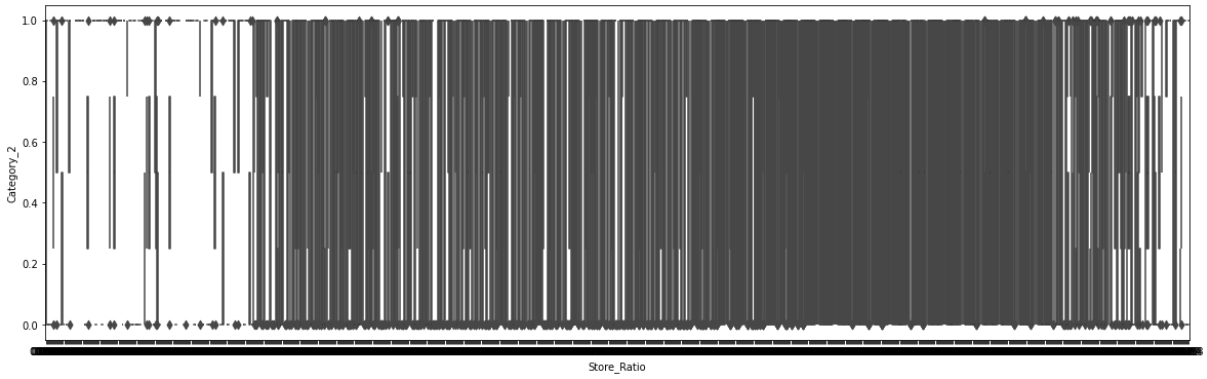
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09e83f1dd0>
```



there is positive correlation between store ratio and store score.

In [ ]:
```
#Perform boxen plot between Store ratio and Category 2
```

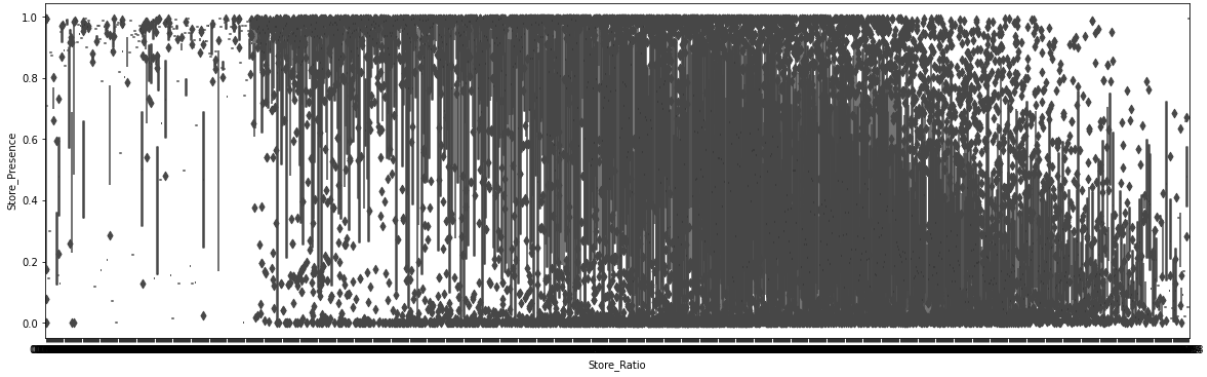
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09e60f9dd0>
```



there is weak correlation between store ratio and category 2

In [ ]:
```
#Perform boxen plot between Store ratio and Store presence
```

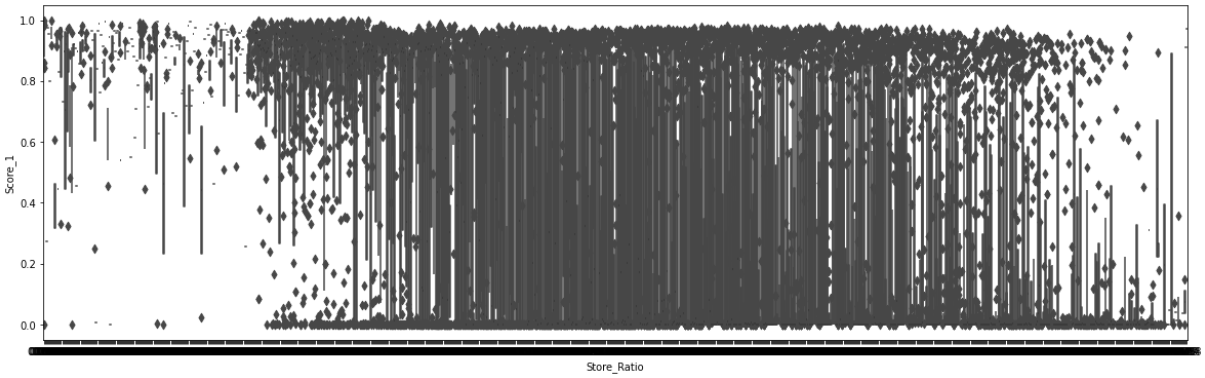
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09e2fb77d0>
```



there is strong negative correlation between store presence and store ratio

In [ ]:
```
#Perform boxen plot between Store ratio and Score 1
```

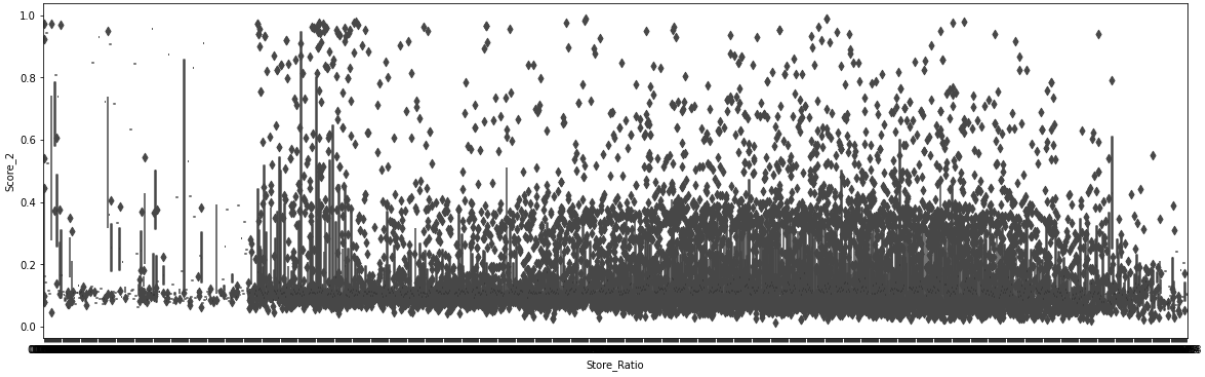
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09e0507650>
```



there is weak correlation between store ratio and score1

In [ ]:
```
#Perform boxen plot between Store ratio and Score 2
```

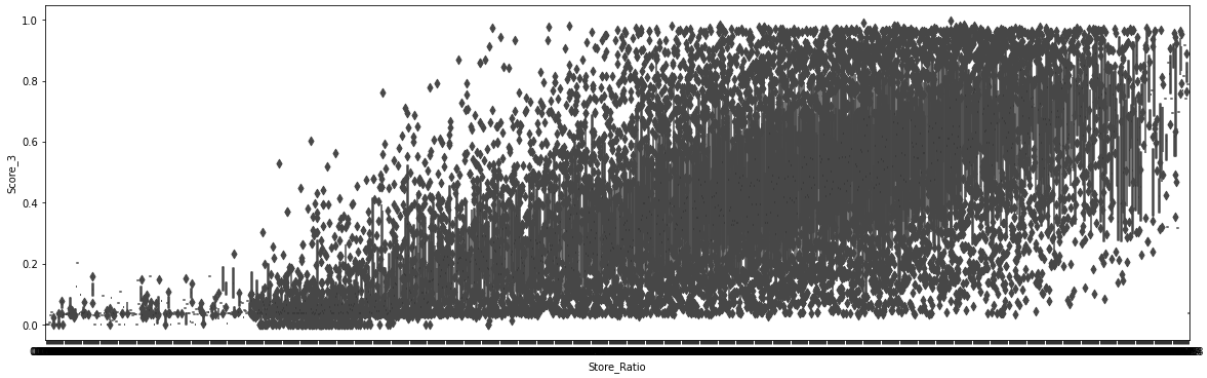
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09dfb75210>
```



there is no relation between store ratio and score 2

In [ ]:
```
#Perform boxen plot between Store ratio and Score 3
```

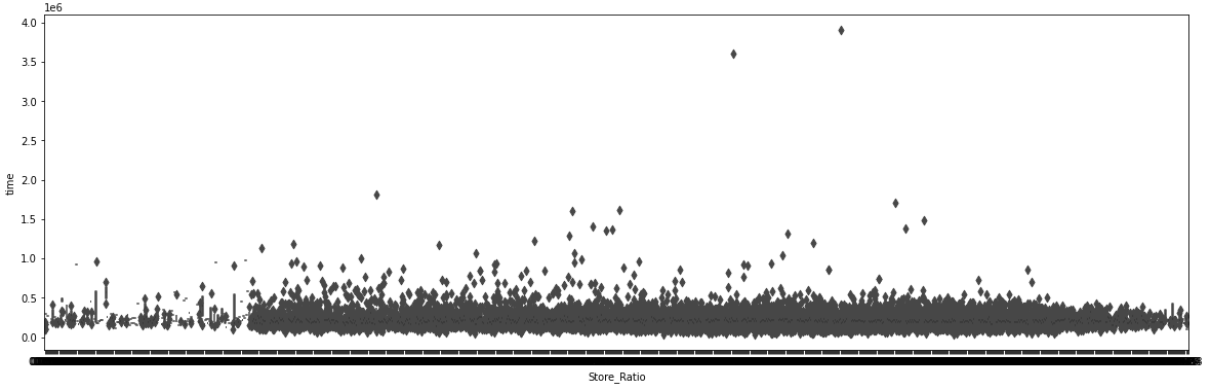
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09dab32e50>
```



there is positive correlation with store ratio and score 3

In [ ]:
```
#Perform boxen plot between Store ratio and time
```

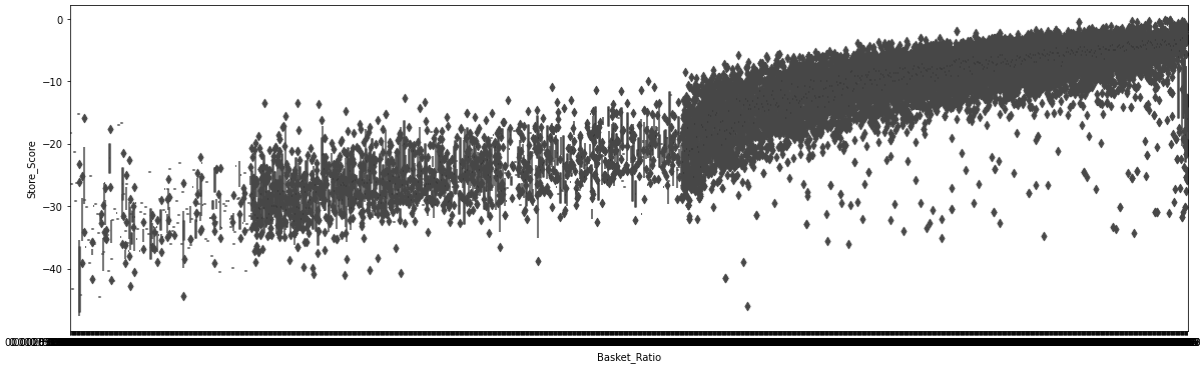
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09d4d00e50>
```



there is no relation between store ratio and time

In [ ]:
```
#Perform boxen plot between Basket Ratio and Score Score
```

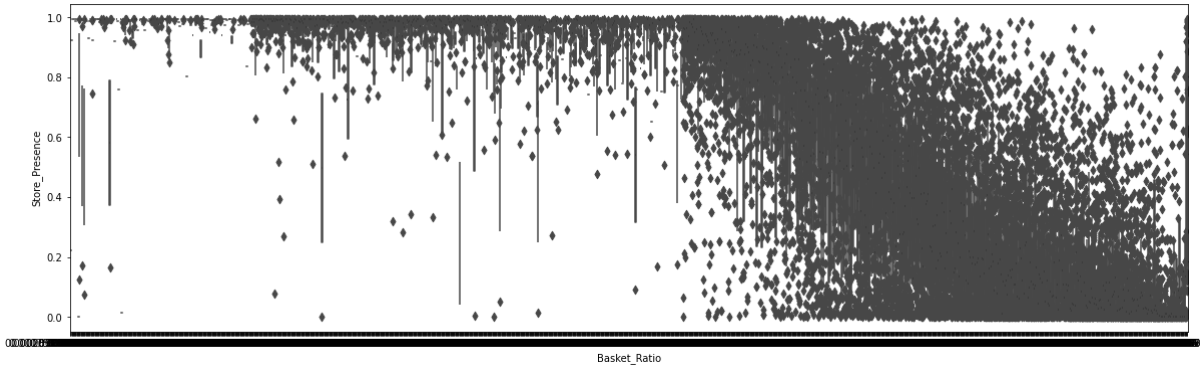
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f09ca826410>
```



there is a positive correlation between store score and basket ratio

In [ ]:
```
#Perform boxen plot between Basket Ratio and Store presence
```

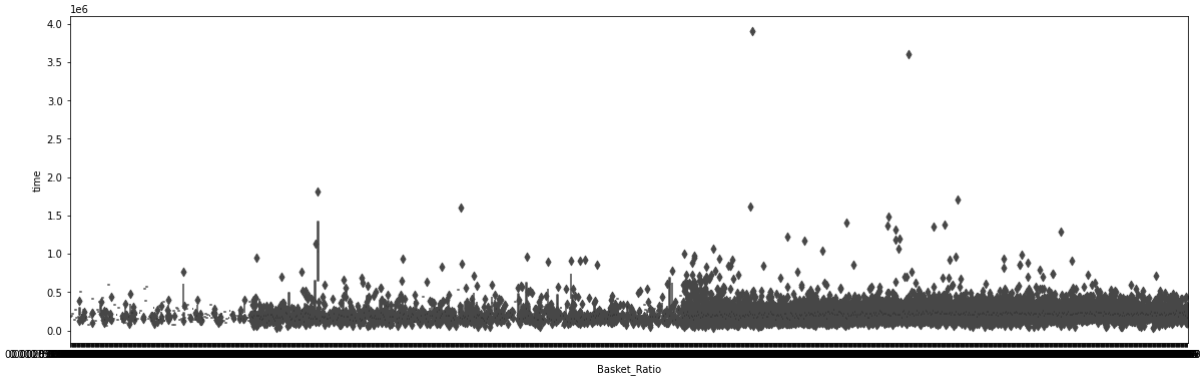
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35181fc650>
```



after a certain point as the value ok basket ratio is increasing value of store presence is decreasing

In [ ]:
```
#Perform boxen plot between Basket ratio and time
```

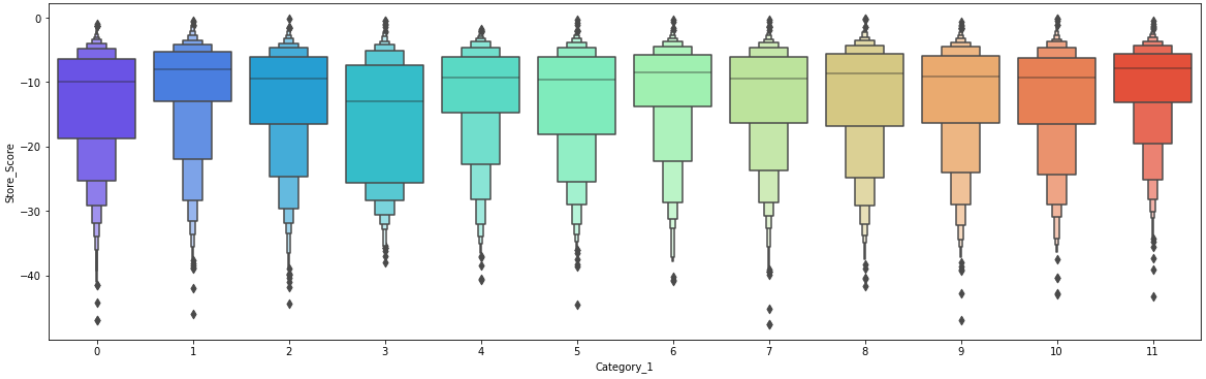
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3506658bd0>
```



there is no relation between time and basker ratio

In [ ]:
```
#Perform boxen plot between Category 1 and Store score
```

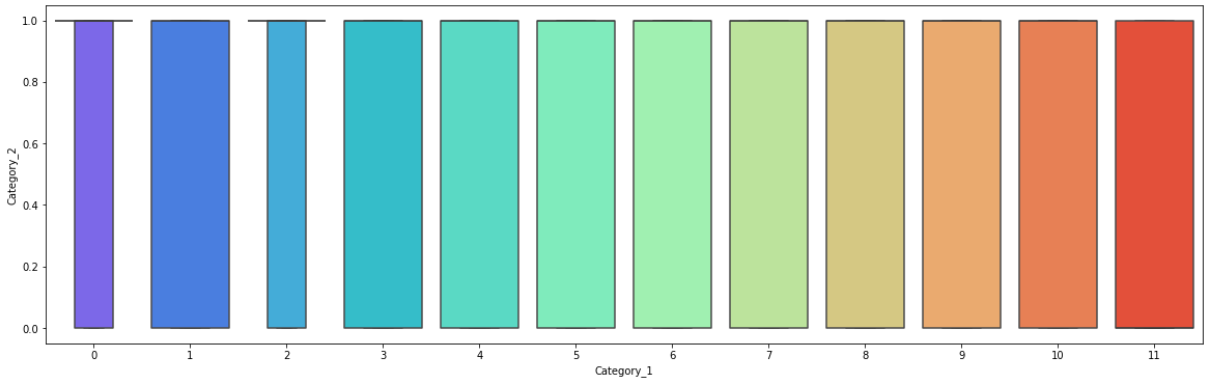
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f1876310>
```



there is no relation between between store score and category1

In [ ]:
```
#Perform boxen plot between Category 1 and Category 2
```

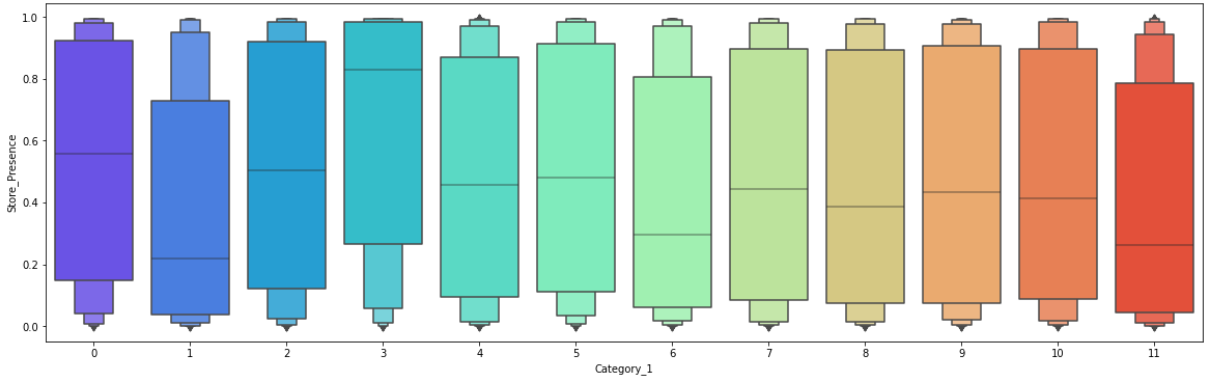
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f1366490>
```



there is weak relation between category1 and category2

In [ ]:
```
#Perform boxen plot between Category 1 and Store presence
```

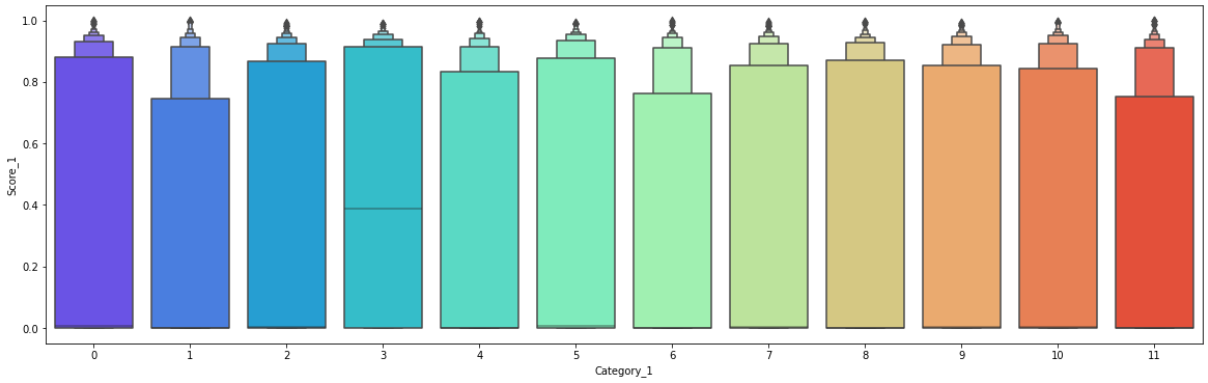
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f19c28d0>
```



there is weak relation between category1 and store presence

In [ ]:
```
#Perform boxen plot between Category 1 and Score 1
```

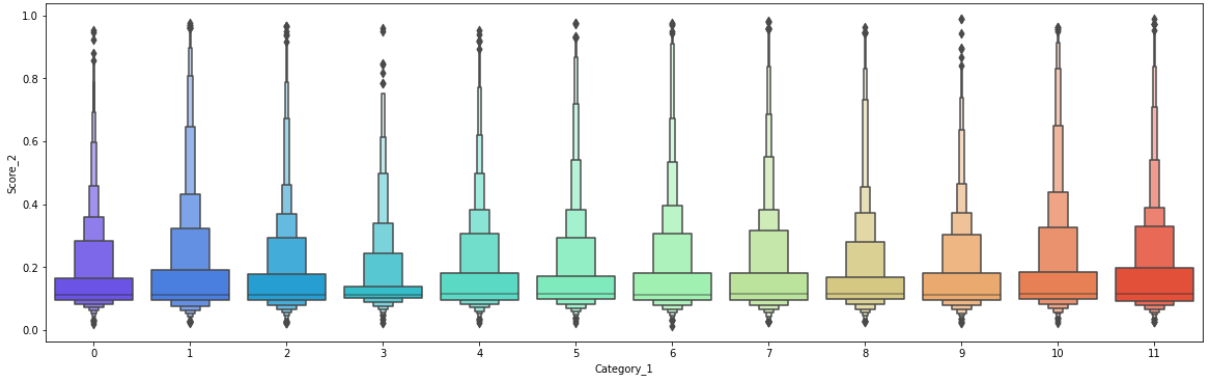
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f19aec50>
```



there is weak relation between category1 and score1

In [ ]:
```
#Perform boxen plot between Category 1 and Score 2
```

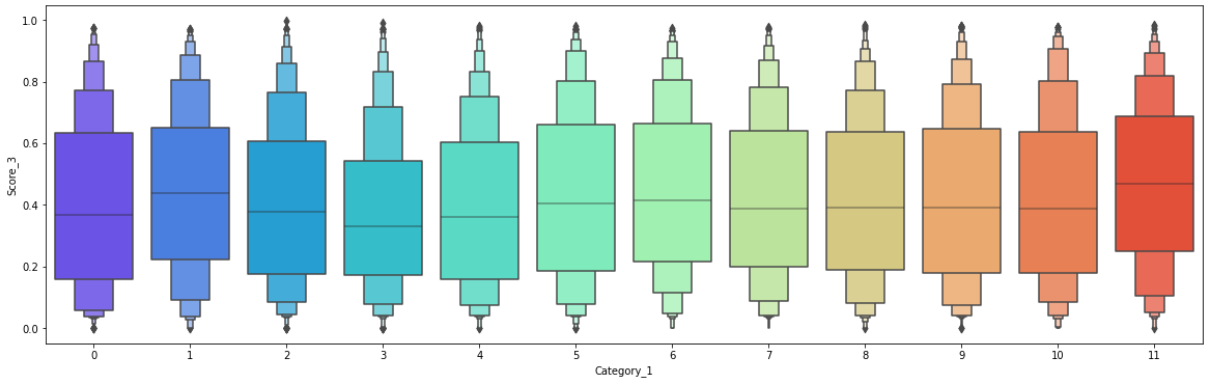
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f168f150>
```



there is no relation between category1 and score2

In [ ]:
```
#Perform boxen plot between Category 1 and Score 3
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f18a7710>
```



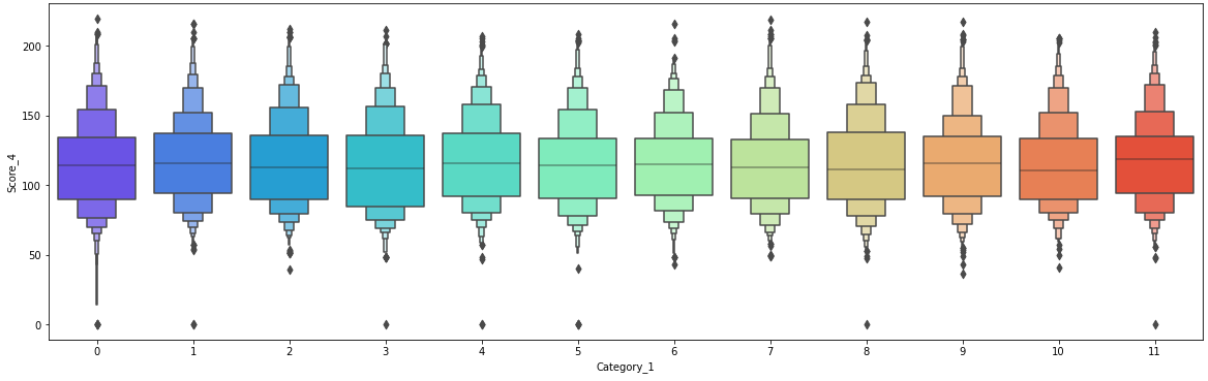
from above boxen plot:

The distribution between lower adjacent value and upper adjacent value is symmetrical.

there is no relation between category1 and score3

In [ ]:
```
#Perform boxen plot between Category 1 and Score 4
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f1afc690>
```

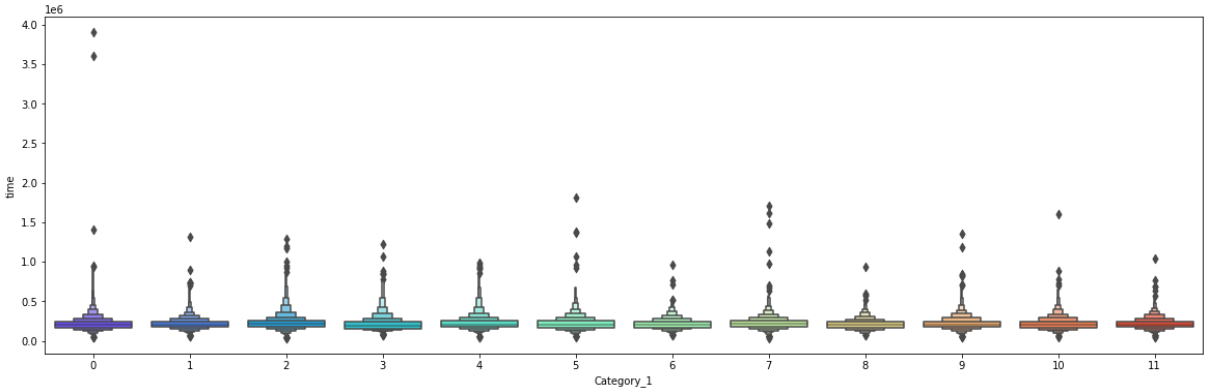


from above boxen plot:

The distribution between lower adjacent value and upper adjacent value is symmetrical.

In [ ]:
```
#Perform boxen plot between Category 1 and time
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f163ebd0>
```

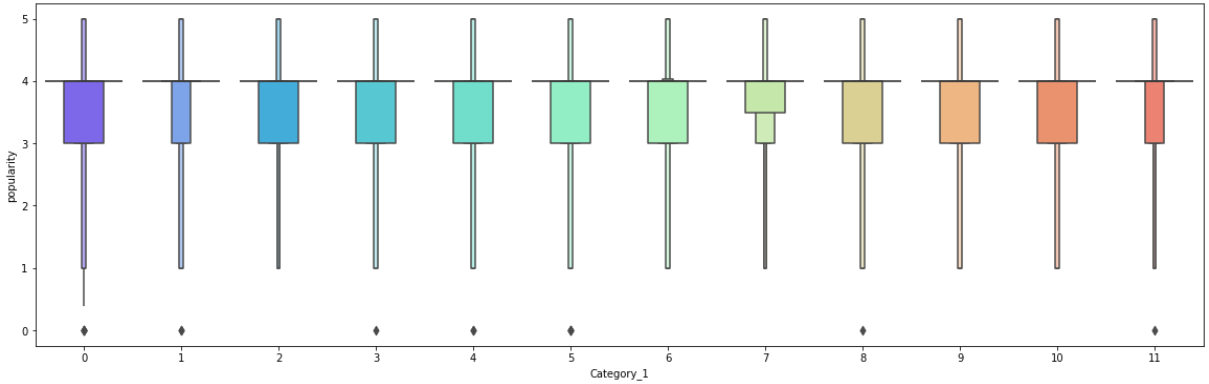


from above boxen plot:

There is no relation between time and category1

In [ ]:
```
#Perform boxen plot between Category 1 and popularity
```

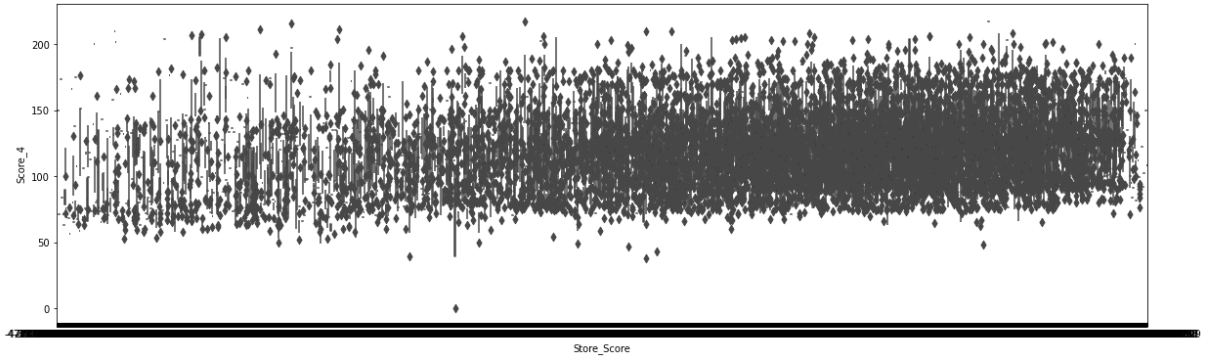
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f34f14ce710>
```



there is no relation between category1 and popularity

In [ ]:
```
#Perform boxen plot between Store score and Score 4
```

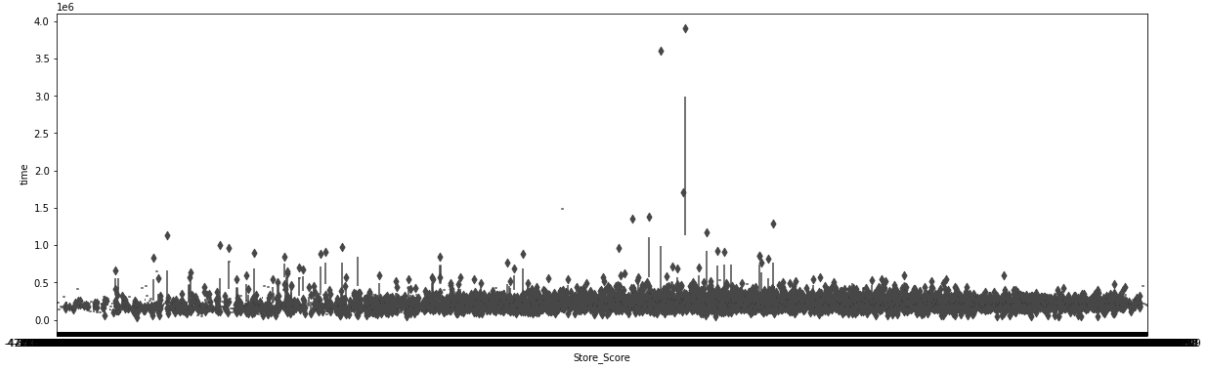
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8d55a5e10>
```



there is weak relation between store score and score2

In [ ]:
```
#Perform boxen plot between Store score and time
```

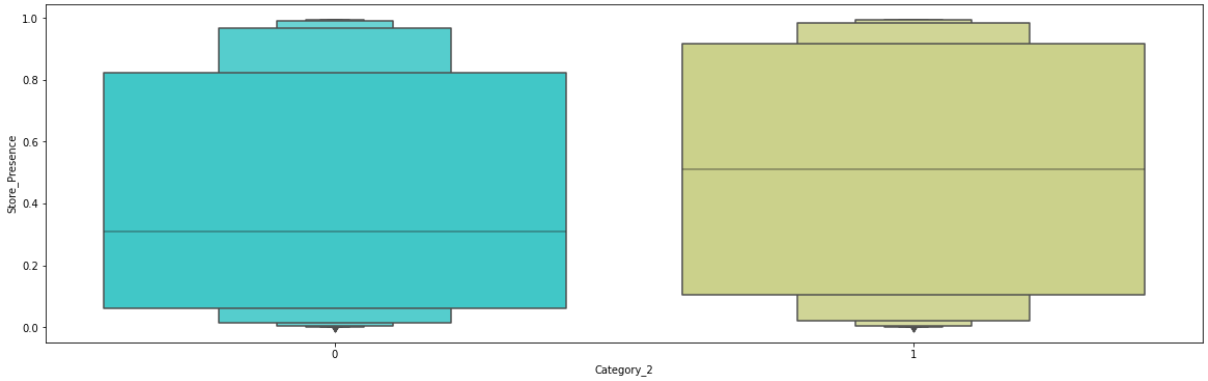
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe90f712610>
```



there is no relation between time and store score

In [ ]:
```
#Perform boxen plot between Store presence and category2
```

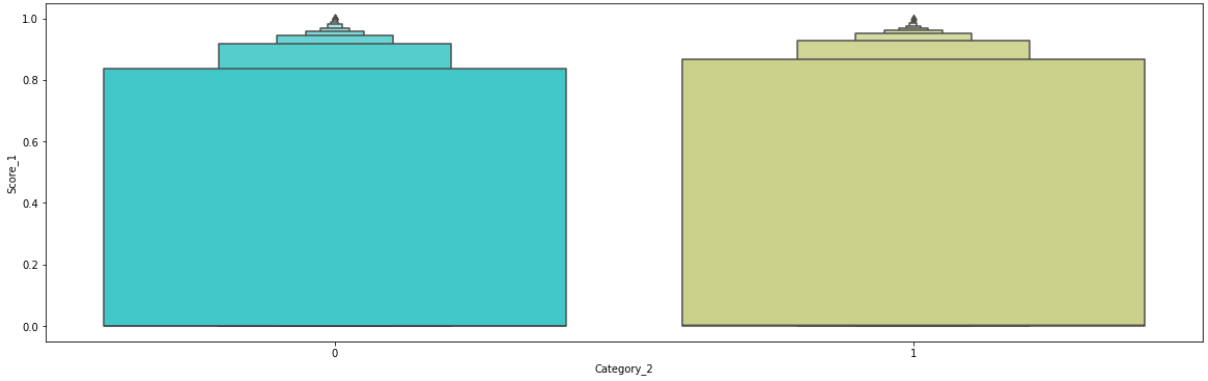
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fe2d2c10>



there is no relation between store presence and category2

In [ ]:
```
#Perform boxen plot between score1  and categort2
```

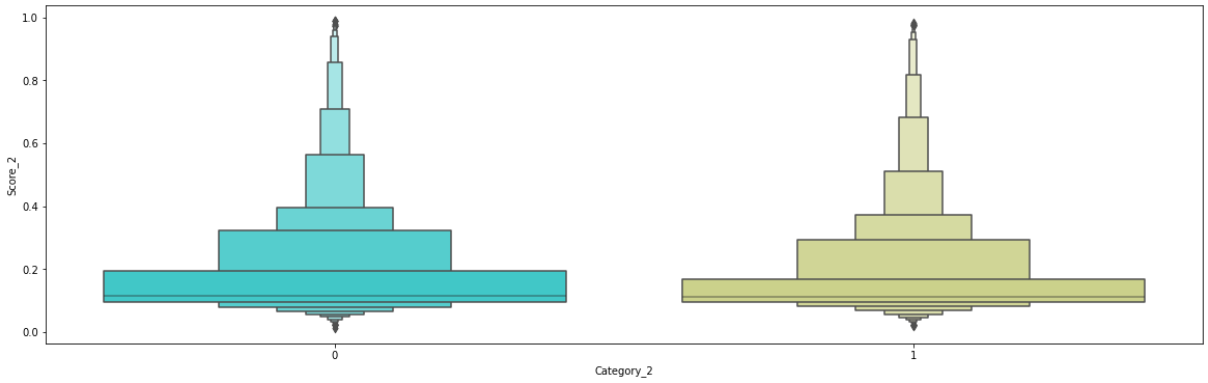
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fe3e0850>



there is no relation between score1 and category2

In [ ]:
```
#Perform boxen plot between category2  and score2
```

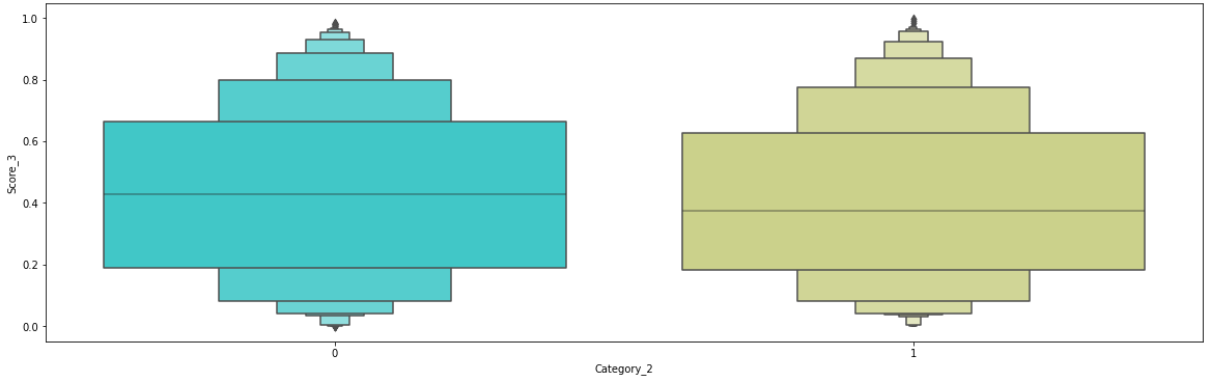
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fe2a5b90>



there is no relation between score2 and category2

In [ ]:
```
#Perform boxen plot between category2  and score3
```

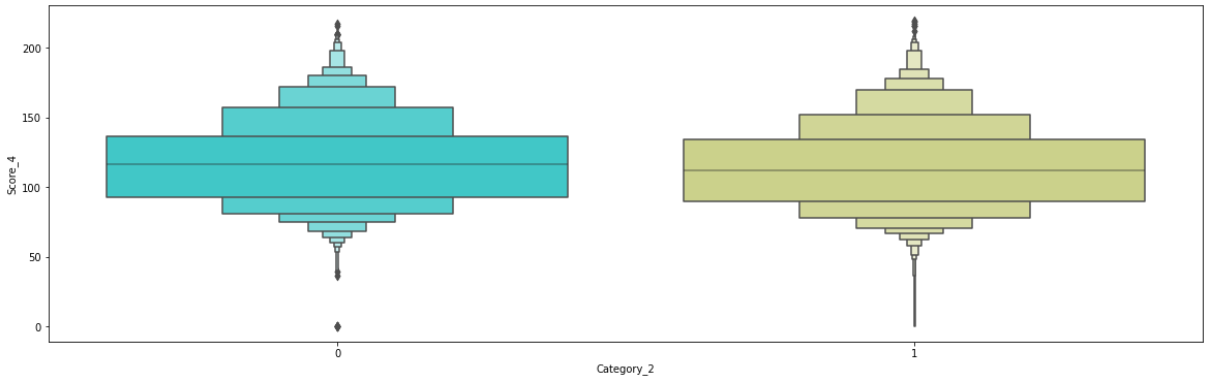
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8fe2b33d0>
```



there is no relation between score3 and category2

In [ ]:
```
#Perform boxen plot between category2  and score4
```

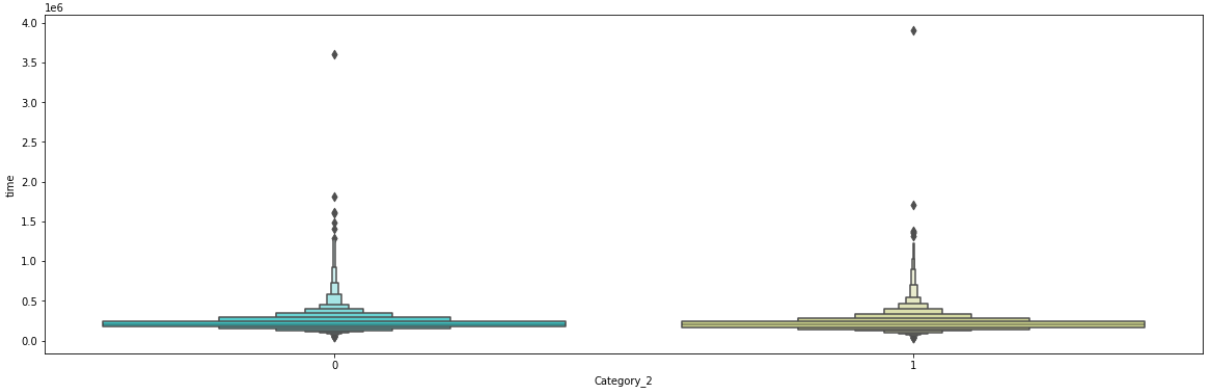
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8fe544a50>
```



there is no relation between score4 and category2

In [ ]:
```
#Perform boxen plot between category2  and time
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fdee0e90>



there is no relation between time and category2

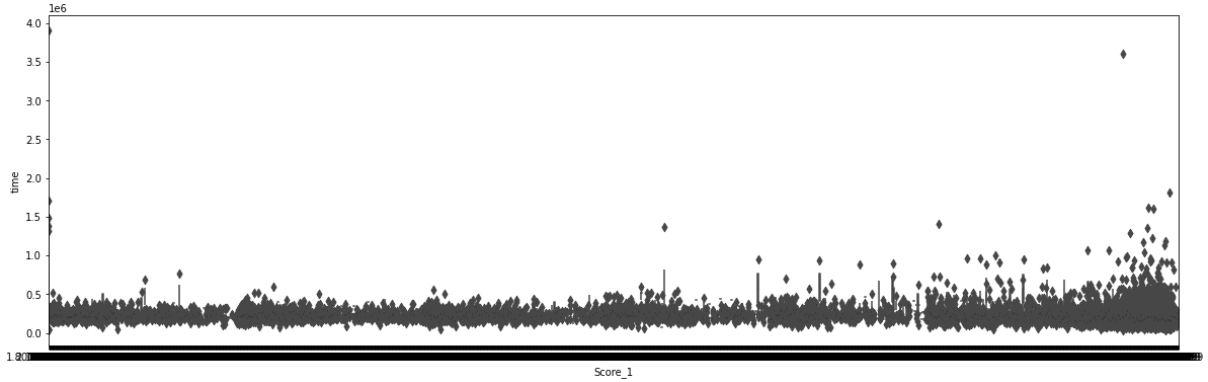
In [ ]:
```
#Perform boxen plot between category2  and popularity
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8fdfcff50>

there is no relation between popularity and category2

In [ ]:
```
#Perform boxen plot between time  and score1
```

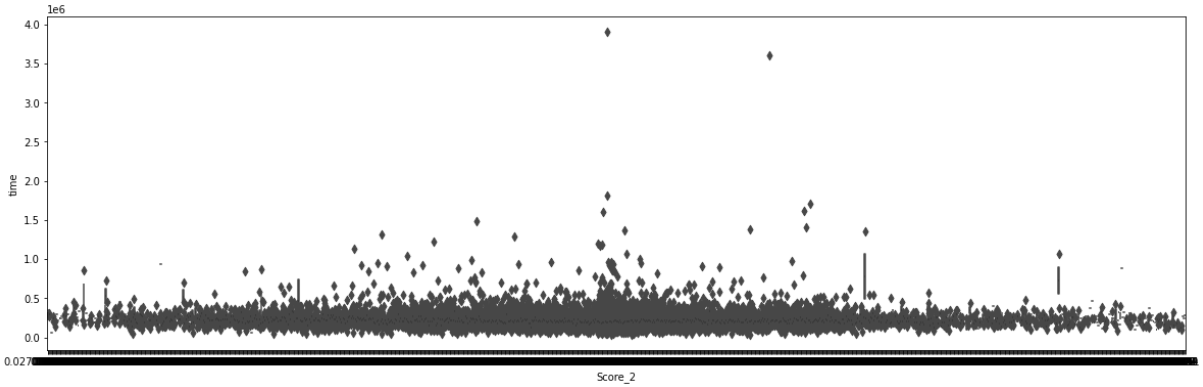
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe920edd250>
```



there is no relation between score1 and time

In [ ]:
```
#Perform boxen plot between time  and score2
```

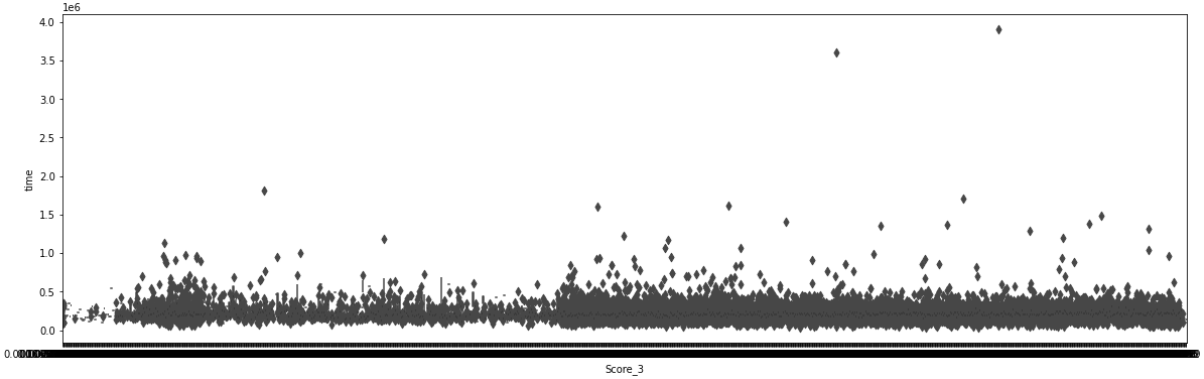
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe934014110>
```



there is no relation between score2 and time

In [ ]:
```
#Perform boxen plot between time  and score3
```

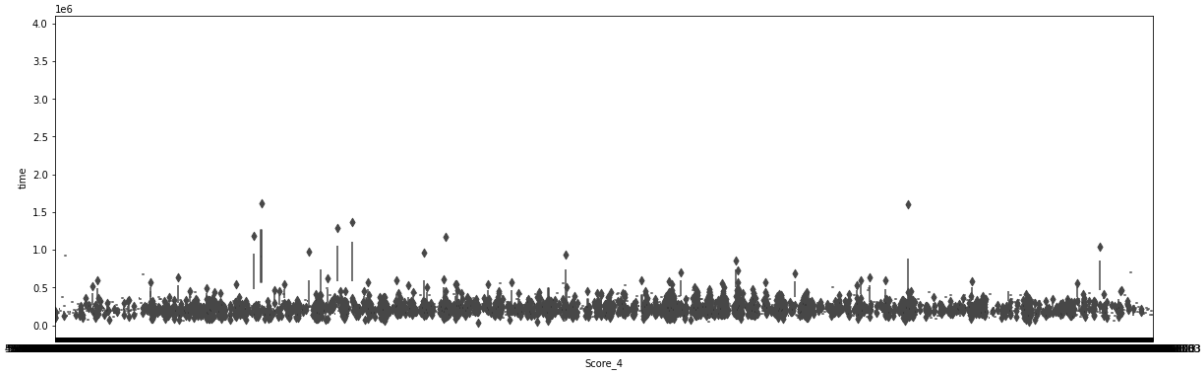
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe95ad8bdd0>
```



there is no relation between score3 and time

In [ ]:
```
#Perform boxen plot between time  and score4
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9861fde10>
```



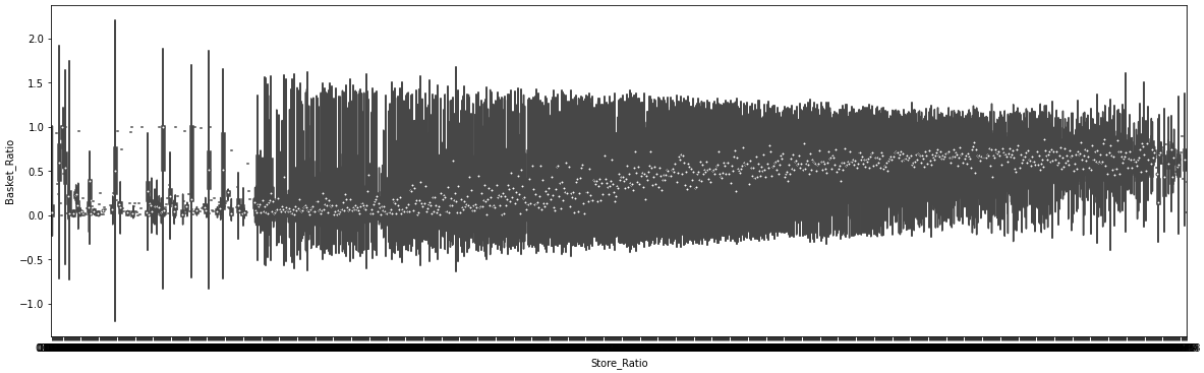
there is no relation between score4 and time

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 store ratio and basket ratio
```

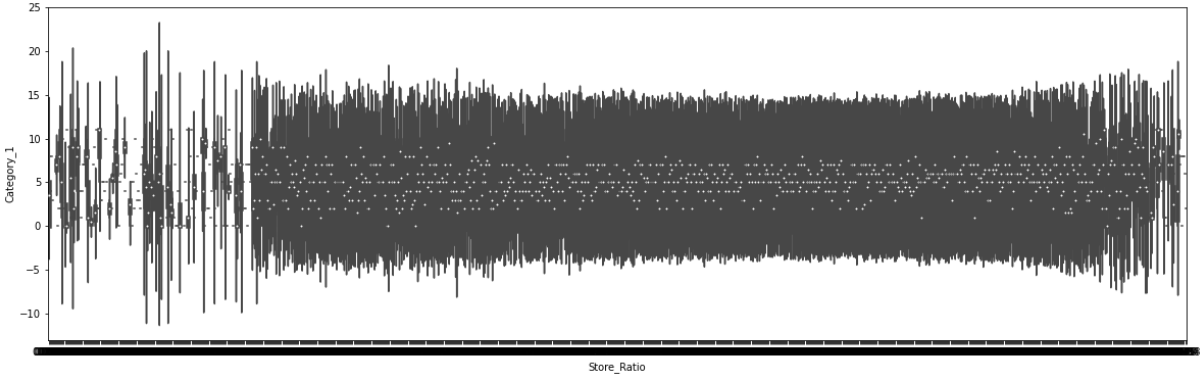
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2b99b3310>
```



there is a positive correlation between basket ratio and store ratio

In [ ]:
```
# perform a violin plot between store ratio and category1
```

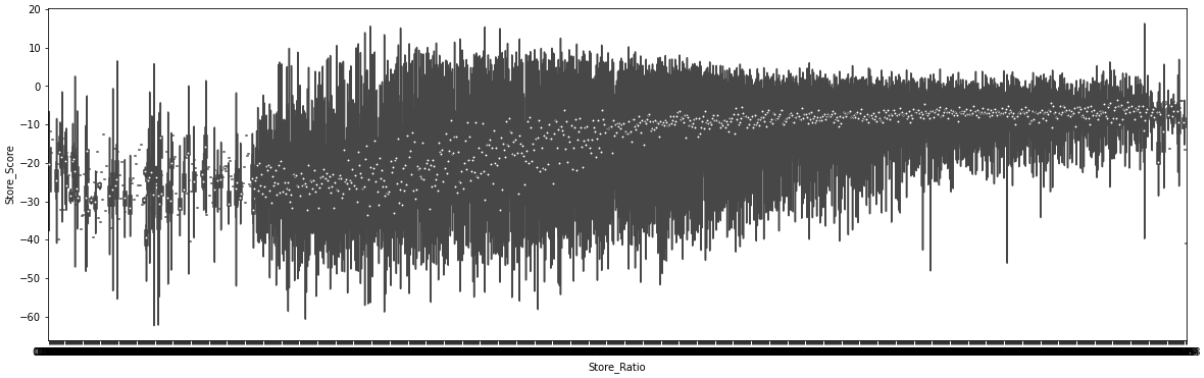
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2a13b7cd0>
```



there is a weak relation between store ratio and category1

In [ ]:
```
# perform a violin plot between store ratio and store presence
```

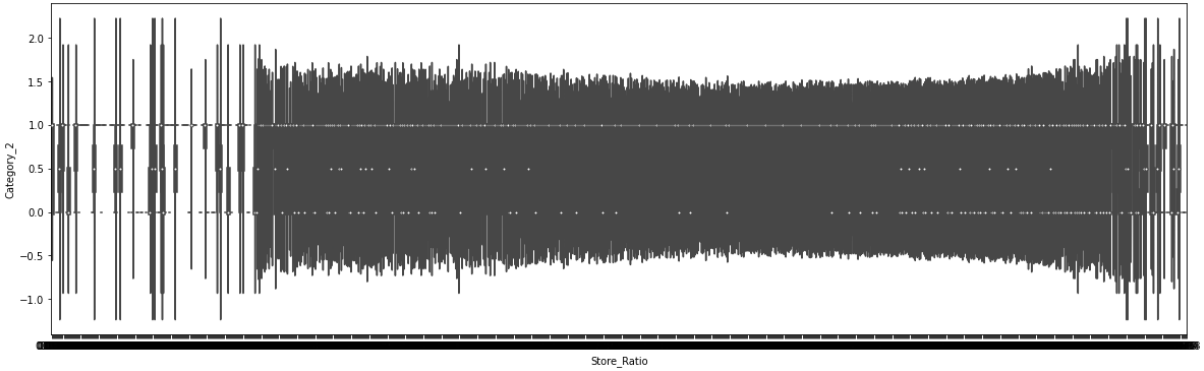
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc29d175510>
```



store score and store ratio are correlated to each other

In [ ]:
```
# perform a violin plot between store ratio and category2
```

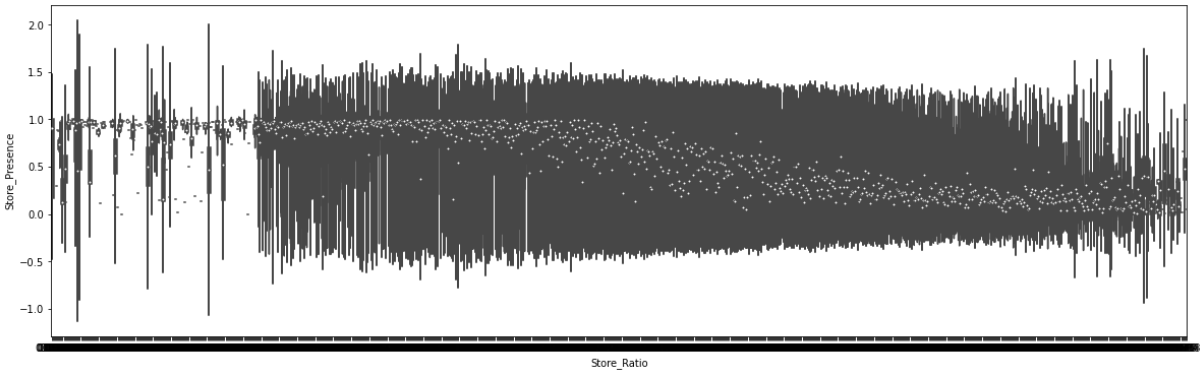
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc29d35d910>
```



there is a week correlation between store ratio and category2

In [ ]:
```
# perform a violin plot between store ratio and store presence
```

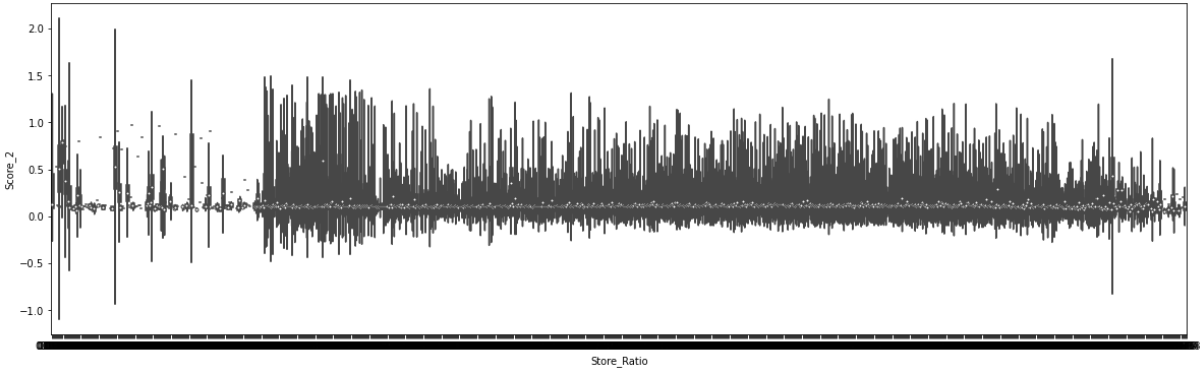
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc298124e90>
```



there is strong negative correlation between store ratio and store presence

In [ ]:
```
# perform a violin plot between store ratio and score2
```

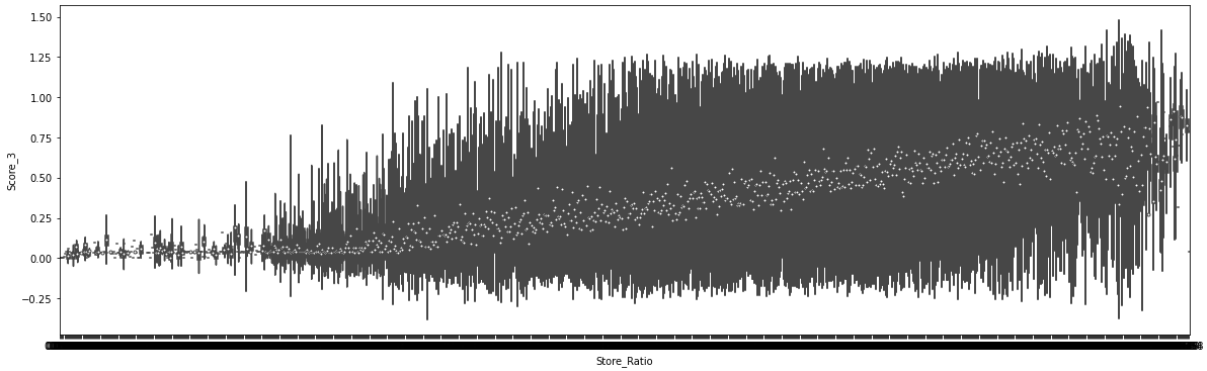
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc28d2bded0>
```



there is no relation between score2 and store ratio

In [ ]:
```
# perform a violin plot between store ratio and score3
```

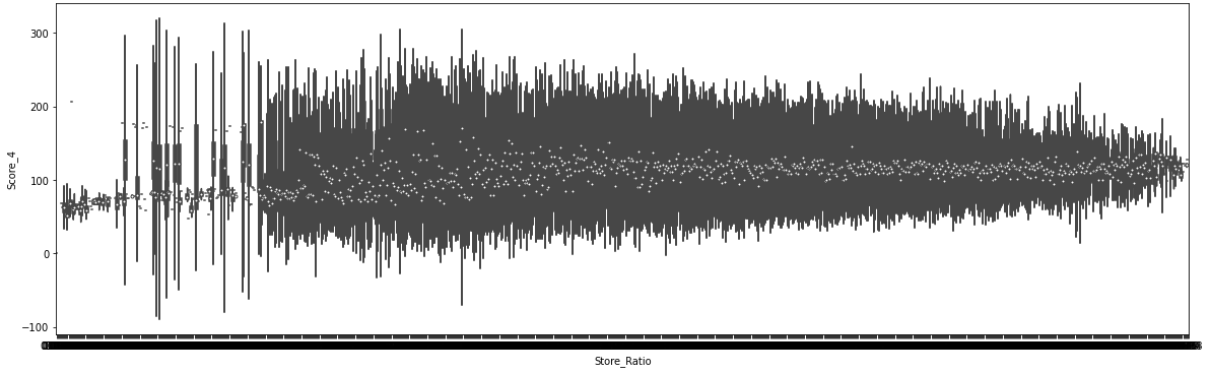
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc28991b090>
```



there is positive correlation between store ratio and score3

In [ ]:
```
# perform a violin plot between store ratio and score4
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2861f0e90>
```

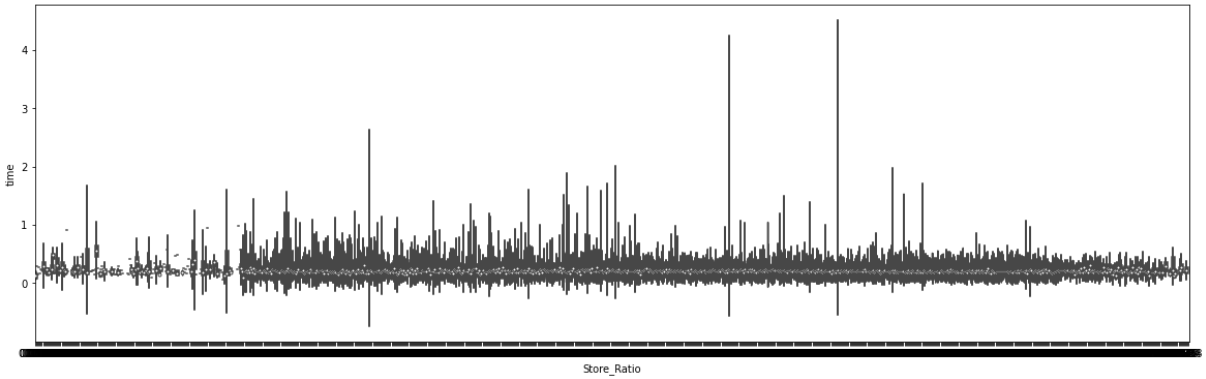


there is no relation between score4 and store ratio

In [ ]:
```
# perform a violin plot between store ratio and time
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2829bb0d0>
```

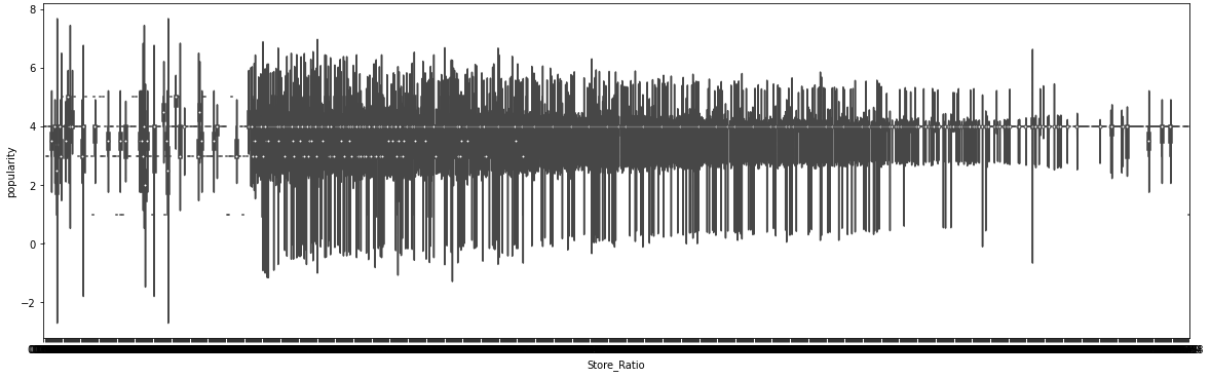
1e6



there is no relation between time and store ratio

In [ ]:
```
# perform a violin plot between store ratio and popularuty
```

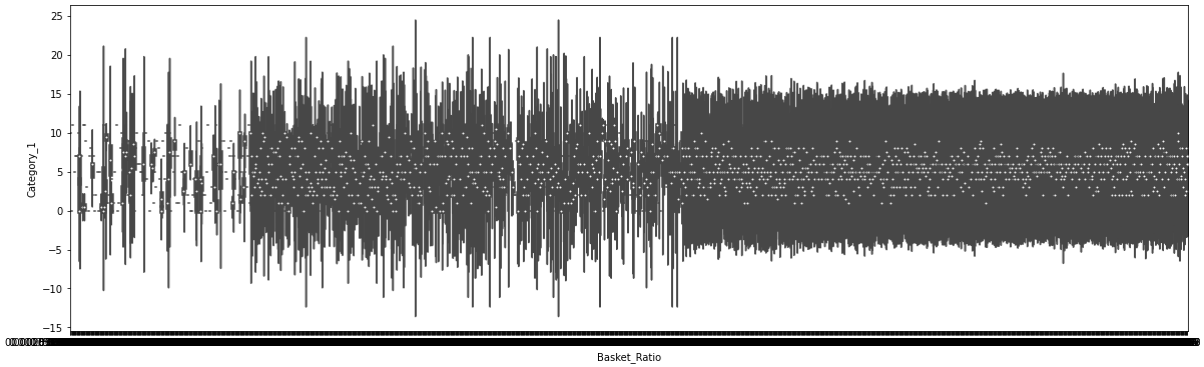
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc27edb4fd0>
```



there is weak relation between popularity and store ratio

In [ ]:
```
# perform a violin plot between basket ratio and category1
```

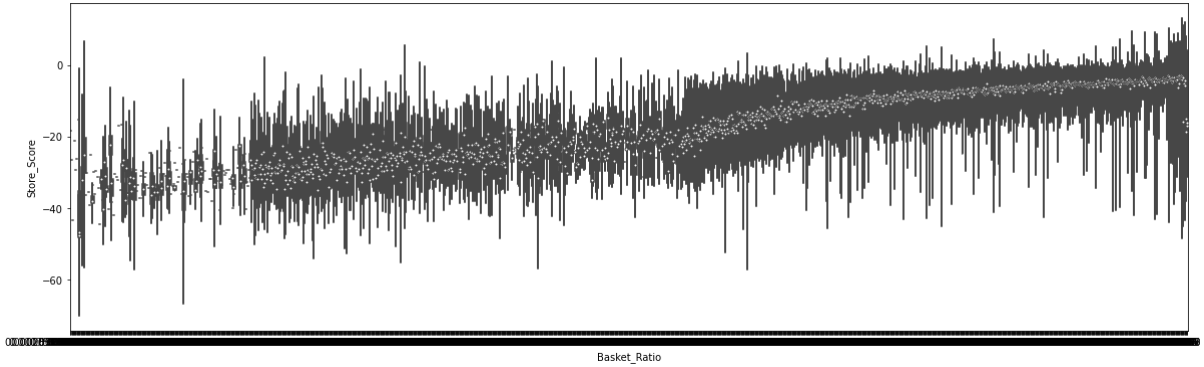
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc27bca8150>
```



there is no relation between category1 and basket ratio

In [ ]:
```
# perform a violin plot between basket ratio and store score
```

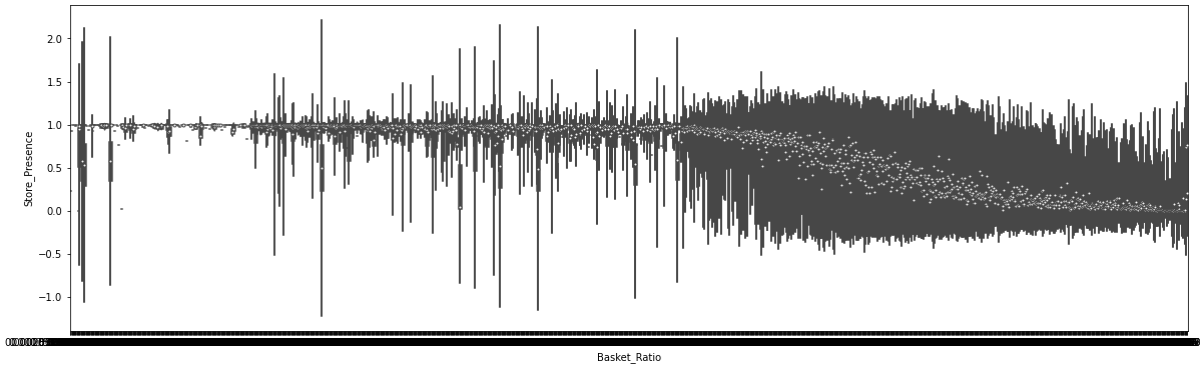
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2755af150>
```



there is positive correlation between basket ratio and store score

In [ ]:
```
# perform a violin plot between basket ratio and store presence
```

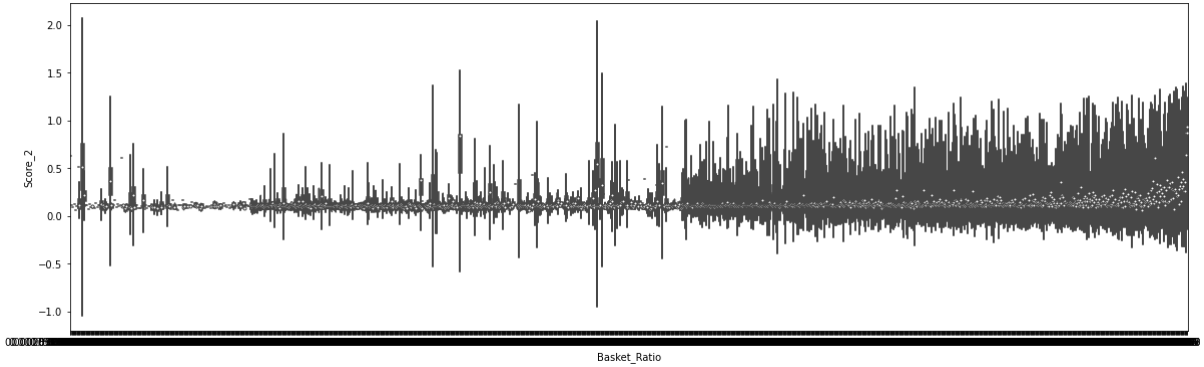
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc269aa7fd0>
```



there is weak relation between basket ratio and store presence

In [ ]:
```
# perform a violin plot between basket ratio and score2
```

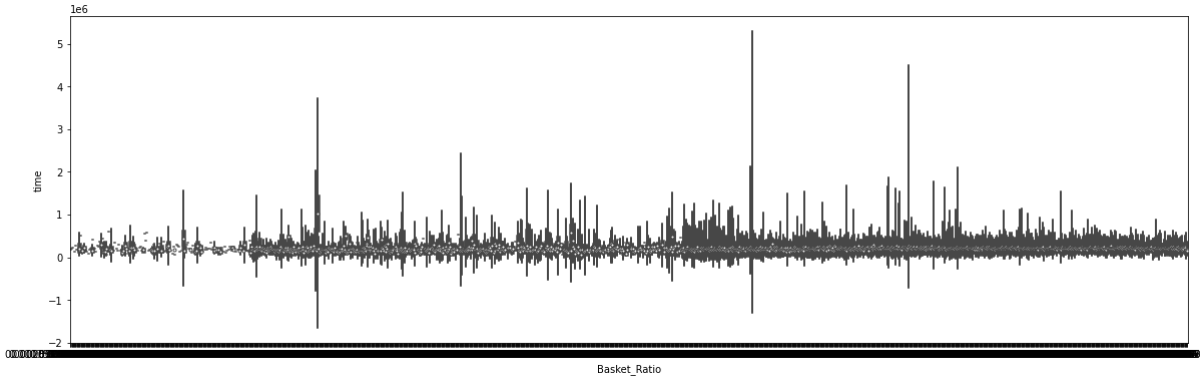
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc26312aad0>
```



there is very weak relation between score2 and basket ratio

In [ ]:
```
# perform a violin plot between time and basket ratio
```

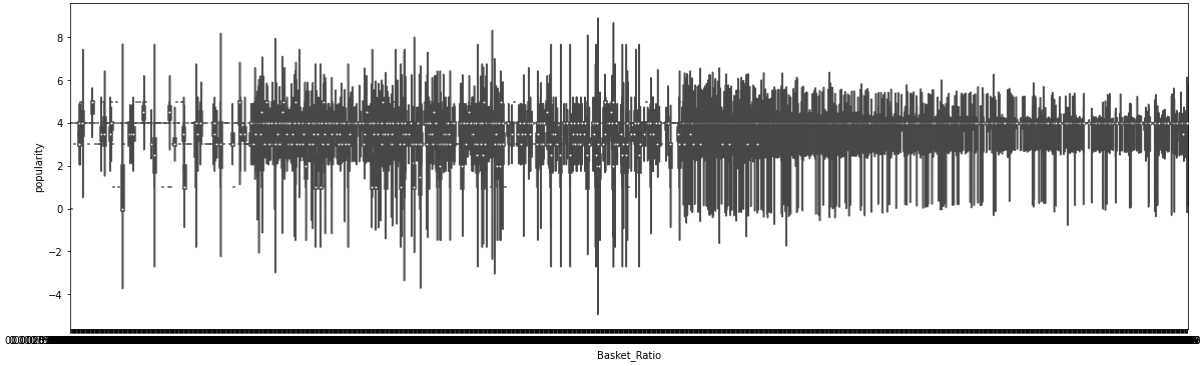
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2569f2c50>
```



there is no relation between time and basket ratio

In [ ]:
```
# perform a violin plot between basket ratio and popularity
```

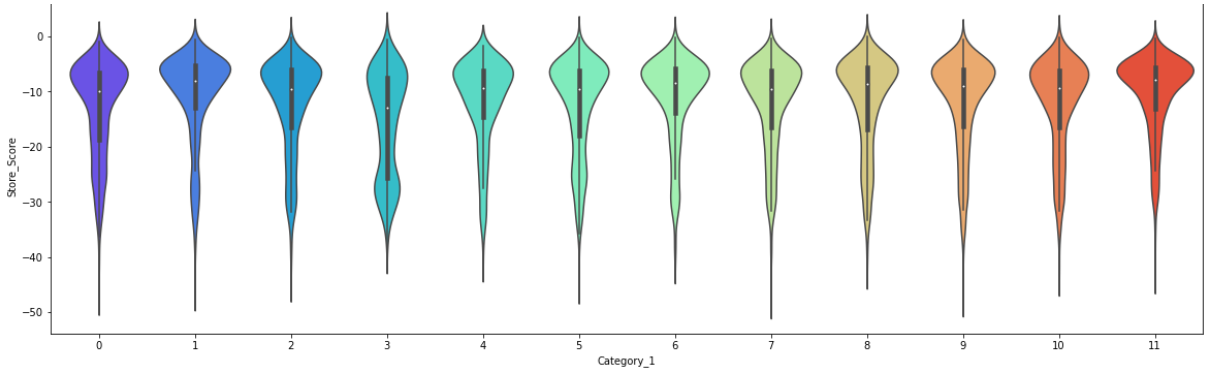
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc2433d6610>
```



there is weak relation between basket ratio and popularity

In [ ]:
```
# perform a violin plot between category1 and store score
```

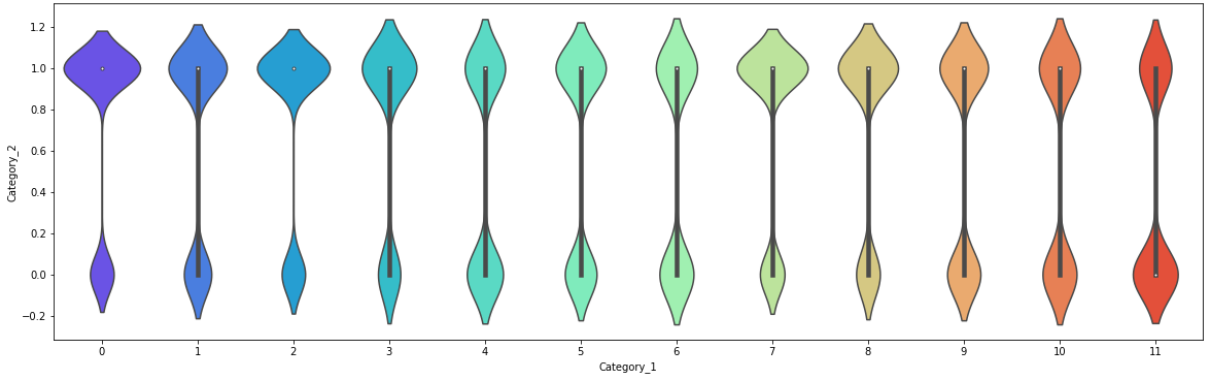
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d698d50>
```



there is no relation between store score and category1

In [ ]:
```
p# perform a violin plot between category2 and category1
```

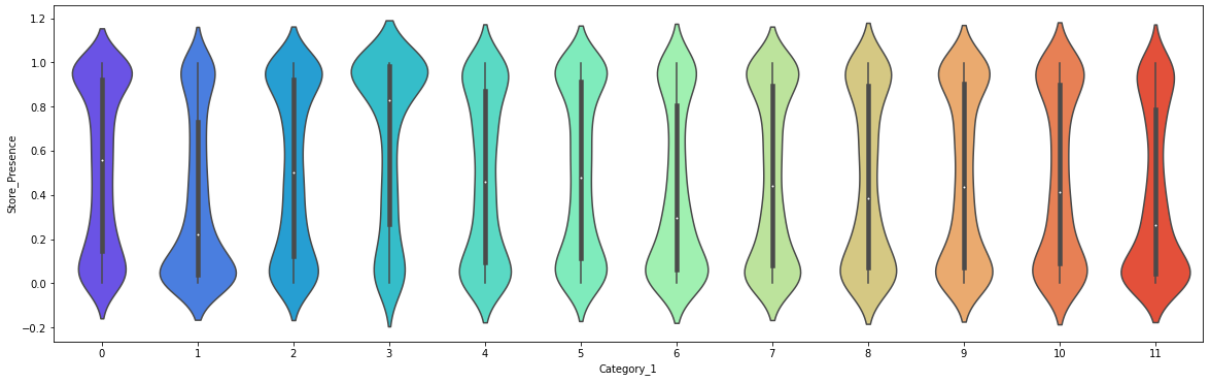
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d93c590>
```



there is no relation between category2 and category1

In [ ]:
```
# perform a violin plot between categort1 and store prsence
```

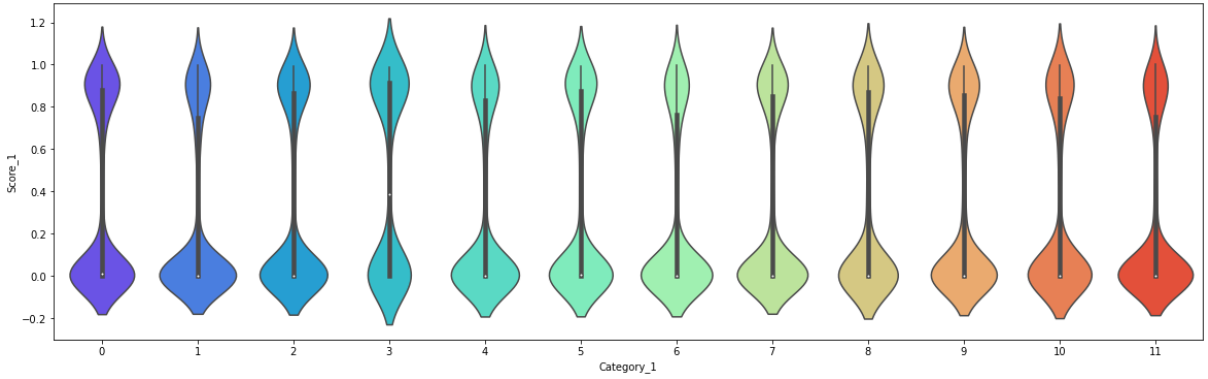
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d37f3d0>
```



there is no relation between store presence and category1

In [ ]:
```
# perform a violin plot between category1 and score1
```

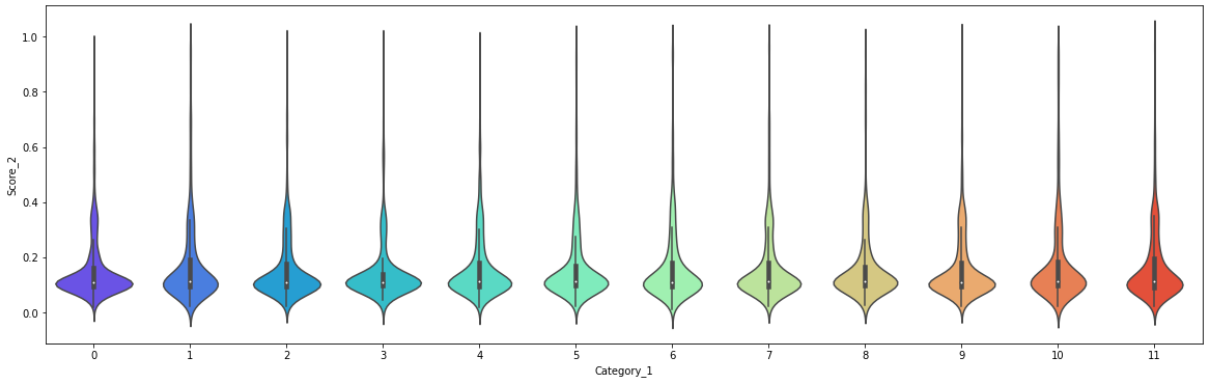
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d482ad0>
```



there is no relation between score1 and category1

In [ ]:
```
# perform a violin plot between category2 and score2
```

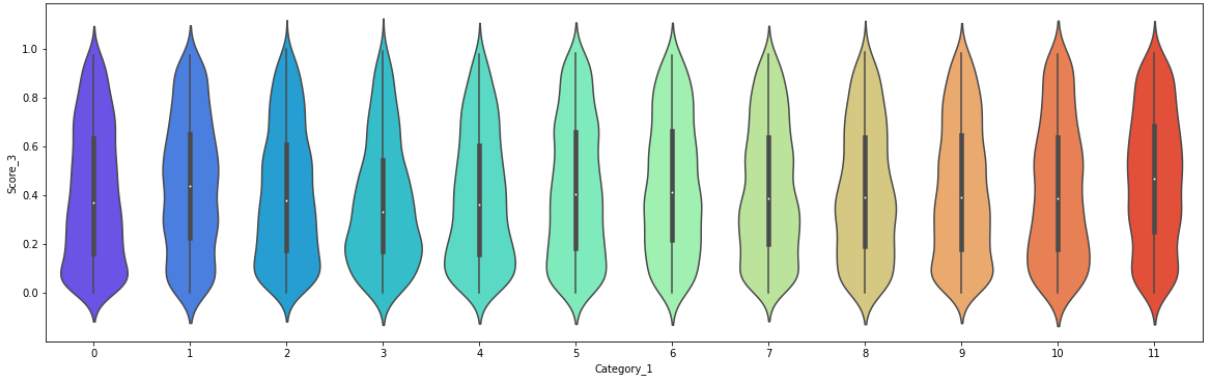
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d482750>
```



there is no relation between score2 and category1

In [ ]:
```
# perform a violin plot between category1 and score3
```

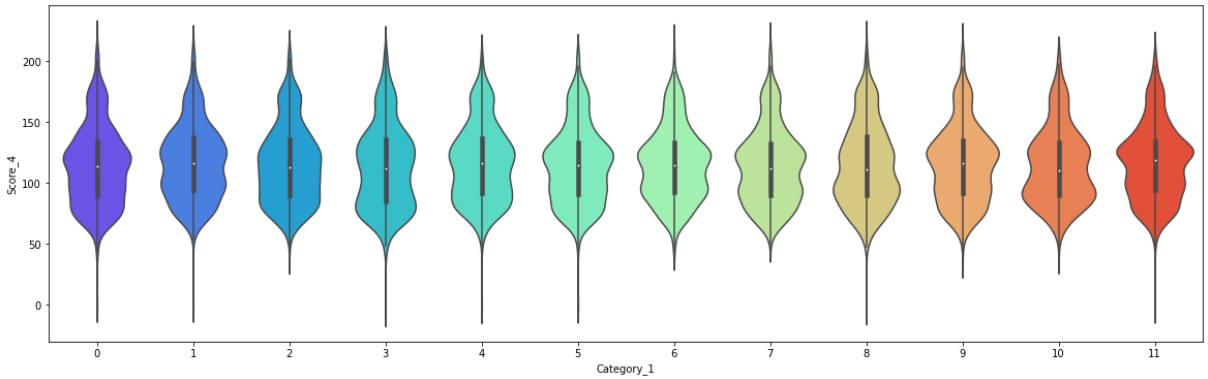
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d5dff10>
```



there is no relation between score3 and category1

In [ ]:
```
# perform a violin plot between score4 and category1
```

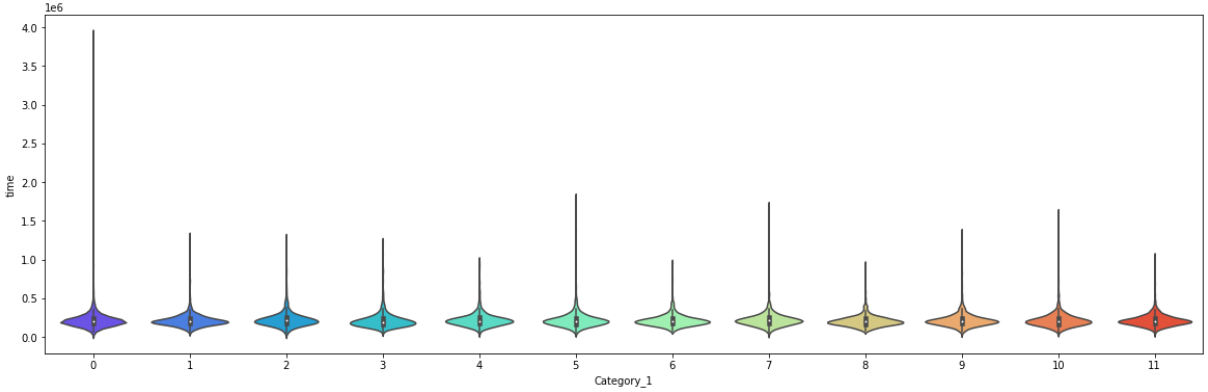
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d88f1d0>
```



there is no relation between score4 and category1

In [ ]:
```
# perform a violin plot between category1 and time
```

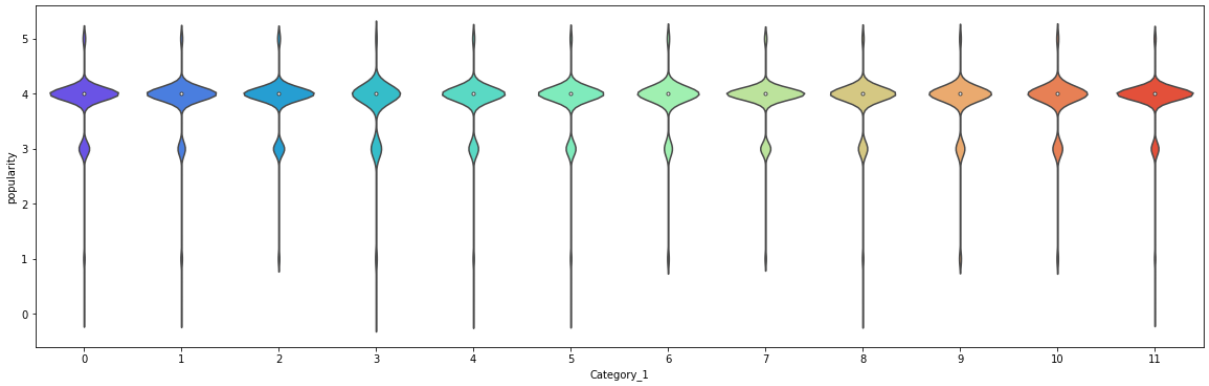
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d903810>
```



there is no relation between time and category1

In [ ]:
```
# perform a violin plot between popularity and category1
```

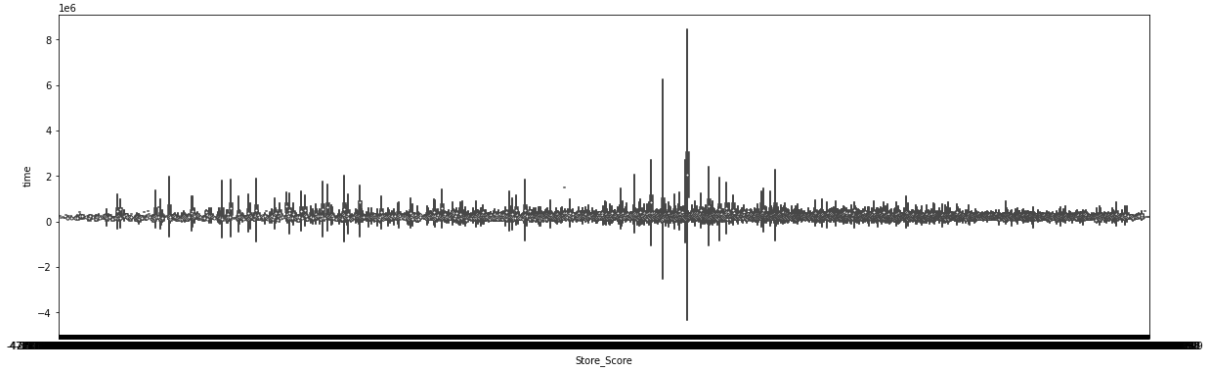
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc23d79e450>
```



there is no relation between popularity and category1

In [ ]:
```
# perform a violin plot between store score and time
```

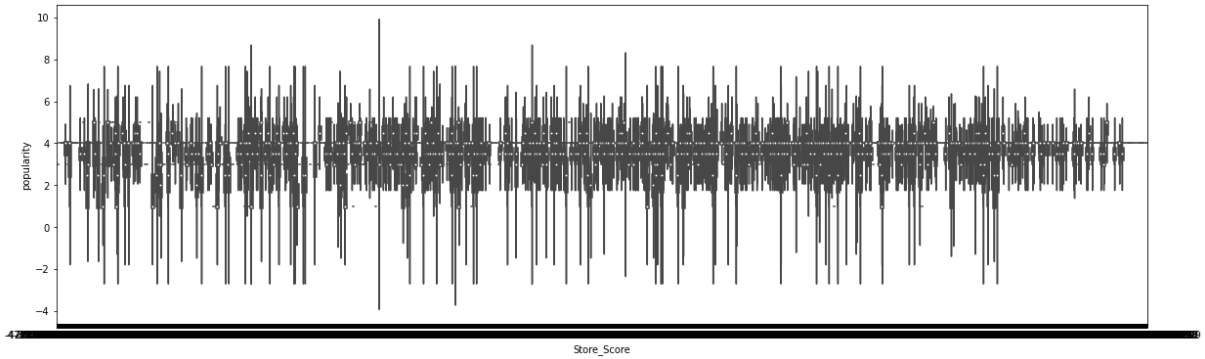
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc1dce0ddd0>



there is no relation between time and store score

In [ ]:
```
# perform a violin plot between store score and popularity
```

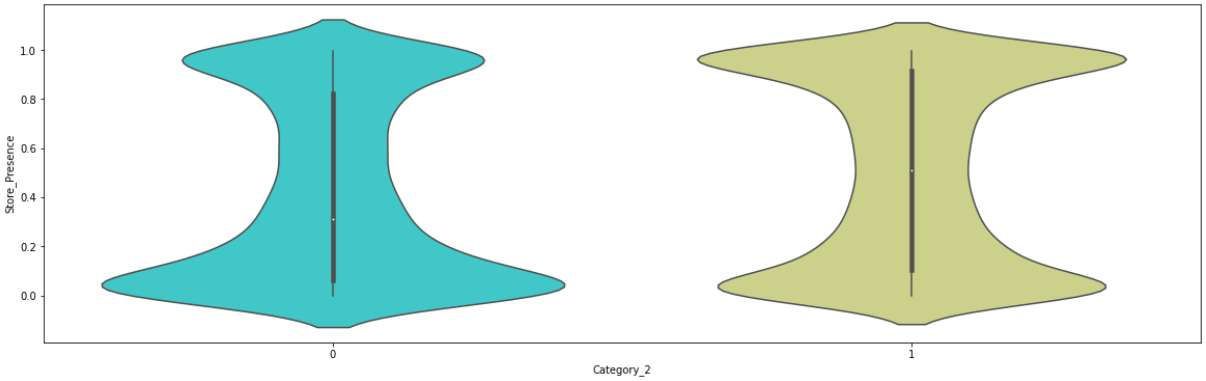
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc1dbbcdb50>



there is weak relation between store score and popularity

In [ ]:
```
# perform a violin plot between category2 and store presence
```

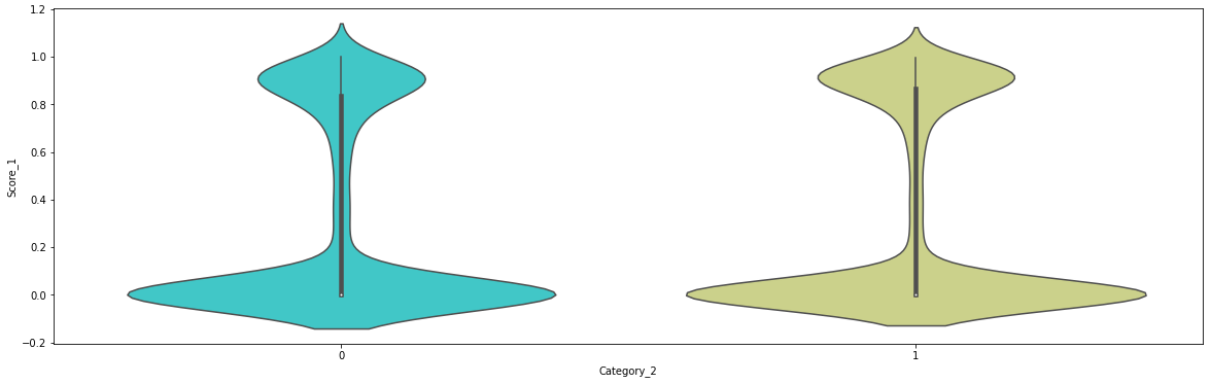
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fea016c7250>
```



there is no relation between category2 and store presence

In [ ]:
```
# perform a violin plot between category2 and score1
```

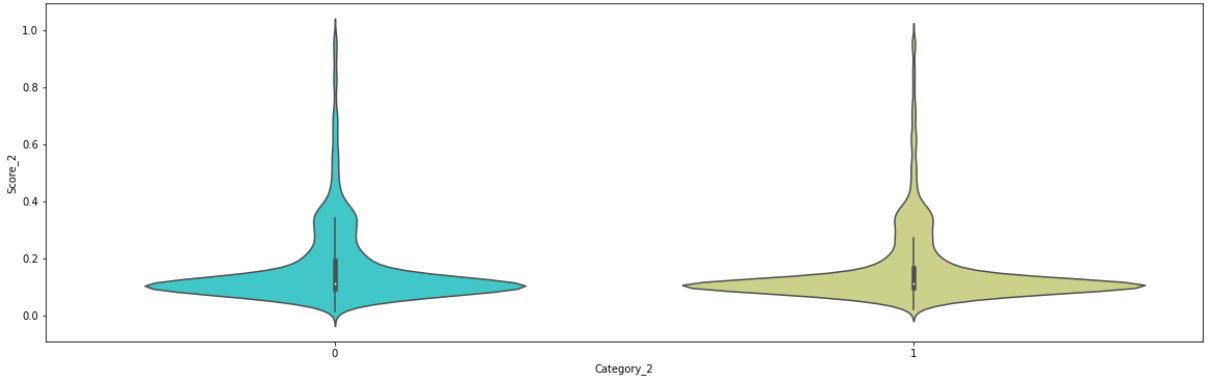
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e7f9ab90>
```



there is no relation between category2 and score1

In [ ]:
```
# perform a violin plot between category2 and score2
```

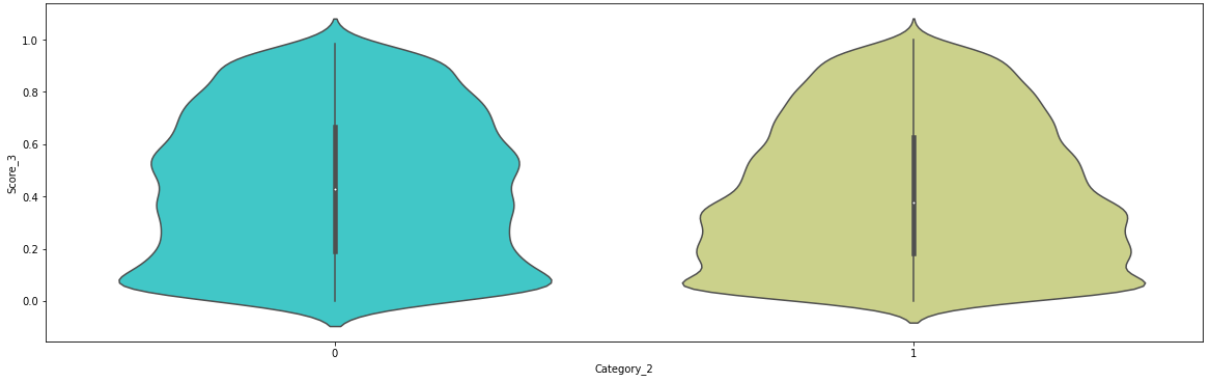
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e6999290>
```



there is no relation between category2 and score2

In [ ]:
```
# perform a violin plot between category2 and score3
```

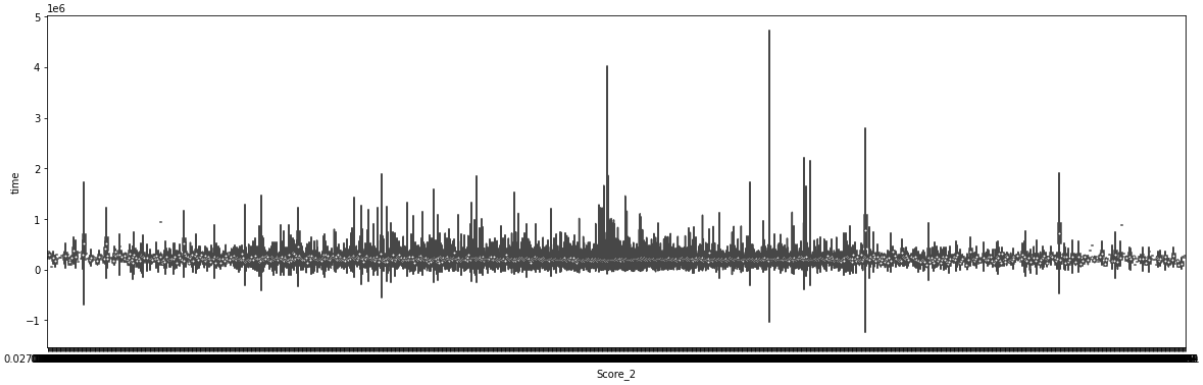
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e6a42350>
```



there is no relation between category2 and score3

In [ ]:
```
# perform a violin plot between score2 and time
```

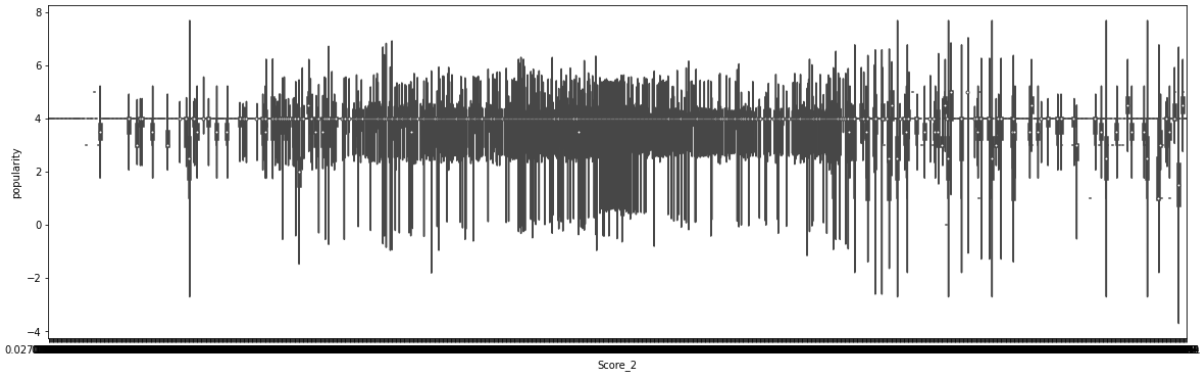
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe999352ed0>
```



there is no relation between time and score

In [ ]:
```
# perform a violin plot between score2 and popularity
```

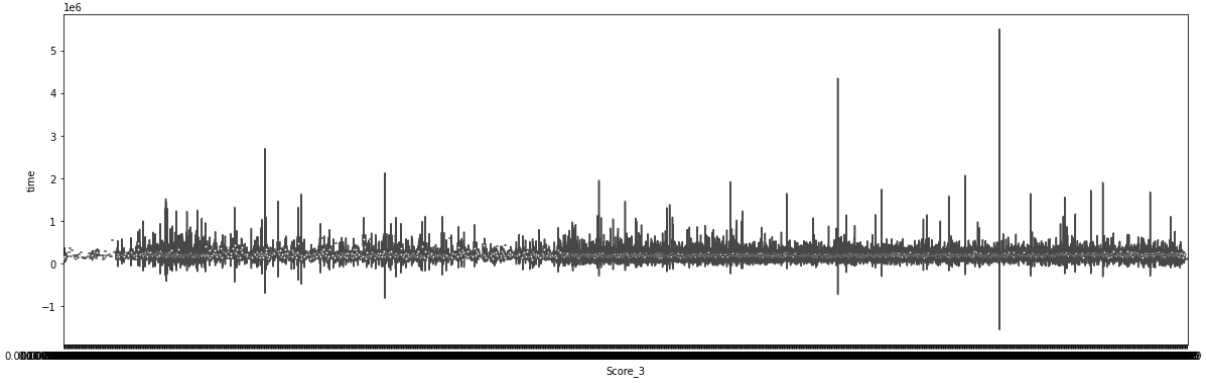
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe998dddd90>
```



there is no relation between score and popularity

In [ ]:
```
# perform a violin plot between score3 and time
```

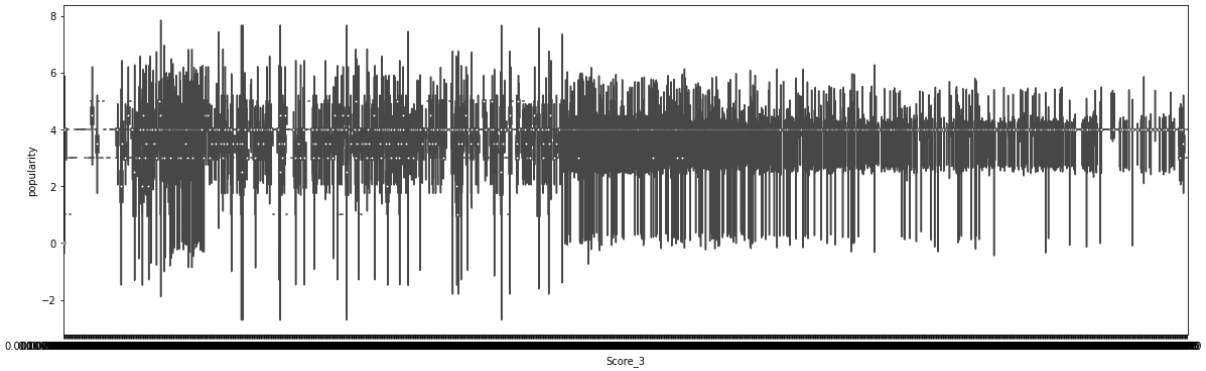
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9a38a5b10>
```



there is no relation between time and score 3

In [ ]:
```
# perform a violin plot between score3 and popularity
```

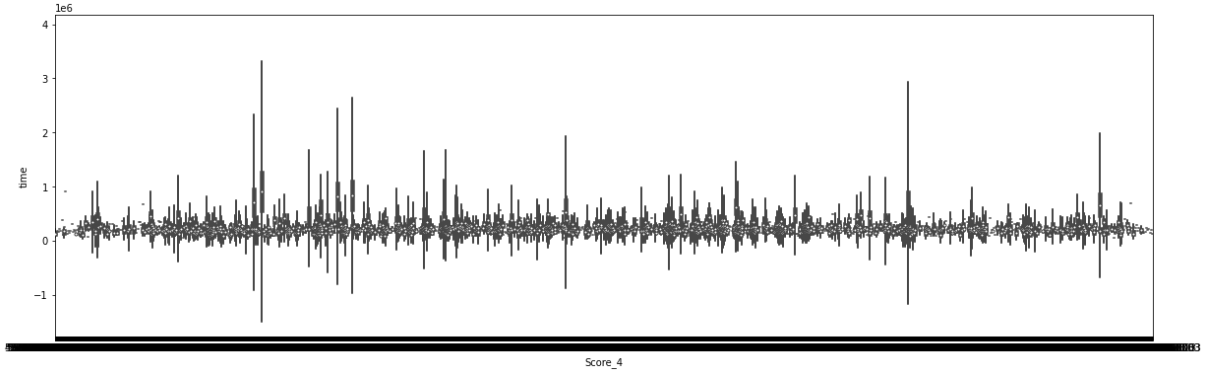
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e5135c50>
```



there is weak relation between popularity and score 3

In [ ]:
```
# perform a violin plot between time and score4
```

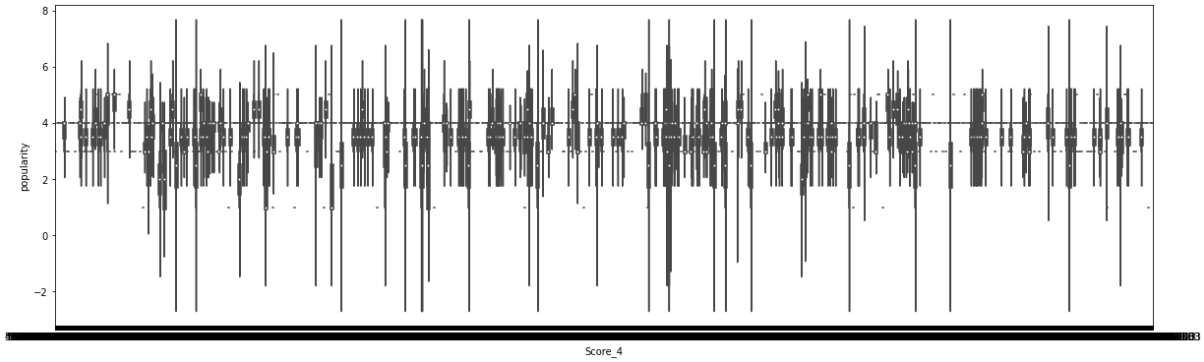
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9bd78b5d0>
```



there is no relation between time and score4

In [ ]:
```
# perform a violin plot between popularity and score4
```

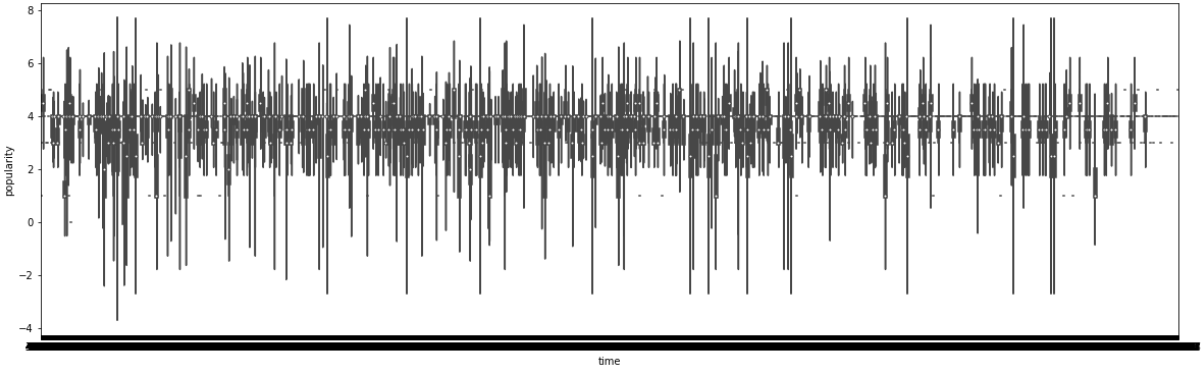
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e523bb90>
```



there is no relation between popularity and score 4

In [ ]:
```
# perform a violin plot between popularity and time
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9e6913990>
```



there is on relation between time and popularity

Preprocessing

In [ ]:
```
#convert the time column into more columns like hour, month, day, year
 , minute
```

In [ ]:
```
# drop popularity from train_df and name that variable as y

#drop popularity n time from train dataset
```

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
```

In [ ]:
```
# Making a list of the column names to be scaled

# passing data and column name for scaling
```

Splitting the data into train and test set

In [ ]:
```
# Import train_test_split from sklearn
# split data into 95% train , 5% test and random state 42
```

In [ ]:
```
# check shape of X_train
```

Out[ ]:
```
(17297, 15)
```

In [ ]:
```
# check for nan value in X_train
```

Out[ ]:
```
False
```

Modelling

In [ ]:
```
# importing necessary libraries for geting metrics of models



# Function for calculating all the relevant metrics
```

In [ ]:
```
# Helper function for Visualizing importance of all the features in the
dataset for the prediction



    # creating dataframe for feature name and feature importance


    # grouping all data and sorting in descending order


    # ploting feature importance data using boxenplot

    # return fig, ax
```

```
# Visualize importance of all the features in the dataset for the predi
ction
```

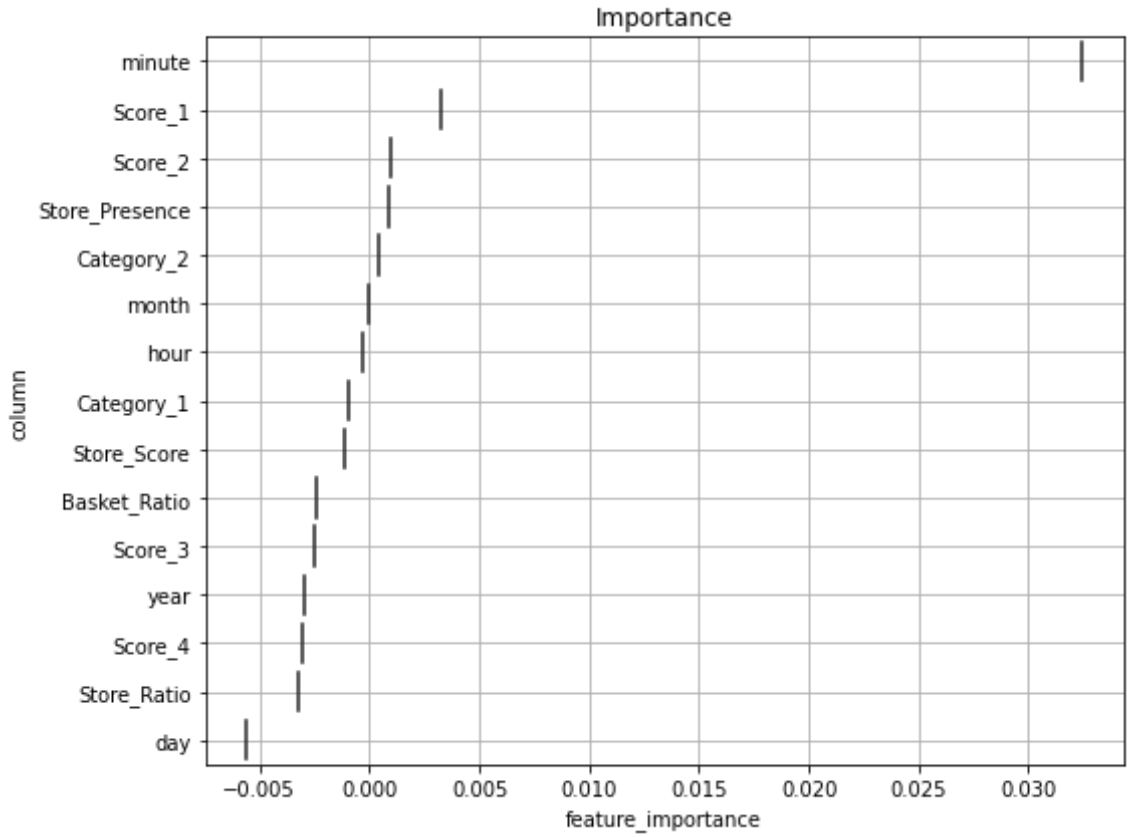
LOGISTIC REGRESSION

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 logisticRegressor


# Instantiate the model


# fitting the model on train data

# print score of the model by calling function

# visualizing importance of features
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.00      0.00      0.00        11
           3       0.00      0.00      0.00       115
```

```
              4       0.84      1.00      0.91       761
              5       0.00      0.00      0.00        22

    accuracy                            0.84       911
   macro avg       0.17      0.20      0.18       911
weighted avg       0.70      0.84      0.76       911

CPU times: user 1.64 s, sys: 1.28 s, total: 2.92 s
Wall time: 1.72 s
```

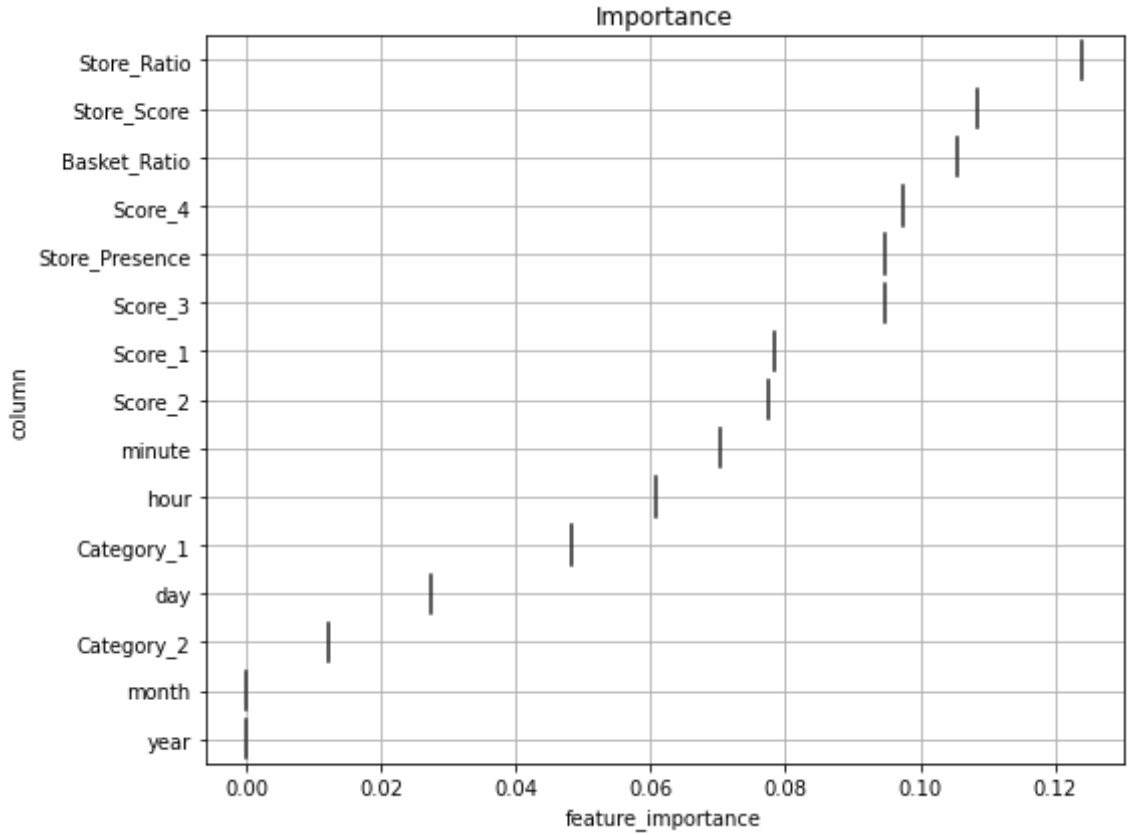


RANDOM FOREST CLASSIFIER

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

# fitting the model on train data

# print score of the model

# visualizing importance of features
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      0.73      0.84        11
           3       0.84      0.33      0.47       115
           4       0.89      0.99      0.94       761
           5       1.00      0.41      0.58        22

    accuracy                           0.89       911
   macro avg       0.95      0.69      0.77       911
weighted avg       0.89      0.89      0.87       911

CPU times: user 8.04 s, sys: 3.13 ms, total: 8.04 s
Wall time: 8.17 s
```



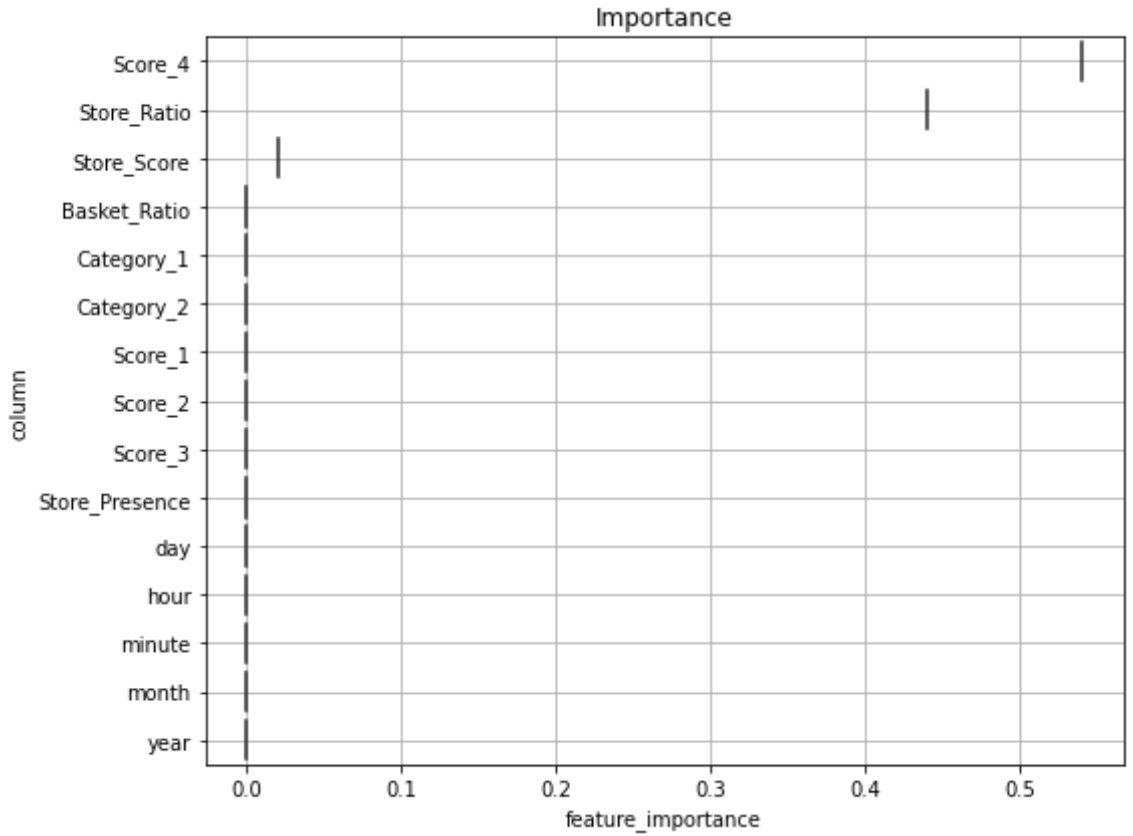
ADA BOOST CLASSIFIER

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

# fitting the model on train data

# print score of the model by calling function

# visualizing importance of features
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       0.00      0.09      0.00        11
           3       0.00      0.00      0.00       115
           4       0.70      0.34      0.45       761
           5       0.00      0.00      0.00        22

    accuracy                           0.28       911
   macro avg       0.34      0.29      0.29       911
weighted avg       0.58      0.28      0.38       911

CPU times: user 1.5 s, sys: 1.82 ms, total: 1.5 s
Wall time: 1.51 s
```



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

#fit the model on train data

# print score of the model by calling function
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.00      0.00      0.00        11
           3       0.00      0.00      0.00       115
           4       0.84      1.00      0.91       761
           5       0.00      0.00      0.00        22

    accuracy                           0.84       911
   macro avg       0.17      0.20      0.18       911
weighted avg       0.70      0.84      0.76       911

CPU times: user 4.2 s, sys: 139 ms, total: 4.34 s
Wall time: 4.33 s
```

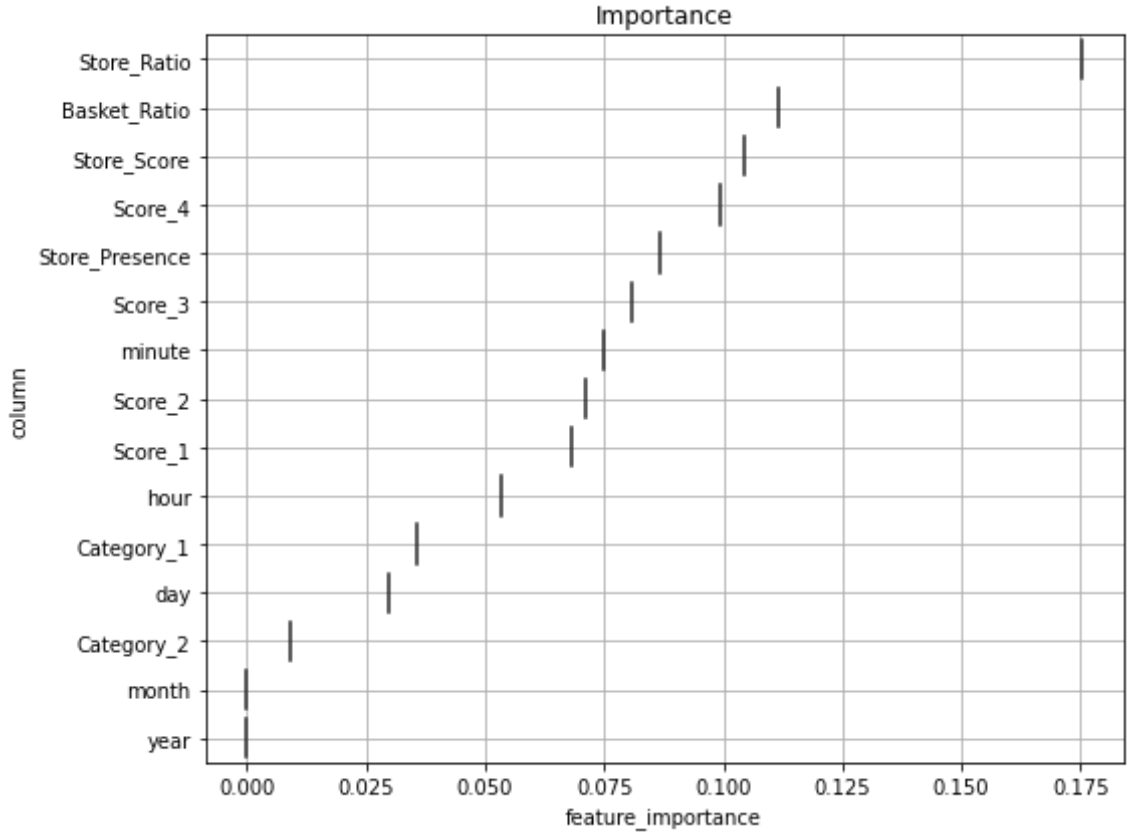
DESISION TREE CLASSICIFIER

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

# fitting the model on train data

# print score of the model by calling function

# visualizing importance of features
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       0.31      0.73      0.43        11
           3       0.47      0.41      0.44       115
           4       0.92      0.90      0.91       761
           5       0.28      0.45      0.34        22

    accuracy                           0.83       911
   macro avg       0.60      0.70      0.63       911
weighted avg       0.84      0.83      0.83       911

CPU times: user 343 ms, sys: 10.6 ms, total: 354 ms
Wall time: 354 ms
```



K NEIGHBOUR CLASSIFIER

In [ ]:
```
%%time
# Fit a K-Neighbour classifier model to the train dataset

# Import KNeighborsClassifier


# Instantiate the model

# fitting the model on train data
```

```
# print score of the model by calling function
```

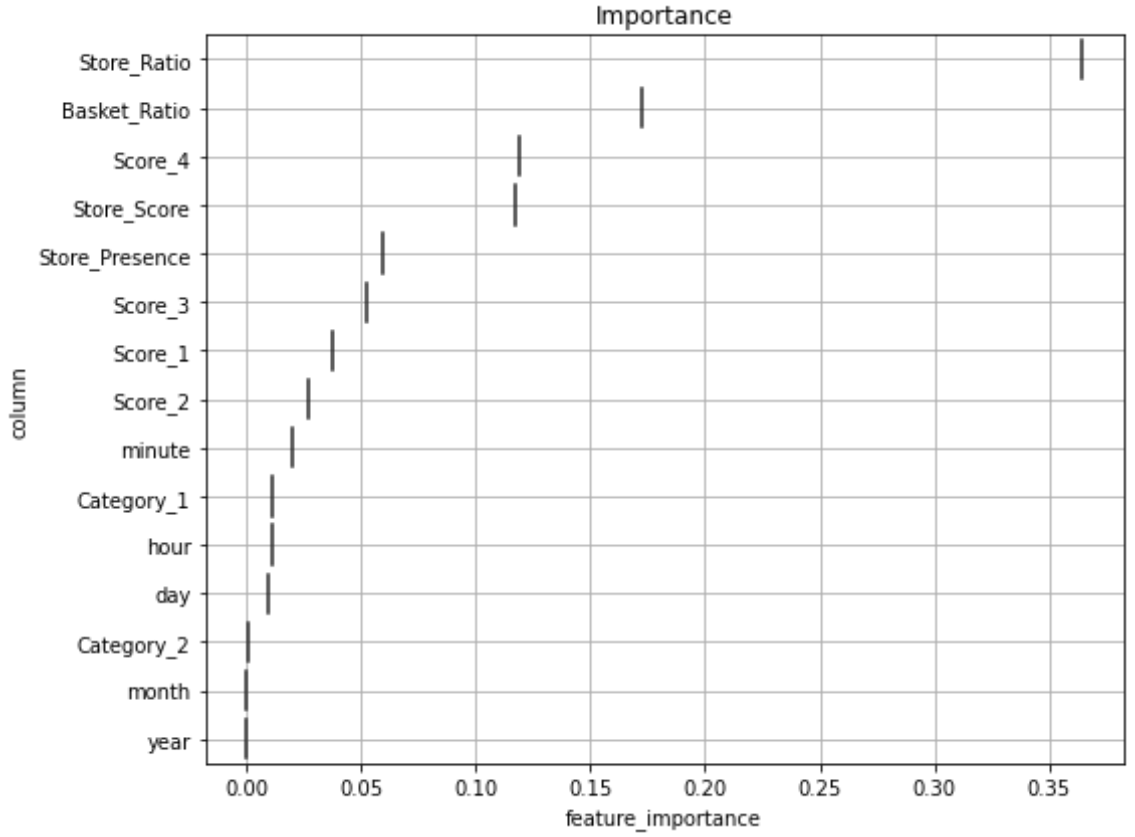
```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.12      0.09      0.11        11
           3       0.50      0.24      0.33       115
           4       0.87      0.96      0.91       761
           5       0.50      0.14      0.21        22

    accuracy                           0.84       911
   macro avg       0.40      0.29      0.31       911
weighted avg       0.80      0.84      0.81       911

CPU times: user 93.2 ms, sys: 0 ns, total: 93.2 ms
Wall time: 95.9 ms
```

GRADIENT BOOSTING CLASSIFIER

In [ ]:
```
%%time
# Fit a Gradient Boosting Classifier model to the train dataset

# Import GradientBoostingClassifier

# Instantiate the model

# fitting the model on train data

# print score of the model

# visualizing importance of features
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
```

```
              1       0.33      0.09      0.14        11
              3       0.46      0.11      0.18       115
              4       0.86      0.98      0.92       761
              5       0.80      0.18      0.30        22

    accuracy                            0.84       911
   macro avg       0.69      0.47      0.51       911
weighted avg       0.80      0.84      0.80       911

CPU times: user 23.4 s, sys: 621 µs, total: 23.4 s
Wall time: 23.4 s
```



BAGGING CLASSIFIER

In [ ]:
```
%%time
# Fit a Bagging Classifier model to the train dataset

# Import BaggingClassifier

# Instantiate the model

# 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         2
           1       0.82      0.82      0.82        11
           3       0.62      0.41      0.49       115
           4       0.90      0.96      0.93       761
           5       0.78      0.32      0.45        22

    accuracy                           0.88       911
   macro avg       0.82      0.70      0.74       911
weighted avg       0.86      0.88      0.86       911

CPU times: user 1.71 s, sys: 1.37 ms, total: 1.71 s
Wall time: 1.72 s
```

VOTING CLASSIFIER

In [ ]:
```
%%time
# Fit a VotingClassifier model to the train dataset

# Import VotingClassifier
```

```
# list of classifier objects

# Instantiate the model

# 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         2
           1       0.89      0.73      0.80        11
           3       0.84      0.32      0.47       115
           4       0.89      0.99      0.94       761
           5       1.00      0.23      0.37        22

    accuracy                           0.89       911
   macro avg       0.92      0.65      0.71       911
weighted avg       0.89      0.89      0.86       911

CPU times: user 12 s, sys: 68.4 ms, total: 12.1 s
Wall time: 12.1 s
```

EASY ENSEMBLE CLASSIFIER

In [ ]:
```
%%time
# Fit a EasyEnsembleClassifier model to the train dataset

# Import EasyEnsembleClassifier

# Instantiate the model

# fitting the model on train data

# print score of the model
```

```
Classification Report
```

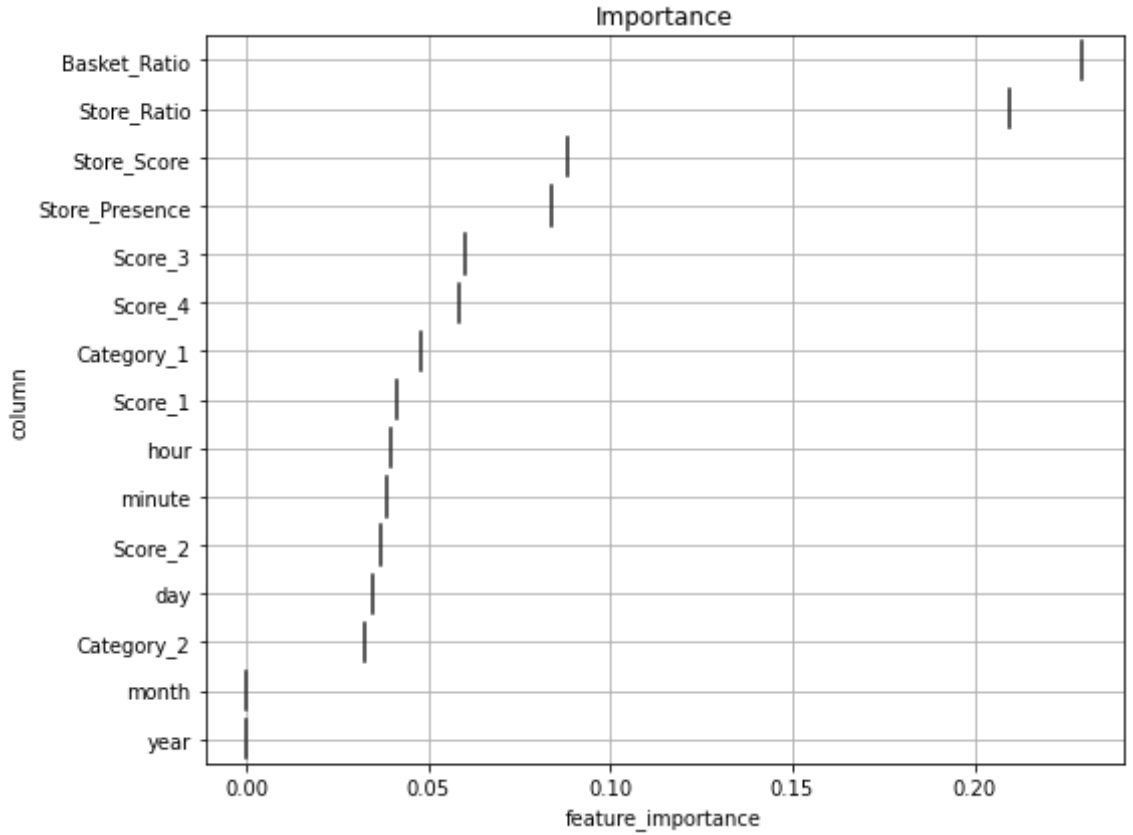
```
                precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       0.03      0.45      0.05        11
           3       0.17      0.21      0.19       115
           4       0.94      0.57      0.71       761
           5       0.06      0.36      0.11        22

    accuracy                           0.51       911
   macro avg       0.44      0.52      0.41       911
weighted avg       0.81      0.51      0.62       911

CPU times: user 884 ms, sys: 24 ms, total: 908 ms
Wall time: 1 s
```

XGB CLASSIFIER

In [ ]:
```
%%time
# Fit a XGBClassifier model to the train dataset

# Import XGBClassifier

# Instantiate the model

# fitting the model on train data

# print score of the model

# visualizing importance of features
```

```
Classification Report
                precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       0.00      0.00      0.00        11
           3       0.58      0.06      0.11       115
           4       0.84      0.99      0.91       761
```

```
           5       0.00      0.00      0.00        22

    accuracy                           0.84       911
   macro avg       0.49      0.41      0.40       911
weighted avg       0.78      0.84      0.78       911

CPU times: user 4.62 s, sys: 25.3 ms, total: 4.64 s
Wall time: 4.88 s
```



Comparing all the model based on metric

In [ ]:
```
# helper function for comparing models matric
```

```
    # the libraries we need

    # create a dataframe with column matric and metric name as value

     # then predict on the test set
```

In [ ]:
```
# list of model objects

# list of model names

# print the comparison of models
```

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 [ ]:
```
 # Helper function to perform hyper parameter tunning with GridSearchCV


    # Grid search of parameters, using 5 fold cross validation

    #fit model_cv

    # print best parameters

    # print best score
```

In [ ]:
```
%%time
# create logistic regressor parameters dict in list for tunning


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

```
0.18232284750190483
```

In [ ]:
```
%%time
# create RandomForest parameters dict in list for tunning


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

In [ ]:
```
%%time
# create KNNRegressor parameters dict in list for tunning


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

In [ ]:
```
%%time
# create GradientBoostRegressor parameters dict in list for tunning


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

In [ ]:
```
%%time
# create DecisionTreeRegressor parameters dict in list for tunning



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

In [ ]:
```
%%time
# create AdaBoostRegressor parameters dict in list for tunning


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

In [ ]:
```
%%time
# create XGBoost parameters dict in list for tunning


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

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

RamdomizedSearchCV

In [ ]:
```
# Helper function to perform hyper parameter tunning with RandomizedSea
rchCV


  # Random search of parameters, using 3 fold cross validation,
  # search across 100 different combinations, and use all available cor
es
```

```
  # Fit the random search model
```

In [ ]:
```
%%time
# create logistic regressor parameters dict for tunning


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

In [ ]:
```
%%time
# create RandomForest classifier parameters dict for tunning


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

In [ ]:
```
%%time
# create KNN classifier parameters dict for tunning

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

In [ ]:
```
%%time
# create GradientBoost classifier parameters dict for tunning


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

In [ ]:
```
%%time
# create DecisionTree classifier parameters dict for tunning


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

In [ ]:
```
%%time
# create AdaBoost classifier parameters dict for tunning


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

In [ ]:
```
%%time
# create XGBoost parameters dict in list for tunning


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

Now working with the test dataset provided

In [ ]:
```
# preparing test data as similarly as done for train data before.
```

In [ ]:
```

```

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
```

In [ ]:
```
# Perforn the prediction on the test dataset
```

Out[ ]:
```
array([4, 4, 4, ..., 4, 4, 4])
```

In [ ]:
```
# creating a dataframe of predicted results
```

In [ ]:
```
# predicted values in dataframe
```

Out[ ]:

	0
0	4
1	4

	0
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.