IST 707 – Data Analytics

Black Friday Sales

Group 6

Kshitij Sankesara

Vijet Muley

Yunxi Dong

Introduction:

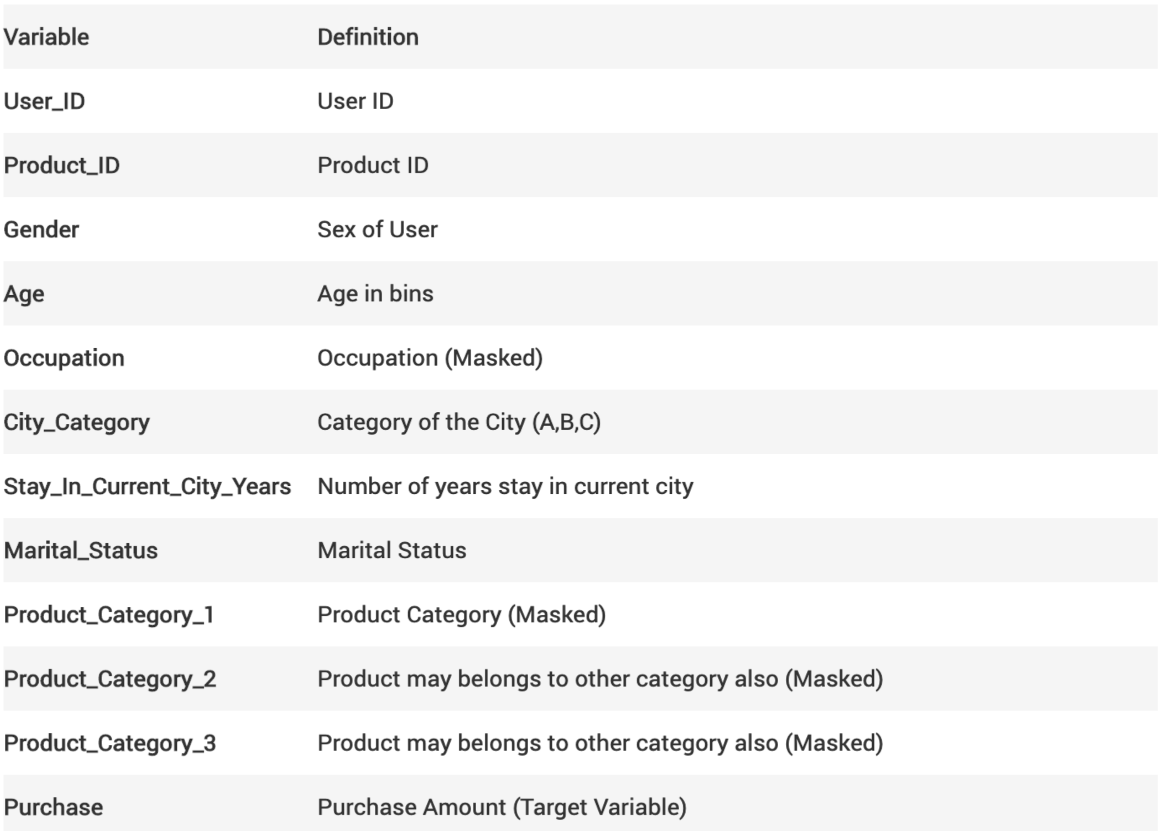
A retail store wants to know which products to stock before the holiday season to make sure that the store doesn't run out of products. For this, the store wants to analyze all the previous year's transactions. They want to make sure that they don't over stock or under stock any particular category of product.

To help the store owners we analyze the complete Black Friday dataset which is a sample of the transactions made in the retail store. We get to know the customer purchase behavior against different products.

The report will showcase how we have gone through with our analysis, starting from data cleaning, preprocessing the data, descriptive statistics, visualizations, generating models for the analysis, comparison of our models and conclusion based on all the whole analysis.

Data Description:

The Black Friday Dataset was obtained from Kaggle which consists of 538k rows and 12 attributes. The attributes are User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_in_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3 and Purchase.



Variable	Definition
User_ID	User ID
Product_ID	Product ID
Gender	Sex of User
Age	Age in bins
Occupation	Occupation (Masked)
City_Category	Category of the City (A,B,C)
Stay_In_Current_City_Years	Number of years stay in current city
Marital_Status	Marital Status
Product_Category_1	Product Category (Masked)
Product_Category_2	Product may belongs to other category also (Masked)
Product_Category_3	Product may belongs to other category also (Masked)
Purchase	Purchase Amount (Target Variable)

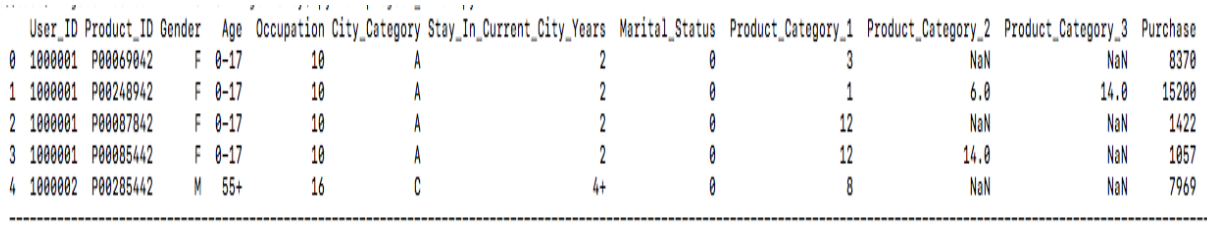
Attributes and their Definitions

Data Preprocessing:

First, we found out the missing values in our dataset. There were missing values in two columns 'Product_Category_2' and 'Product_Category_3'. We tried different methods for the missing values but later we decided to replace all the missing values with 0. We replaced it with 0 as it made more sense for our later analysis.

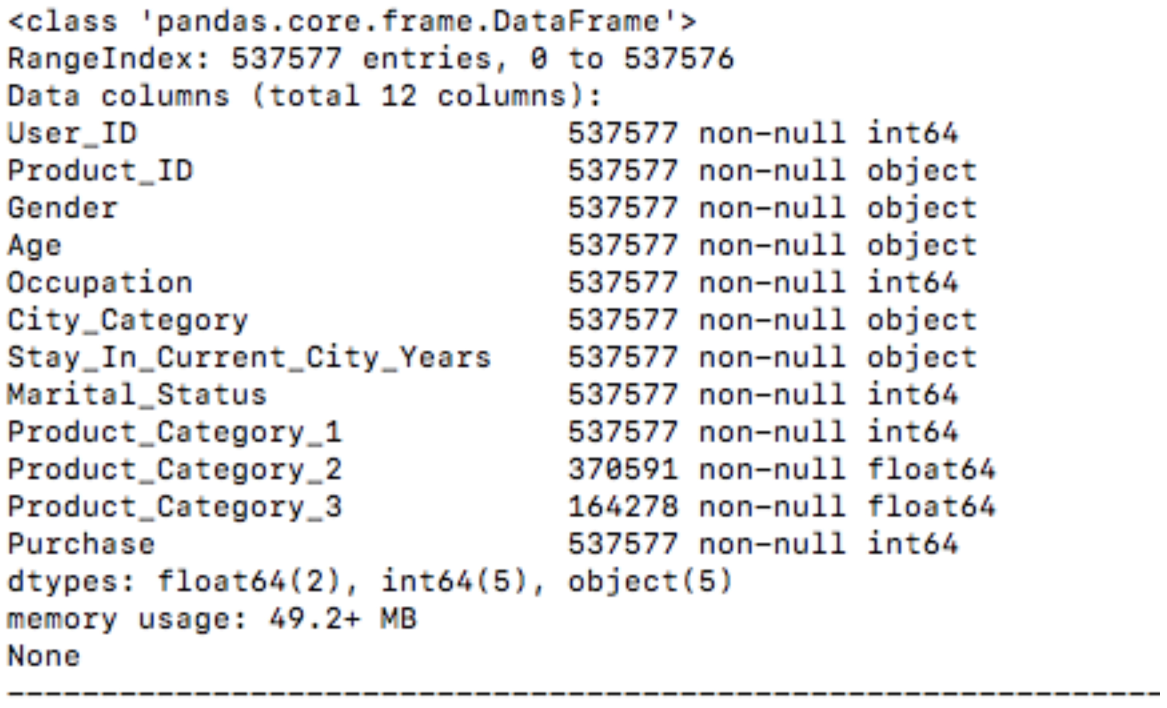
It took a while for us to understand as to why this may be the case. Rest of the data has no missing value whatsoever. However, only these columns seem to display missing values. The answer to this was actually in the data description. The product category columns are present in order to cover the versatility of a product. Meaning, the product category bins in 'Product_Category_1' column categorizes every product into a certain category (say, for eg., Electronics, sports etc.) The way the other 2 columns are supposed to be interpreted is that one product may belong to more than one category. If it doesn't, the other 2 columns simply won't contain any data. And that is the reason why those 2 columns, in particular, have missing data.

We decided to implement a simple solution for this missing data problem. The categories start from '1' and range till '18'. So, we decided to replace all the NaN values with 0. Thus, we made a new category '0' for all the missing or uncategorized data points (uncategorized in the sense that they don't belong to more than one or more than two categories).



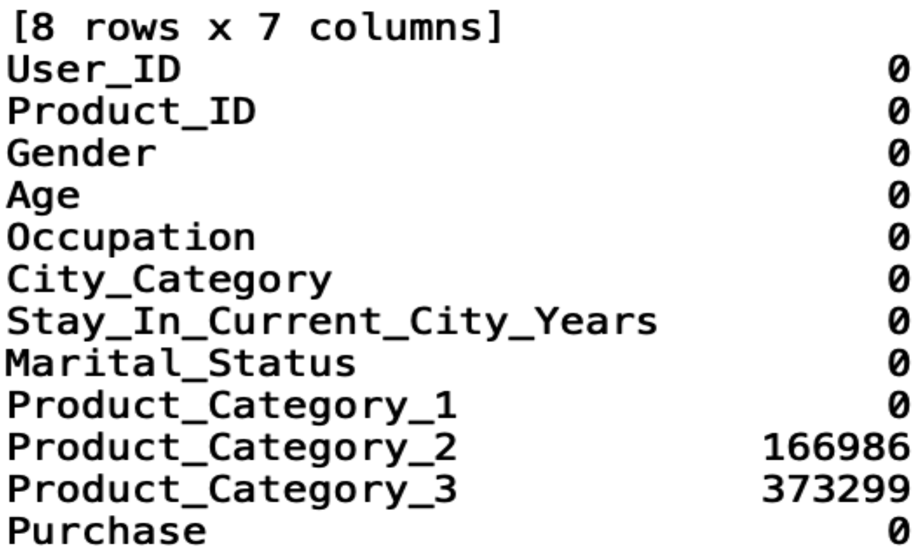
```
   User_ID Product_ID Gender   Age  Occupation City_Category Stay_In_Current_City_Years  Marital_Status  Product_Category_1  Product_Category_2  Product_Category_3  Purchase
0  1000001  P00069042      F  0-17          10             A                          2               0                   3                 NaN                 NaN      8370
1  1000001  P00248942      F  0-17          10             A                          2               0                   1                 6.0                14.0     15200
2  1000001  P00087842      F  0-17          10             A                          2               0                  12                 NaN                 NaN      1422
3  1000001  P00085442      F  0-17          10             A                          2               0                  12                14.0                 NaN      1057
4  1000002  P00285442      M   55+          16             C                         4+               0                   8                 NaN                 NaN      7969
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Data in its raw form



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null object
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            370591 non-null float64
Product_Category_3            164278 non-null float64
Purchase                      537577 non-null int64
dtypes: float64(2), int64(5), object(5)
memory usage: 49.2+ MB
None
-------------------------------------------------------------
```

Metadata

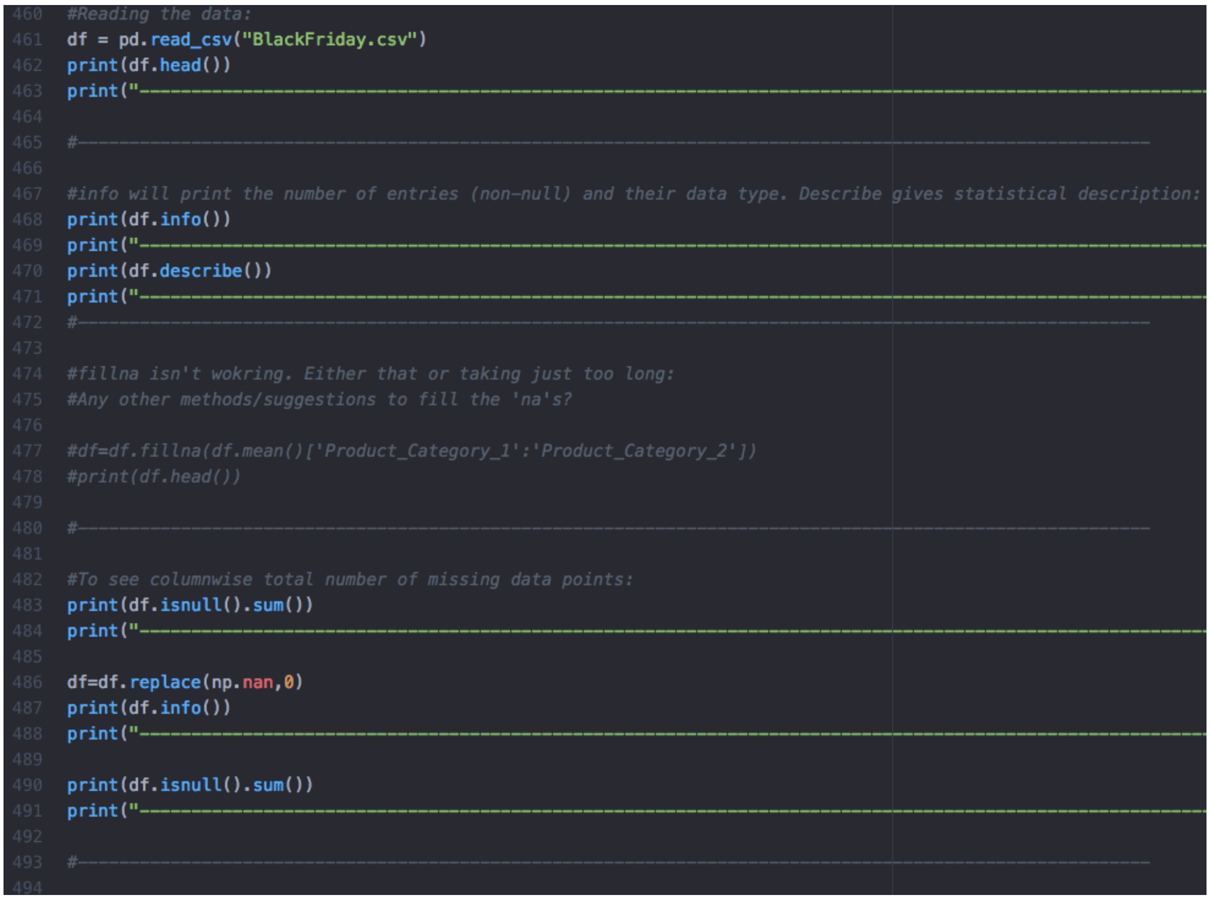


```
[8 rows x 7 columns]
User_ID                            0
Product_ID                         0
Gender                             0
Age                                0
Occupation                         0
City_Category                      0
Stay_In_Current_City_Years         0
Marital_Status                     0
Product_Category_1                 0
Product_Category_2            166986
Product_Category_3            373299
Purchase                           0
```

Missing Values

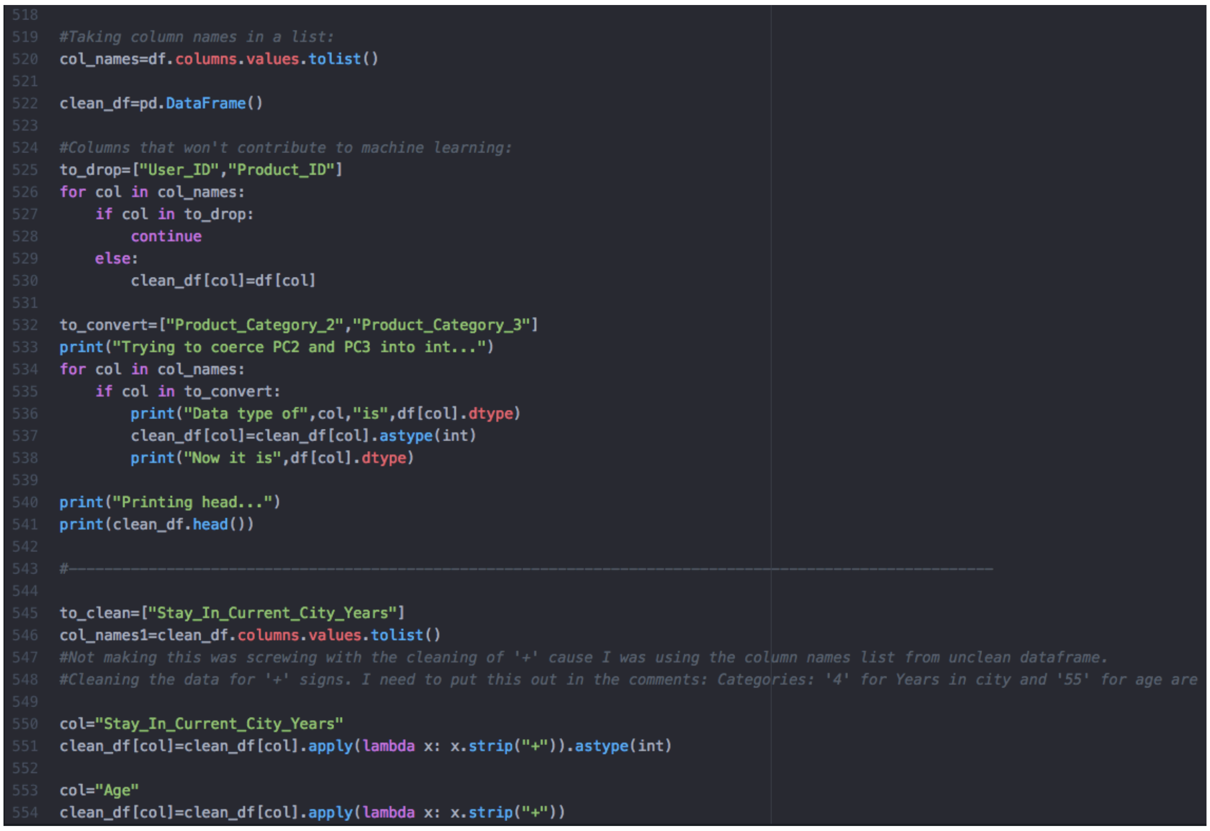
We removed 2 columns, User_ID and Product_ID as both the attributes were not significant for our analysis. Both the columns won't be that helpful when we choose to carry out a predictive analysis on our data. The reason is that, we are trying to predict the purchase amount as a part of this project. The IDs are a label, mostly for data storing, and being so distinct as they are, they'd rather make the model more specific and result in overfitting, if included at all. Thus, we decided to remove those 2 columns.

Another data inconsistency problem was the presence of '+' sign in 'Age' and 'Stay_In_Current_City_Years'. We are planning to use these bins as levels, giving them an order in their occurrence. '+' sign will introduce an inconsistency and thus we decided upon removing the '+' sign. We used the 'strip' function in combination with 'apply' function on these particular columns to get rid of the '+' signs.



```
#Reading the data:
df = pd.read_csv("BlackFriday.csv")
print(df.head())
print("-----------------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

#info will print the number of entries (non-null) and their data type. Describe gives statistical description:
print(df.info())
print("-----------------------------------------------------------------------------------------------
print(df.describe())
print("-----------------------------------------------------------------------------------------------
#-----------------------------------------------------------------------------------------------

#fillna isn't wokring. Either that or taking just too long:
#Any other methods/suggestions to fill the 'na's?

#df=df.fillna(df.mean()['Product_Category_1':'Product_Category_2'])
#print(df.head())

#-----------------------------------------------------------------------------------------------

#To see columnwise total number of missing data points:
print(df.isnull().sum())
print("-----------------------------------------------------------------------------------------------

df=df.replace(np.nan,0)
print(df.info())
print("-----------------------------------------------------------------------------------------------

print(df.isnull().sum())
print("-----------------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------

```

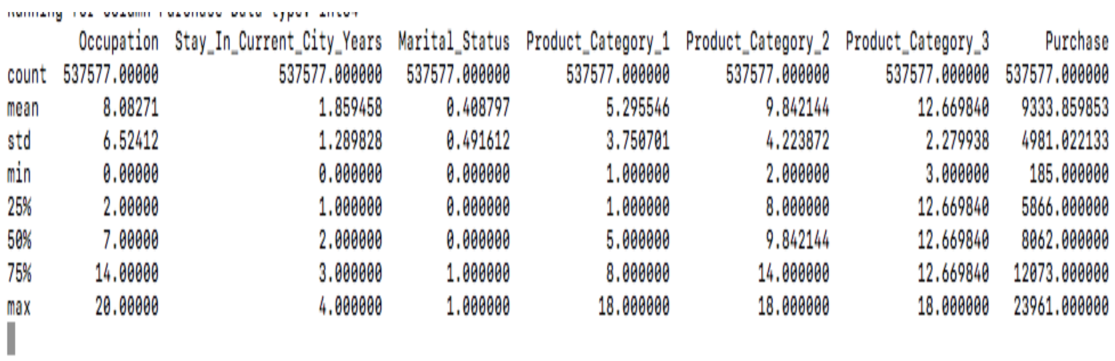
Data Cleaning - 1



```python

#Taking column names in a list:
col_names=df.columns.values.tolist()

clean_df=pd.DataFrame()

#Columns that won't contribute to machine learning:
to_drop=["User_ID","Product_ID"]
for col in col_names:
    if col in to_drop:
        continue
    else:
        clean_df[col]=df[col]

to_convert=["Product_Category_2","Product_Category_3"]
print("Trying to coerce PC2 and PC3 into int...")
for col in col_names:
    if col in to_convert:
        print("Data type of",col,"is",df[col].dtype)
        clean_df[col]=clean_df[col].astype(int)
        print("Now it is",df[col].dtype)

print("Printing head...")
print(clean_df.head())

#-----------------------------------------------------------------------------------------------------------

to_clean=["Stay_In_Current_City_Years"]
col_names1=clean_df.columns.values.tolist()
#Not making this was screwing with the cleaning of '+' cause I was using the column names list from unclean dataframe.
#Cleaning the data for '+' signs. I need to put this out in the comments: Categories: '4' for Years in city and '55' for age are

col="Stay_In_Current_City_Years"
clean_df[col]=clean_df[col].apply(lambda x: x.strip("+")).astype(int)

col="Age"
clean_df[col]=clean_df[col].apply(lambda x: x.strip("+"))
```

Data Cleaning - 2

We had also gone through the Descriptive Analysis to get to know better the numerical attributes of our Dataset.



```
       Occupation  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  Product_Category_2  Product_Category_3       Purchase
count 537577.00000               537577.000000   537577.000000       537577.000000       537577.000000       537577.000000  537577.000000
mean       8.08271                    1.859458        0.408797            5.295546            9.842144           12.669840    9333.859853
std        6.52412                    1.289828        0.491612            3.750701            4.223872            2.279938    4981.022133
min        0.00000                    0.000000        0.000000            1.000000            2.000000            3.000000     185.000000
25%        2.00000                    1.000000        0.000000            1.000000            8.000000           12.669840    5866.000000
50%        7.00000                    2.000000        0.000000            5.000000            9.842144           12.669840    8062.000000
75%       14.00000                    3.000000        1.000000            8.000000           14.000000           12.669840   12073.000000
max       20.00000                    4.000000        1.000000           18.000000           18.000000           18.000000   23961.000000
```

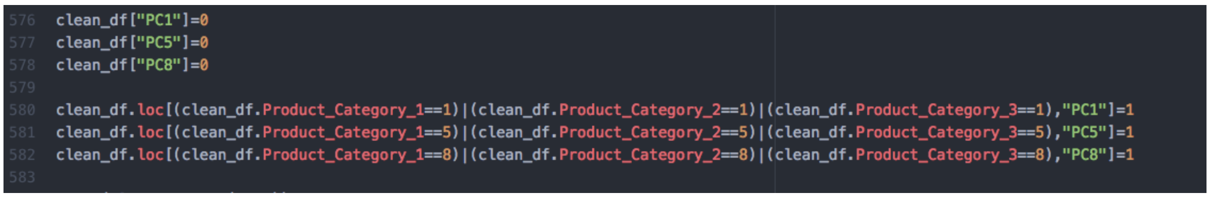
Descriptive Analysis

For the Product Category 1,2 and 3 we first thought to analyze it by creating a new attribute Product Category. The Product Category will be calculated by using all 3 Product Categories. Product Category = Product_Category_1 * 10000 + Product_Category_2 * 100 + Product_Category_3 * 1.

However, our assumptions related to Product Categories were wrong. This way of calculating Product Category won't be significant while generating models. So, we re-evaluated our method and later changed it.

We thought of using dummies as an alternative approach. We had 18 categories in our Product Category, so making 17 dummy variables wasn't feasible. That's why we decided to go with 3 product categories 1, 5 and 8 as their frequency in the data was very high. This is the method which we later used for our Product Category attribute.

1. Dummy_1=1, if Product_Category_1==1 or Product_Category_2==1 or Product_Category_3==1
2. Dummy_5=1, if Product_Category_1==5 or Product_Category_2==5 or Product_Category_3==5
3. Dummy_8=1, if Product_Category_1==8 or Product_Category_2==8 or Product_Category_3==8



```
clean_df["PC1"]=0
clean_df["PC5"]=0
clean_df["PC8"]=0

clean_df.loc[(clean_df.Product_Category_1==1)|(clean_df.Product_Category_2==1)|(clean_df.Product_Category_3==1),"PC1"]=1
clean_df.loc[(clean_df.Product_Category_1==5)|(clean_df.Product_Category_2==5)|(clean_df.Product_Category_3==5),"PC5"]=1
clean_df.loc[(clean_df.Product_Category_1==8)|(clean_df.Product_Category_2==8)|(clean_df.Product_Category_3==8),"PC8"]=1
```

Product Category Code

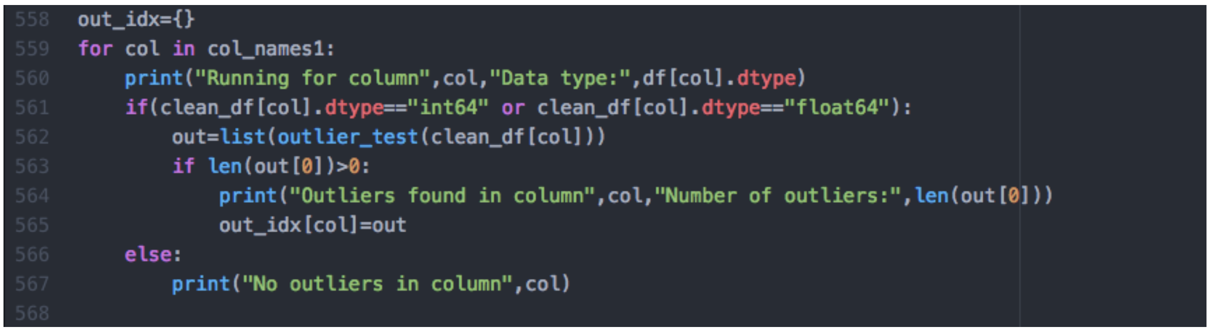
<u>Challenges and Solution:</u>

A further visual analysis showed us that 3 of the 18 product categories in our data showed a significantly high number of sales, when compared to others. But it is to be noted that the column labels are mere labels, barring any ordering whatsoever. So, it won't make any sense if we used these product category labels directly for our analysis as they don't mean anything. For all intent and purposes, we may replace 1 with apples, 2 with oranges and so on and ultimately it won't deprecate our analysis in any manner. This problem was realized a further down the road, and it required a pragmatic solution. As mentioned before, 3 categories showed higher sale than other. Those categories were '1', '5' and '8'. Thus, we decided to make dummy variables representing these 3 product categories. The approach used was that if either of the 3 product category columns had product classified as one of these 3 categories, that particular dummy will take the value 1. Being neither or these 3 categories but belonging to any other category was held as baseline, for both individual dummies and all 3 dummies as collective.

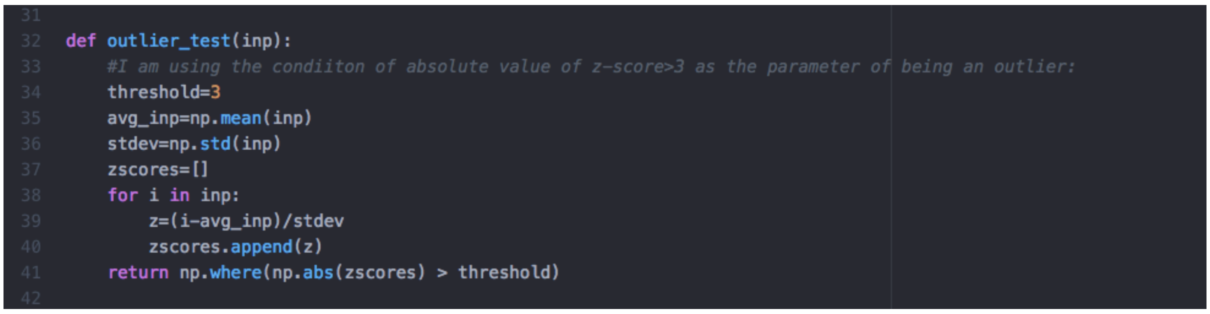
Thus, for eg., if a product belonged to category 1 and category 5, the PC1 and PC5 dummy would be 1 and PC8 dummy would be 0. If the product belonged only to category 0, then, PC1, PC5 and PC8 would all be 0.

<u>Outliers:</u>

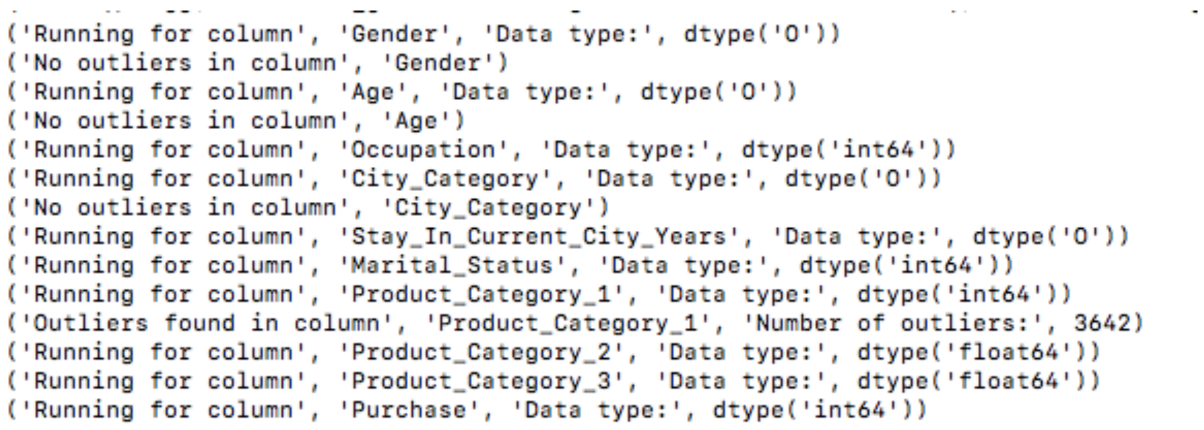
We also chose to have a look at the outliers, if any. We wrote a custom function for that, which took one column at a time as an argument. It calculated the mean and standard deviation for that column and then eventually calculated the z-scores. We decided to take a z-score absolute value of 3 as a threshold for outlier and our function returned the row numbers of all the observations determined to be outliers, as an assistance in handling them. While calling the function, we made sure that the function was called only for the columns that had the data type integer or float, for only they can be evaluated for outliers.



```
out_idx={}
for col in col_names1:
    print("Running for column",col,"Data type:",df[col].dtype)
    if(clean_df[col].dtype=="int64" or clean_df[col].dtype=="float64"):
        out=list(outlier_test(clean_df[col]))
        if len(out[0])>0:
            print("Outliers found in column",col,"Number of outliers:",len(out[0]))
            out_idx[col]=out
    else:
        print("No outliers in column",col)
```



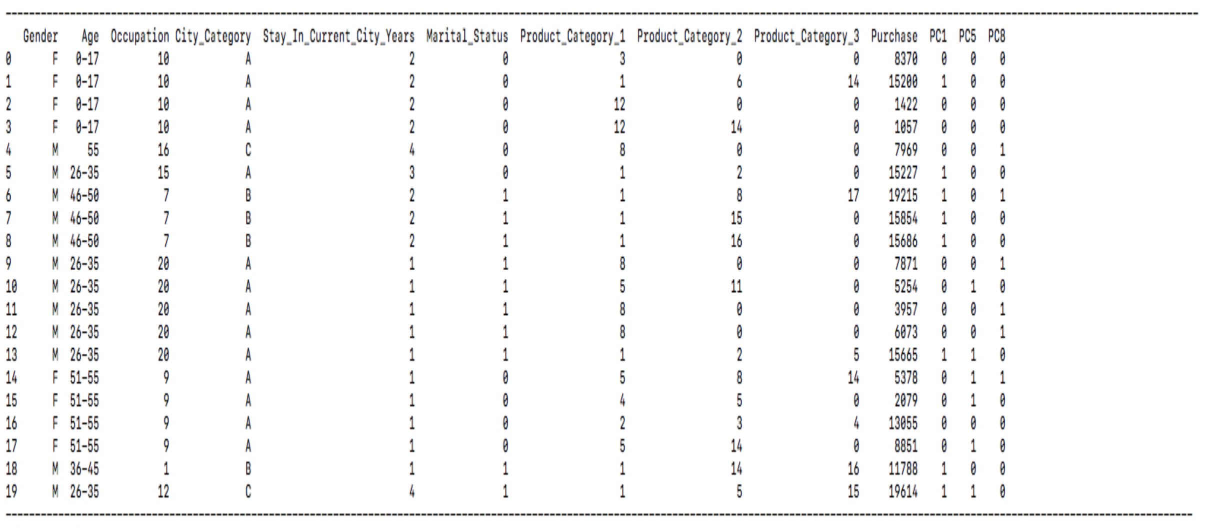
```
def outlier_test(inp):
    #I am using the condiiton of absolute value of z-score>3 as the parameter of being an outlier:
    threshold=3
    avg_inp=np.mean(inp)
    stdev=np.std(inp)
    zscores=[]
    for i in inp:
        z=(i-avg_inp)/stdev
        zscores.append(z)
    return np.where(np.abs(zscores) > threshold)
```



```
('Running for column', 'Gender', 'Data type:', dtype('O'))
('No outliers in column', 'Gender')
('Running for column', 'Age', 'Data type:', dtype('O'))
('No outliers in column', 'Age')
('Running for column', 'Occupation', 'Data type:', dtype('int64'))
('Running for column', 'City_Category', 'Data type:', dtype('O'))
('No outliers in column', 'City_Category')
('Running for column', 'Stay_In_Current_City_Years', 'Data type:', dtype('O'))
('Running for column', 'Marital_Status', 'Data type:', dtype('int64'))
('Running for column', 'Product_Category_1', 'Data type:', dtype('int64'))
('Outliers found in column', 'Product_Category_1', 'Number of outliers:', 3642)
('Running for column', 'Product_Category_2', 'Data type:', dtype('float64'))
('Running for column', 'Product_Category_3', 'Data type:', dtype('float64'))
('Running for column', 'Purchase', 'Data type:', dtype('int64'))
```

It shows outliers only for the column ‘Product_Category_1’. Ironically, it won’t count to anything, as discussed earlier, they are all labels and they don’t have a fixed order amongst the product categories. All the other columns are outlier free. (P.S.: It shows running for each column, but only numeric columns get the 2nd output, i.e. either no outliers found, or number of outliers found. The non-numeric ones don’t call the function. Corroborated from the snippet of code in image 4 and image 5)

Thus finally, the cleaned data looks as follows:



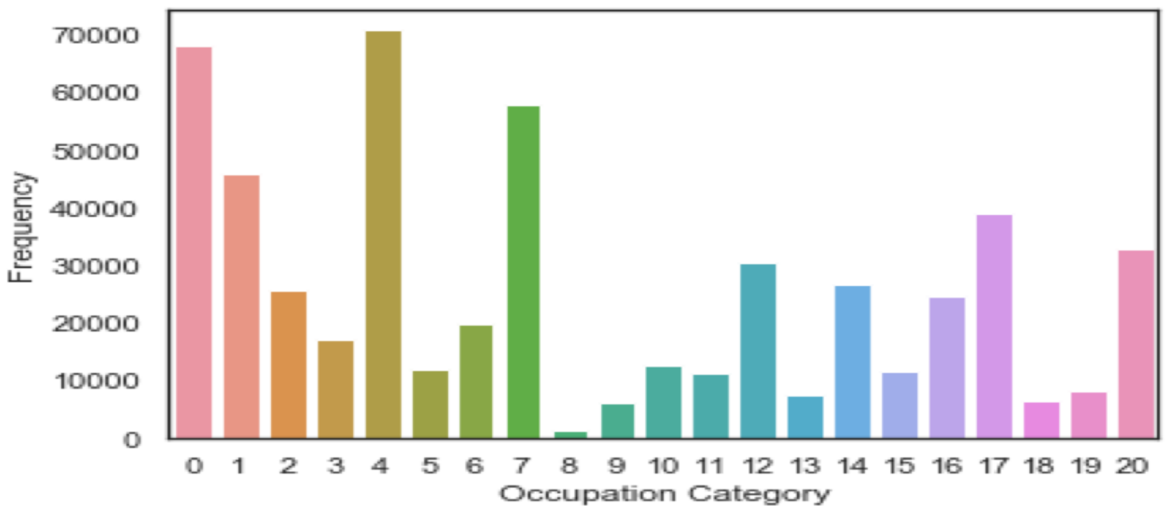
```
   Gender   Age  Occupation City_Category  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  Product_Category_2  Product_Category_3  Purchase  PC1  PC5  PC8
0       F  0-17          10             A                           2               0                   3                   0                   0      8370    0    0    0
1       F  0-17          10             A                           2               0                   1                   6                  14     15200    1    0    0
2       F  0-17          10             A                           2               0                  12                   0                   0      1422    0    0    0
3       F  0-17          10             A                           2               0                  12                  14                   0      1057    0    0    0
4       M    55          16             C                           4               0                   8                   0                   0      7969    0    0    1
5       M 26-35          15             A                           3               0                   1                   2                   0     15227    1    0    0
6       M 46-50           7             B                           2               1                   1                   8                  17     19215    1    0    1
7       M 46-50           7             B                           2               1                   1                  15                   0     15854    1    0    0
8       M 46-50           7             B                           2               1                   1                  16                   0     15686    1    0    0
9       M 26-35          20             A                           1               1                   8                   0                   0      7871    0    0    1
10      M 26-35          20             A                           1               1                   5                  11                   0      5254    0    1    0
11      M 26-35          20             A                           1               1                   8                   0                   0      3957    0    0    1
12      M 26-35          20             A                           1               1                   8                   0                   0      6073    0    0    1
13      M 26-35          20             A                           1               1                   1                   2                   5     15665    1    1    0
14      F 51-55           9             A                           1               0                   5                   8                  14      5378    0    1    1
15      F 51-55           9             A                           1               0                   4                   5                   0      2079    0    1    0
16      F 51-55           9             A                           1               0                   2                   3                   4     13055    0    0    0
17      F 51-55           9             A                           1               0                   5                  14                   0      8851    0    1    0
18      M 36-45           1             B                           1               1                   1                  14                  16     11788    1    0    0
19      M 26-35          12             C                           4               1                   1                   5                  15     19614    1    1    0
```

Another point to be noted is that, although this is termed to be clean data, this is still raw data when it comes to be used for analysis. Every analysis and predictive modeling method will require more cleaning and munging so that the data is fit and ready for that particular predictive model.

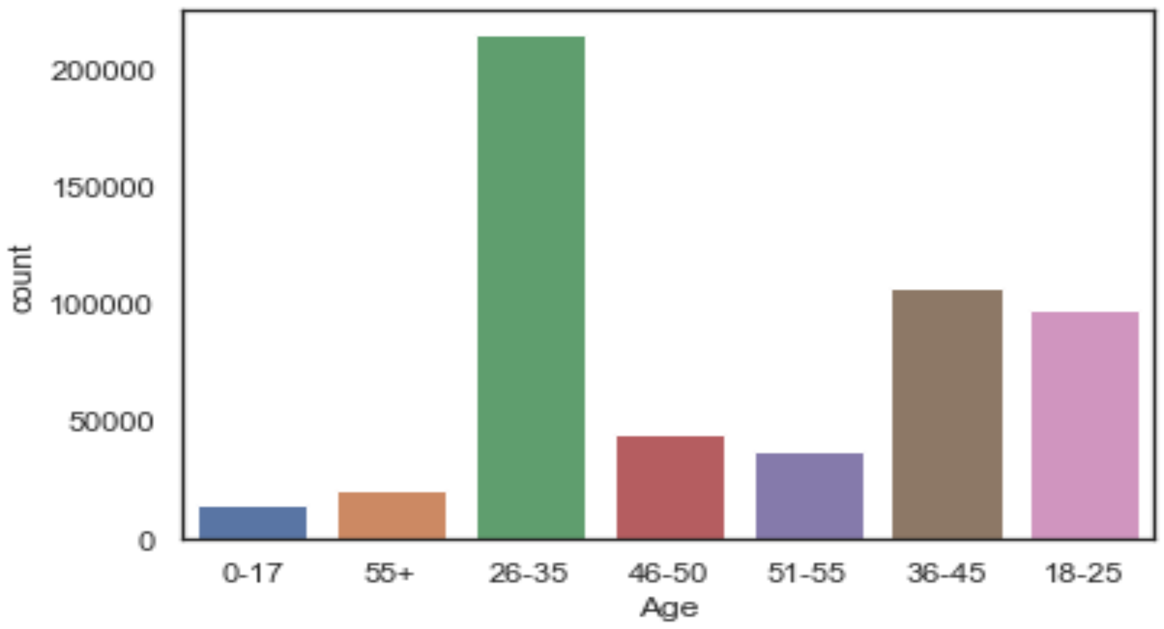
Descriptive Statistics:

We created visualization graphs of significant attributes with their frequency to better visualize our data.

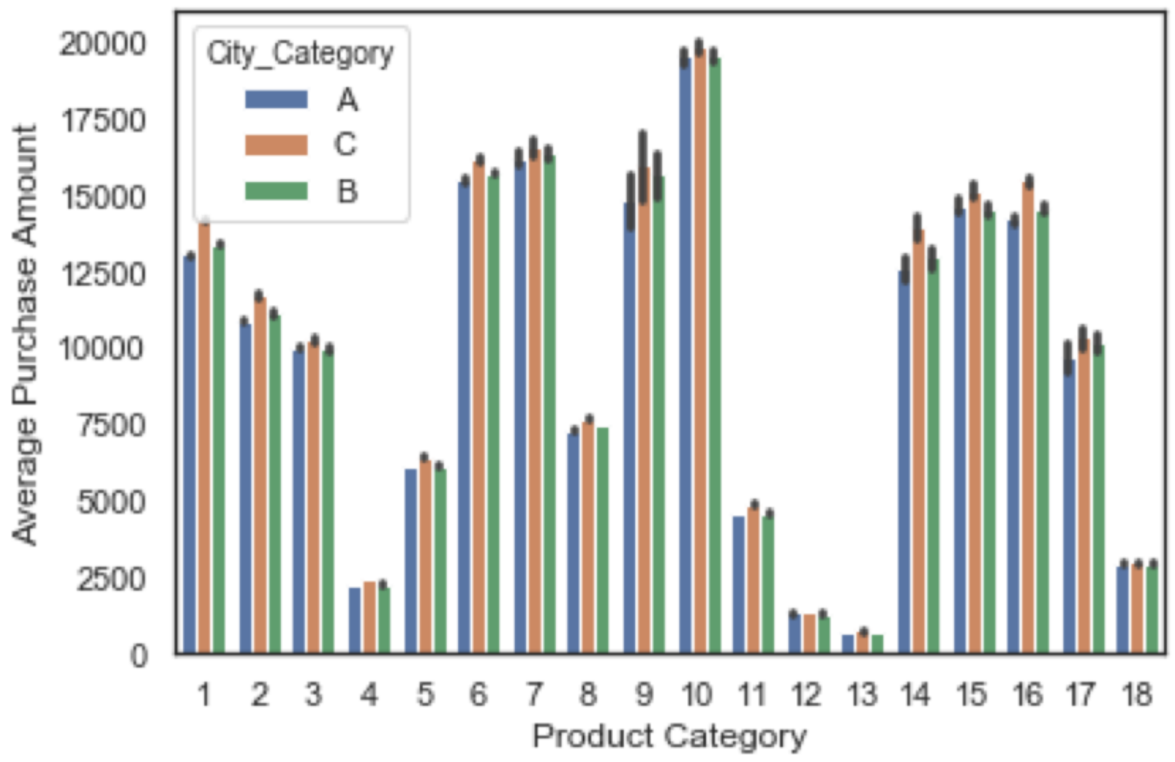
1. Occupation – Below is the Occupation of people with respect to their purchase count. We found out that the greatest number of purchase count were by occupation 0, 1, 4, 7, 17 and 20.



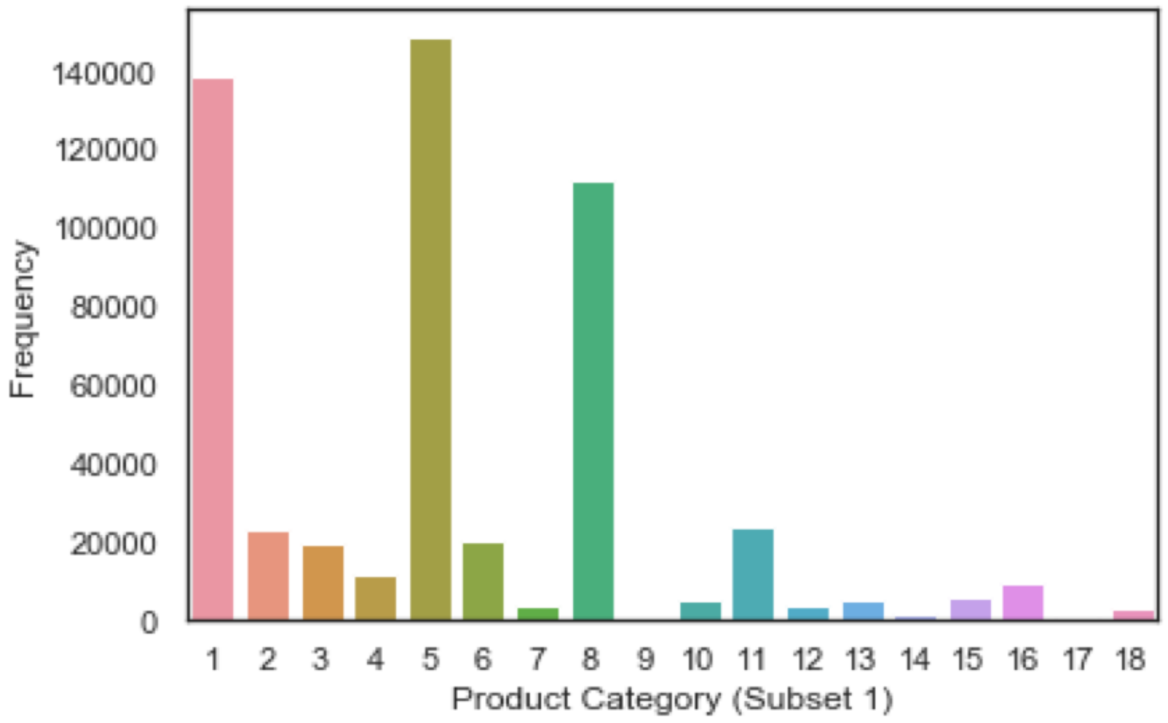
2. Age – Age Group of people and their respective count of purchase. The age groups between 18-45 are the ones who purchases the most.



3. Product & City – Here is the product and city category with respect to their Purchase Amount.



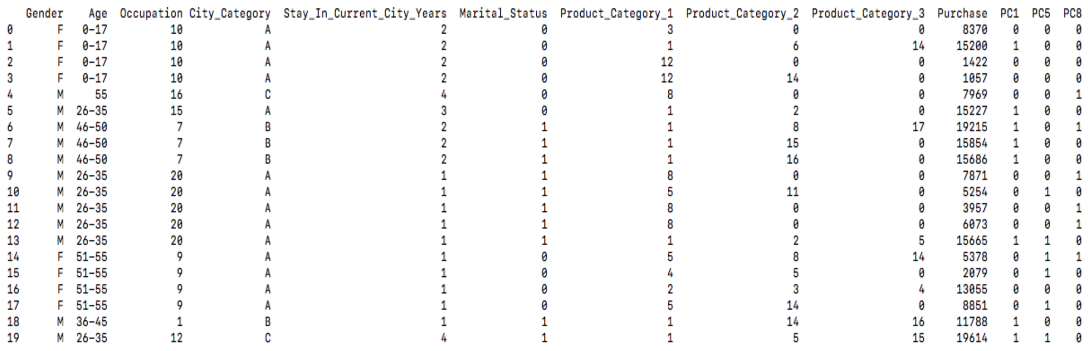
4. Product Category – Below is the Product Category 1 and its Frequency. From this we get to know that Product Category 1, 5 and 8 were purchased the most.



Linear Regression:

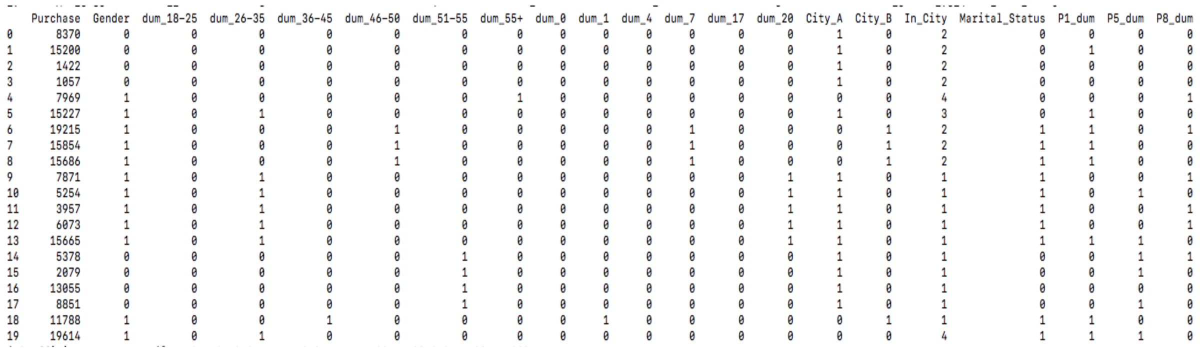
For generating Linear Regression model, we have created dummy variables. The dummy variables which we have created are:

1. For ‘Gender’: We took ‘F’ as our baseline, i.e. 0 and ‘M’ as 1.
2. For ‘Age’: We had 7 bins originally, so we made 6 dummy variables and 0-17 bin is the baseline (all dummies 0)
3. For ‘Occupation’: We selected 6 categories out of 20 that had highest number of observations. The selected categories are 0,1,4,7,17,20. All the other categories are the baseline.
4. For ‘City Category’: We took C as our baseline and 2 dummy variables for category A and B.
5. For ‘Stay in current city’, ‘Marital status’ and dummies for product categories 1,5 and 8 are taken as it is.



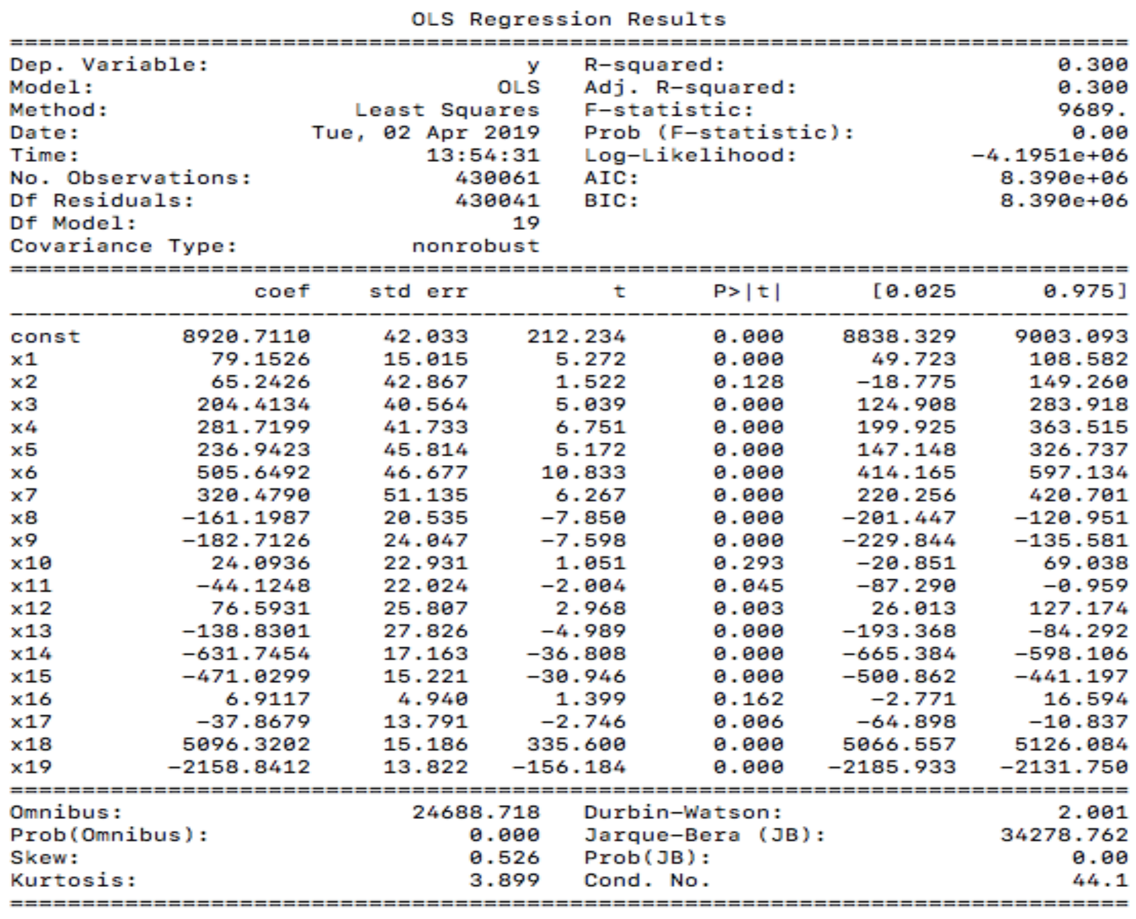
```
   Gender    Age  Occupation City_Category  Stay_In_Current_City_Years  Marital_Status  Product_Category_1  Product_Category_2  Product_Category_3  Purchase  PC1  PC5  PC8
0       F   0-17          10             A                           2               0                   3                   0                   0      8370    0    0    0
1       F   0-17          10             A                           2               0                   1                   6                  14     15200    1    0    0
2       F   0-17          10             A                           2               0                  12                   0                   0      1422    0    0    0
3       F   0-17          10             A                           2               0                  12                  14                   0      1057    0    0    0
4       M     55          16             C                           4               0                   8                   0                   0      7969    0    0    1
5       M  26-35          15             A                           3               0                   1                   2                   0     15227    1    0    0
6       M  46-50           7             B                           2               1                   1                   8                  17     19215    1    0    1
7       M  46-50           7             B                           2               1                   1                  15                   0     15854    1    0    0
8       M  46-50           7             B                           2               1                   1                  16                   0     15686    1    0    0
9       M  26-35          20             A                           1               1                   8                   0                   0      7871    0    0    1
10      M  26-35          20             A                           1               1                   5                  11                   0      5254    0    1    0
11      M  26-35          20             A                           1               1                   8                   0                   0      3957    0    0    1
12      M  26-35          20             A                           1               1                   8                   0                   0      6073    0    0    1
13      M  26-35          20             A                           1               1                   1                   2                   5     15665    1    1    0
14      F  51-55           9             A                           1               0                   5                   8                  14      5378    0    1    1
15      F  51-55           9             A                           1               0                   4                   5                   0      2079    0    1    0
16      F  51-55           9             A                           1               0                   2                   3                   4     13055    0    0    0
17      F  51-55           9             A                           1               0                   5                  14                   0      8851    0    1    0
18      M  36-45           1             B                           1               1                   1                  14                  16     11788    1    0    0
19      M  26-35          12             C                           4               1                   1                   5                  15     19614    1    1    0
```

Original Data



```
    Purchase  Gender  dum_18-25  dum_26-35  dum_36-45  dum_46-50  dum_51-55  dum_55+  dum_0  dum_1  dum_4  dum_7  dum_17  dum_20  City_A  City_B  In_City  Marital_Status  P1_dum  P5_dum  P8_dum
0       8370       0          0          0          0          0          0        0      0      0      0      0       0       0       1       0        2               0       0       0       0
1      15200       0          0          0          0          0          0        0      0      0      0      0       0       0       1       0        2               0       1       0       0
2       1422       0          0          0          0          0          0        0      0      0      0      0       0       0       1       0        2               0       0       0       0
3       1057       0          0          0          0          0          0        0      0      0      0      0       0       0       1       0        2               0       0       0       0
4       7969       1          0          0          0          0          0        1      0      0      0      0       0       0       0       0        4               0       0       0       1
5      15227       1          0          1          0          0          0        0      0      0      0      0       0       0       1       0        3               0       1       0       0
6      19215       1          0          0          0          1          0        0      0      0      0      1       0       0       0       1        2               1       1       0       1
7      15854       1          0          0          0          1          0        0      0      0      0      1       0       0       0       1        2               1       1       0       0
8      15686       1          0          0          0          1          0        0      0      0      0      1       0       0       0       1        2               1       1       0       0
9       7871       1          0          1          0          0          0        0      0      0      0      0       0       1       1       0        1               1       0       0       1
10      5254       1          0          1          0          0          0        0      0      0      0      0       0       1       1       0        1               1       0       1       0
11      3957       1          0          1          0          0          0        0      0      0      0      0       0       1       1       0        1               1       0       0       1
12      6073       1          0          1          0          0          0        0      0      0      0      0       0       1       1       0        1               1       0       0       1
13     15665       1          0          1          0          0          0        0      0      0      0      0       0       1       1       0        1               1       1       1       0
14      5378       0          0          0          0          0          1        0      0      0      0      0       0       0       1       0        1               0       0       1       1
15      2079       0          0          0          0          0          1        0      0      0      0      0       0       0       1       0        1               0       0       1       0
16     13055       0          0          0          0          0          1        0      0      0      0      0       0       0       1       0        1               0       0       0       0
17      8851       0          0          0          0          0          1        0      0      0      0      0       0       0       1       0        1               0       0       1       0
18     11788       1          0          0          1          0          0        0      0      1      0      0       0       0       0       1        1               1       1       0       0
19     19614       1          0          1          0          0          0        0      0      0      0      0       0       0       0       0        4               1       1       1       0
```

Data with Dummy Variables



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.300
Model:                            OLS   Adj. R-squared:                  0.300
Method:                 Least Squares   F-statistic:                     9689.
Date:                Tue, 02 Apr 2019   Prob (F-statistic):               0.00
Time:                        13:54:31   Log-Likelihood:            -4.1951e+06
No. Observations:              430061   AIC:                         8.390e+06
Df Residuals:                  430041   BIC:                         8.390e+06
Df Model:                          19
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       8920.7110     42.033    212.234      0.000    8838.329    9003.093
x1            79.1526     15.015      5.272      0.000      49.723     108.582
x2            65.2426     42.867      1.522      0.128     -18.775     149.260
x3           204.4134     40.564      5.039      0.000     124.908     283.918
x4           281.7199     41.733      6.751      0.000     199.925     363.515
x5           236.9423     45.814      5.172      0.000     147.148     326.737
x6           505.6492     46.677     10.833      0.000     414.165     597.134
x7           320.4790     51.135      6.267      0.000     220.256     420.701
x8          -161.1987     20.535     -7.850      0.000    -201.447    -120.951
x9          -182.7126     24.047     -7.598      0.000    -229.844    -135.581
x10           24.0936     22.931      1.051      0.293     -20.851      69.038
x11          -44.1248     22.024     -2.004      0.045     -87.290      -0.959
x12           76.5931     25.807      2.968      0.003      26.013     127.174
x13         -138.8301     27.826     -4.989      0.000    -193.368     -84.292
x14         -631.7454     17.163    -36.808      0.000    -665.384    -598.106
x15         -471.0299     15.221    -30.946      0.000    -500.862    -441.197
x16            6.9117      4.940      1.399      0.162      -2.771      16.594
x17          -37.8679     13.791     -2.746      0.006     -64.898     -10.837
x18         5096.3202     15.186    335.600      0.000    5066.557    5126.084
x19        -2158.8412     13.822   -156.184      0.000   -2185.933   -2131.750
==============================================================================
Omnibus:                    24688.718   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            34278.762
Skew:                           0.526   Prob(JB):                         0.00
Kurtosis:                       3.899   Cond. No.                         44.1
==============================================================================
```

Output

Results:

The take outs from Regression Model are -

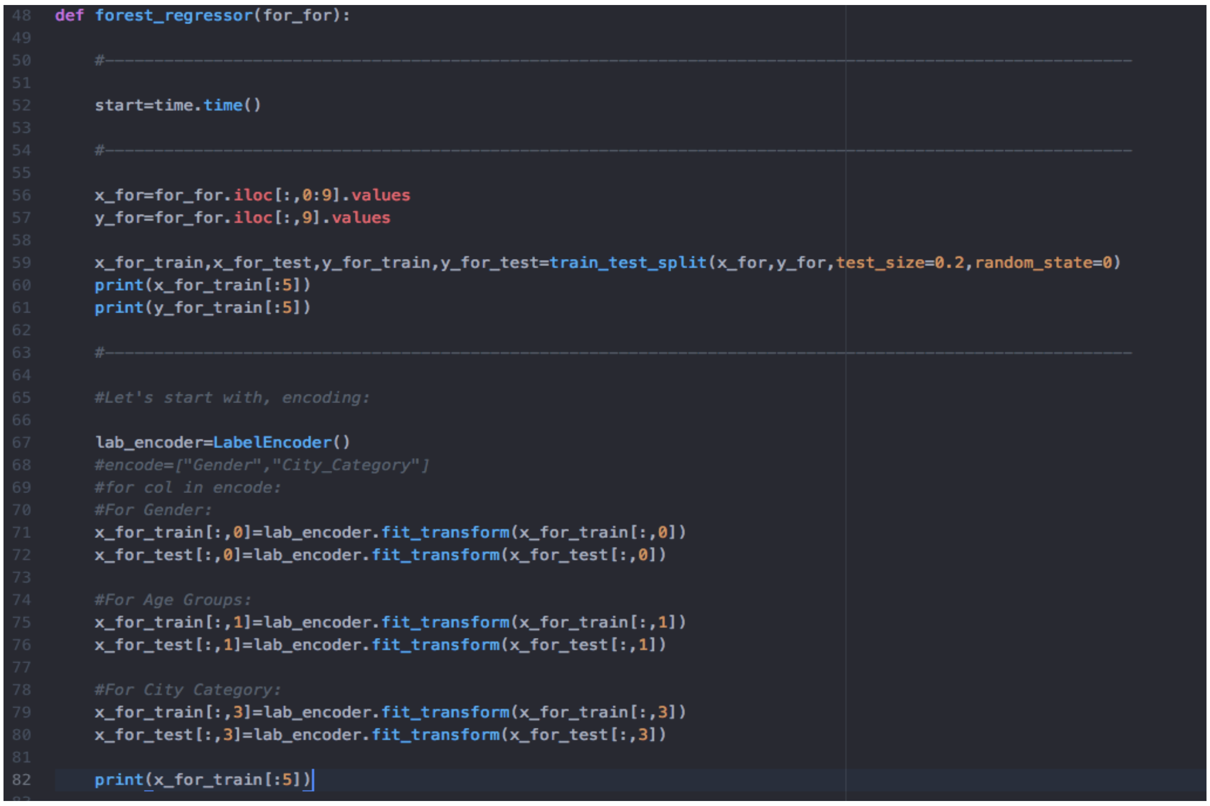
1. There were 3 explanatory variables having a p-value higher than 0.05, thus showing they are less significant.
2. The value of R-squared and adjusted R-squared was 0.3.
3. For this data set, males tend to spend more than females.
4. Regression shows that every additional year in the city leads to a decrease in the amount of purchase.
5. Spending pattern over age is not plainly increasing or decreasing.

Random Forest Regressor:

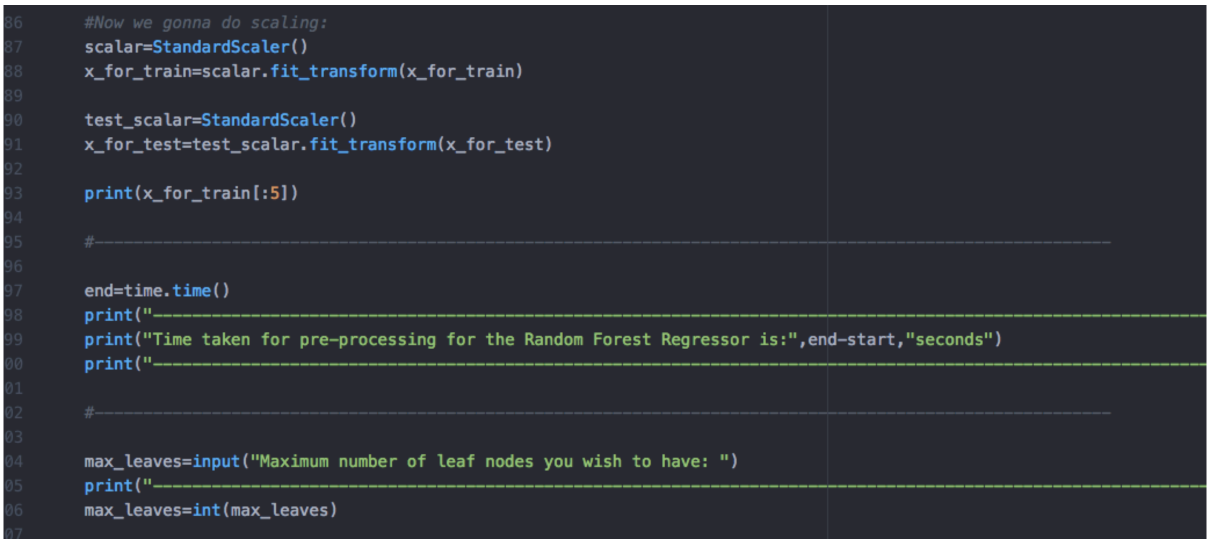
The random forest regressor model will make a forest out of a certain number of decision trees. The decision trees are made by pulling out random samples of data from input data. The size of sample stays the same, but bootstrapping enables the model to pull different samples for making different trees.

The trees have a specified number of maximum leaf nodes. These are the maximum number of data points to be assigned to one leaf node. The data is classified into trees by finding patterns in the data. Data points showcasing similar patterns in explanatory and target variable end up being assigned to the same leaf node. What random forest regressor will do is that when it is used to finally predict for an unknown, never seen before data record, it will start traversing from the root node and follow the branches on the basis of the value of the explanatory variables from the data entry that is used for prediction. It will eventually reach one of the leaf nodes based on the values of these variables, thus classifying the data entry into a group showcasing similar patterns. Eventually, an average of the values of target variables for all the data points assigned to that particular leaf node is given to the new data point as a result of prediction.

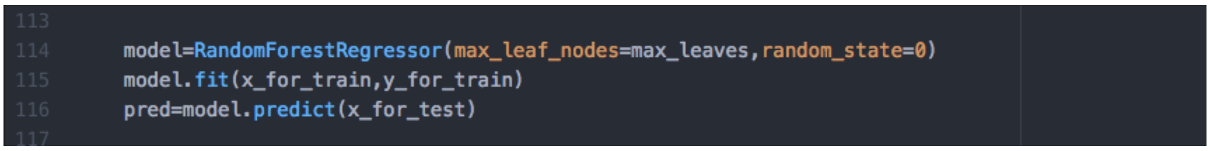
We wrote a function for random forest regressor that took a data frame as its input. The data frame from earlier section (cleaned data) is passed to this function.



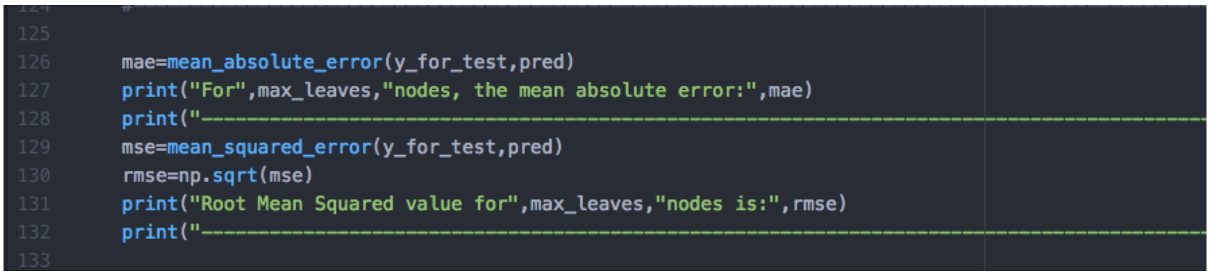
```python
def forest_regressor(for_for):

    #-----------------------------------------------------------------------------------------------

    start=time.time()

    #-----------------------------------------------------------------------------------------------

    x_for=for_for.iloc[:,0:9].values
    y_for=for_for.iloc[:,9].values

    x_for_train,x_for_test,y_for_train,y_for_test=train_test_split(x_for,y_for,test_size=0.2,random_state=0)
    print(x_for_train[:5])
    print(y_for_train[:5])

    #-----------------------------------------------------------------------------------------------

    #Let's start with, encoding:

    lab_encoder=LabelEncoder()
    #encode=["Gender","City_Category"]
    #for col in encode:
    #For Gender:
    x_for_train[:,0]=lab_encoder.fit_transform(x_for_train[:,0])
    x_for_test[:,0]=lab_encoder.fit_transform(x_for_test[:,0])

    #For Age Groups:
    x_for_train[:,1]=lab_encoder.fit_transform(x_for_train[:,1])
    x_for_test[:,1]=lab_encoder.fit_transform(x_for_test[:,1])

    #For City Category:
    x_for_train[:,3]=lab_encoder.fit_transform(x_for_train[:,3])
    x_for_test[:,3]=lab_encoder.fit_transform(x_for_test[:,3])

    print(x_for_train[:5])
```



```
#Now we gonna do scaling:
scalar=StandardScaler()
x_for_train=scalar.fit_transform(x_for_train)

test_scalar=StandardScaler()
x_for_test=test_scalar.fit_transform(x_for_test)

print(x_for_train[:5])

#--------------------------------------------------------------------------------------------------


end=time.time()
print("------------------------------------------------------------------------------------------------
print("Time taken for pre-processing for the Random Forest Regressor is:",end-start,"seconds")
print("------------------------------------------------------------------------------------------------

#--------------------------------------------------------------------------------------------------

max_leaves=input("Maximum number of leaf nodes you wish to have: ")
print("------------------------------------------------------------------------------------------------
max_leaves=int(max_leaves)
```



```
model=RandomForestRegressor(max_leaf_nodes=max_leaves,random_state=0)
model.fit(x_for_train,y_for_train)
pred=model.predict(x_for_test)
```

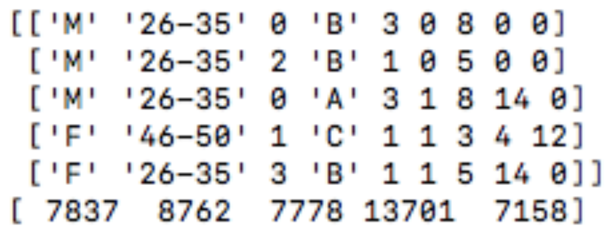


```
mae=mean_absolute_error(y_for_test,pred)
print("For",max_leaves,"nodes, the mean absolute error:",mae)
print("-----------------------------------------------------------------------------------------
mse=mean_squared_error(y_for_test,pred)
rmse=np.sqrt(mse)
print("Root Mean Squared value for",max_leaves,"nodes is:",rmse)
print("-----------------------------------------------------------------------------------------
```

The following snippet of codes show the pre-processing involved for the random forest regressor, the implementation of the model and the calculation of metrics involved in evaluating the accuracy of the model.

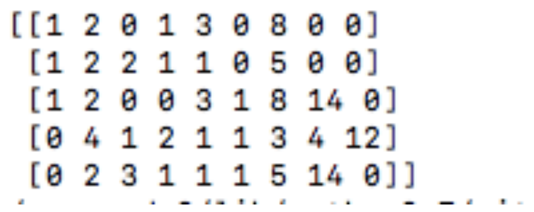
The first step is to separate the input data into explanatory and target variables, done here by splitting the data into 2 numpy arrays, ‘x’ and ‘y’. Then we use the train_test_split function from sklearn package’s model_selection class to split these 2 arrays into 2 subsections, one for training the model and another for testing its performance. We chose to split the data 80:20, i.e. 80% training and 20% testing. We are choosing the data not to be sampled randomly.

Here's a look at the raw split training data (both explanatory and target):



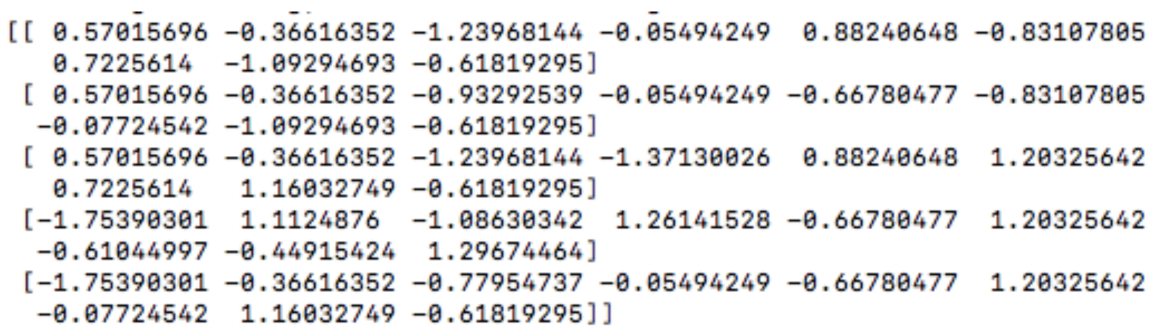
```
[['M' '26-35' 0 'B' 3 0 8 0 0]
 ['M' '26-35' 2 'B' 1 0 5 0 0]
 ['M' '26-35' 0 'A' 3 1 8 14 0]
 ['F' '46-50' 1 'C' 1 1 3 4 12]
 ['F' '26-35' 3 'B' 1 1 5 14 0]]
[ 7837  8762  7778 13701  7158]
```

We then used LabelEncoder so that the categories in our data get encoded into simpler labels. LabelEncoder is a function from sklearn package's preprocessing class. The simpler labels are general numbers starting from 0 and so forth. The labels are assigned as per the comparision of models. For eg, for age, 0-17 will be assigned as 0, since the starting of this bin is 0 (lowest bin) and so on. We used LabelEncoder on 3 columns, 0,1 and 3 which are gender, age and city category. The outcome of label encoding is as follows:



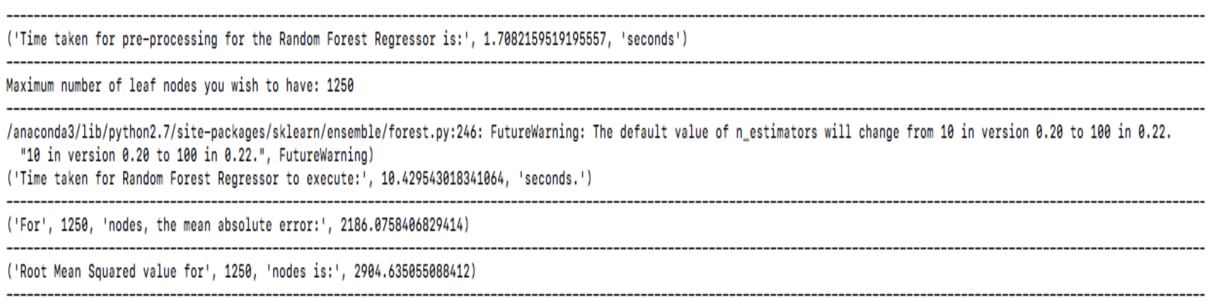
```
[[1 2 0 1 3 0 8 0 0]
 [1 2 2 1 1 0 5 0 0]
 [1 2 0 0 3 1 8 14 0]
 [0 4 1 2 1 1 3 4 12]
 [0 2 3 1 1 1 5 14 0]]
```

Finally, we decided to scale the data. Although, there is not a drastic change in the range of data, the 12s and 14s (and similar data), may drive the decisions comparatively more than the other variables. So, we have used the StandardScaler from the sklearn's preprocessing class in order to scale the data to have mean 0 and standard deviation 1, for every column individually, thus maintaining the data's characteristics of variance. The output is as follows:



```
[[ 0.57015696 -0.36616352 -1.23968144 -0.05494249  0.88240648 -0.83107805
   0.7225614  -1.09294693 -0.61819295]
 [ 0.57015696 -0.36616352 -0.93292539 -0.05494249 -0.66780477 -0.83107805
  -0.07724542 -1.09294693 -0.61819295]
 [ 0.57015696 -0.36616352 -1.23968144 -1.37130026  0.88240648  1.20325642
   0.7225614   1.16032749 -0.61819295]
 [-1.75390301  1.1124876  -1.08630342  1.26141528 -0.66780477  1.20325642
  -0.61044997 -0.44915424  1.29674464]
 [-1.75390301 -0.36616352 -0.77954737 -0.05494249 -0.66780477  1.20325642
  -0.07724542  1.16032749 -0.61819295]]
```

Finally, we make an object instance of Random Forest regressor and we use the training explanatory and target arrays to fit the model. Then we use the testing explanatory array to make predictions. We resorted to evaluating 3 types of forecast errors; mean absolute error, mean squared error and root mean squared error. We chose to make our model more user driven so that we can see the change in performance by tuning a few parameters. Thus, we made a functionality where the user can enter the maximum number of leaf nodes he or she wants. By default, the algorithm will make a forest with 100 trees (that's the default value of n_estimators). Bootstrapping is used as a default of the algorithm and we didn't see any point in not having bootstrapping, as it will help create a more generalized model due to the randomness. The default criterion is 'mse' (mean squared error) and the algorithm will try making a model that minimizes the mean squared error. The output in pictures is for a forest with all the trees having 1250 number of maximum leaf nodes:



```
----------------------------------------------------------------------------------------------------------------------------------
('Time taken for pre-processing for the Random Forest Regressor is:', 1.7082159519195557, 'seconds')
----------------------------------------------------------------------------------------------------------------------------------
Maximum number of leaf nodes you wish to have: 1250
----------------------------------------------------------------------------------------------------------------------------------
/anaconda3/lib/python2.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
('Time taken for Random Forest Regressor to execute:', 10.429543018341064, 'seconds.')
----------------------------------------------------------------------------------------------------------------------------------
('For', 1250, 'nodes, the mean absolute error:', 2186.0758406829414)
----------------------------------------------------------------------------------------------------------------------------------
('Root Mean Squared value for', 1250, 'nodes is:', 2904.635055088412)
----------------------------------------------------------------------------------------------------------------------------------
```

Challenges and tweaked approach:

No major challenges were faced when carrying out the building and execution of this model. Reason behind this is the code was pretty straight forward. Understanding the concept and thinking of the data munging was a challenge. We tried making the code as dynamic as possible, but we couldn't make the code generalized enough for encoding; we had to hard code the column names and so forth.

Looking back, we think that we may have been able to create a better model had we used some different components of data. We interpreted this as a forest of decision trees making decision on the basis of patterns in the explanatory variables and hence, despite knowing that the labels in product categories are not ordered, we chose to use them as they still identify product categories distinctly. But we think that there is a possibility that rather than using the product category columns, the model would have performed better, had we used the dummy variables we made for the linear regression model.

Random Forest Classifier:

For most of the parts, random forest classifier works same as the random forest regressor. The modeling technique will make a forest of a certain number of trees that would map the pattern in the explanatory variables. The broad difference is that rather than having an average of the target variable as the value of the node, the classifier requires binning or discretizing the numeric target variable and eventually, it will classify data points on the basis of the explanatory variables and a given number of maximum data points will be assigned to a leaf node. The mode of the bins of these data points is assigned as the resultant category for this leaf node.

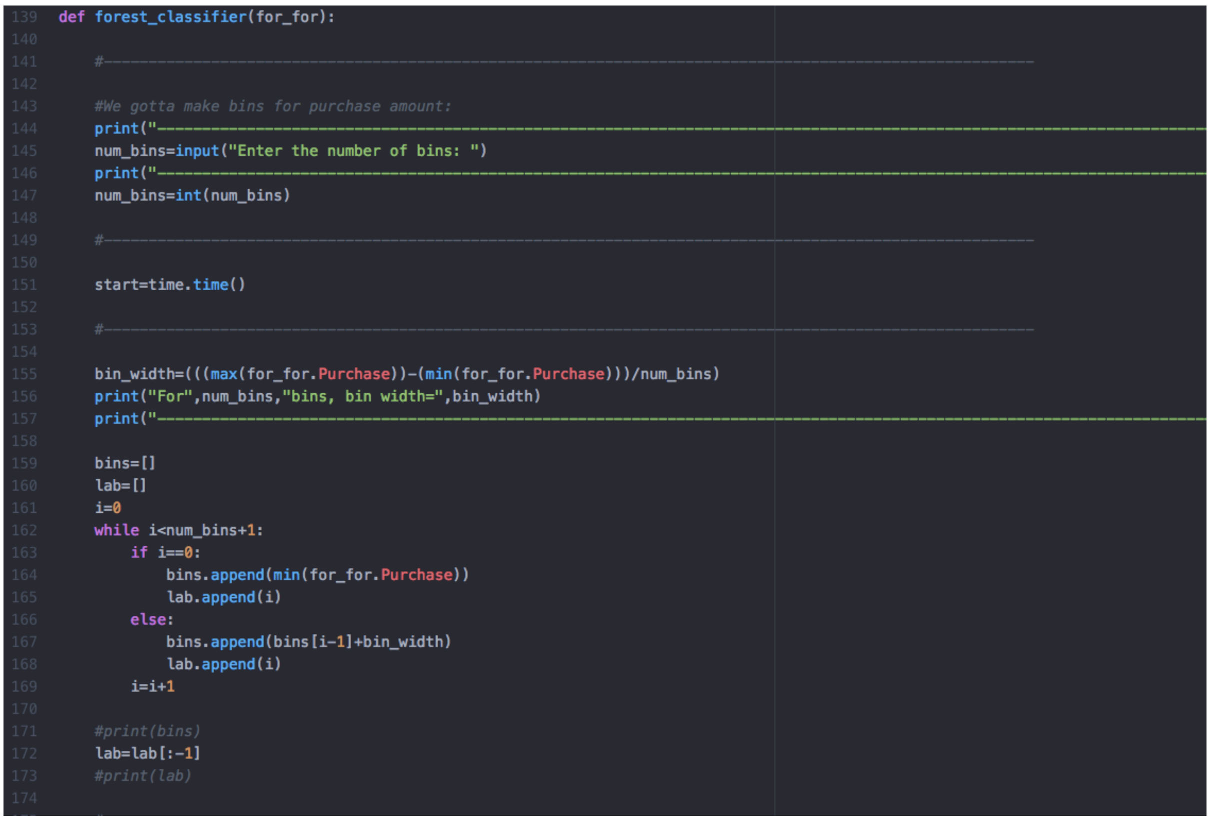
We strived and succeeded in making the model user driven. The system will ask for number of bins the user wishes to discretize the 'Purchase' column into. Based on the number entered, the system will calculate the bin width by using the maximum and minimum value from the target column. After getting the width and the number of bins, we make 2 lists, one with bin start value and one with respective bin label. The we used the 'cut' function from pandas which will take the array or data-frame to be discretized, the bin start point list and the bin label list. We then used the LabelEncoder on the same column as we used it on in the regressor model, i.e. 0,1 and 3; gender, age and city category respectively. Finally, we used Standard Scaler on the data.

Running a Random Forest Classifier requires us to initialize model instance. We than fit the model on the training data (both explanatory and discretized target). Finally, we will use the trained model and testing explanatory data to make predictions. In order to evaluate the model's accuracy we used 3 parameter; confusion matrix, classification report and accuracy score. All these are implemented using confusion_matrix, classification_report, accuracy_score functions from sklearn package's metrics class.

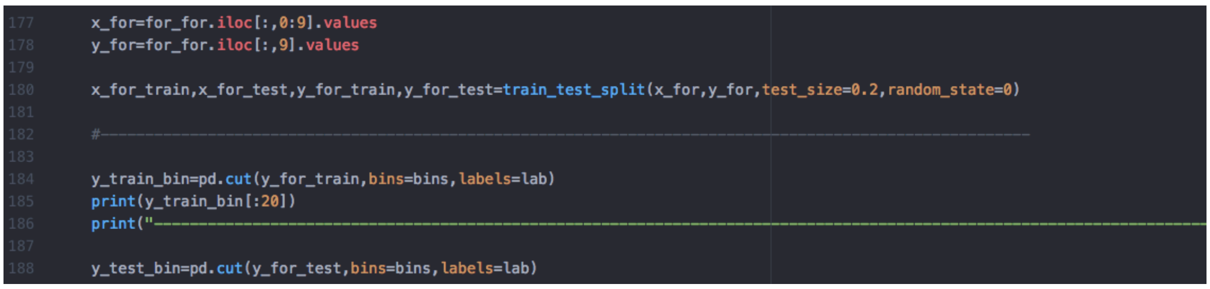
A bit about the Random Forest Classifier function: We have made the maximum number of leaf nodes parameter to be user driven so that the user can actually see the variation in performance of model when this and number of bins parameter is tuned. We used a different number of trees parameter (n_estimators) than the default one. We fixed it to be 150. However, we can also make it user driven, it is a matter of 2 lines of code. The criterion used is 'Gini Index'. Another available criterion was 'information gain' (Entropy). Both are a measure of impurities. Entropy will be 0 if all the nodes belong to 1 category (Entropy is the change or variance) and will be the highest of all the nodes are evenly distributed over different categories. Exact same goes for Gini Index. Gini Index is faster, due to the mathematical ease of computation. Both give more or less the same output.

$$Entropy = -\sum_j p_j \log_2 p_j$$

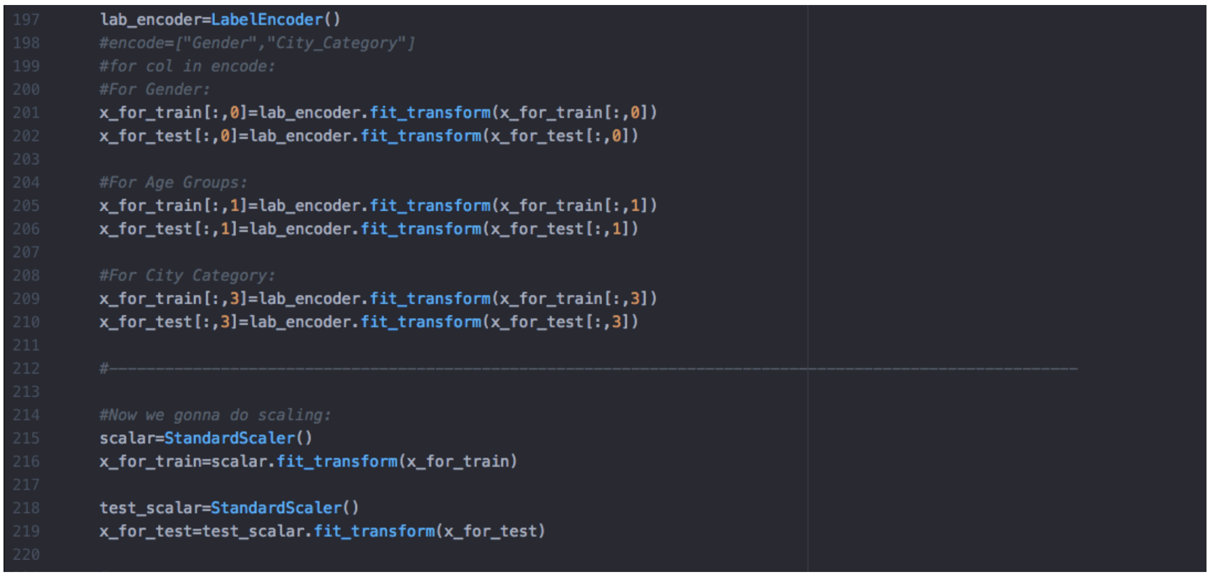
$$Gini = 1 - \sum_j p_j^2$$



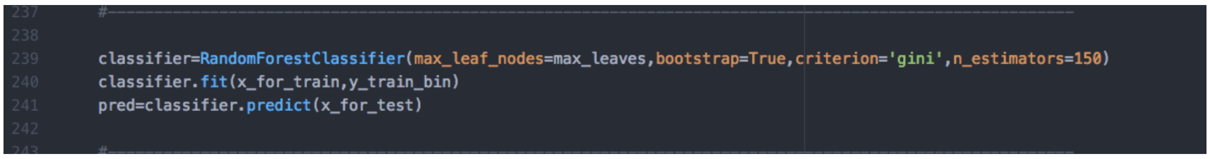
```
def forest_classifier(for_for):

    #-----------------------------------------------------------------------------------------------

    #We gotta make bins for purchase amount:
    print("--------------------------------------------------------------------------------------------------------------------
    num_bins=input("Enter the number of bins: ")
    print("--------------------------------------------------------------------------------------------------------------------
    num_bins=int(num_bins)

    #-----------------------------------------------------------------------------------------------

    start=time.time()

    #-----------------------------------------------------------------------------------------------

    bin_width=(((max(for_for.Purchase))-(min(for_for.Purchase)))/num_bins)
    print("For",num_bins,"bins, bin width=",bin_width)
    print("--------------------------------------------------------------------------------------------------------------------

    bins=[]
    lab=[]
    i=0
    while i<num_bins+1:
        if i==0:
            bins.append(min(for_for.Purchase))
            lab.append(i)
        else:
            bins.append(bins[i-1]+bin_width)
            lab.append(i)
        i=i+1

    #print(bins)
    lab=lab[:-1]
    #print(lab)

```



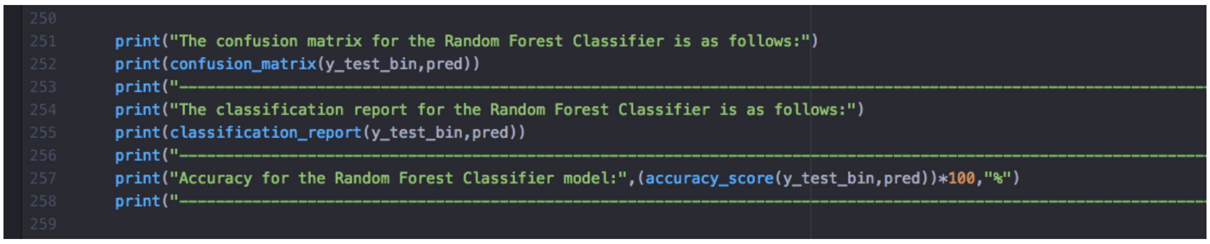
```
    x_for=for_for.iloc[:,0:9].values
    y_for=for_for.iloc[:,9].values

    x_for_train,x_for_test,y_for_train,y_for_test=train_test_split(x_for,y_for,test_size=0.2,random_state=0)

    #-----------------------------------------------------------------------------------------------

    y_train_bin=pd.cut(y_for_train,bins=bins,labels=lab)
    print(y_train_bin[:20])
    print("--------------------------------------------------------------------------------------------------------------------

    y_test_bin=pd.cut(y_for_test,bins=bins,labels=lab)
```



```python
lab_encoder=LabelEncoder()
#encode=["Gender","City_Category"]
#for col in encode:
#For Gender:
x_for_train[:,0]=lab_encoder.fit_transform(x_for_train[:,0])
x_for_test[:,0]=lab_encoder.fit_transform(x_for_test[:,0])

#For Age Groups:
x_for_train[:,1]=lab_encoder.fit_transform(x_for_train[:,1])
x_for_test[:,1]=lab_encoder.fit_transform(x_for_test[:,1])

#For City Category:
x_for_train[:,3]=lab_encoder.fit_transform(x_for_train[:,3])
x_for_test[:,3]=lab_encoder.fit_transform(x_for_test[:,3])

#--------------------------------------------------------------------------------------------------------

#Now we gonna do scaling:
scalar=StandardScaler()
x_for_train=scalar.fit_transform(x_for_train)

test_scalar=StandardScaler()
x_for_test=test_scalar.fit_transform(x_for_test)
```



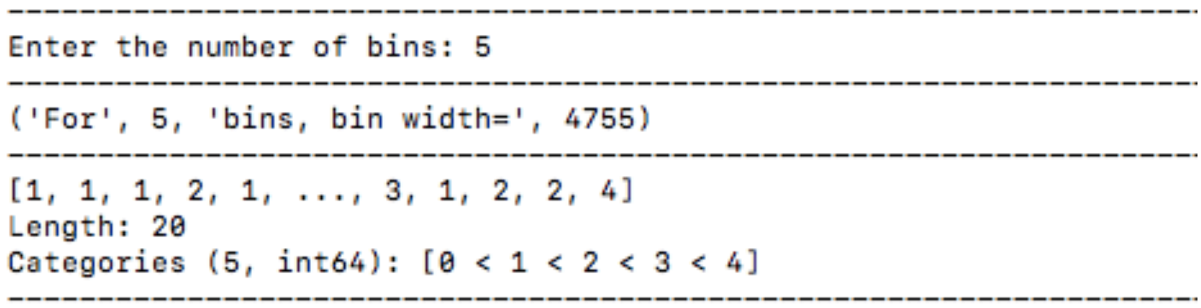
```python
#--------------------------------------------------------------------------------------------------------

classifier=RandomForestClassifier(max_leaf_nodes=max_leaves,bootstrap=True,criterion='gini',n_estimators=150)
classifier.fit(x_for_train,y_train_bin)
pred=classifier.predict(x_for_test)

#--------------------------------------------------------------------------------------------------------
```



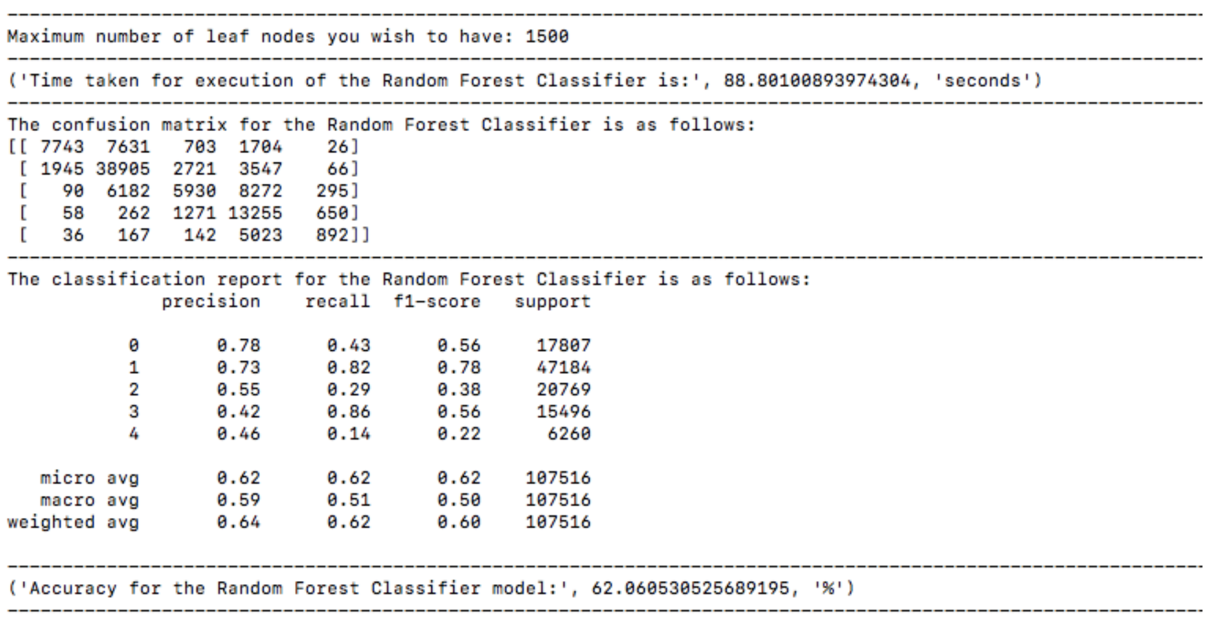
```python
print("The confusion matrix for the Random Forest Classifier is as follows:")
print(confusion_matrix(y_test_bin,pred))
print("------------------------------------------------------------------------------------------------------
print("The classification report for the Random Forest Classifier is as follows:")
print(classification_report(y_test_bin,pred))
print("------------------------------------------------------------------------------------------------------
print("Accuracy for the Random Forest Classifier model:",(accuracy_score(y_test_bin,pred))*100,"%")
print("------------------------------------------------------------------------------------------------------
```

Output:

The output is for 5 bins, 1500 maximum number of leaf nodes, 150 trees in forest and bootstrapped model:



```
------------------------------------------------------------------
Enter the number of bins: 5
------------------------------------------------------------------
('For', 5, 'bins, bin width=', 4755)
------------------------------------------------------------------
[1, 1, 1, 2, 1, ..., 3, 1, 2, 2, 4]
Length: 20
Categories (5, int64): [0 < 1 < 2 < 3 < 4]
------------------------------------------------------------------
```



```
------------------------------------------------------------------------------------------------------------
Maximum number of leaf nodes you wish to have: 1500
------------------------------------------------------------------------------------------------------------
('Time taken for execution of the Random Forest Classifier is:', 88.80100893974304, 'seconds')
------------------------------------------------------------------------------------------------------------
The confusion matrix for the Random Forest Classifier is as follows:
[[ 7743  7631   703  1704    26]
 [ 1945 38905  2721  3547    66]
 [   90  6182  5930  8272   295]
 [   58   262  1271 13255   650]
 [   36   167   142  5023   892]]
------------------------------------------------------------------------------------------------------------
The classification report for the Random Forest Classifier is as follows:
              precision    recall  f1-score   support

           0       0.78      0.43      0.56     17807
           1       0.73      0.82      0.78     47184
           2       0.55      0.29      0.38     20769
           3       0.42      0.86      0.56     15496
           4       0.46      0.14      0.22      6260

   micro avg       0.62      0.62      0.62    107516
   macro avg       0.59      0.51      0.50    107516
weighted avg       0.64      0.62      0.60    107516

------------------------------------------------------------------------------------------------------------
('Accuracy for the Random Forest Classifier model:', 62.060530525689195, '%')
------------------------------------------------------------------------------------------------------------
```

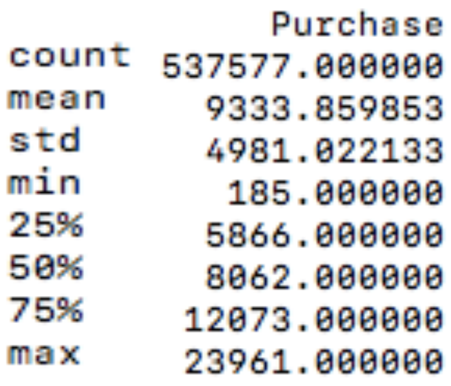
The accuracy obtained is 62.06%. However, the highest accuracy we got was 62.18% for the same setting, except that we had ‘information gain’ criterion instead of ‘Gini Index’ criterion.

Challenges and setbacks:

This model was not only the best in all the models we implemented but was also the most complicated and challenging one. We faced a couple of challenges that required some obvious yet oblivious solutions. Initially, we tried fitting the model by making 20 bins. This was when everything was still in development and we had to hard code the number of models. Thus, we couldn’t run multiple iterations for different number of models. But even before we could successfully execute the model for 20 bins, we were faced with an error:

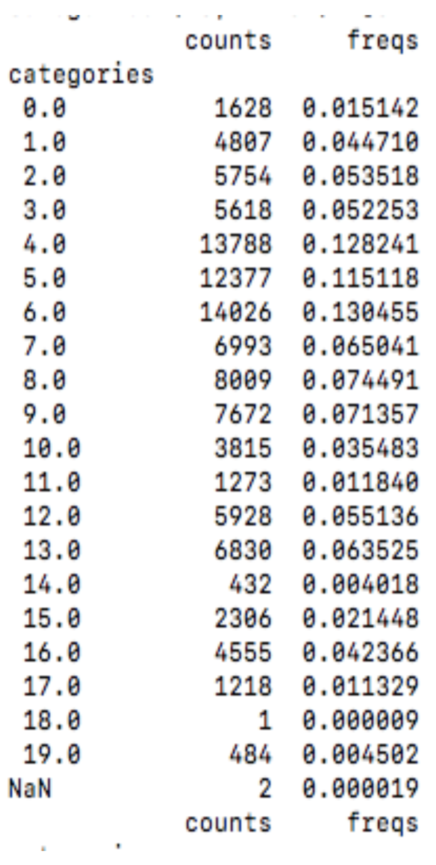
ValueError: Input contains NaN, infinity or a value too large for dtype('float64’).

We couldn’t get the complete meaning of the error at first. After some thinking, we found out that the binning was faulty. So we had a look at the bins and the maximum number in the ‘Purchase’ column:

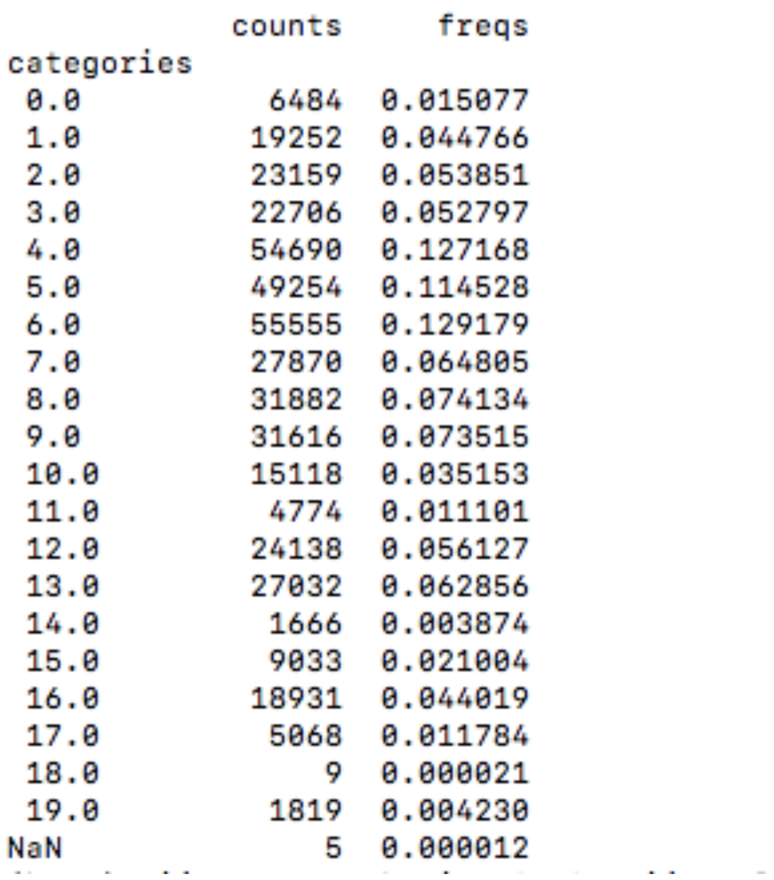


```
            Purchase
count  537577.000000
mean     9333.859853
std      4981.022133
min       185.000000
25%      5866.000000
50%      8062.000000
75%     12073.000000
max     23961.000000
```

The problem was that the bin width was 1188.8. There were a few data points that were higher than the closing value of the last bin, by only a slight decimal margin. As a result, the ‘cut’ function couldn’t put them in a bin and thus labeled them to have a bin value NaN. And since the Random Forest Classifier can’t handle NaN values in the target variable, we were getting these errors.

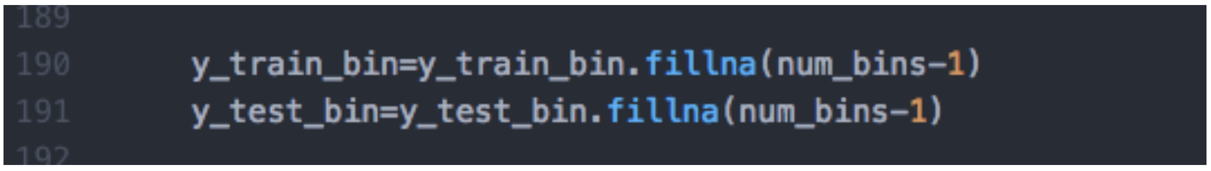


```
            counts     freqs
categories
 0.0          1628  0.015142
 1.0          4807  0.044710
 2.0          5754  0.053518
 3.0          5618  0.052253
 4.0         13788  0.128241
 5.0         12377  0.115118
 6.0         14026  0.130455
 7.0          6993  0.065041
 8.0          8009  0.074491
 9.0          7672  0.071357
 10.0         3815  0.035483
 11.0         1273  0.011840
 12.0         5928  0.055136
 13.0         6830  0.063525
 14.0          432  0.004018
 15.0         2306  0.021448
 16.0         4555  0.042366
 17.0         1218  0.011329
 18.0            1  0.000009
 19.0          484  0.004502
NaN              2  0.000019
            counts     freqs
```



```
            counts     freqs
categories
 0.0          6484  0.015077
 1.0         19252  0.044766
 2.0         23159  0.053851
 3.0         22706  0.052797
 4.0         54690  0.127168
 5.0         49254  0.114528
 6.0         55555  0.129179
 7.0         27870  0.064805
 8.0         31882  0.074134
 9.0         31616  0.073515
 10.0        15118  0.035153
 11.0         4774  0.011101
 12.0        24138  0.056127
 13.0        27032  0.062856
 14.0         1666  0.003874
 15.0         9033  0.021004
 16.0        18931  0.044019
 17.0         5068  0.011784
 18.0            9  0.000021
 19.0         1819  0.004230
NaN              5  0.000012
```

So, we just wrote 2 lines of code that filled the NaN values (using fillna function) and the NaN were replaced with the last bin label, thus coercing them into the last bin.



```
    y_train_bin=y_train_bin.fillna(num_bins-1)
    y_test_bin=y_test_bin.fillna(num_bins-1)
```

Then we could finally execute the model. After execution, we faced another issue. The issue was low accuracy. With a setting in which we made a forest with 150 trees, 1500 maximum leaf nodes and Gini Index criterion for the data discretized into 20 bins, we got an accuracy of 34.49%.

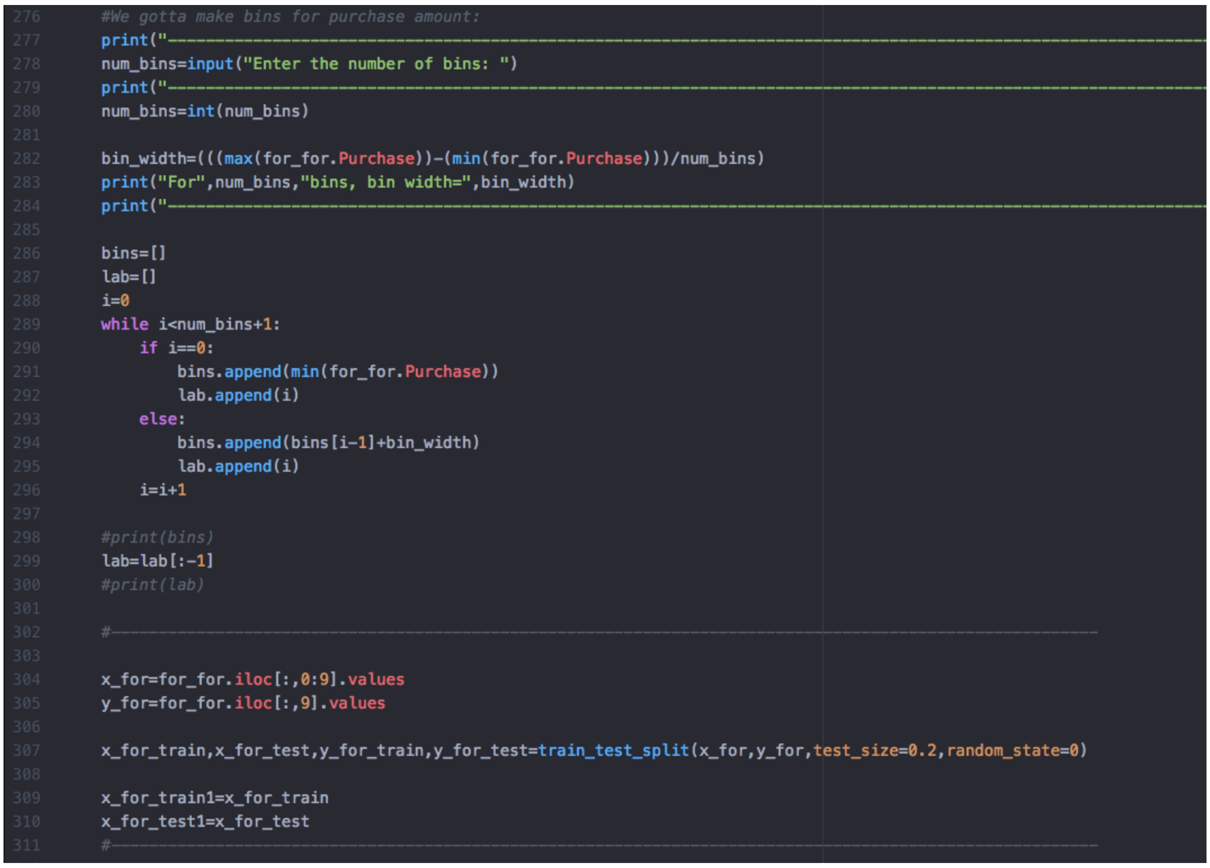
Eventually, we arrived at the conclusion that 20 bins were resulting in too specific of a model and hence the overfitting impeded the model from having a better accuracy. Thus we decided to reduce down the number of bins and did the same thing (hard coding) for 5 bins rather than 20 and our accuracy went up to what was achieved.

Naïve Bayes Model:

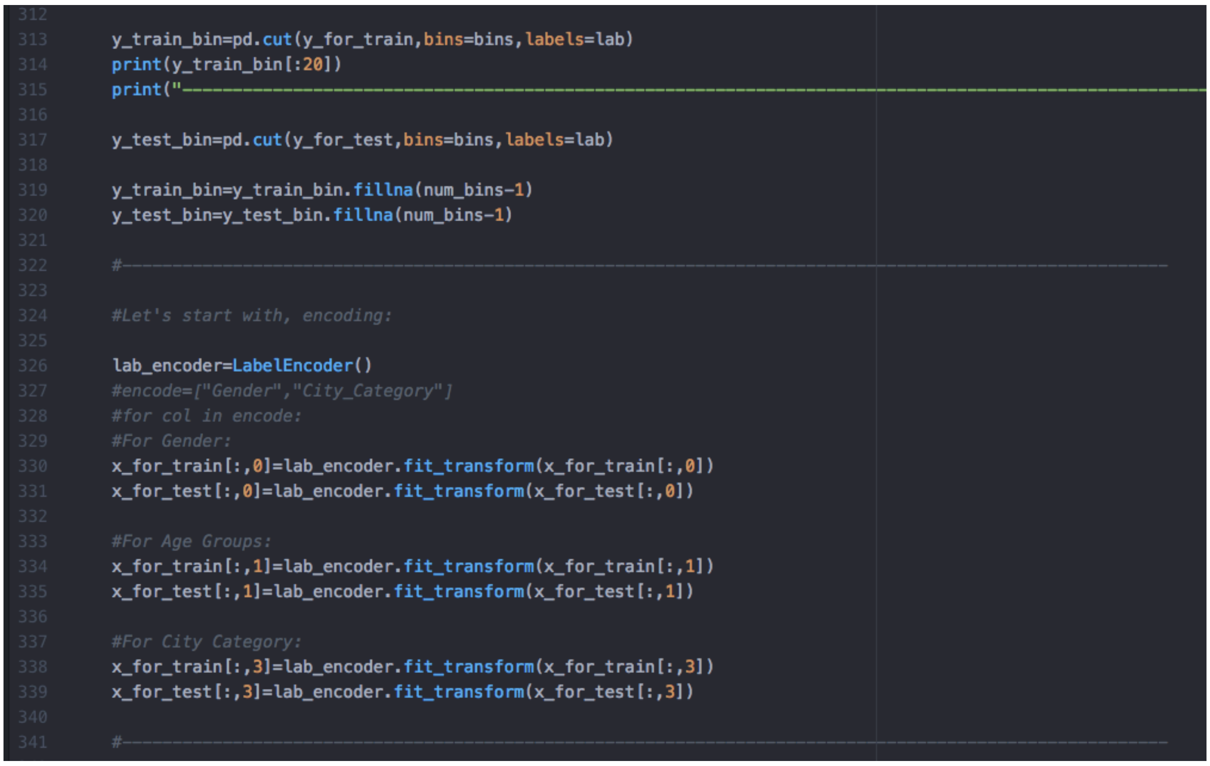
Naïve Bayes algorithm uses Bayes theorem of probability to predict the class (or in our case bin) of an unknown data set. The base assumption in Naïve Bayes method is that all the predictors or the explanatory variables are independent of each other. Naïve Bayes is very simple to implement and a quick predictive technique. The algorithm focuses on the frequency and the resulting conditional probability and likelihood of a data point belonging to a certain bin.

Naïve Bayes required the exact same pre-processing of data as Random Forest Classifier. We made Naïve Bayes as user driven as the Random Forest Classifier. The user is asked to enter the number of bins to be used to discretize the numeric ‘Purchase’ column. Then we calculate the bin width and using that we make 2 lists for bin start values and bin labels. Then we use the ‘cut’ function to discretize the column. We then use ‘fillna’ to replace the NaN values (result of slightly larger data values being binned as NaN) by the label of the last bin. The we split data into training and testing data (80:20). We make a copy of the explanatory data set before scaling.

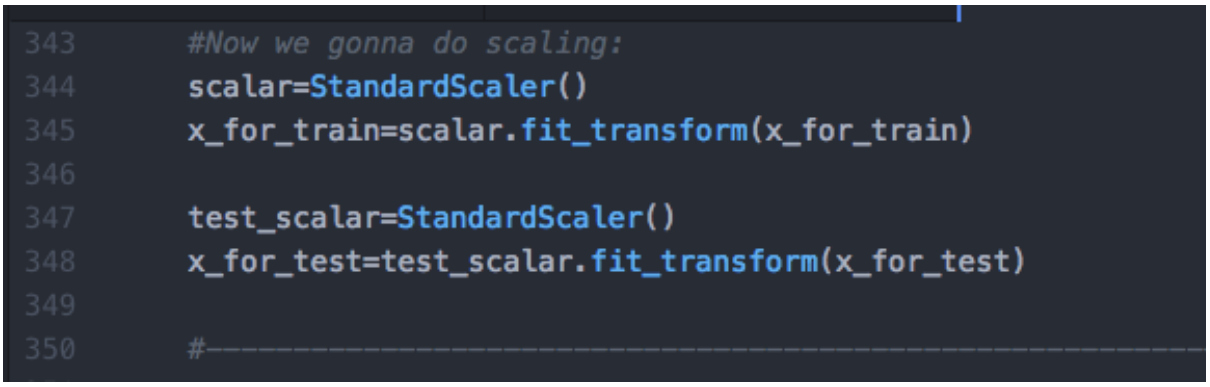
Since we are using Gaussian Naïve Bayes, which requires data to be continuous, we’d be scaling the data. Before scaling, we’d use LabelEncoder on the columns 0,1 and 3; gender, age and city category to label them. We are using GaussianNB from sklearn package’s naïve_bayes class. We create an instance for the model and fir the model onto the labeled and scaled explanatory training data and corresponding binned testing data. Then we use the same performance metrics as Random Forest Classifier to evaluate the performance of the model.



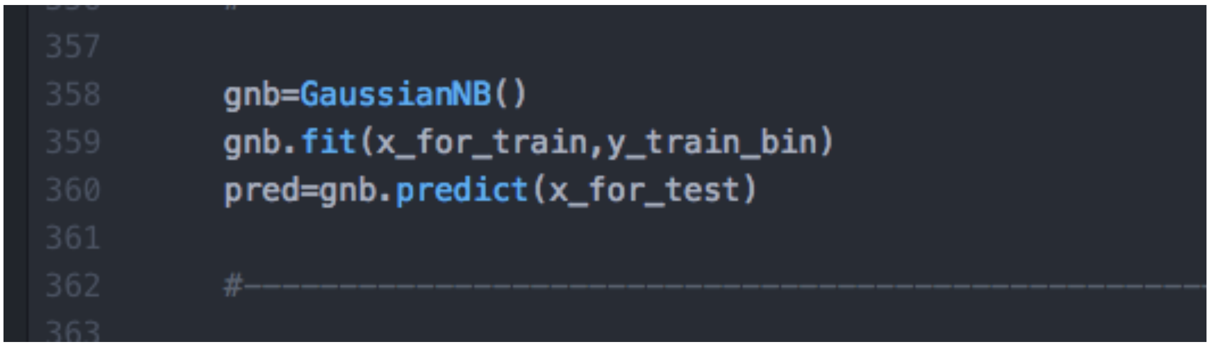
```python
#We gotta make bins for purchase amount:
print("------------------------------------------------------------------------------------------------------------
num_bins=input("Enter the number of bins: ")
print("------------------------------------------------------------------------------------------------------------
num_bins=int(num_bins)

bin_width=(((max(for_for.Purchase))-(min(for_for.Purchase)))/num_bins)
print("For",num_bins,"bins, bin width=",bin_width)
print("------------------------------------------------------------------------------------------------------------

bins=[]
lab=[]
i=0
while i<num_bins+1:
    if i==0:
        bins.append(min(for_for.Purchase))
        lab.append(i)
    else:
        bins.append(bins[i-1]+bin_width)
        lab.append(i)
    i=i+1

#print(bins)
lab=lab[:-1]
#print(lab)

#------------------------------------------------------------------------------------------------------------

x_for=for_for.iloc[:,0:9].values
y_for=for_for.iloc[:,9].values

x_for_train,x_for_test,y_for_train,y_for_test=train_test_split(x_for,y_for,test_size=0.2,random_state=0)

x_for_train1=x_for_train
x_for_test1=x_for_test
#------------------------------------------------------------------------------------------------------------
```



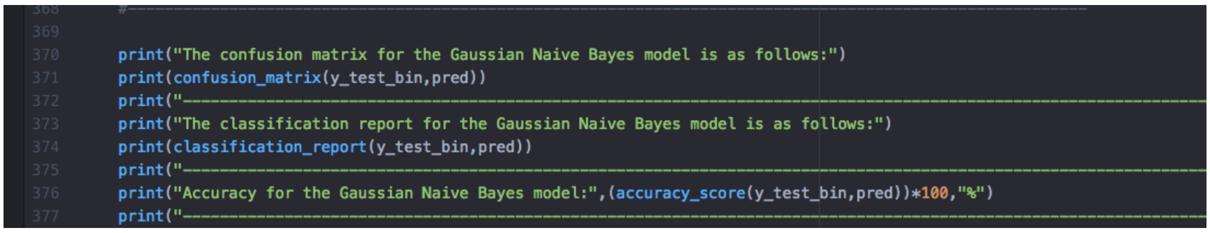
```python

y_train_bin=pd.cut(y_for_train,bins=bins,labels=lab)
print(y_train_bin[:20])
print("------------------------------------------------------------------------------------------------------------

y_test_bin=pd.cut(y_for_test,bins=bins,labels=lab)

y_train_bin=y_train_bin.fillna(num_bins-1)
y_test_bin=y_test_bin.fillna(num_bins-1)

#------------------------------------------------------------------------------------------------------------

#Let's start with, encoding:

lab_encoder=LabelEncoder()
#encode=["Gender","City_Category"]
#for col in encode:
#For Gender:
x_for_train[:,0]=lab_encoder.fit_transform(x_for_train[:,0])
x_for_test[:,0]=lab_encoder.fit_transform(x_for_test[:,0])

#For Age Groups:
x_for_train[:,1]=lab_encoder.fit_transform(x_for_train[:,1])
x_for_test[:,1]=lab_encoder.fit_transform(x_for_test[:,1])

#For City Category:
x_for_train[:,3]=lab_encoder.fit_transform(x_for_train[:,3])
x_for_test[:,3]=lab_encoder.fit_transform(x_for_test[:,3])

#------------------------------------------------------------------------------------------------------------
```



```python
#Now we gonna do scaling:
scalar=StandardScaler()
x_for_train=scalar.fit_transform(x_for_train)

test_scalar=StandardScaler()
x_for_test=test_scalar.fit_transform(x_for_test)

#-----------------------------------------------------------
```

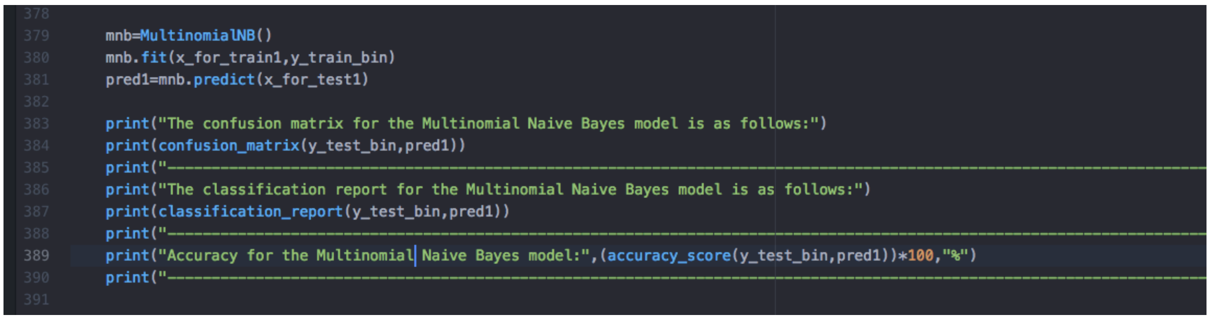


```python

gnb=GaussianNB()
gnb.fit(x_for_train,y_train_bin)
pred=gnb.predict(x_for_test)

#-------------------------------------------------
```



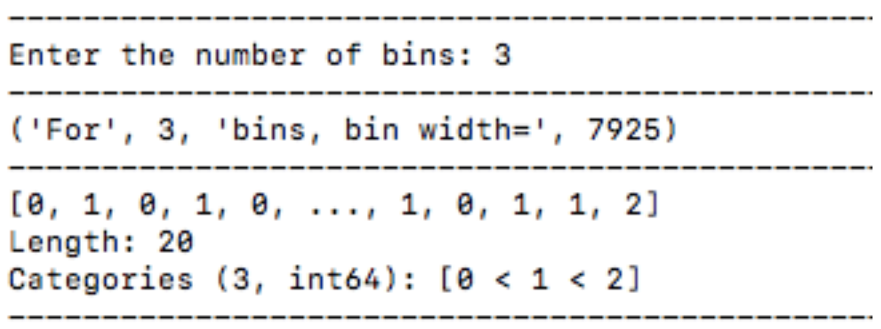
```python
#-----------------------------------------------------------------------------------------------


print("The confusion matrix for the Gaussian Naive Bayes model is as follows:")
print(confusion_matrix(y_test_bin,pred))
print("-----------------------------------------------------------------------------------------------
print("The classification report for the Gaussian Naive Bayes model is as follows:")
print(classification_report(y_test_bin,pred))
print("-----------------------------------------------------------------------------------------------
print("Accuracy for the Gaussian Naive Bayes model:",(accuracy_score(y_test_bin,pred))*100,"%")
print("-----------------------------------------------------------------------------------------------
```

As we mentioned during the final presentation, we thought that multinomial Naïve Bayes may be a better model for our data and much of it is categorical data and multinomial Naïve Bayes fits better on such data. We included multinomial Naïve Bayes functionality under the Gaussian Naïve Bayes method, using the unscaled and unlabeled data set. However, Gaussian Naïve Bayes proved to be performing better, so out original work holds better of the new work. Both the models, Gaussian and Multinomial, by default, carry out Laplace smoothing on the data.

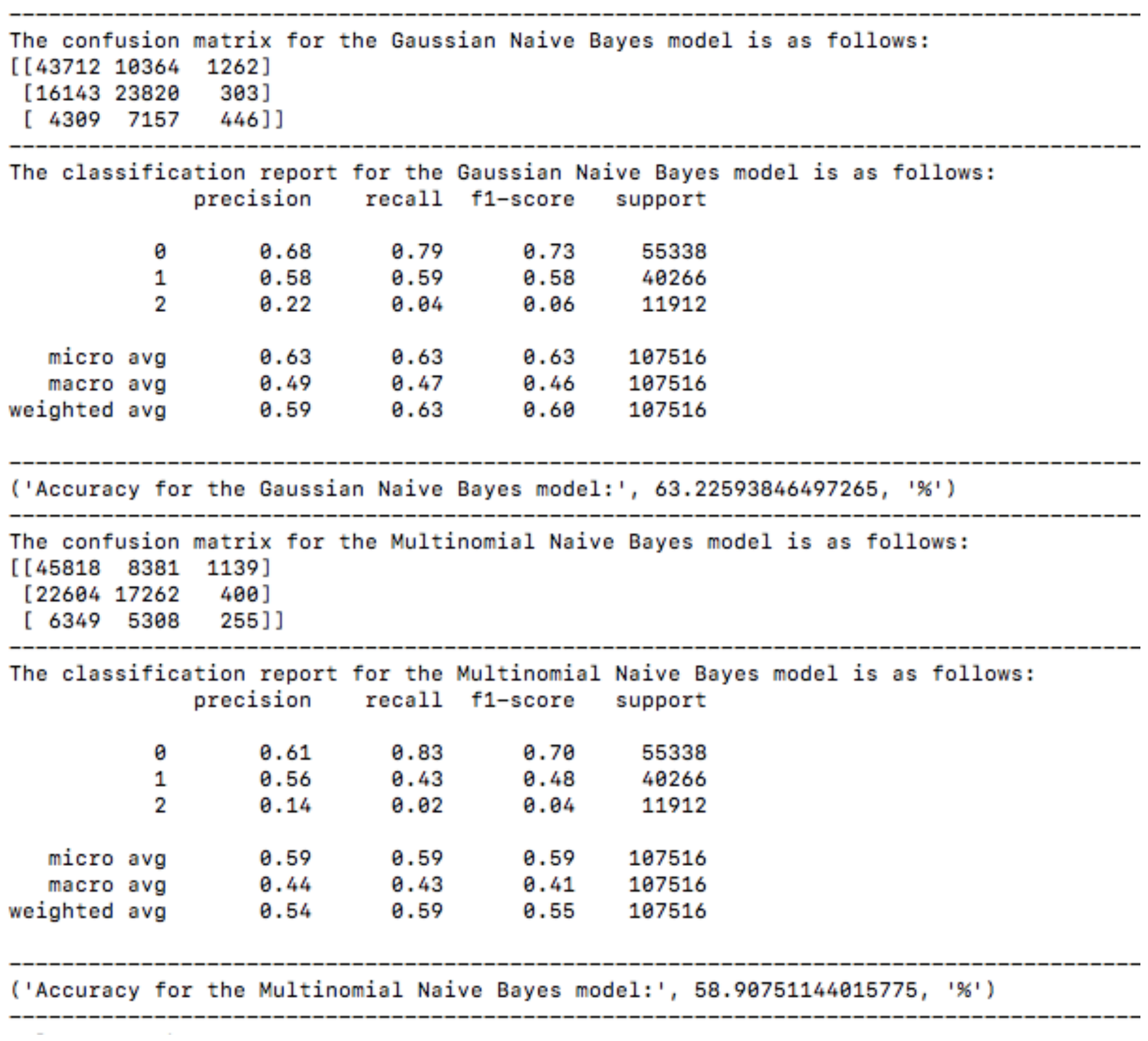


```python

mnb=MultinomialNB()
mnb.fit(x_for_train1,y_train_bin)
pred1=mnb.predict(x_for_test1)

print("The confusion matrix for the Multinomial Naive Bayes model is as follows:")
print(confusion_matrix(y_test_bin,pred1))
print("-------------------------------------------------------------------------------------------------
print("The classification report for the Multinomial Naive Bayes model is as follows:")
print(classification_report(y_test_bin,pred1))
print("-------------------------------------------------------------------------------------------------
print("Accuracy for the Multinomial Naive Bayes model:",(accuracy_score(y_test_bin,pred1))*100,"%")
print("-------------------------------------------------------------------------------------------------

```

Output of the Naïve Bayes predictor (both) for 3 bins:



```
----------------------------------------------
Enter the number of bins: 3
----------------------------------------------
('For', 3, 'bins, bin width=', 7925)
----------------------------------------------
[0, 1, 0, 1, 0, ..., 1, 0, 1, 1, 2]
Length: 20
Categories (3, int64): [0 < 1 < 2]
----------------------------------------------
```

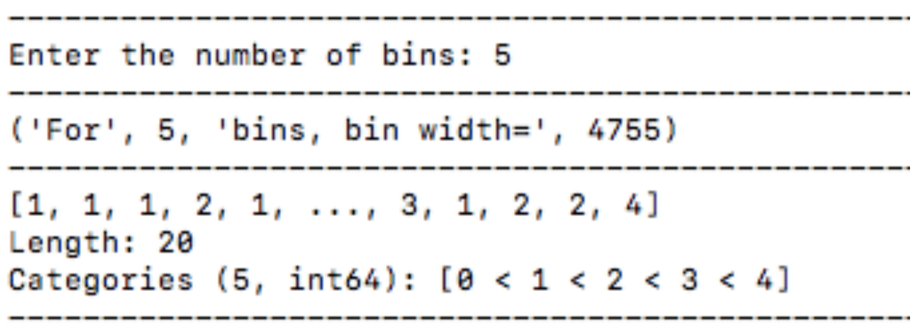


```
--------------------------------------------------------------------------------
The confusion matrix for the Gaussian Naive Bayes model is as follows:
[[43712 10364  1262]
 [16143 23820   303]
 [ 4309  7157   446]]
--------------------------------------------------------------------------------
The classification report for the Gaussian Naive Bayes model is as follows:
              precision    recall  f1-score   support

           0       0.68      0.79      0.73     55338
           1       0.58      0.59      0.58     40266
           2       0.22      0.04      0.06     11912

   micro avg       0.63      0.63      0.63    107516
   macro avg       0.49      0.47      0.46    107516
weighted avg       0.59      0.63      0.60    107516

--------------------------------------------------------------------------------
('Accuracy for the Gaussian Naive Bayes model:', 63.22593846497265, '%')
--------------------------------------------------------------------------------
The confusion matrix for the Multinomial Naive Bayes model is as follows:
[[45818  8381  1139]
 [22604 17262   400]
 [ 6349  5308   255]]
--------------------------------------------------------------------------------
The classification report for the Multinomial Naive Bayes model is as follows:
              precision    recall  f1-score   support

           0       0.61      0.83      0.70     55338
           1       0.56      0.43      0.48     40266
           2       0.14      0.02      0.04     11912

   micro avg       0.59      0.59      0.59    107516
   macro avg       0.44      0.43      0.41    107516
weighted avg       0.54      0.59      0.55    107516

--------------------------------------------------------------------------------
('Accuracy for the Multinomial Naive Bayes model:', 58.90751144015775, '%')
--------------------------------------------------------------------------------
```

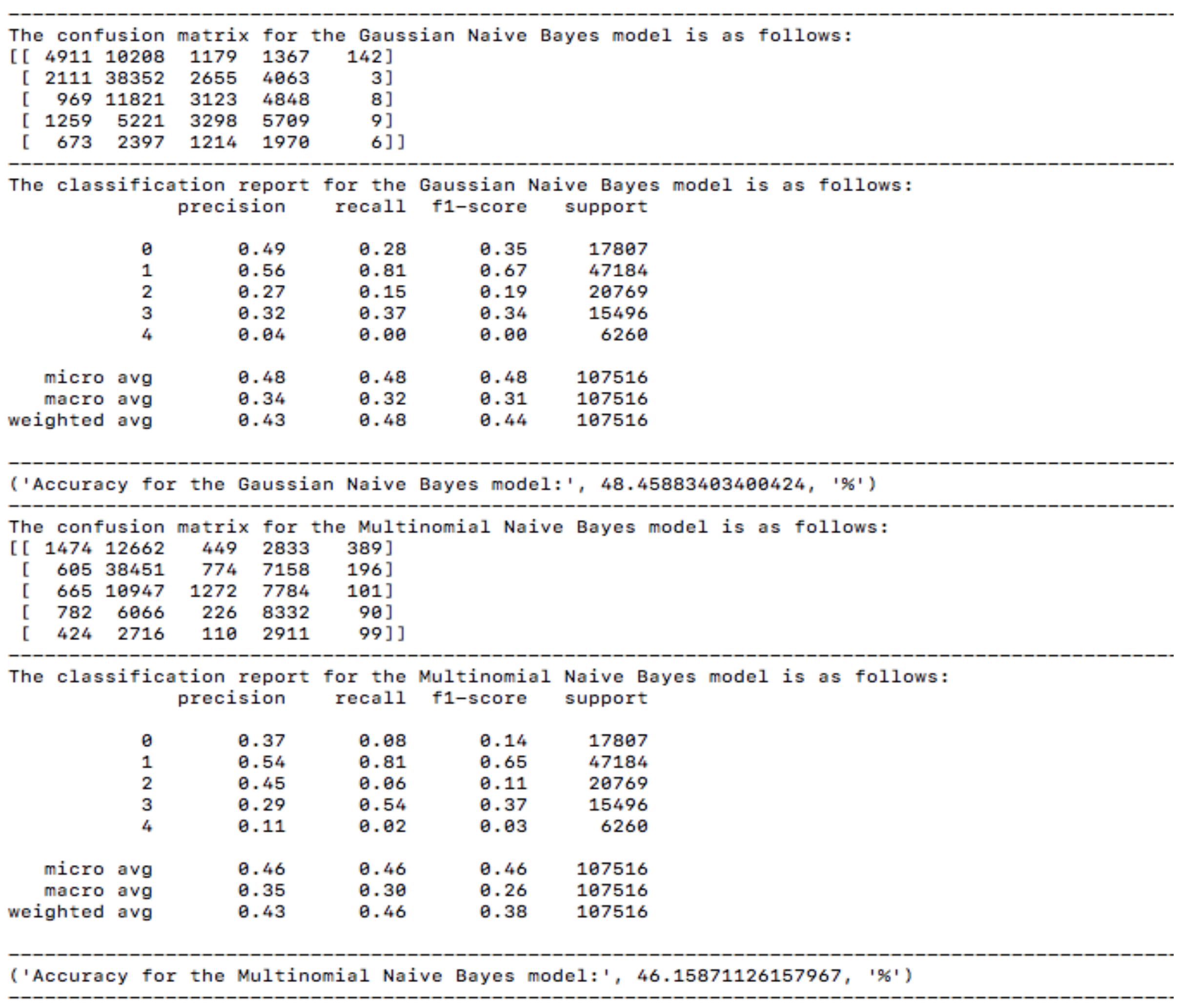
<u>Challenges and issues:</u>

We got an accuracy of 63.23%, which is the highest we have achieved so far. However, despite performing so well, Naïve Bayes model isn't that flexible when it comes to changing number of bins. When we tuned the number of bins from 3 to 5, the accuracy drops down massively to

48.46%. Now, comparing on a plain level field, Naïve Bayes for 5 bins has an accuracy of 48.46%, while, Random Forest Classifier for 5 bins had the highest accuracy of 62.18%, thus making it the better model. Apart from this issue, we didn’t face any other challenge or issue while implementing Naïve Bayes model. It is mostly because most of the procedure mirrors that of the Random Forest Classifier (pre-processing) and only the modeling code changed, which was pretty straight forward.



```
------------------------------------------------
Enter the number of bins: 5
------------------------------------------------
('For', 5, 'bins, bin width=', 4755)
------------------------------------------------
[1, 1, 1, 2, 1, ..., 3, 1, 2, 2, 4]
Length: 20
Categories (5, int64): [0 < 1 < 2 < 3 < 4]
------------------------------------------------
```



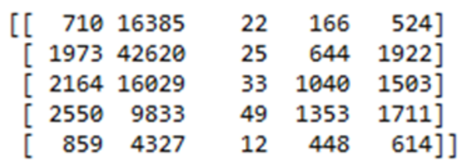
```
------------------------------------------------------------------------------------------------
The confusion matrix for the Gaussian Naive Bayes model is as follows:
[[ 4911 10208  1179  1367   142]
 [ 2111 38352  2655  4063     3]
 [  969 11821  3123  4848     8]
 [ 1259  5221  3298  5709     9]
 [  673  2397  1214  1970     6]]
------------------------------------------------------------------------------------------------
The classification report for the Gaussian Naive Bayes model is as follows:
              precision    recall  f1-score   support

           0       0.49      0.28      0.35     17807
           1       0.56      0.81      0.67     47184
           2       0.27      0.15      0.19     20769
           3       0.32      0.37      0.34     15496
           4       0.04      0.00      0.00      6260

   micro avg       0.48      0.48      0.48    107516
   macro avg       0.34      0.32      0.31    107516
weighted avg       0.43      0.48      0.44    107516

------------------------------------------------------------------------------------------------
('Accuracy for the Gaussian Naive Bayes model:', 48.45883403400424, '%')
------------------------------------------------------------------------------------------------
The confusion matrix for the Multinomial Naive Bayes model is as follows:
[[ 1474 12662   449  2833   389]
 [  605 38451   774  7158   196]
 [  665 10947  1272  7784   101]
 [  782  6066   226  8332    90]
 [  424  2716   110  2911    99]]
------------------------------------------------------------------------------------------------
The classification report for the Multinomial Naive Bayes model is as follows:
              precision    recall  f1-score   support

           0       0.37      0.08      0.14     17807
           1       0.54      0.81      0.65     47184
           2       0.45      0.06      0.11     20769
           3       0.29      0.54      0.37     15496
           4       0.11      0.02      0.03      6260

   micro avg       0.46      0.46      0.46    107516
   macro avg       0.35      0.30      0.26    107516
weighted avg       0.43      0.46      0.38    107516

------------------------------------------------------------------------------------------------
('Accuracy for the Multinomial Naive Bayes model:', 46.15871126157967, '%')
------------------------------------------------------------------------------------------------
```

Support Vector Machine Model:

Another algorithm we used here is Support Vector Machine. SVM algorithm maximizes the margin between the two separating hyperplanes by finding the maximum of the functional. In addition, SVM could solve both linearly separable and inseparable problems by implementing different kernels. Because we do not how is our data set separate, we used both Linear kernel and RBF kernel to test which one is better.

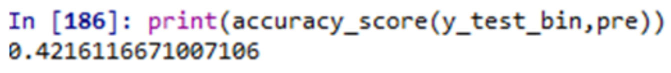
Another problem we faced here is that running SVM usually take lots of time. In order to accelerate the processing time, we use two algorithms here. First is Bagging Classifier, A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. By using Bagging Classifier, it could reduce the error and avoid model overfitting. Second method is One Vs Rest Classifier, this strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational, one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. By implementing those two methods together, we could get a better result with high accuracy and low time cost. The data processing steps for SVM models are same as Random Forest model, we encode the categorical variables into numbers and scale the numerical variables. The bins we set here are 5 bins, which equally separate purchase amount in to 5 intervals. 20% of original data as test data set, and 80% as train data set.

Linear Kernel:



```
[[  710 16385    22   166   524]
 [ 1973 42620    25   644  1922]
 [ 2164 16029    33  1040  1503]
 [ 2550  9833    49  1353  1711]
 [  859  4327    12   448   614]]
```

Confusion Matrix

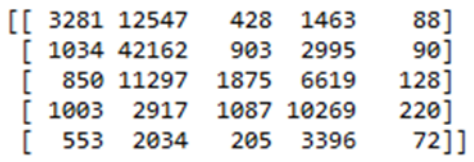


```
In [186]: print(accuracy_score(y_test_bin,pre))
0.4216116671007106
```

Accuracy

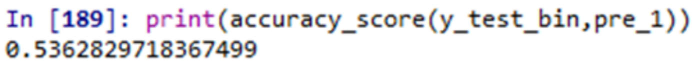
The rows of confusion matrix are the instances which model predicts for one to five bins. The columns are the original instances in test data set. For example, on the first columns, 710 means that at bin one, there are 710 instances are correctly predicting as bin one, 1973 instances are predicting as bin two, 2164 are predicting as bin three, 2550 are predicting as bin four, 859 are predict as bin five. Hence, we could find that bin two has the best prediction. Bin two has the worst prediction result. The overall accuracy for Linear kernel method is 42.16%. The time consumption of this model is 2023 seconds.

<u>RBF Kernel:</u>



```
[[ 3281 12547   428  1463    88]
 [ 1034 42162   903  2995    90]
 [  850 11297  1875  6619   128]
 [ 1003  2917  1087 10269   220]
 [  553  2034   205  3396    72]]
```

Confusion Matrix



```
In [189]: print(accuracy_score(y_test_bin,pre_1))
0.5362829718367499
```

Accuracy

In here, we could find that the RBF kernel gives us a much higher accuracy than linear kernel. This also indicate that our data set is not linearly separable. Because we separate our data randomly each time, so there are some differences in bin numbers, but we could still find that bin 2 has the highest accuracy result. However, the time cost of RBF kernel model is 6864 seconds, which is three times higher than Linear Kernel, this is because the RBF kernel is more complicated than Linear kernel.

<u>Comparison and Summary of Models:</u>

<u>Linear regression:</u> Adjusted R-squared: 0.3 ,16 out of 19 parameters are significant, 5.88 seconds on time consumption.

<u>Random Forest Regressor:</u> smallest mean squared error 2902, maximum number of leaf nodes 2000, 11.36 seconds on time consumption.

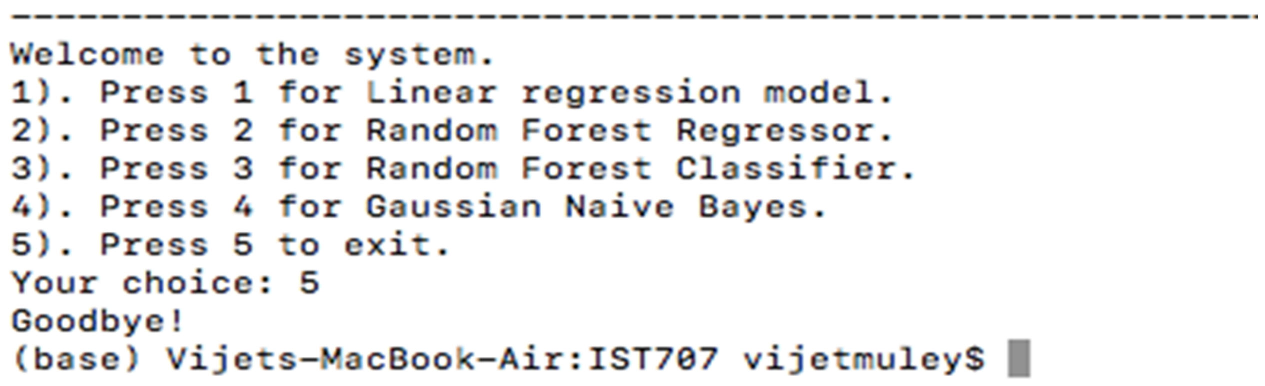
<u>Random Forest Classifier:</u> accuracy rate 62.18% with 5 bins, Number of tress 150, maximum leaf nodes 1500, entropy criteria, 90.13 seconds on time consumption.

<u>Naïve Bayes:</u> accuracy rate 63.23% with 3 bins 5.36 seconds on time consumption.

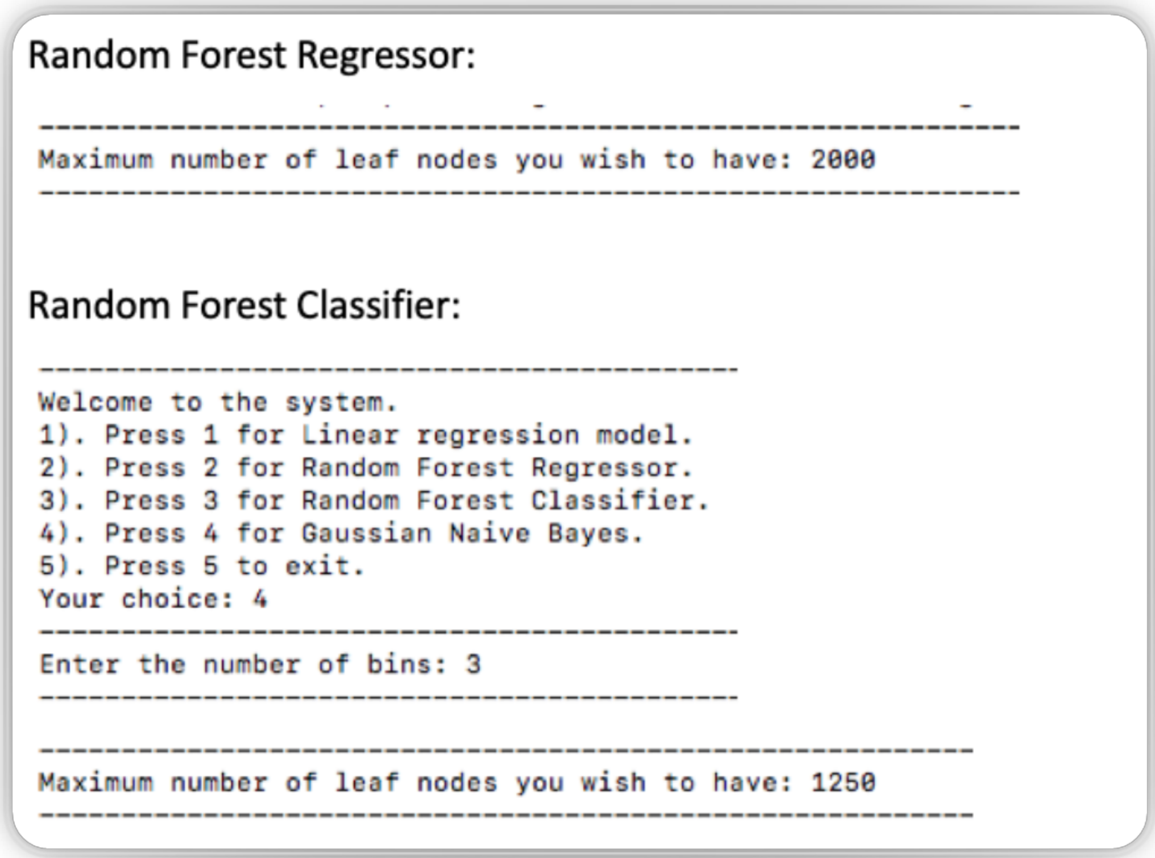
<u>SVM:</u> accuracy rate 53.6% with 5 bins. 6864 seconds on time consumption, RBF Kernel.

Apparently, Naïve Bayes model has the highest accuracy. However, this rate is achieved when the target bins is 3. If we increase the bins to 4, the accuracy drops severely down to 48.46%. because the when the number of bins increase, the number of borders is increasing. This could cause incorrectly classifying. In addition, if we only cut the purchase amount into 3 bins, the prediction will be too broad. In this case, we want to predict the purchase amount of a consumer as accurate as possible, so ideally, we want to use as many as bins as we could. Hence the Naïve bays model is not the best choice. In summary, if we consider the time cost as a bias, the random forest classifier is our best model here, it has a 62% accuracy rate with 90 seconds of time cost. Although it has the second highest processing time, the 90 second time consumption is not beyond our tolerance. If we change the number of bins to 4 or 5, it will not have too much drop on accuracy. However, if we do not care about the time cost, the SVM model will be our first choice. The SVM model has accuracy of 54% with 5 bins. In addition, if we want to predict a certain number of purchase amount instead of a range of purchase amount, the random forest regressor will be the best model.

<u>User interface and flexibility:</u>



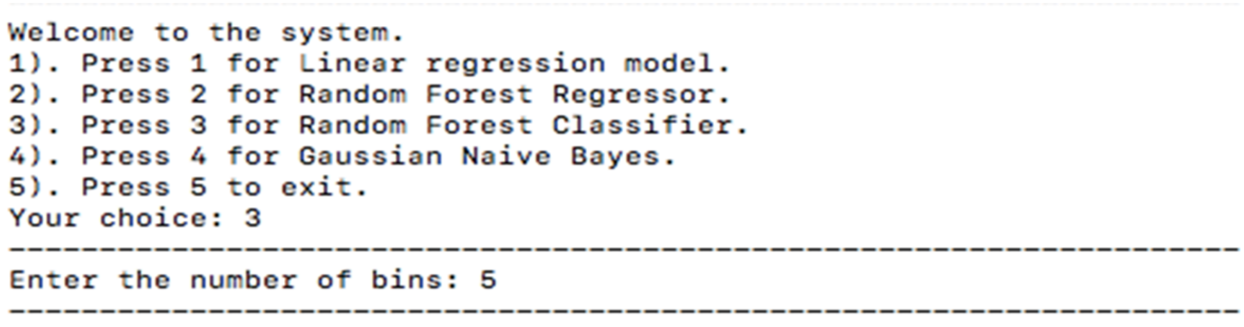
```
-----------------------------------------------------------
Welcome to the system.
1). Press 1 for Linear regression model.
2). Press 2 for Random Forest Regressor.
3). Press 3 for Random Forest Classifier.
4). Press 4 for Gaussian Naive Bayes.
5). Press 5 to exit.
Your choice: 5
Goodbye!
(base) Vijets-MacBook-Air:IST707 vijetmuley$ 
```


Random Forest Regressor:

```
------------------------------------------------------------
Maximum number of leaf nodes you wish to have: 2000
------------------------------------------------------------
```

Random Forest Classifier:

```
-------------------------------------------
Welcome to the system.
1). Press 1 for Linear regression model.
2). Press 2 for Random Forest Regressor.
3). Press 3 for Random Forest Classifier.
4). Press 4 for Gaussian Naive Bayes.
5). Press 5 to exit.
Your choice: 4
-------------------------------------------
Enter the number of bins: 3
-------------------------------------------

---------------------------------------------------------
Maximum number of leaf nodes you wish to have: 1250
---------------------------------------------------------
```



```
Welcome to the system.
1). Press 1 for Linear regression model.
2). Press 2 for Random Forest Regressor.
3). Press 3 for Random Forest Classifier.
4). Press 4 for Gaussian Naive Bayes.
5). Press 5 to exit.
Your choice: 3
---------------------------------------------------------------------
Enter the number of bins: 5
---------------------------------------------------------------------
```

In order to make our programming user friendly, we design a simple user interface which allow them select model. For specific model such random forest, they could select bins and leaf nodes they want to use. Finally, the program will return the time cost and accuracy rate of model the select.

Conclusion:

In this project, our goal is to find the best machine learning model which could give us a best predicting result. The biggest problem we have is how to format the data set to fit into our models. We spend lots time on data cleaning and re-formatting the data. Finally. We successfully implement 5 models to learn patterns from the data and help in forecasting sales amount. The Random Forest Classifier is the best model if we want to predict a range of purchase amount of a consumer. The Random Forest Regression can be used when we want to find a specific amount of purchase of a consumer. The user interface panel offers other people a easy and flexible way to do the analysis they want with our program.