CAPSTONE PROJECT

TELCOM CUSTOMER CHURN - APPENDIX

11 APRIL 2021

Jotinder Singh Matta

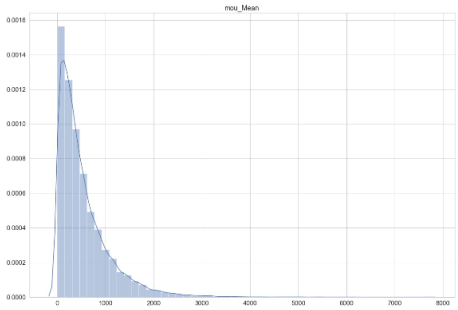
PGPDSBA Online Mar20_A

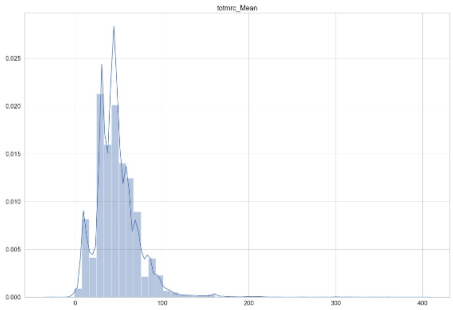
CONTENTS

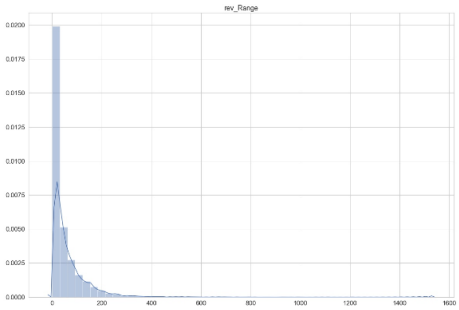
APPENDIX

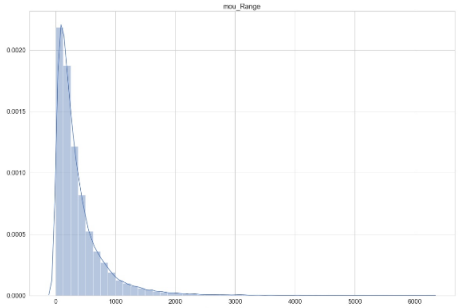
APPENDIX 2.1 - UNI-VARIATE ANALYSIS

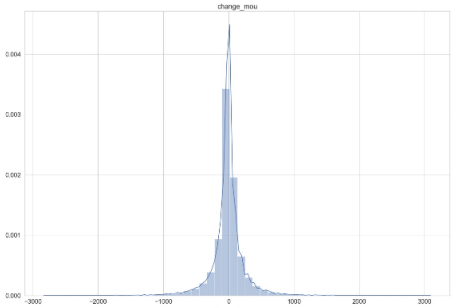
All the graphs for univariate analysis.

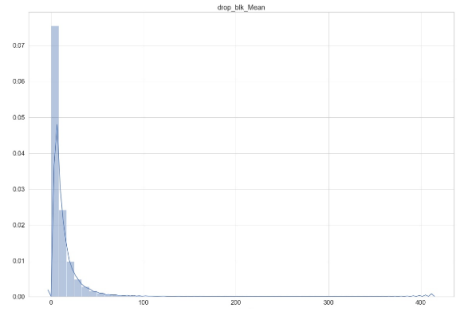


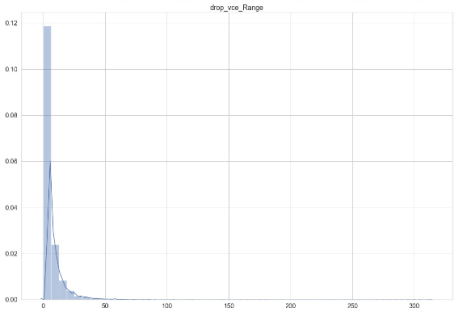


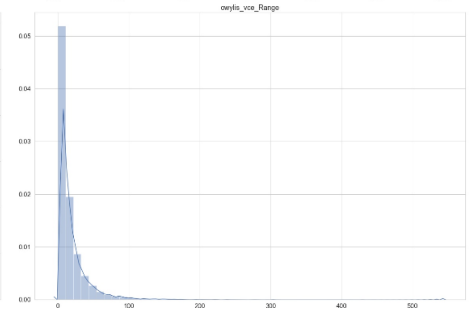


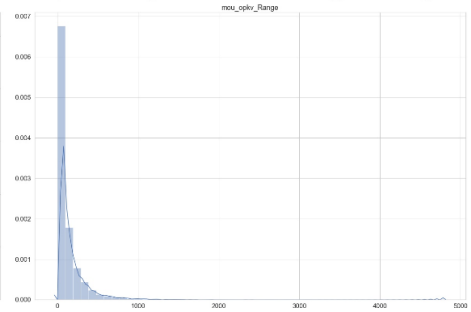


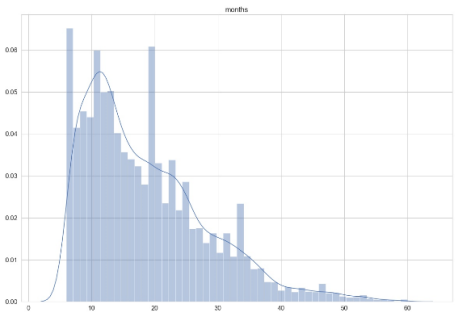


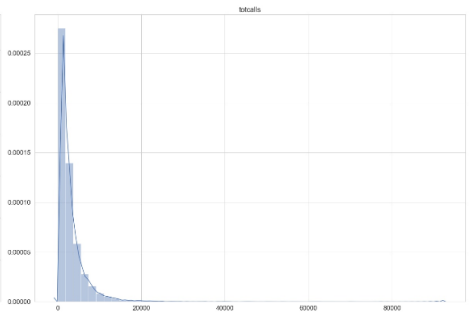


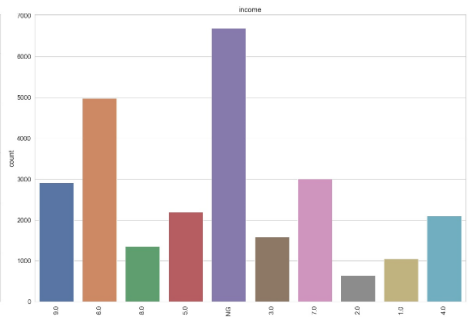






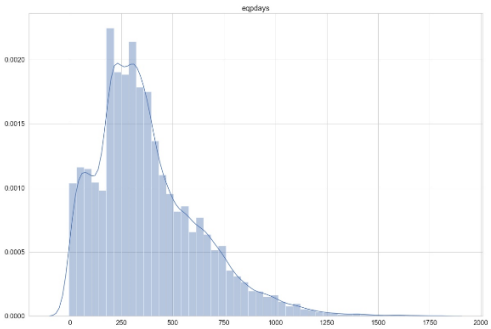


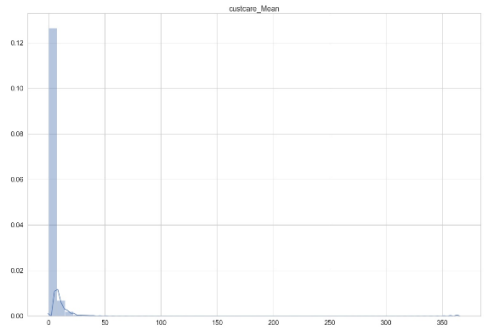


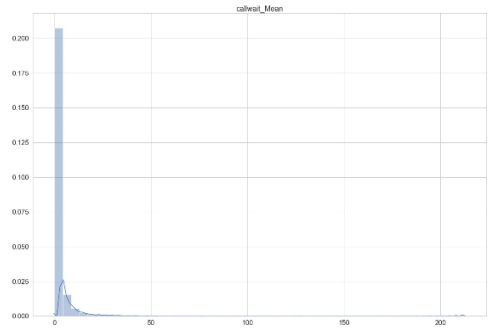


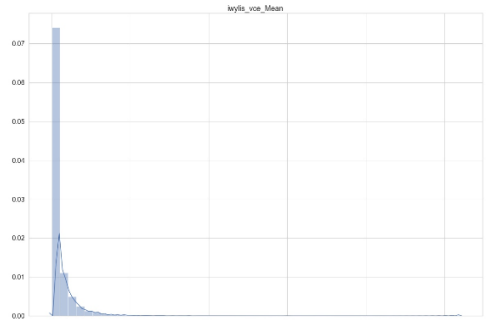
APPENDIX

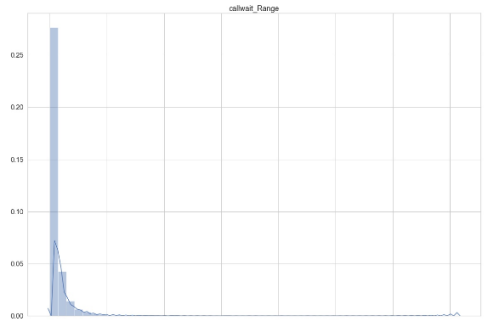
APPENDIX 2.1 - UNI-VARIATE ANALYSIS

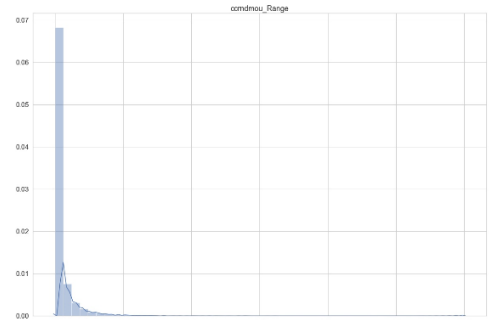


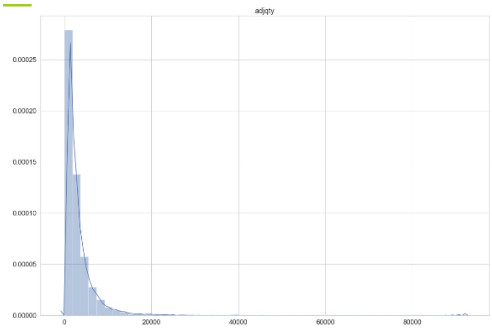


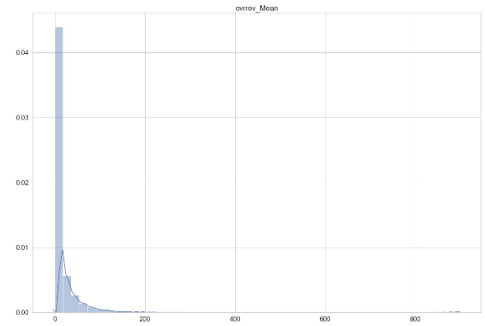


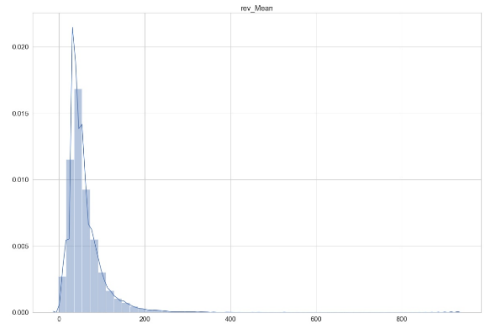


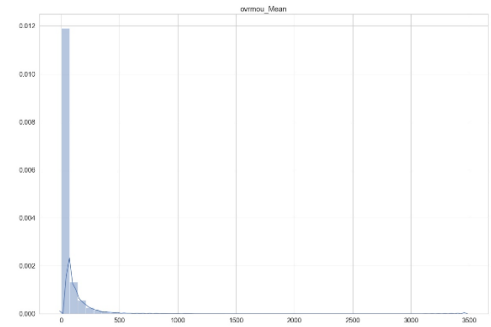


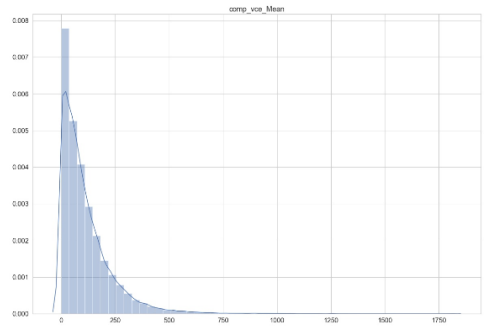


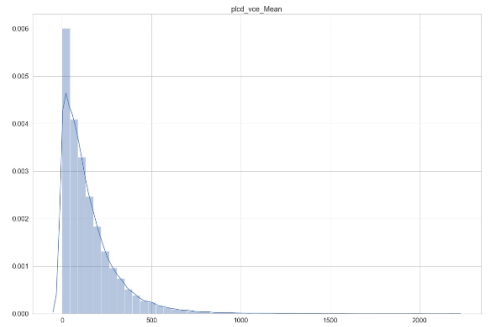








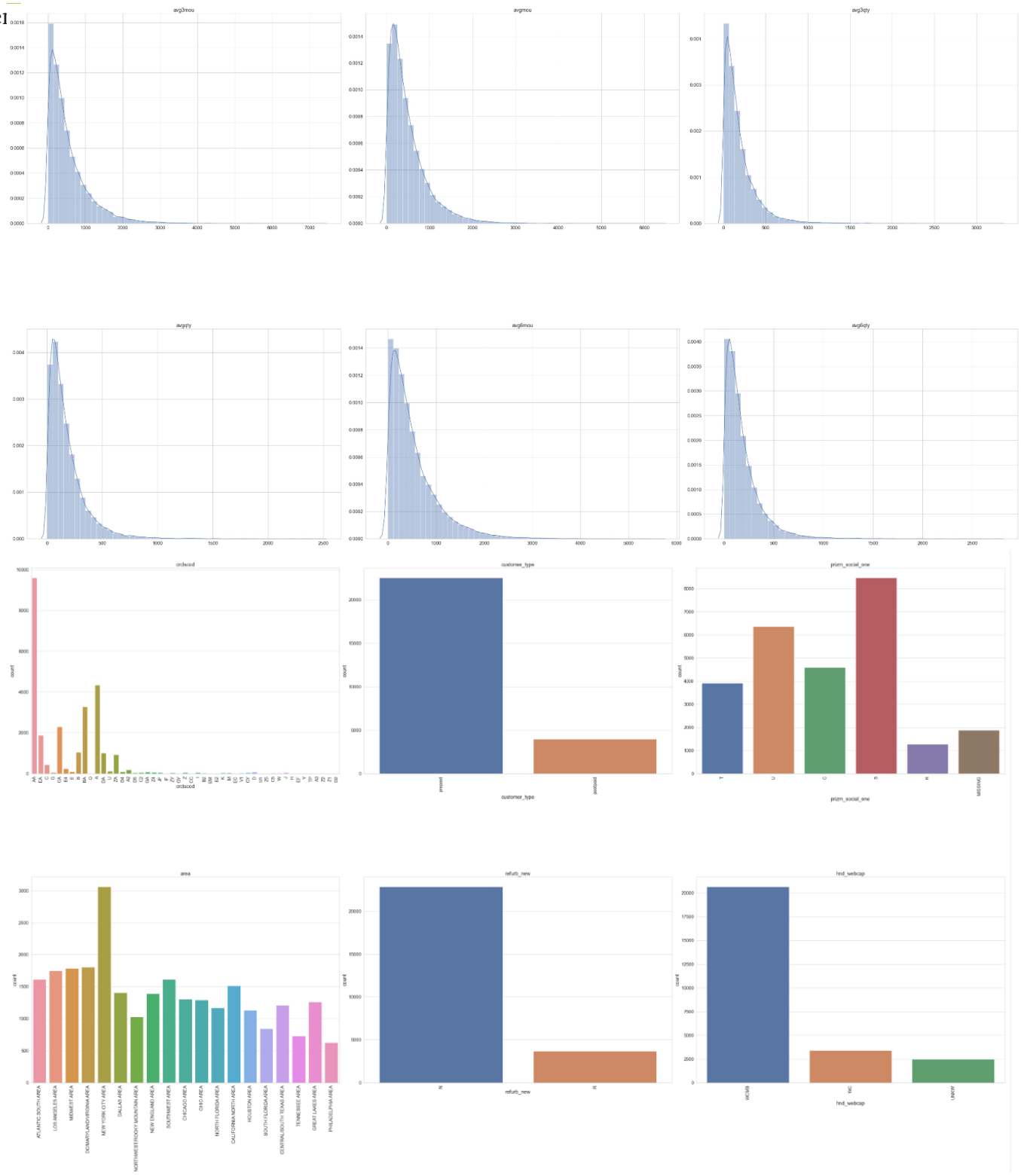




APPENDIX

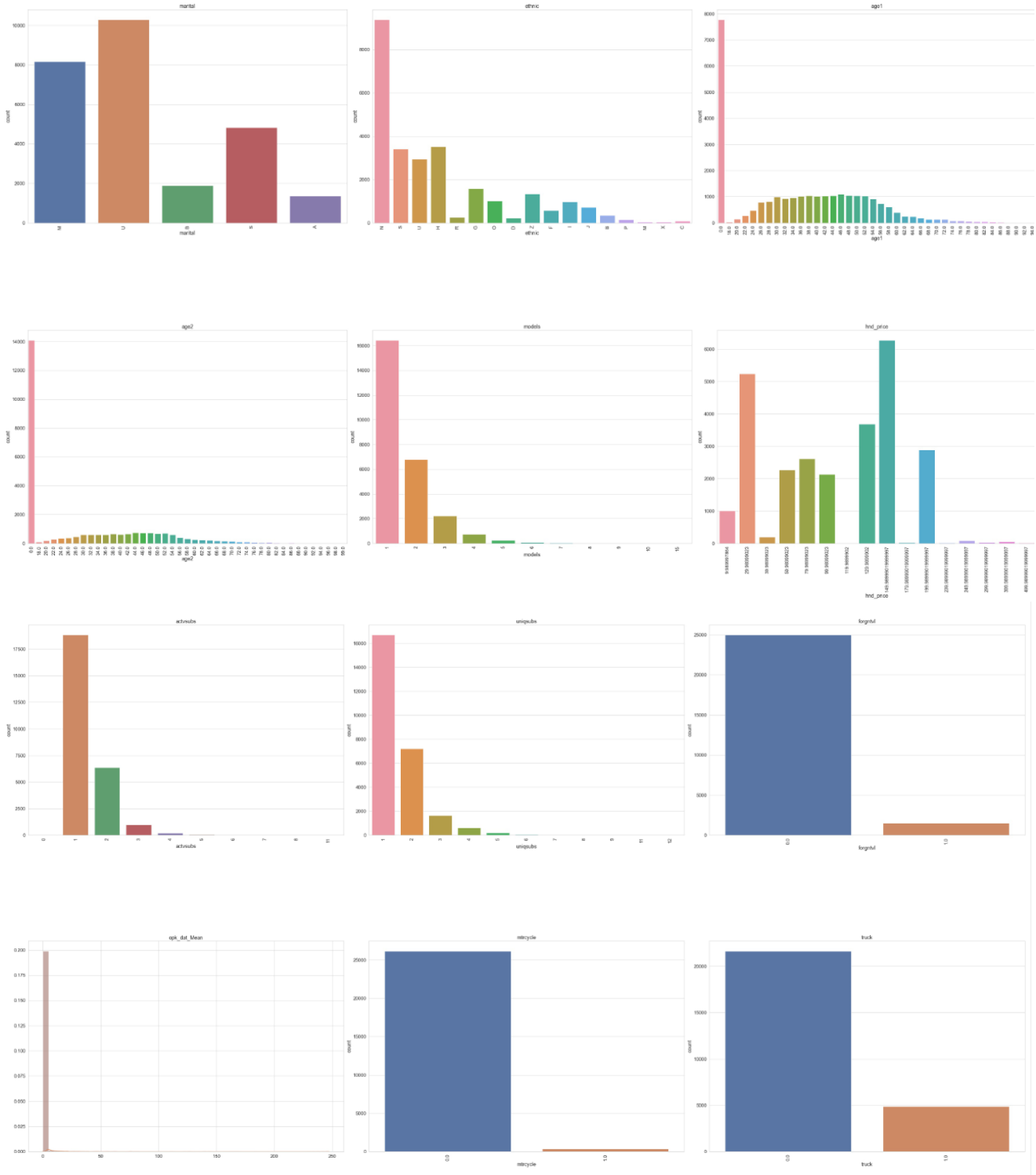
APPENDIX 2.1 - UNI-VARIATE ANALYSIS

Rel



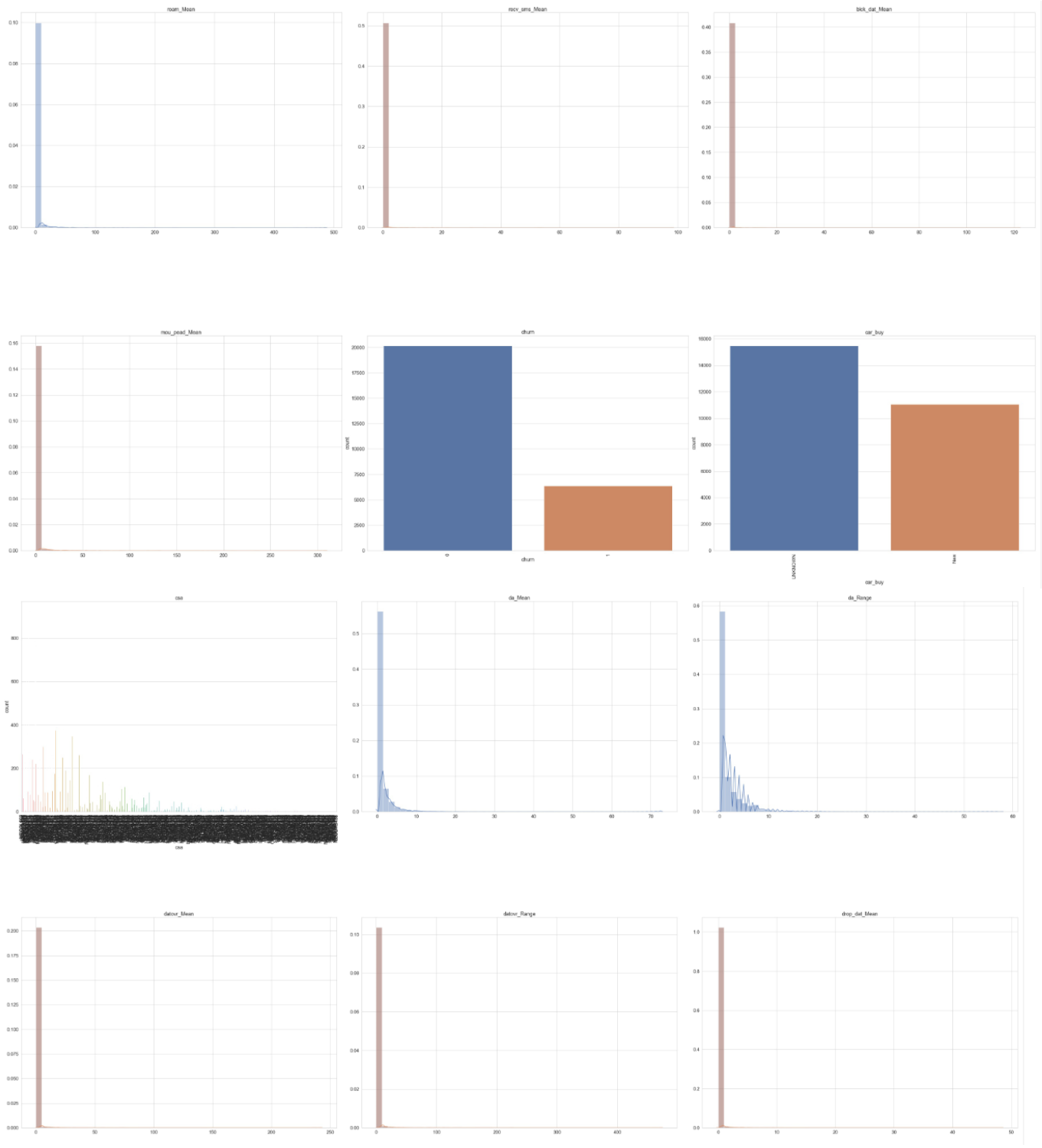
APPENDIX

APPENDIX 2.1 - UNI-VARIATE ANALYSIS



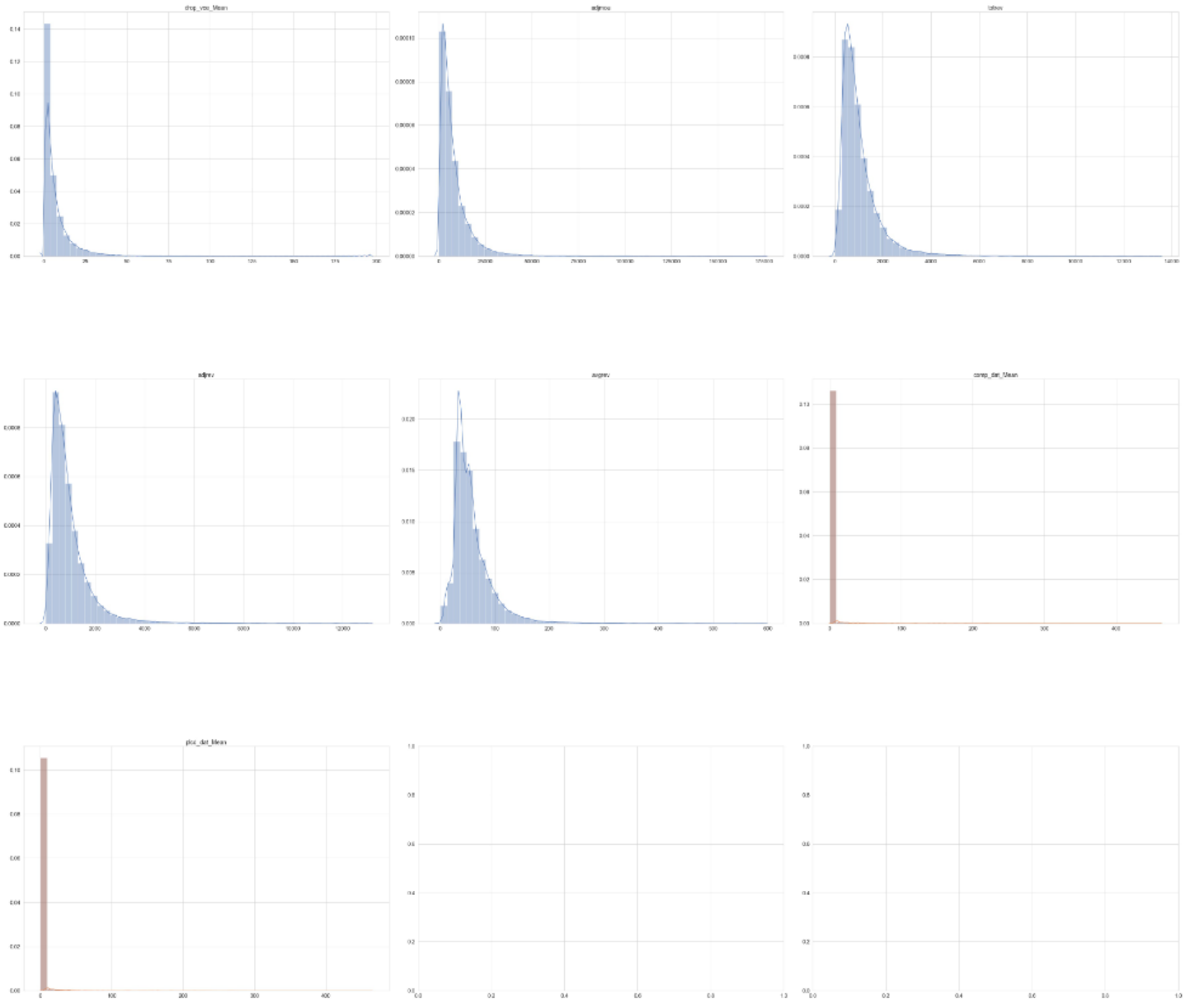
APPENDIX

APPENDIX 2.1 - UNI-VARIATE ANALYSIS



APPENDIX

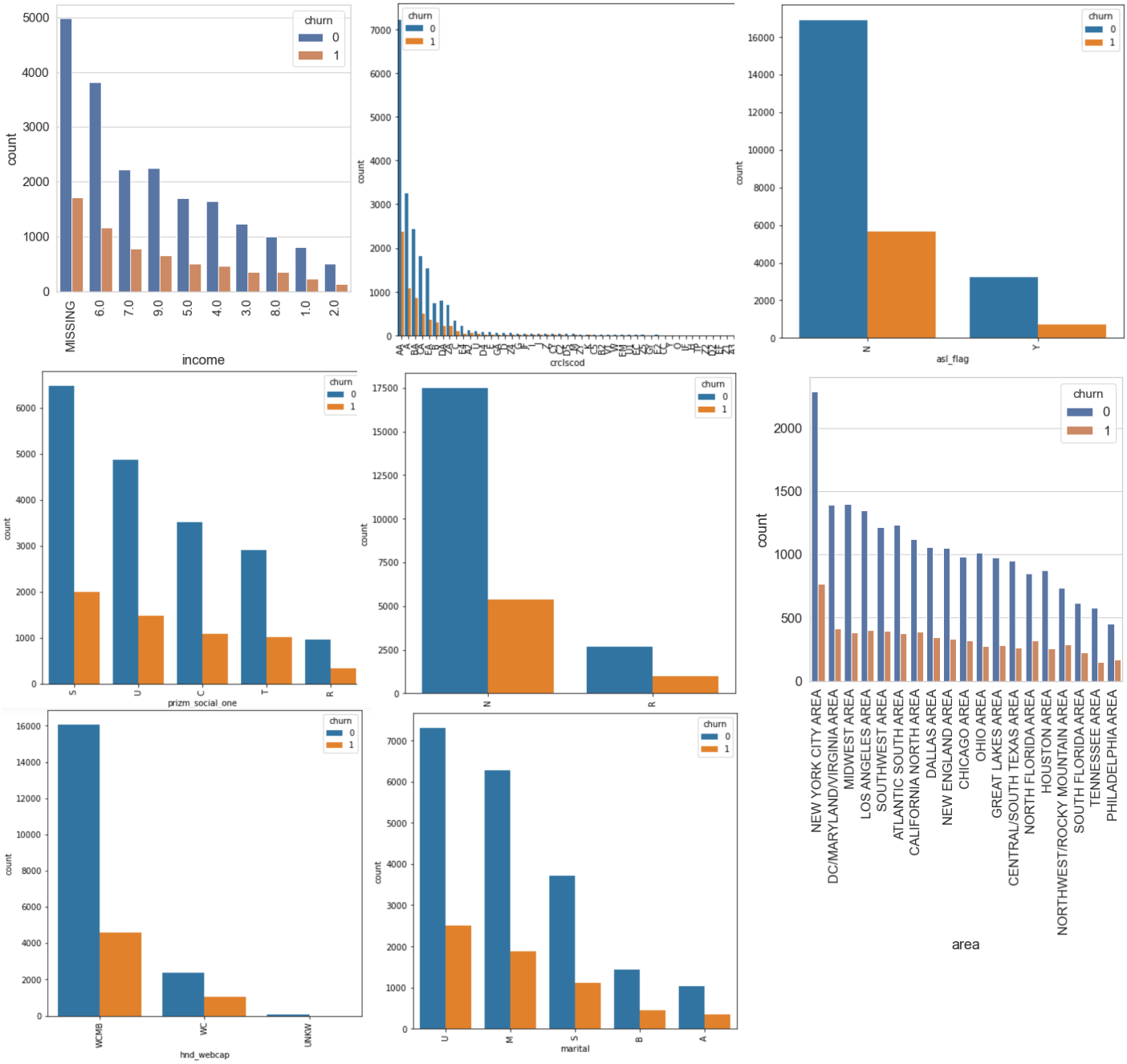
APPENDIX 2.1 - UNI-VARIATE ANALYSIS



APPENDIX

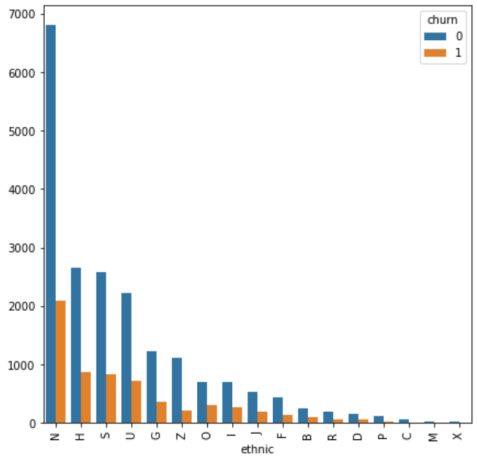
APPENDIX 2.2 - BI-VARIATE ANALYSIS

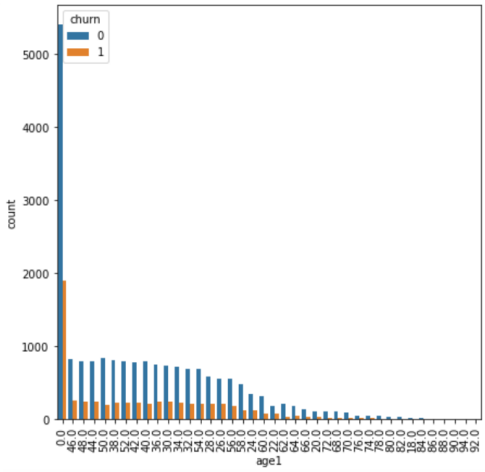
All the graphs for bi-variate analysis.

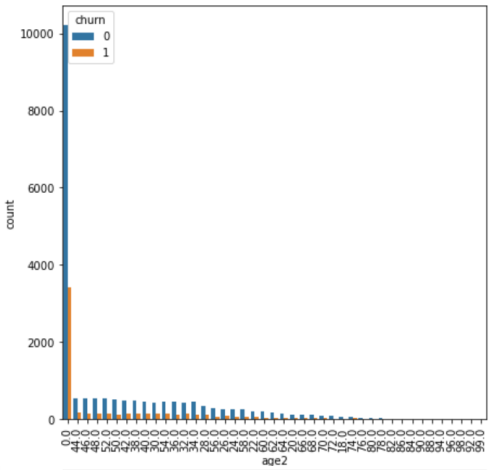


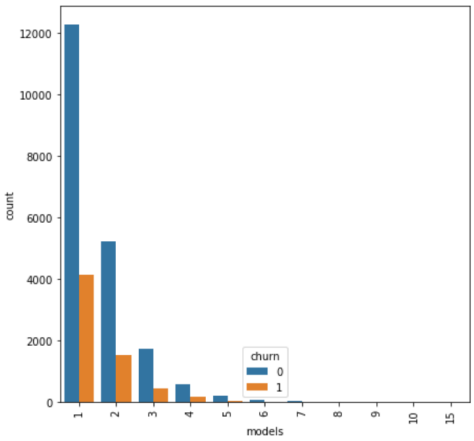
APPENDIX

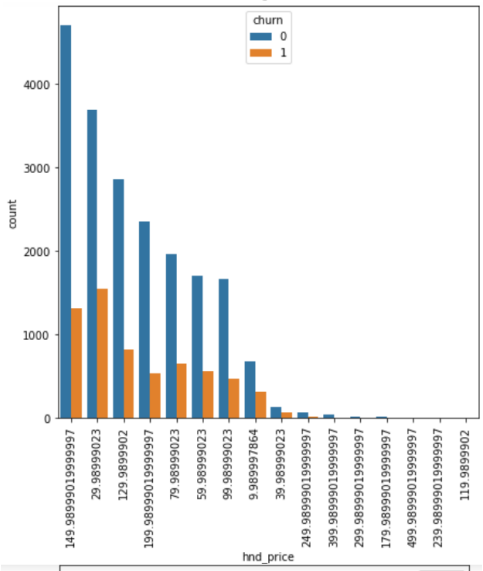
APPENDIX 2.2 - BI-VARIATE ANALYSIS

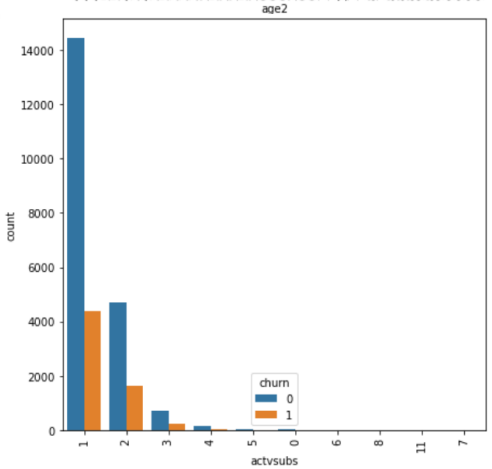


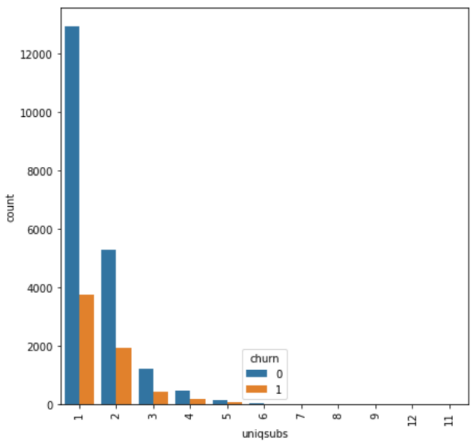


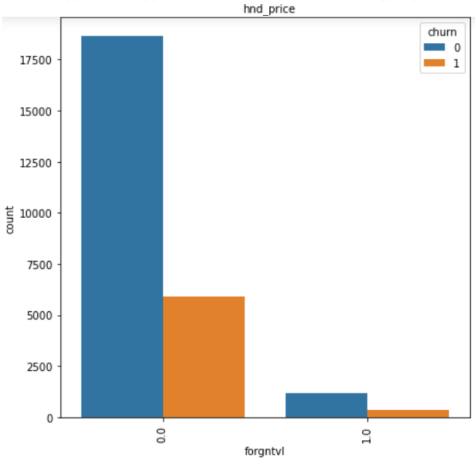


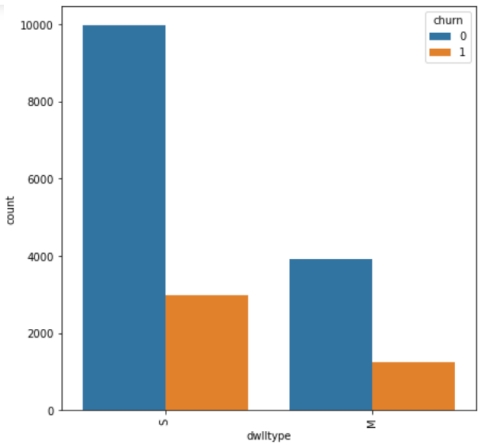






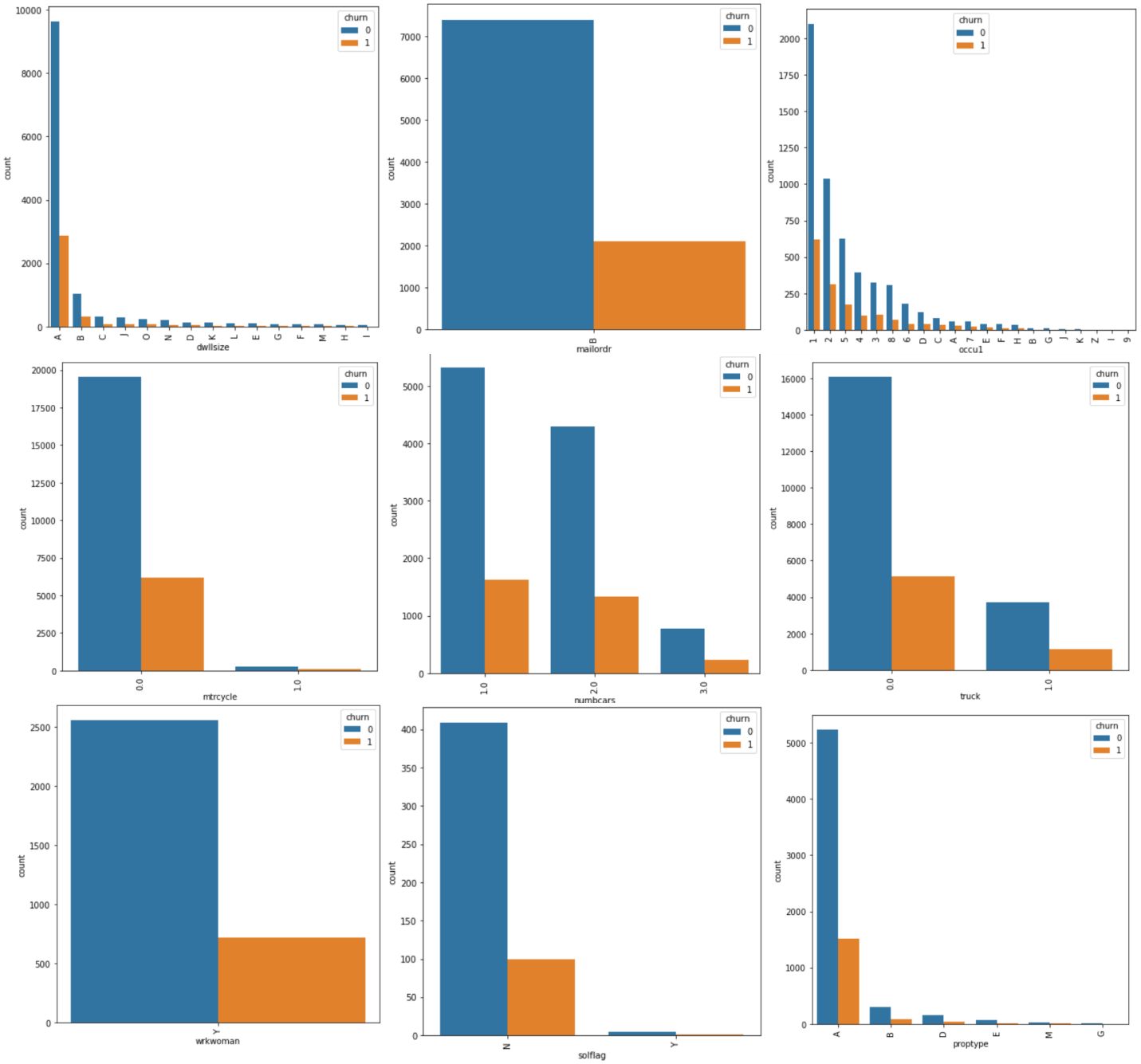






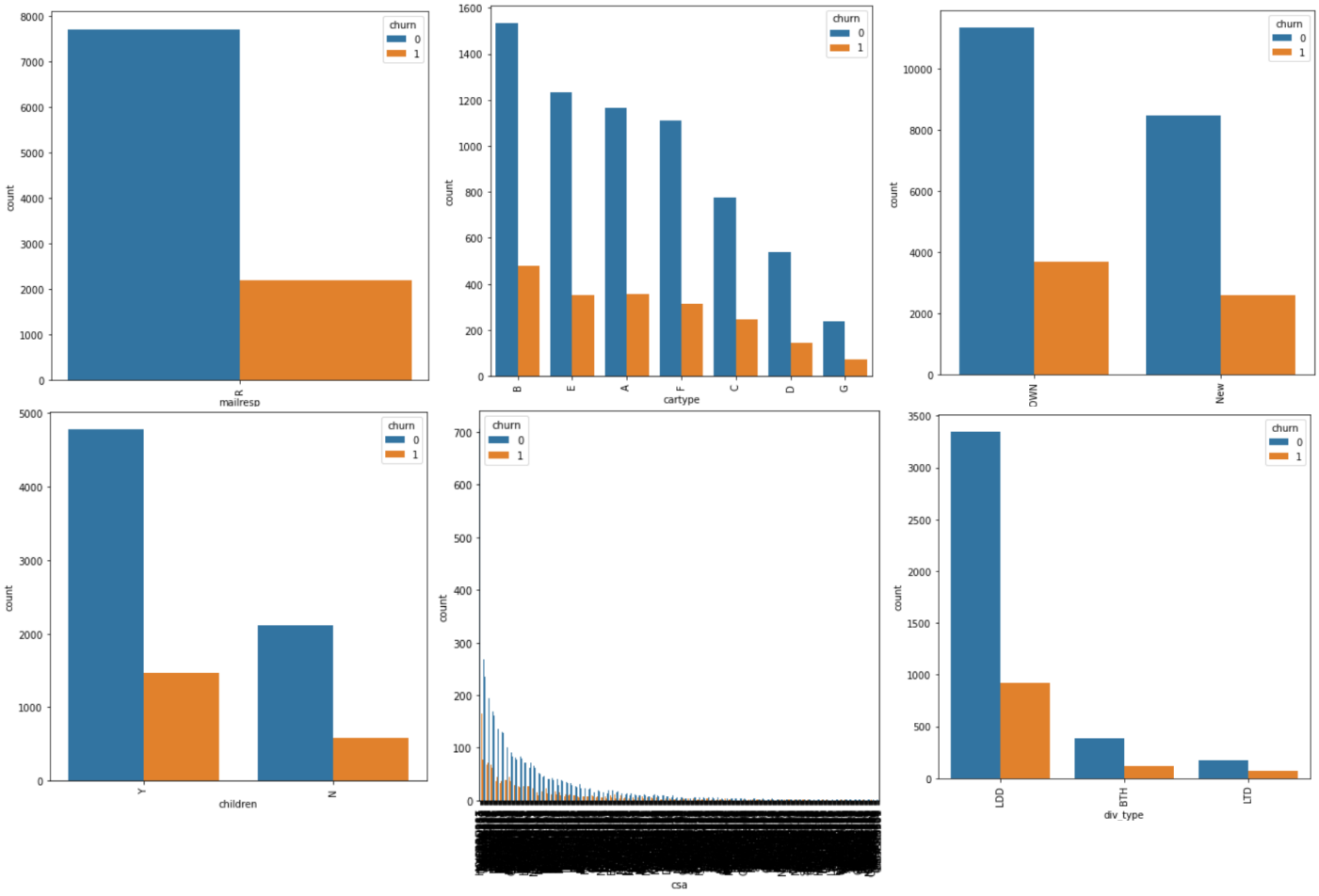
APPENDIX

APPENDIX 2.2 - BI-VARIATE ANALYSIS



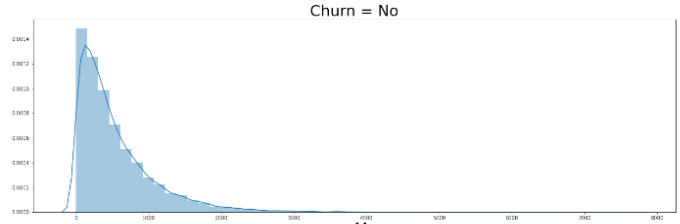
APPENDIX

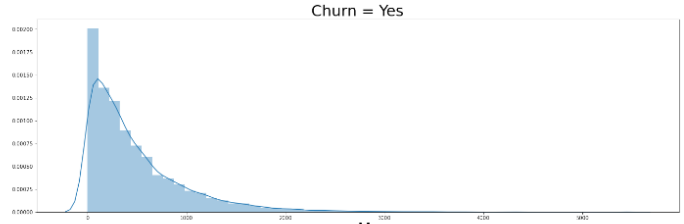
APPENDIX 2.2 - BI-VARIATE ANALYSIS



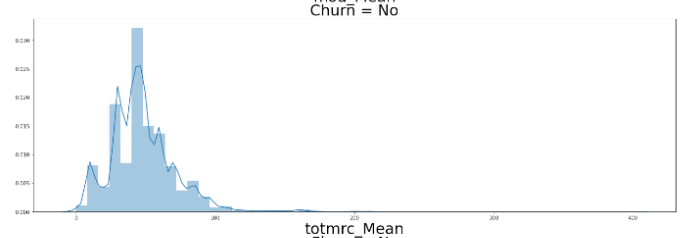
APPENDIX

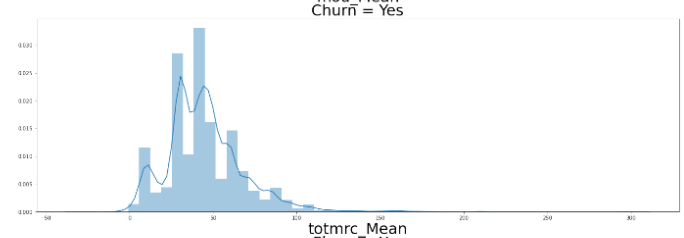
APPENDIX 2.2 - BI-VARIATE ANALYSIS



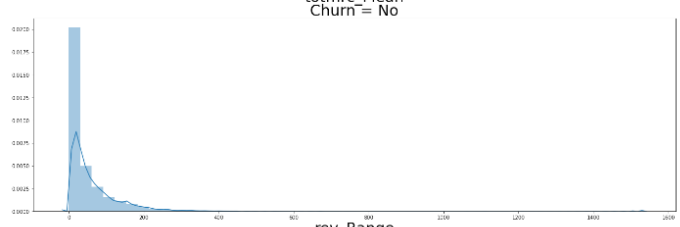


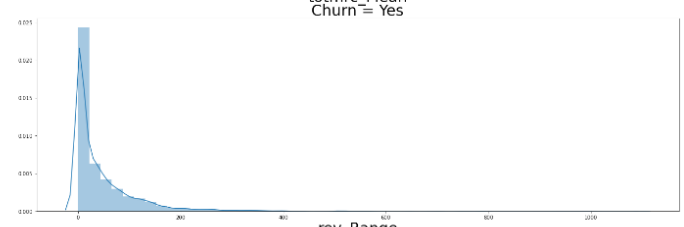
mou_Mean



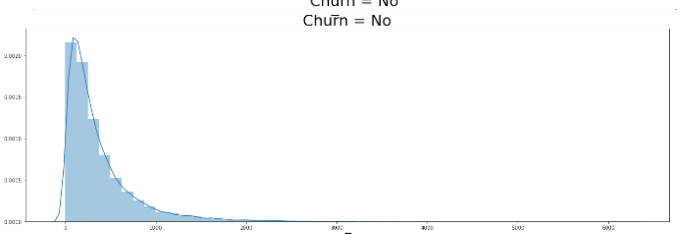


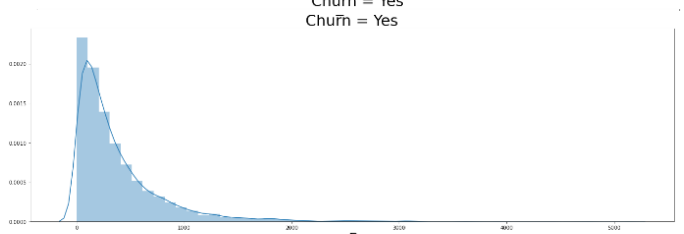
totmrc_Mean



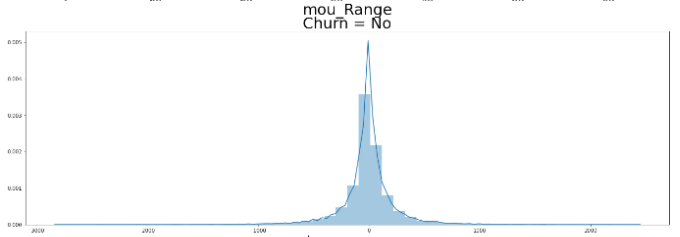


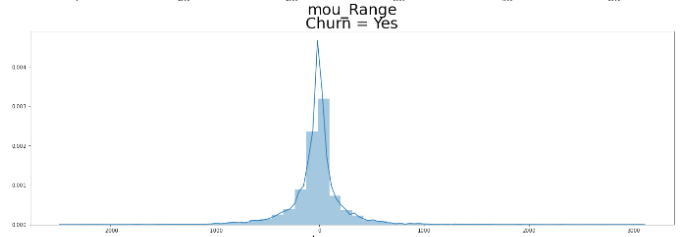
rev_Range



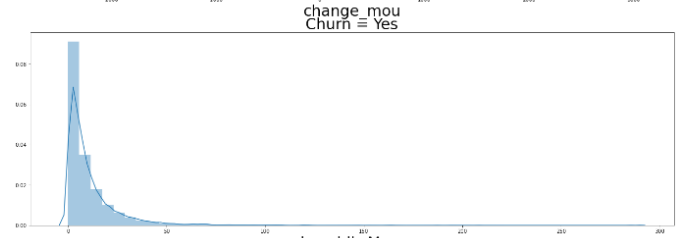


mou_Range





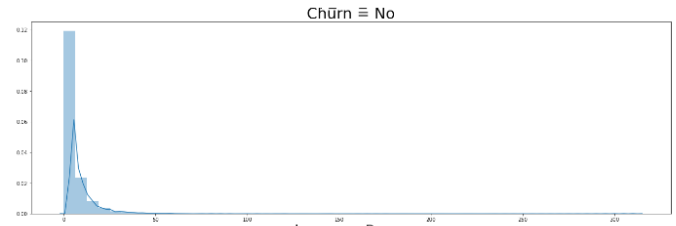
change_mou

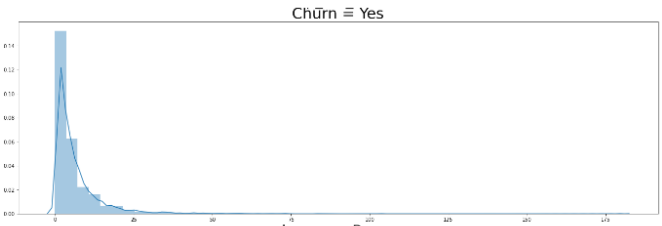


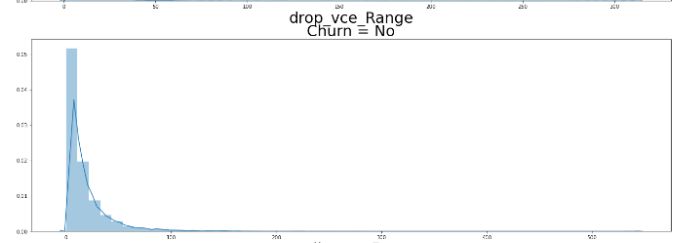
drop_blk_Mean

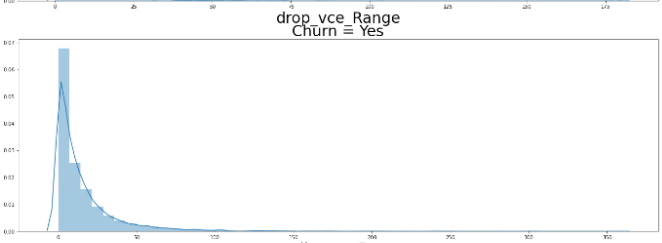
APPENDIX

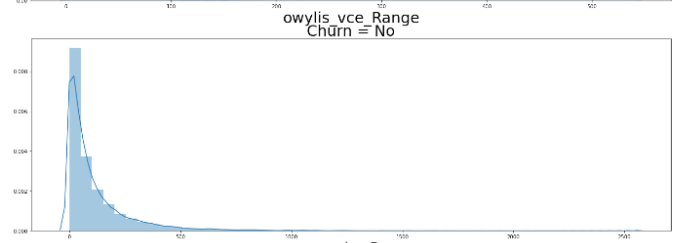
APPENDIX 2.2 - BI-VARIATE ANALYSIS

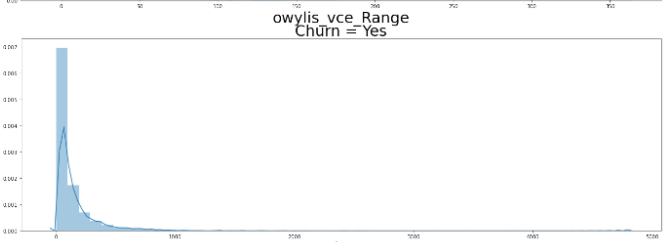


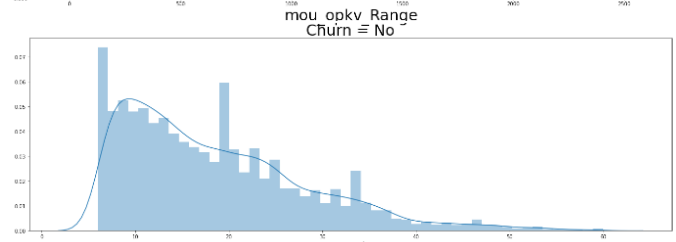


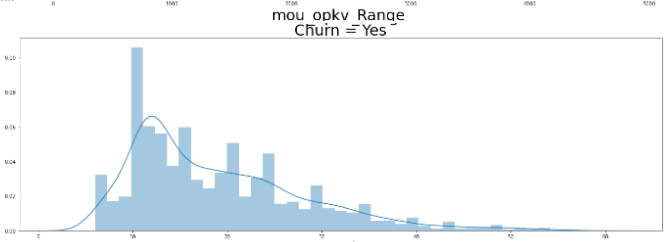


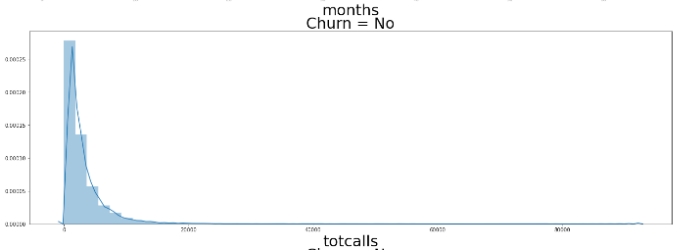


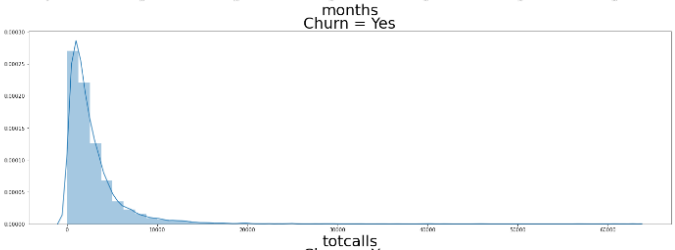


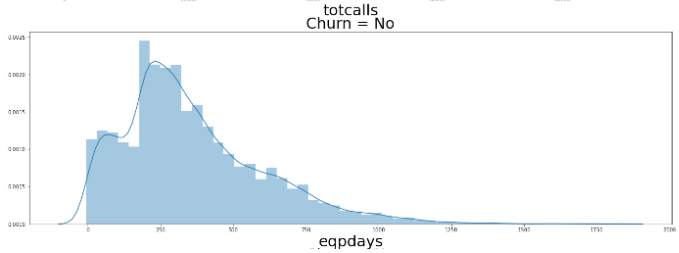


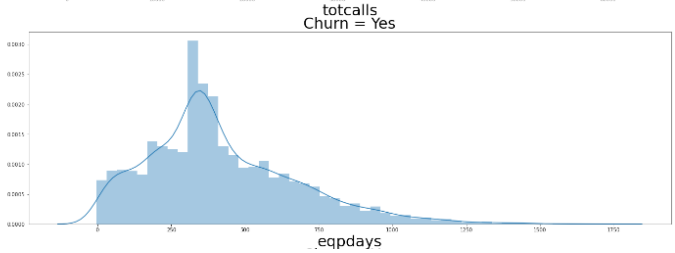






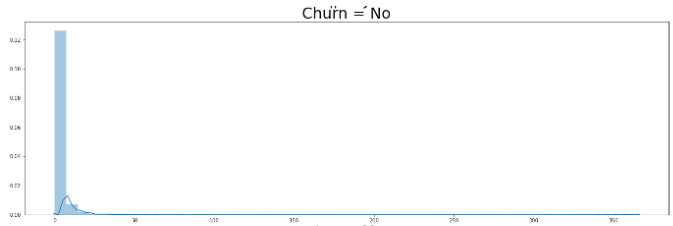


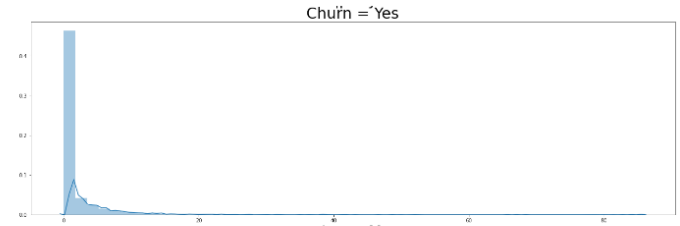


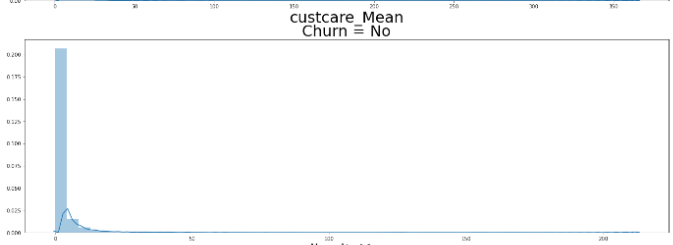


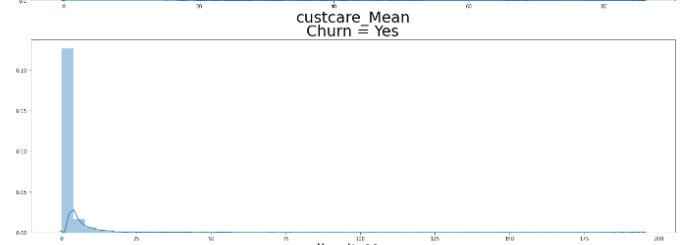
APPENDIX

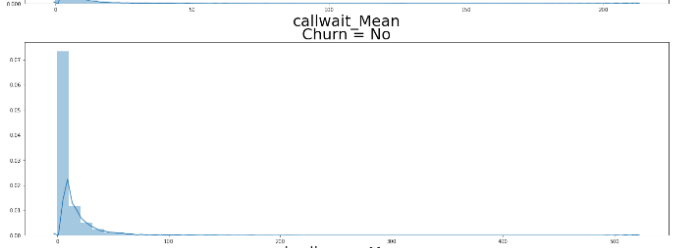
APPENDIX 2.2 - BI-VARIATE ANALYSIS

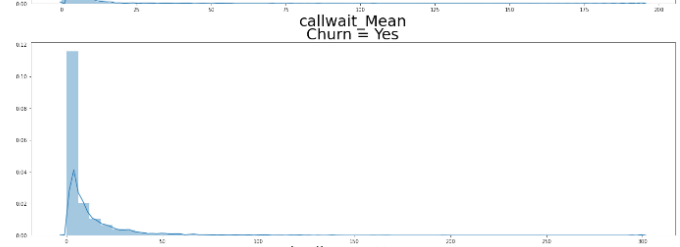


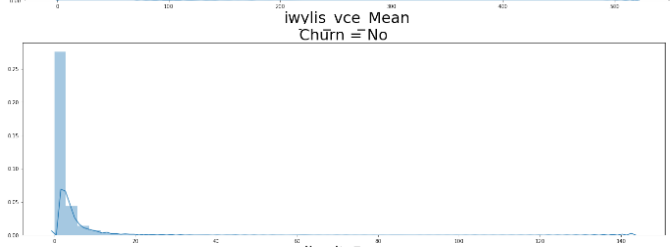


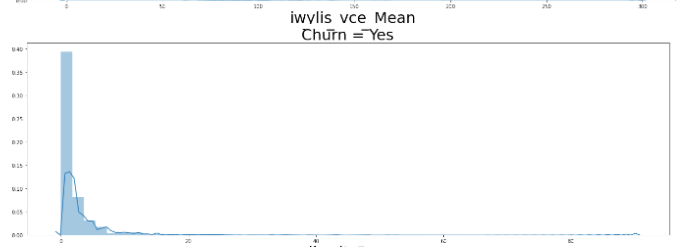


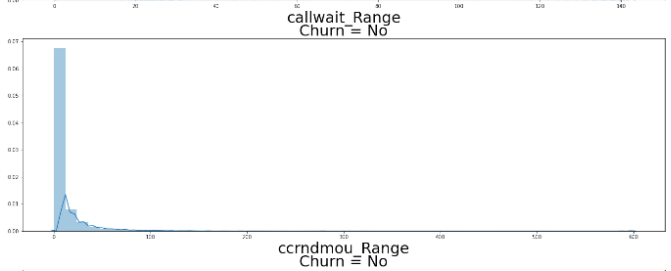


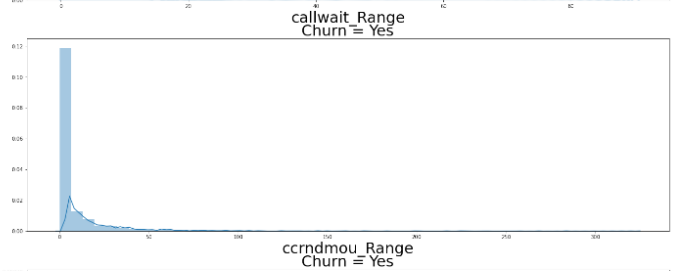


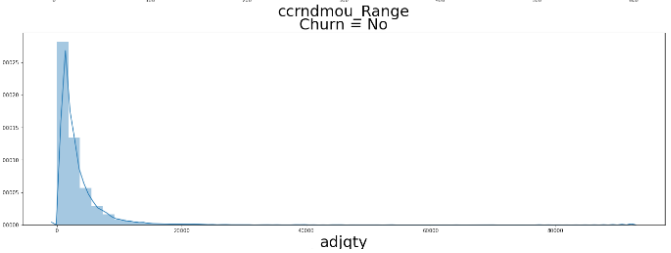


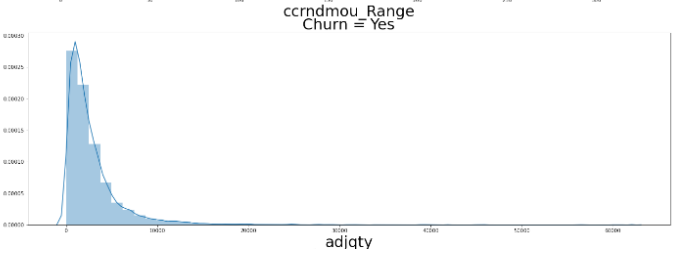






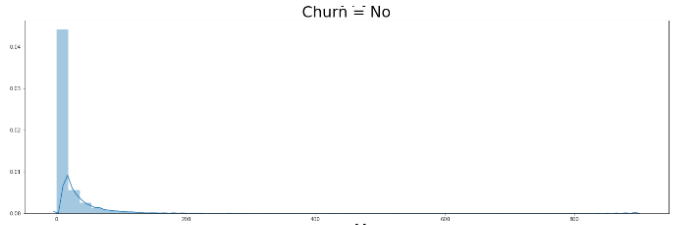


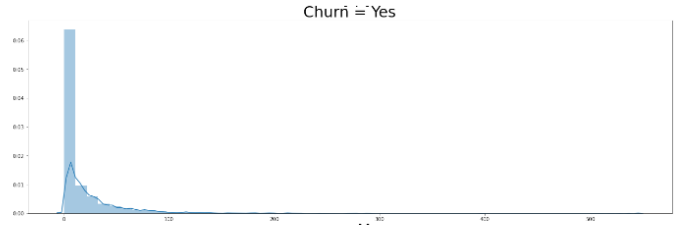


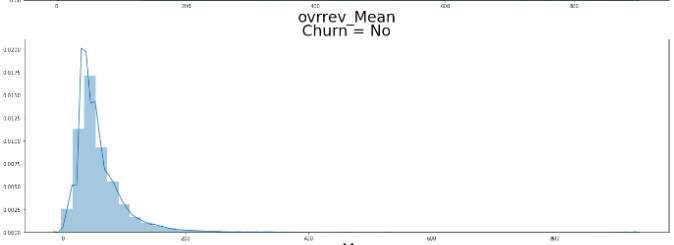


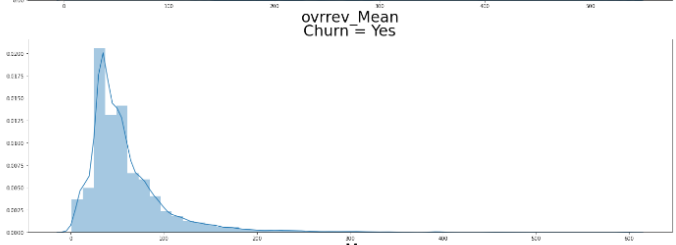
APPENDIX

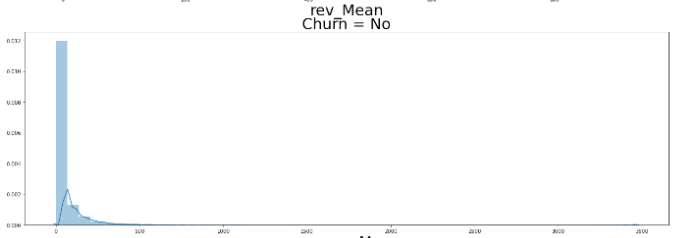
APPENDIX 2.2 - BI-VARIATE ANALYSIS

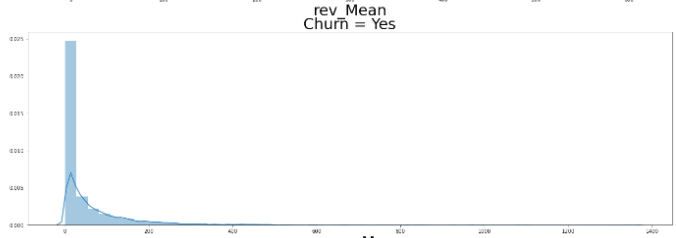


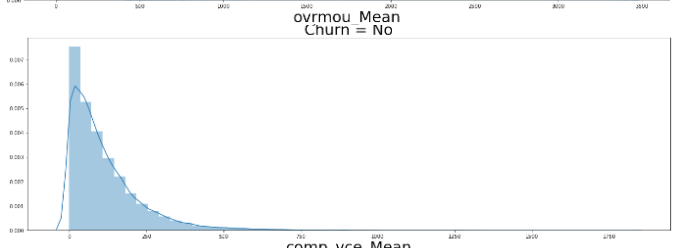


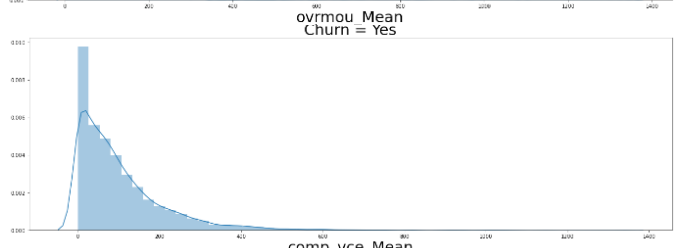


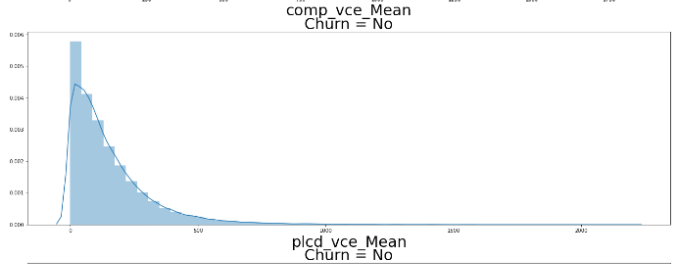


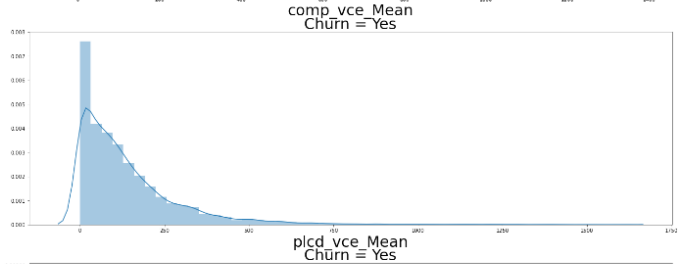


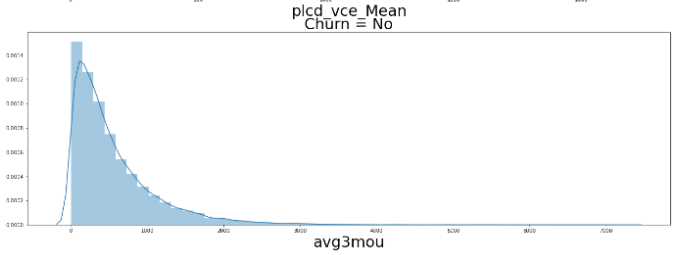


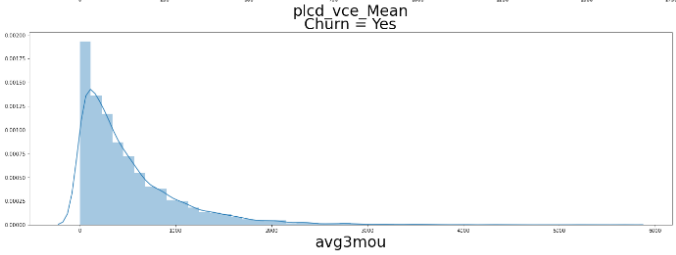


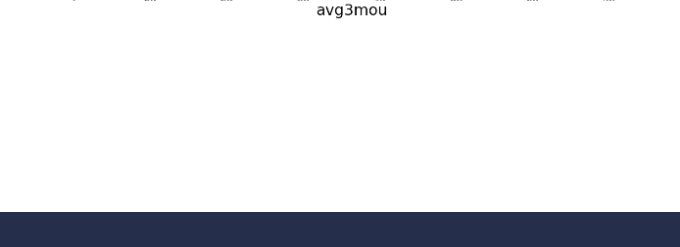


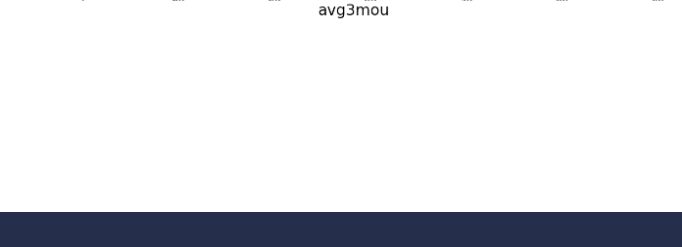








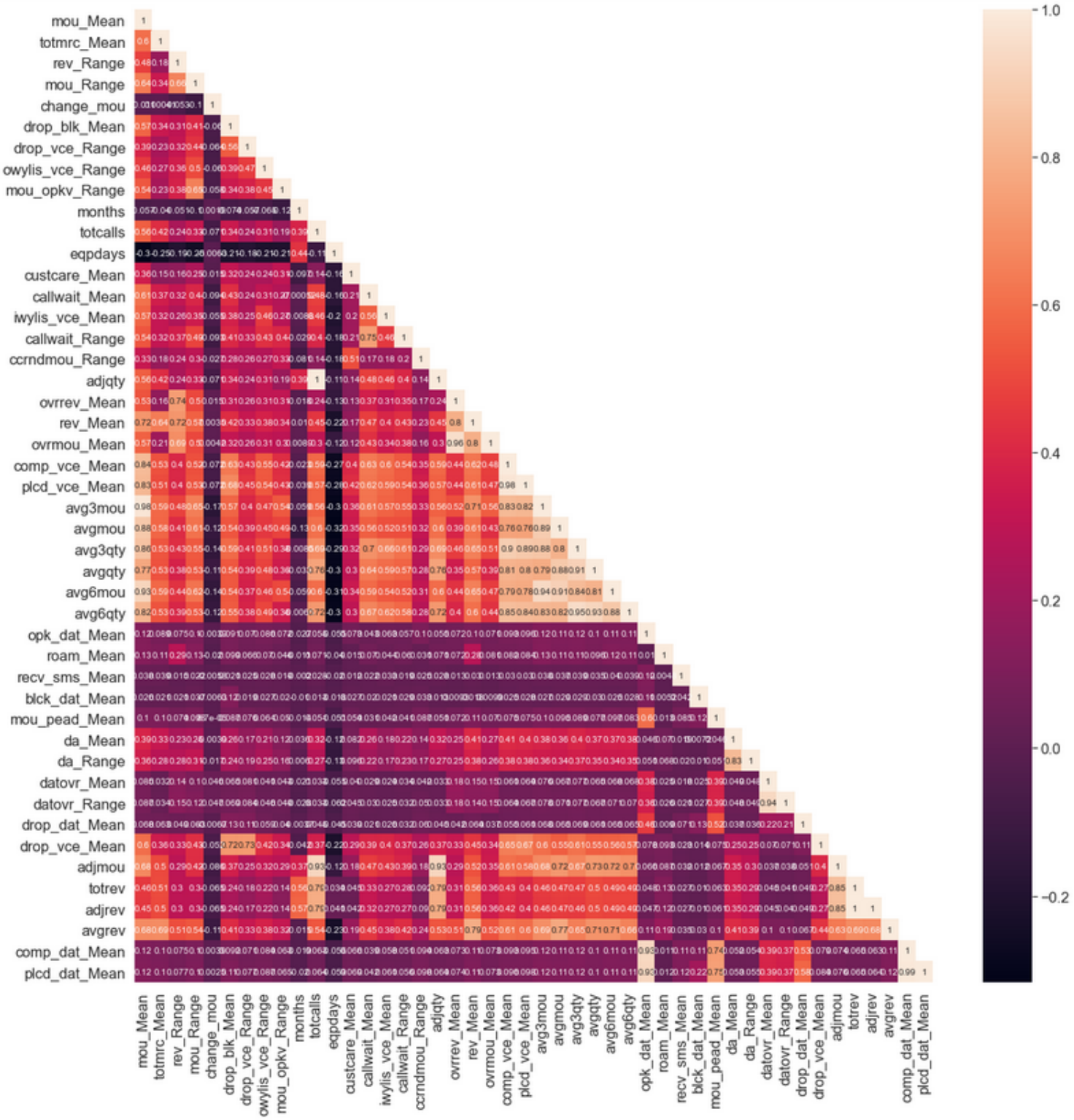




APPENDIX

APPENDIX 2.3 - CORRELATION

Below is the correlation heatmap.



APPENDIX

APPENDIX 2.3 - CORRELATION

Below is the list of al the columns having correlation of more than 90%.

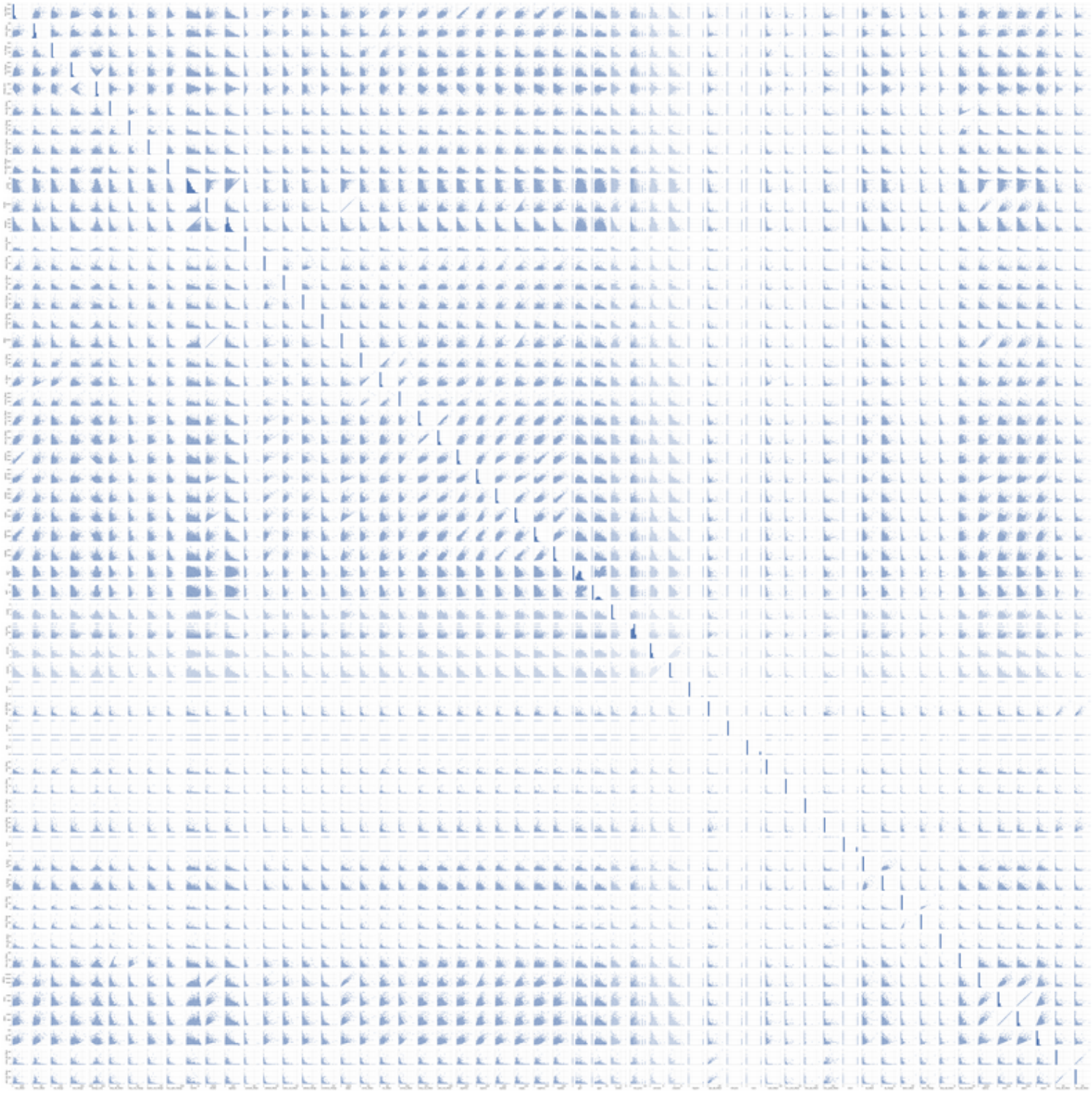
Column1	Column2
mou_Mean	avg3mou
mou_Mean	avg6mou
totcalls	adjqty
totcalls	adjmou
adjqty	totcalls
adjqty	adjmou
ovrrev_Mean	ovrmou_Mean
ovrmou_Mean	ovrrev_Mean
comp_vce_Mean	plcd_vce_Mean
plcd_vce_Mean	comp_vce_Mean
avg3mou	mou_Mean
avg3mou	avg6mou
avgmou	avg6mou
avg3qty	avgqty
avg3qty	avg6qty
avgqty	avg3qty
avgqty	avg6qty
avg6mou	mou_Mean
avg6mou	avg3mou
avg6mou	avgmou

Column1	Column2
avg6qty	avg3qty
avg6qty	avgqty
opk_dat_Mean	comp_dat_Mean
opk_dat_Mean	plcd_dat_Mean
datovr_Mean	datovr_Range
datovr_Range	datovr_Mean
adjmou	totcalls
adjmou	adjqty
totrev	adjrev
adjrev	totrev
comp_dat_Mean	opk_dat_Mean
comp_dat_Mean	plcd_dat_Mean
plcd_dat_Mean	opk_dat_Mean
plcd_dat_Mean	comp_dat_Mean

APPENDIX

APPENDIX 2.4 - PAIRPLOT

PFB the pair plot for the all the continuous columns in the data set.



APPENDIX

APPENDIX 2.5 - VIF

PFB the code and the output for VIF analysis.

```
def calculate_vif_(X, thresh = 5):
    cols = X.columns
    variables = np.arange(X.shape[1])
    dropped=True
    while dropped:
        dropped=False
        c = X[cols[variables]].values
        vif = [variance_inflation_factor(c, ix) for ix in np.arange(c.shape[1])]

        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X[cols[variables]].columns[maxloc] + '\' at index: ' + str(maxloc))
            variables = np.delete(variables, maxloc)
            dropped=True


    print('Remaining variables:')
    print(X.columns[variables])
    return X[cols[variables]]
```

```
df_cont=calculate_vif_(df_cont,thresh=5)
```

```
dropping 'rev_Mean' at index: 18
dropping 'avgrev' at index: 31
dropping 'avgmou' at index: 19
dropping 'avgqty' at index: 19
dropping 'mou_Mean' at index: 0
dropping 'totrev' at index: 27
dropping 'plcd_vce_Mean' at index: 17
dropping 'months' at index: 8
dropping 'mou_Range' at index: 2
dropping 'drop_vce_Mean' at index: 23
Remaining variables:
Index(['totmrc_Mean', 'rev_Range', 'change_mou', 'drop_blk_Mean',
       'drop_vce_Range', 'owylis_vce_Range', 'mou_opkv_Range', 'totcalls',
       'eqpdays', 'custcare_Mean', 'callwait_Mean', 'iwylis_vce_Mean',
       'callwait_Range', 'ccrndmou_Range', 'ovrrev_Mean', 'roam_Mean',
       'recv_sms_Mean', 'blck_dat_Mean', 'mou_pead_Mean', 'da_Mean',
       'da_Range', 'datovr_Mean', 'drop_dat_Mean', 'comp_dat_Mean'],
      dtype='object')
```

APPENDIX

APPENDIX 2.6 - CHI SQUARE TEST

PFB the code and the output for Chi Square Test.

```
def chi_square_test_cat(df,df_cat,dep_var='churn'):
    dict_chi={"ColumnName":[],"Chi_Value":[],"p_Value":[],"DoF":[],"Frequency":[]}
    for col in df_cat:
        if col == dep_var:
            continue
        else:
            ct_temp=pd.crosstab(df[col],df[dep_var])
            chi,p_val,dof,freq=stats.chi2_contingency(ct_temp)
            dict_chi["ColumnName"].append(col)
            dict_chi["Chi_Value"].append(chi)
            dict_chi["p_Value"].append(p_val)
            dict_chi["DoF"].append(dof)
            dict_chi["Frequency"].append(freq)
    return dict_chi
```

ColumnName	Chi_Value	p_Value	ColumnName	Chi_Value	p_Value
income	36.455494	3.290607e-05	forgntvl	0.100936	0.750710
crclscod	141.026655	4.380207e-11	mtrcycle	0.574078	0.448643
customer_type	85.965095	1.831399e-20	truck	0.515877	0.472606
prizm_social_one	12.716710	2.618318e-02	car_buy	3.333988	0.067862
area	55.749581	9.760999e-06			
refurb_new	25.197643	5.174539e-07			
hnd_webcap	148.218206	6.528681e-33			
marital	19.859459	5.323321e-04			
ethnic	98.815523	5.772519e-14			
age1	95.920474	1.066356e-06			
age2	64.751107	1.362996e-02			
models	42.176879	6.974271e-06			
hnd_price	230.652506	1.174373e-40			
actvsubs	28.304678	8.482123e-04			
uniqsubs	66.669940	1.940148e-10			
csa	830.139312	2.479150e-04			

APPENDIX

APPENDIX 3.1 - MISSING VALUE TREATMENT

List of columns with more than 30% missing values.

	Missing %
solflag	98.06
retdays	96.73
wrkwoman	87.66
div_type	81.08
occu1	73.49
proptype	71.60
cartype	67.71
children	66.23
mailordr	64.15
mailresp	62.68
numbcars	48.87
dwllsize	38.05
dwlltype	31.60

List of columns with less than 3% missing values.

```
['marital',
 'ethnic',
 'age1',
 'age2',
 'forgntvl',
 'truck',
 'mtrcycle',
 'car_buy',
 'hnd_price',
 'change_mou',
 'roam_Mean',
 'da_Mean',
 'da_Range',
 'datovr_Mean',
 'ovrrev_Mean',
 'rev_Mean',
 'ovrmou_Mean',
 'datovr_Range',
 'mou_Range',
 'totmrc_Mean',
 'rev_Range',
 'mou_Mean',
 'csa',
 'area',
 'avgqty',
 'drop_vce_Range',
 'blck_dat_Mean',
 'mou_pead_Mean',
 'churn',
 'totcalls',
 'months',
 'mou_opkv_Range',
 'owylis_vce_Range'
 'drop_blk_Mean',
 'crclscod',
 'drop_dat_Mean',
 'drop_vce_Mean',
 'adjmou',
 'totrev',
 'adjrev',
 'avgrev',
 'comp_dat_Mean',
 'recv_sms_Mean',
 'eqpdays',
 'custcare_Mean']
```

APPENDIX

APPENDIX 3.2 - OUTLIER TREATMENT

Code for generating below data.

```
outlier_dict={}
for col in df_cont:
    outlier_list=[]
    q1,q3=np.nanpercentile(df_cont[col],[25,75])
    q1,q3
    iqr=q3-q1
    iqr
    lower=q1-(1.5*iqr)
    upper=q3+(1.5*iqr)
    upper,lower
    for i in df[col]:
        if i < lower or i > upper:
            outlier_list.append(i)
    #print(color.BOLD + color.RED + "Variable: {}".format(col) +color.END)
    #print("Outliers: {}".format(np.sort(outlier_list)))
    #print("Outlier Count: {}".format(len(outlier_list)))
    outlier_dict.update({col:len(outlier_list)})
outlier_cont_df=pd.DataFrame(data=outlier_dict,index=['Outlier Count']).T.sort_values(by='Outlier Count',ascending=False)
outlier_cont_df['Percent Outlier']=outlier_cont_df['Outlier Count']/df.shape[0]*100
outlier_cont_df
```

List of all the columns along with their respective outlier count and percentage.

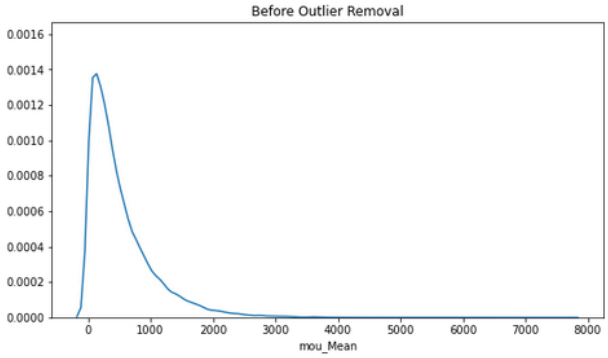
	Outlier Count	Percent Outlier
roam_Mean	4705	18.348087
plcd_dat_Mean	3990	15.559802
callwait_Mean	3831	14.939750
datovr_Mean	3595	14.019421
datovr_Range	3594	14.015521
change_mou	3557	13.871232
comp_dat_Mean	3556	13.867332
custcare_Mean	3461	13.496861
ccrndmou_Range	3244	12.650626
ovrmou_Mean	2983	11.632804
ovrrev_Mean	2927	11.414421
da_Mean	2771	10.806068
da_Range	2664	10.388800
opk_dat_Mean	2577	10.049526
iwylis_vce_Mean	2504	9.764848
mou_pead_Mean	2376	9.265687
callwait_Range	2255	8.793823
mou_opkv_Range	2185	8.520844
rev_Range	2115	8.247865
owylis_vce_Range	1945	7.584916
drop_blk_Mean	1928	7.518621
drop_vce_Mean	1923	7.499123
adjqty	1870	7.292438
totcalls	1869	7.288539
mou_Range	1718	6.699684
drop_vce_Range	1681	6.555395
adjmou	1606	6.262918
rev_Mean	1500	5.849550
adjrev	1492	5.818352

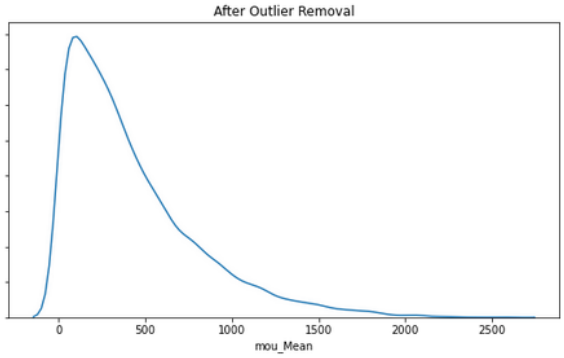
	Outlier Count	Percent Outlier
totrev	1492	5.818352
avg3qty	1401	5.463479
avgqty	1340	5.225598
plcd_vce_Mean	1338	5.217798
avg3mou	1337	5.213899
avg6qty	1333	5.198300
avgrev	1329	5.182701
mou_Mean	1305	5.089108
avg6mou	1296	5.054011
comp_vce_Mean	1264	4.929220
avgmou	1236	4.820029
drop_dat_Mean	724	2.823383
eqpdays	680	2.651796
months	574	2.238428
totmrc_Mean	517	2.016145
blck_dat_Mean	333	1.298600
recv_sms_Mean	215	0.838435

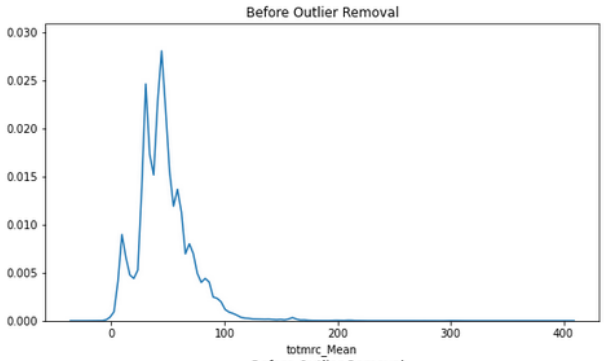
APPENDIX

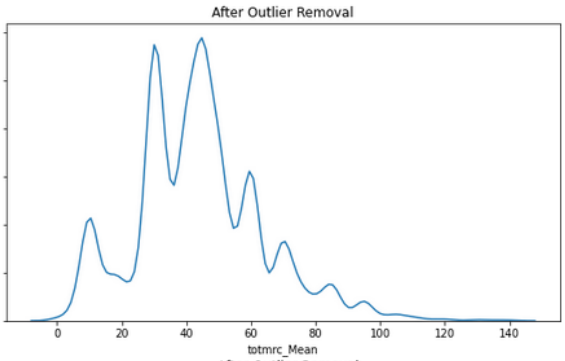
APPENDIX 3.2 - OUTLIER TREATMENT

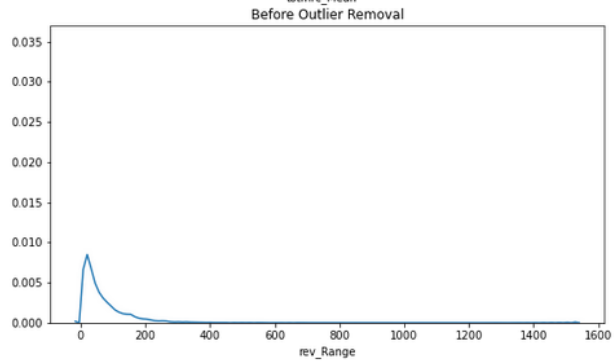
Distribution plot for all the continuous variables before and after outlier treatment.

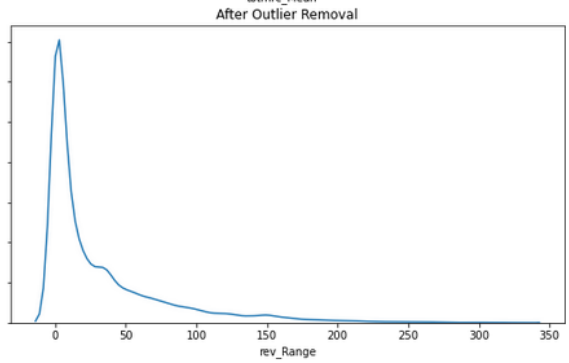


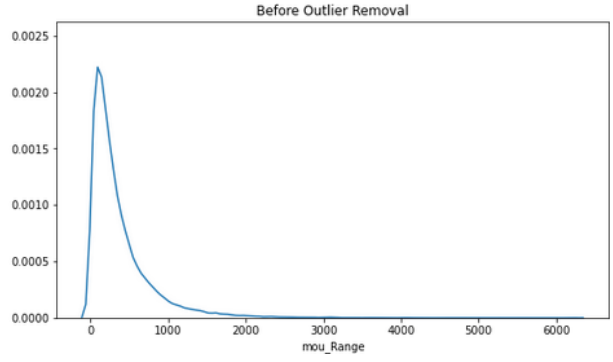


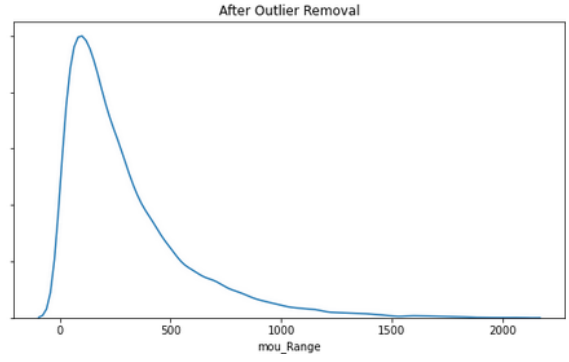






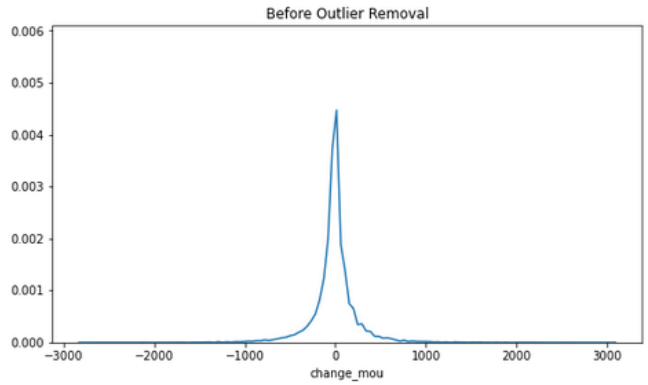


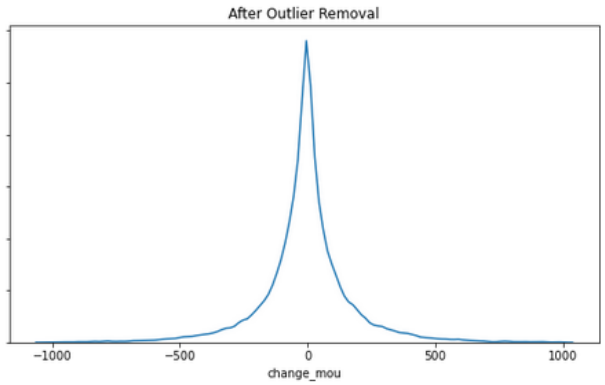


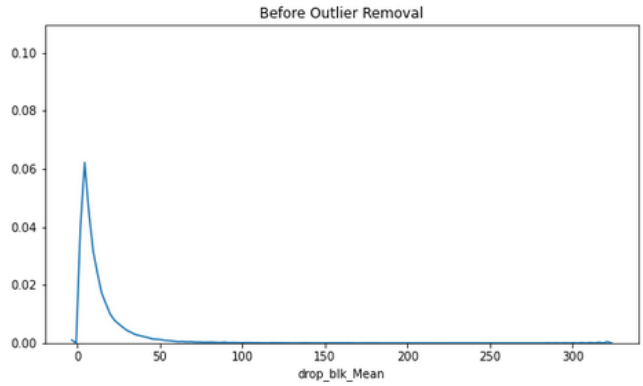


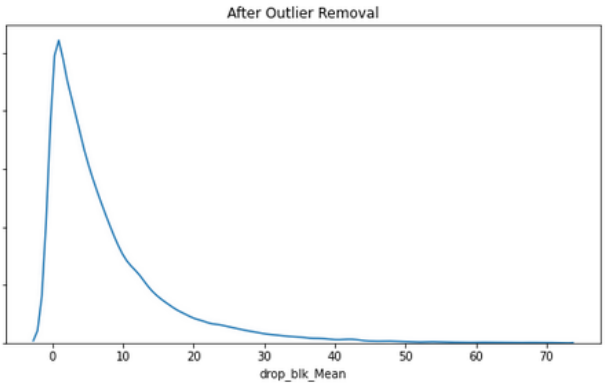
APPENDIX

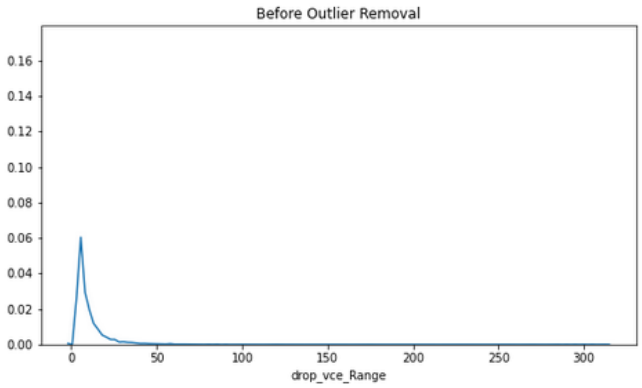
APPENDIX 3.2 - OUTLIER TREATMENT

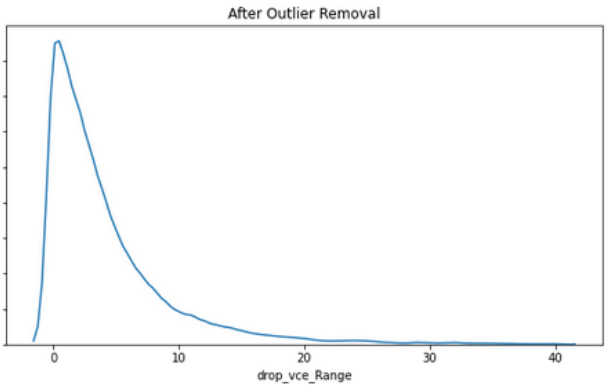


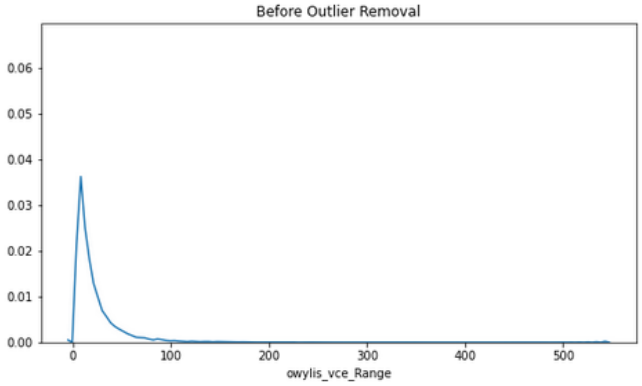


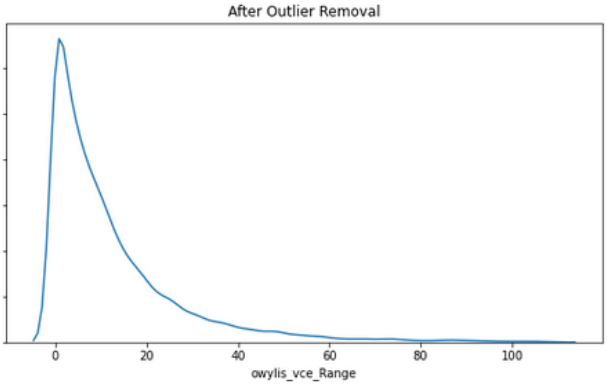






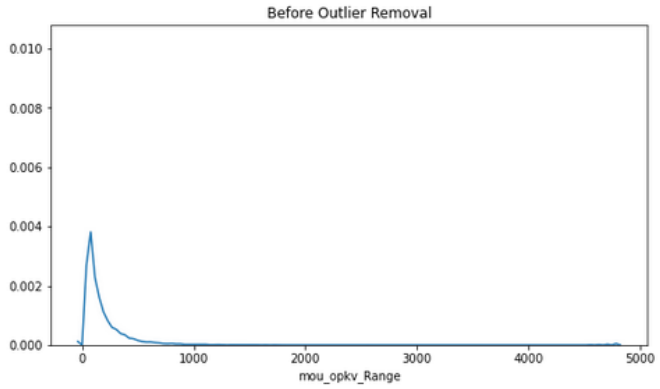


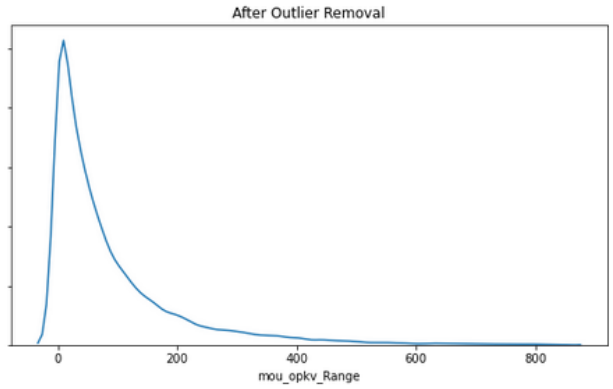


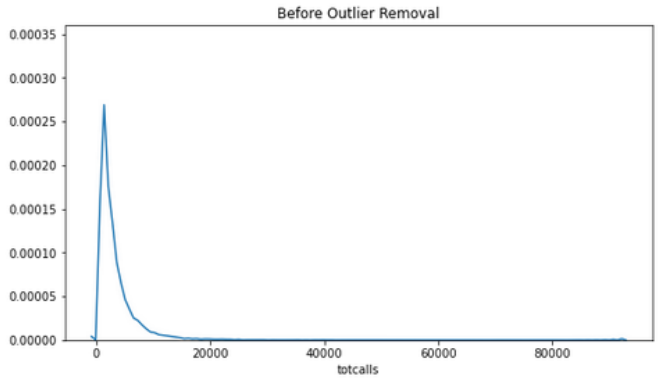


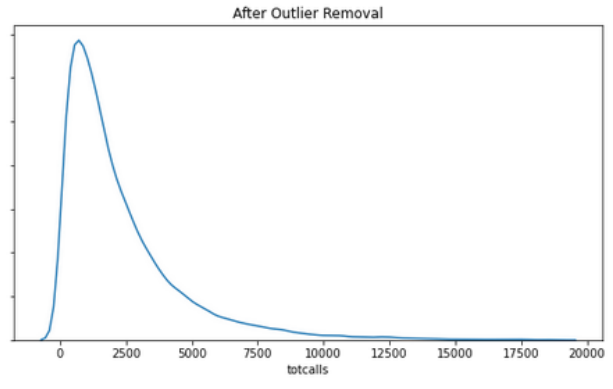
APPENDIX

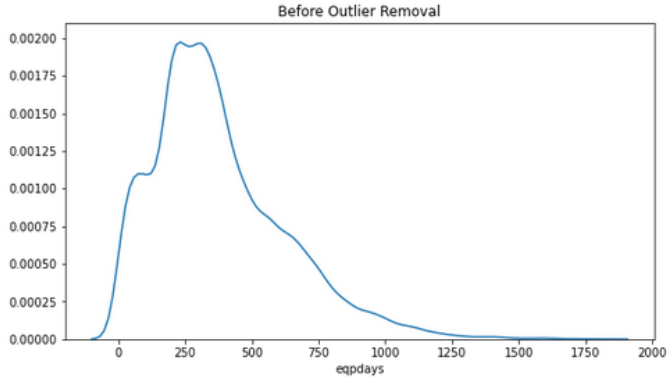
APPENDIX 3.2 - OUTLIER TREATMENT

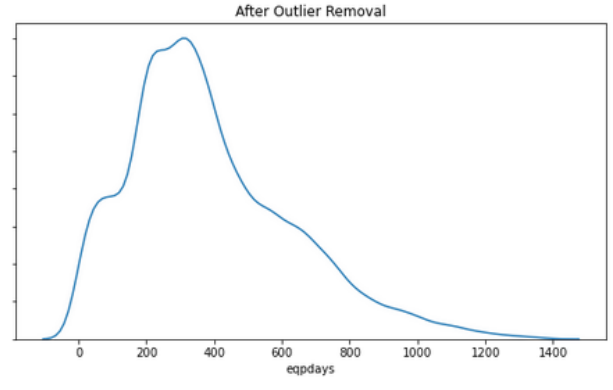






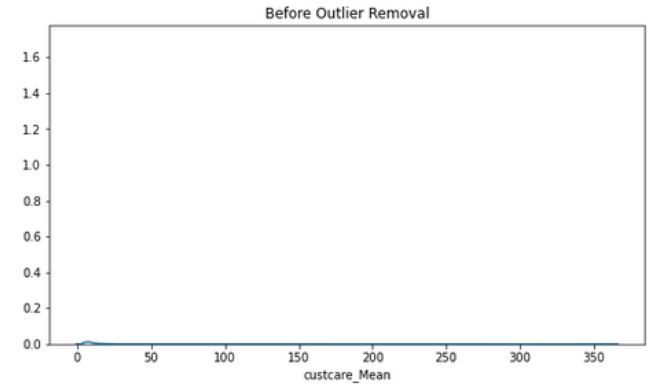


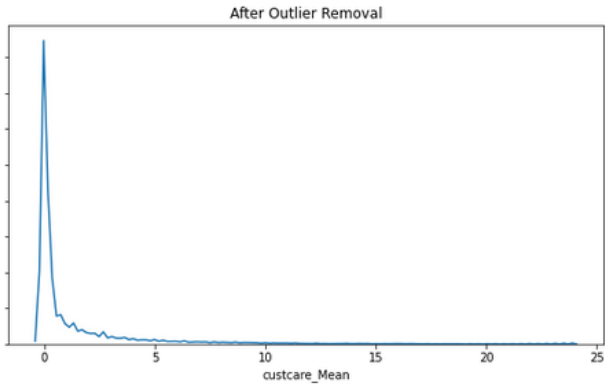


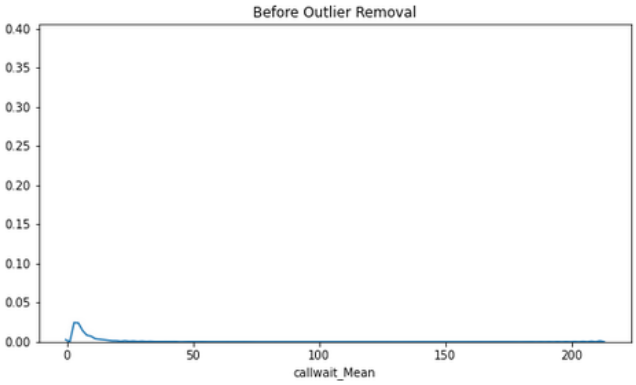


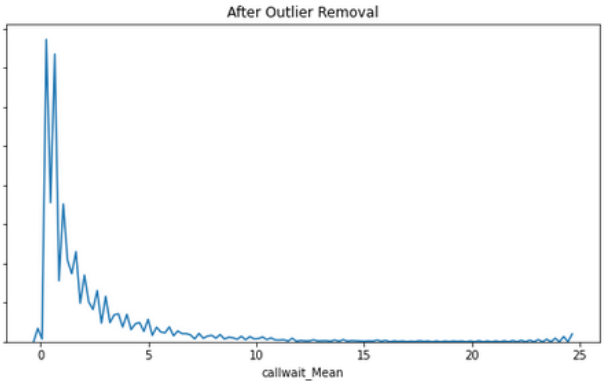
APPENDIX

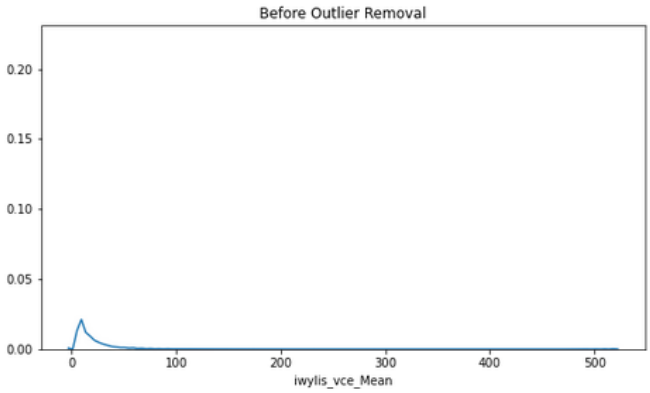
APPENDIX 3.2 - OUTLIER TREATMENT

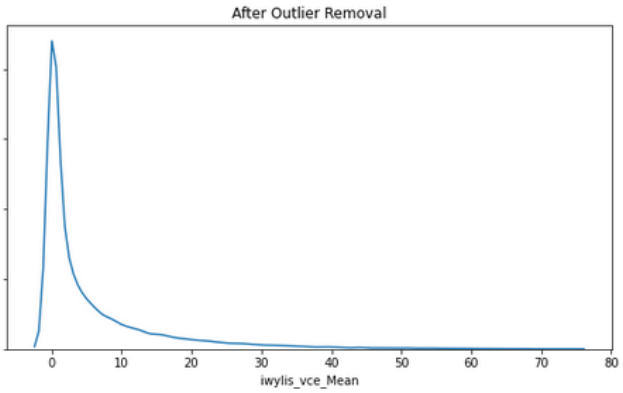


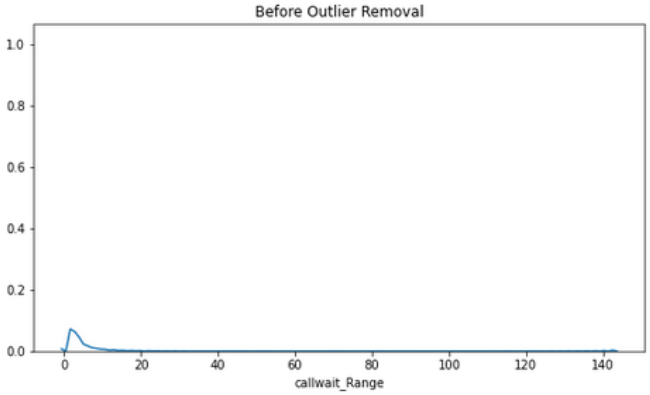


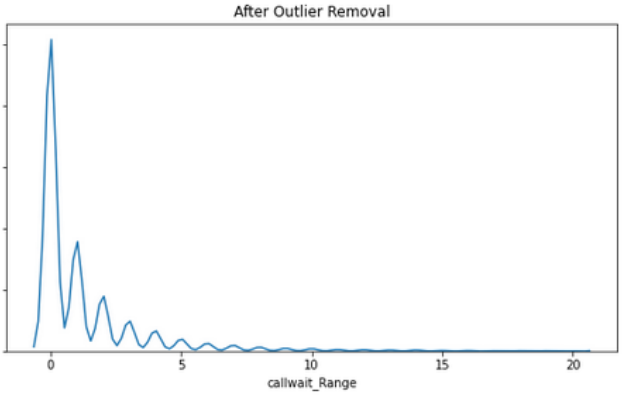






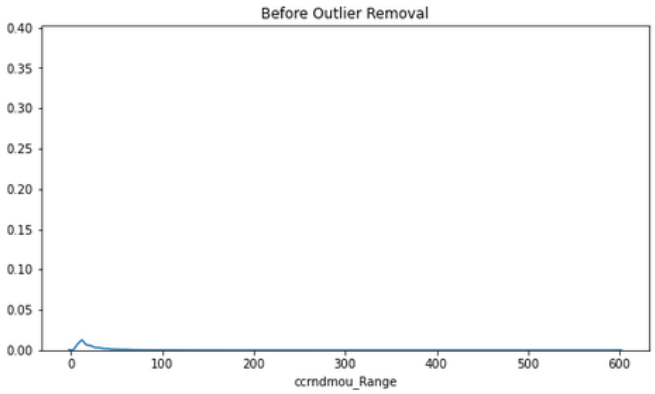


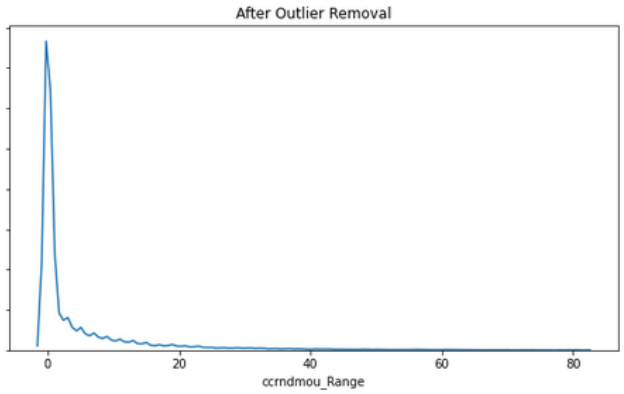


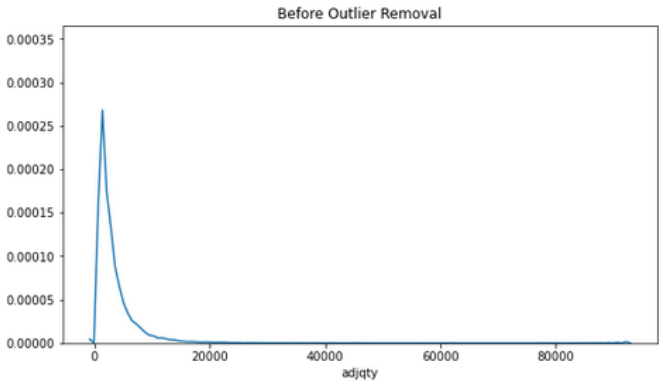


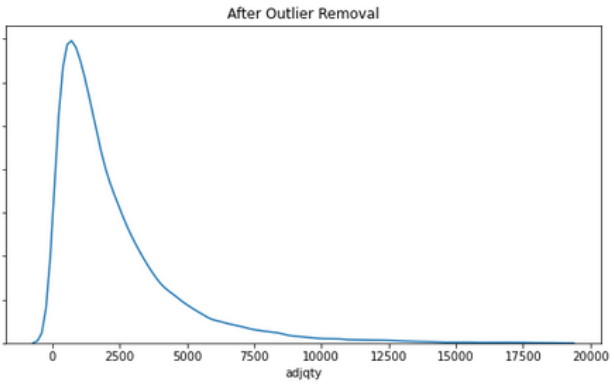
APPENDIX

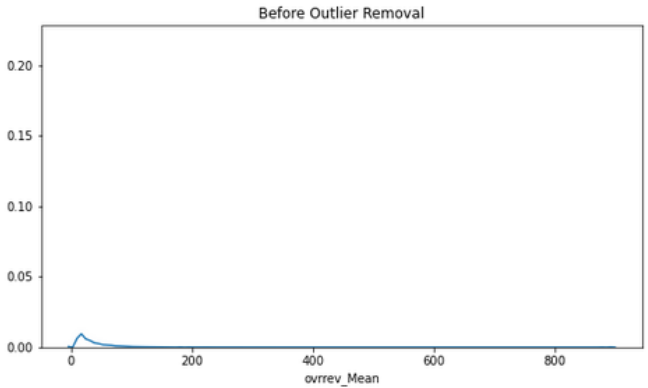
APPENDIX 3.2 - OUTLIER TREATMENT

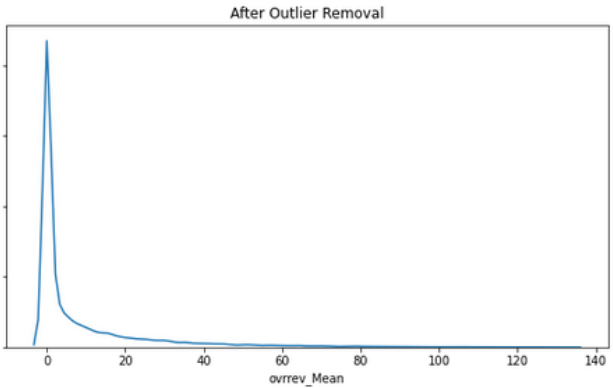


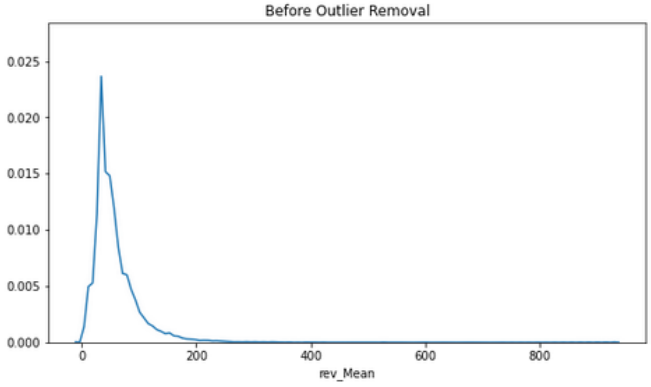


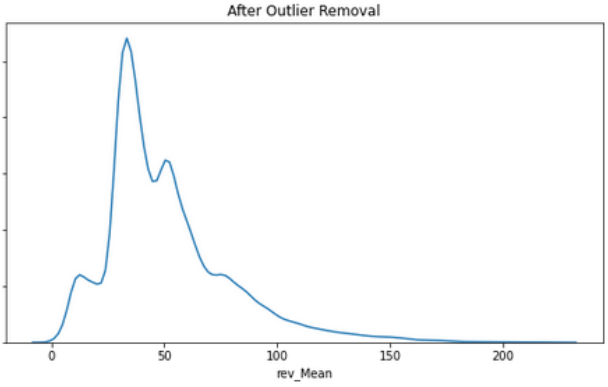






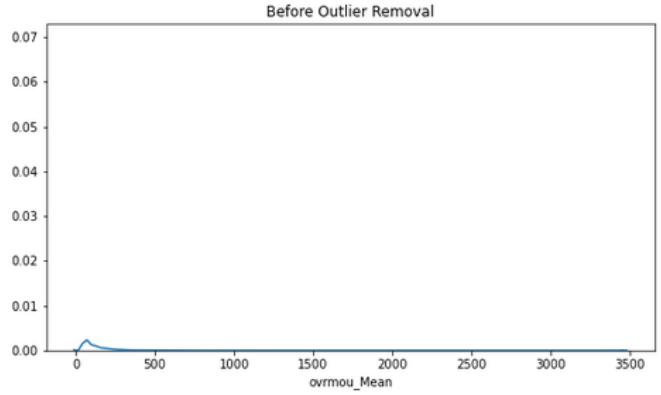


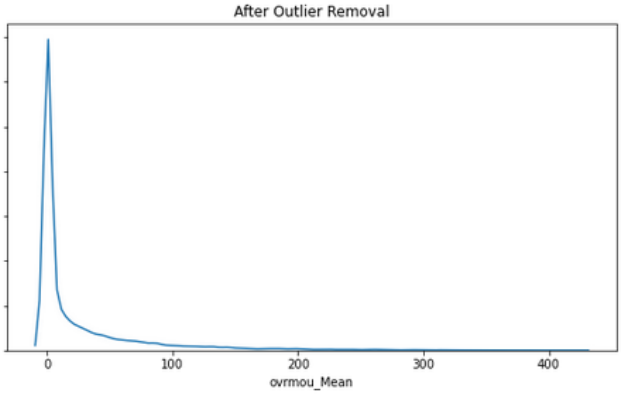


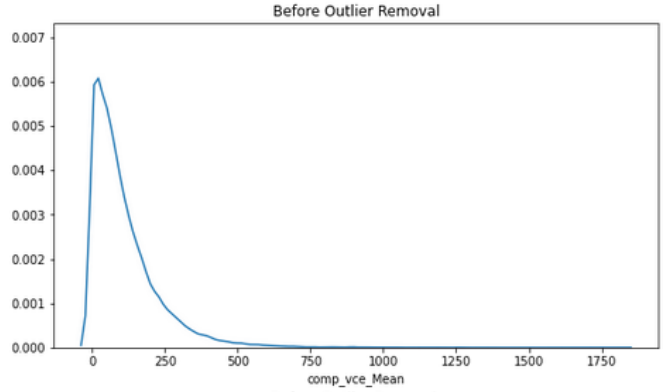


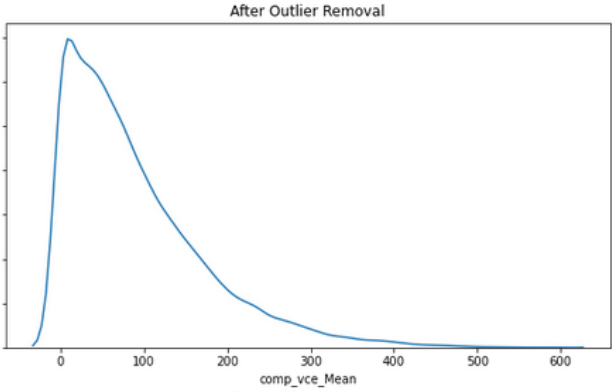
APPENDIX

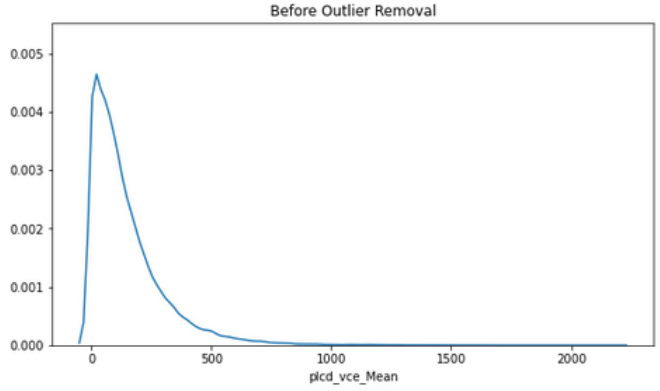
APPENDIX 3.2 - OUTLIER TREATMENT

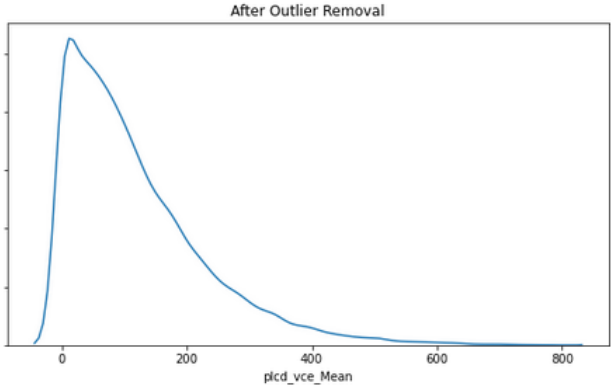


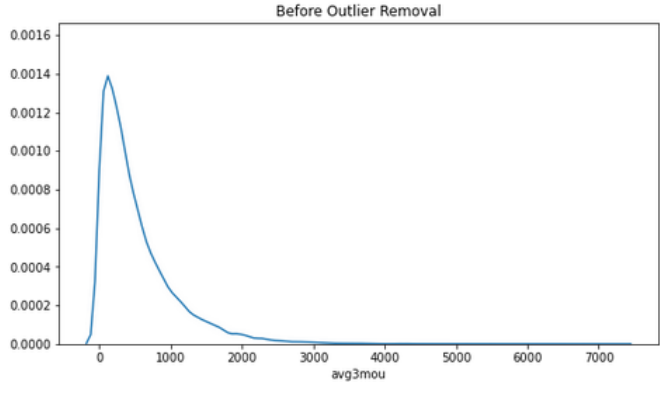


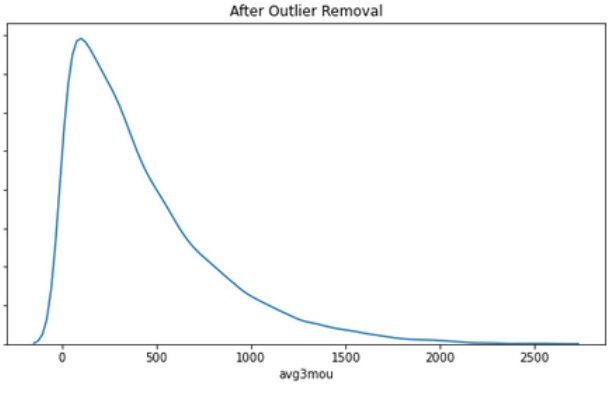






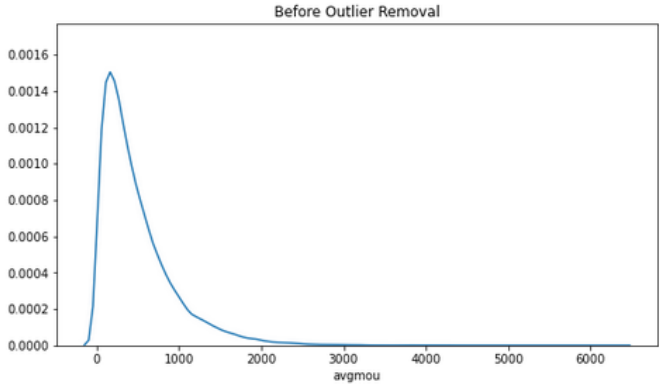


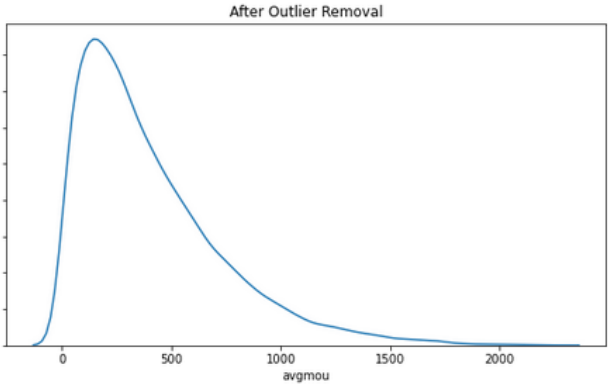


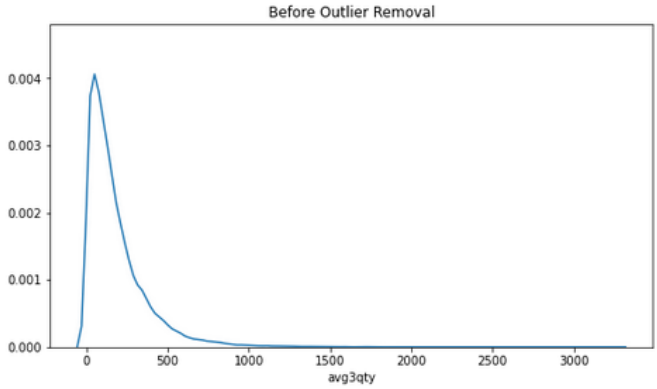


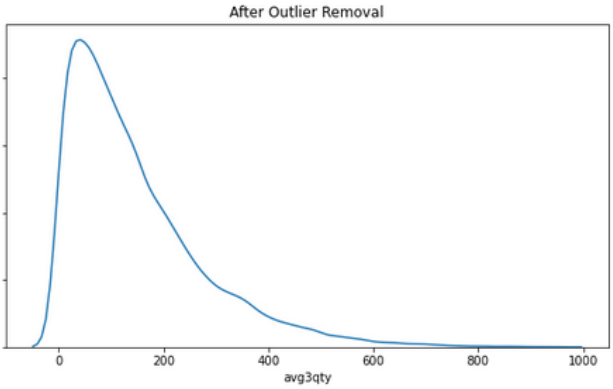
APPENDIX

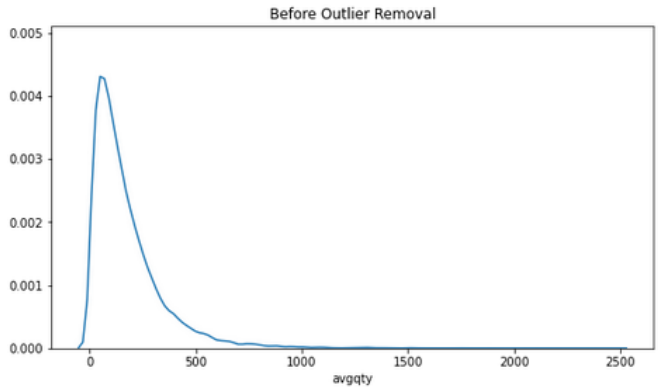
APPENDIX 3.2 - OUTLIER TREATMENT

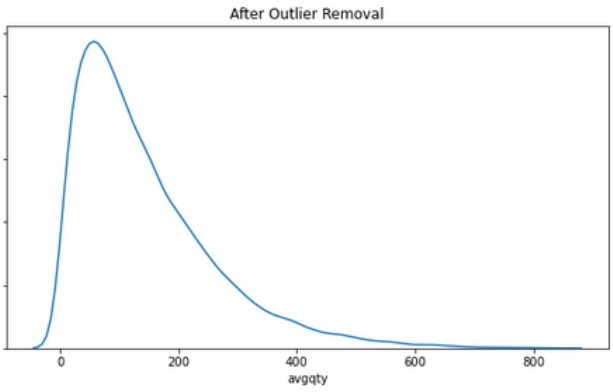


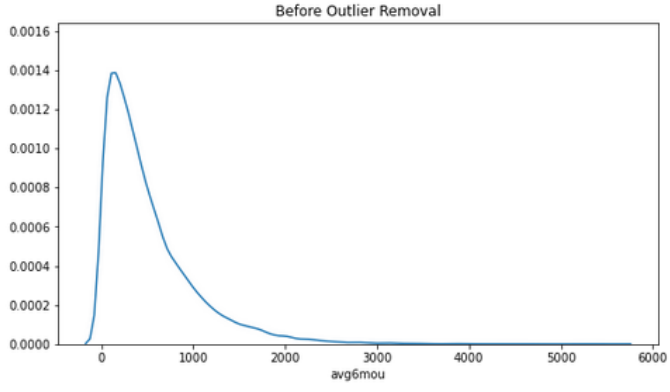


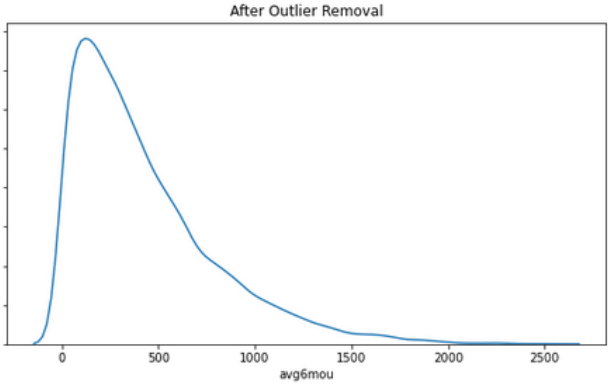






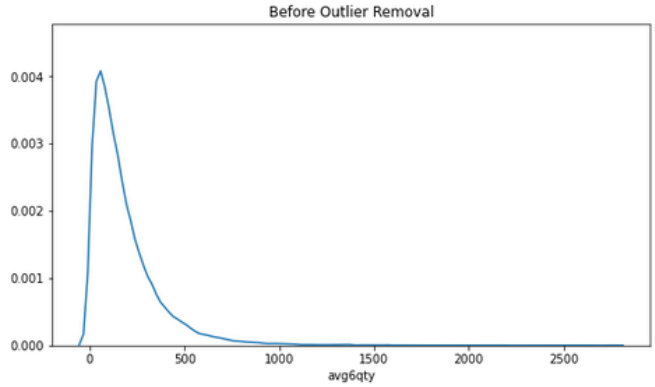


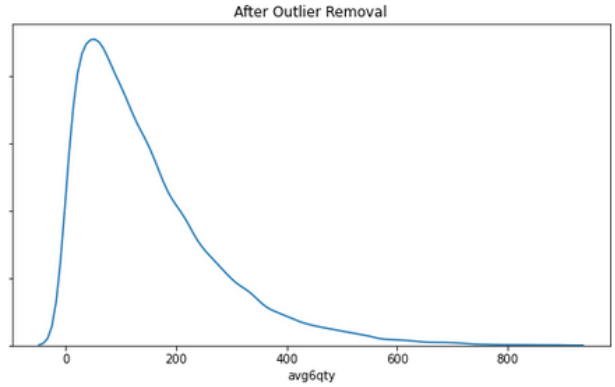


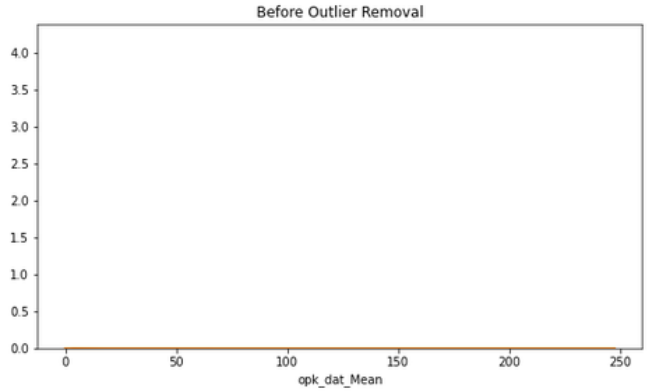


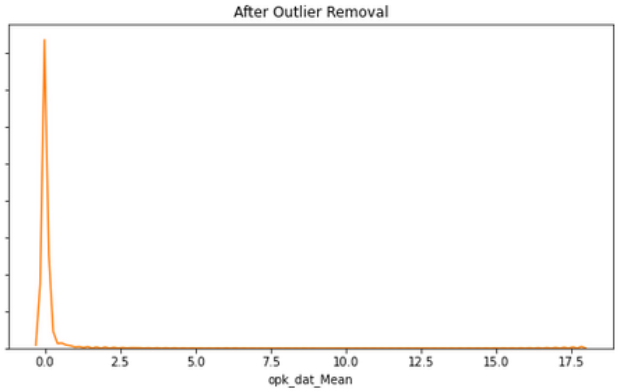
APPENDIX

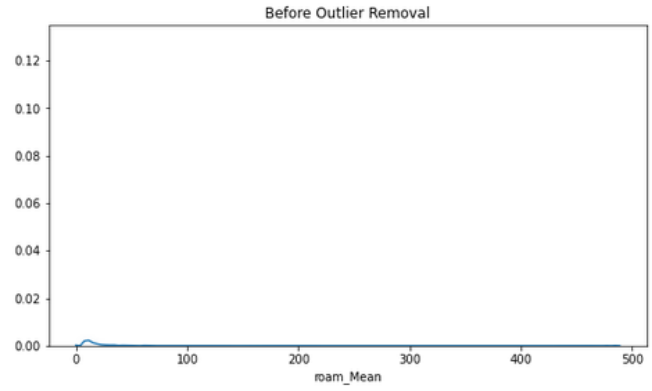
APPENDIX 3.2 - OUTLIER TREATMENT

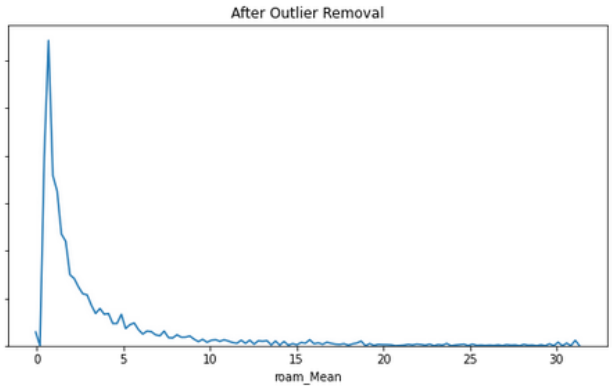


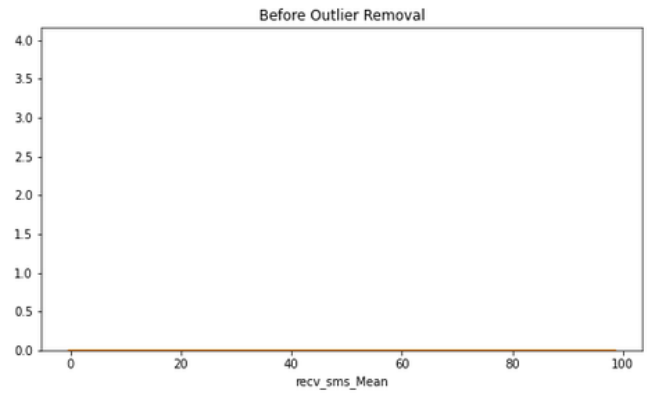


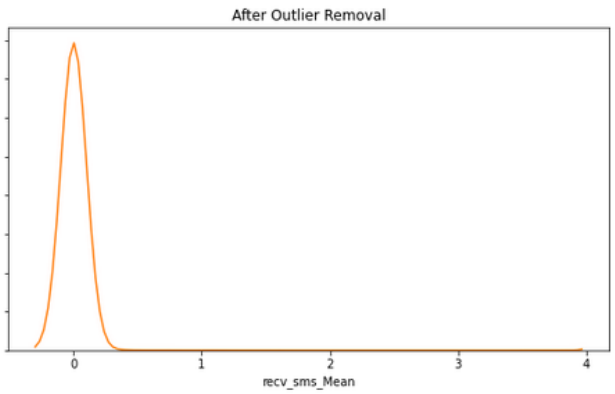






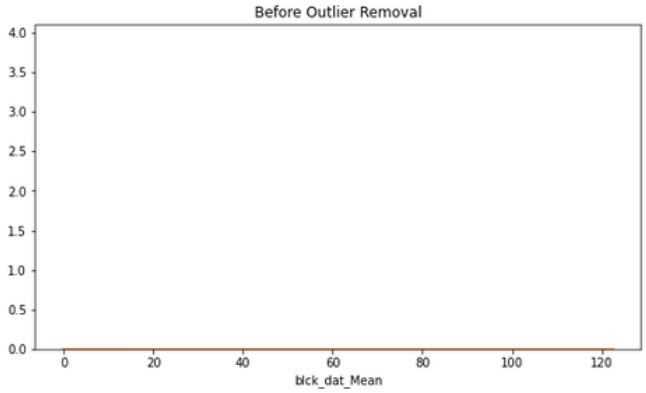


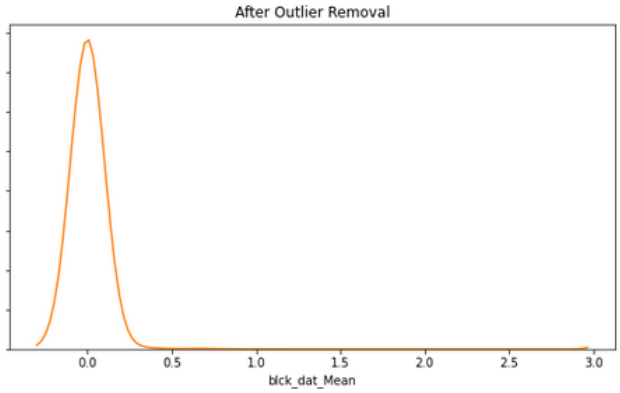


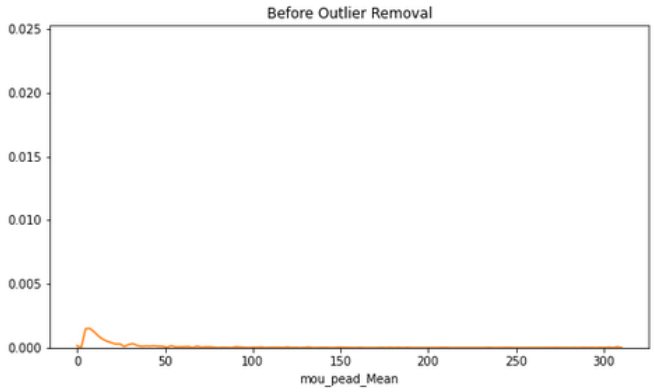


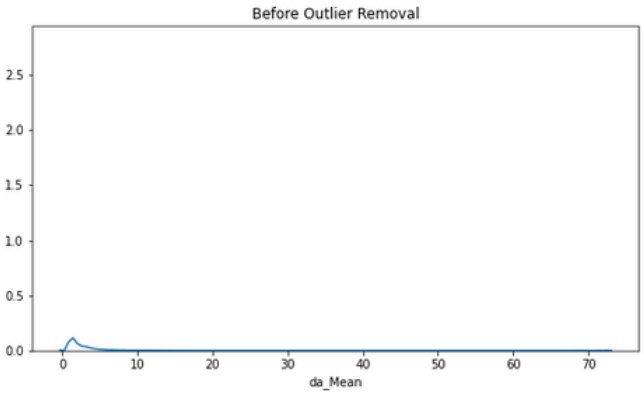
APPENDIX

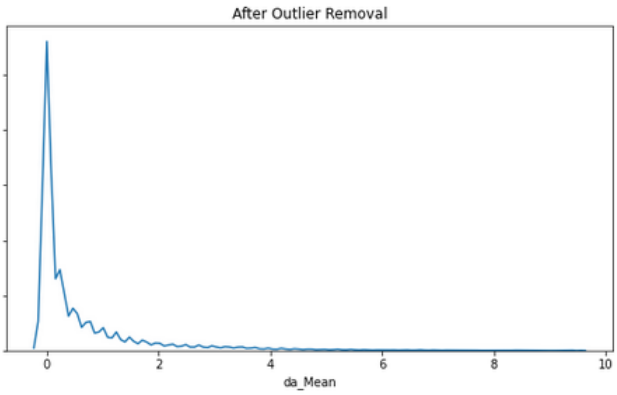
APPENDIX 3.2 - OUTLIER TREATMENT

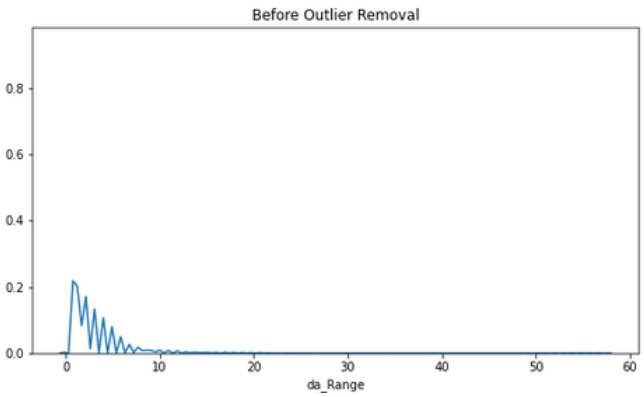


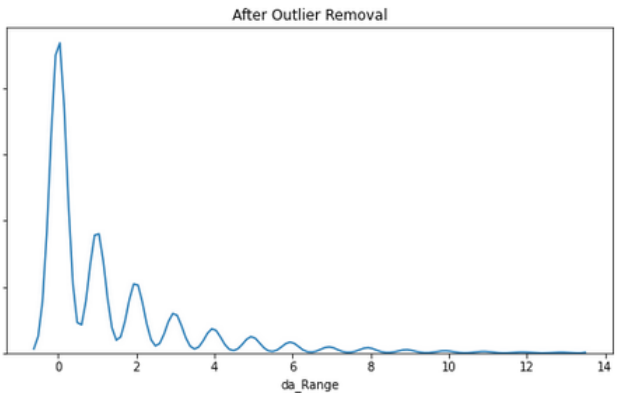






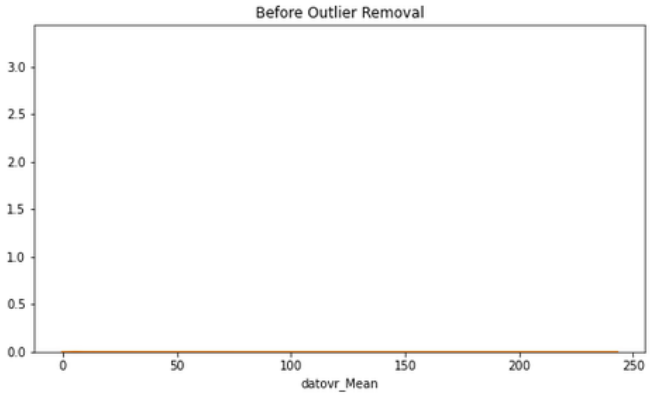


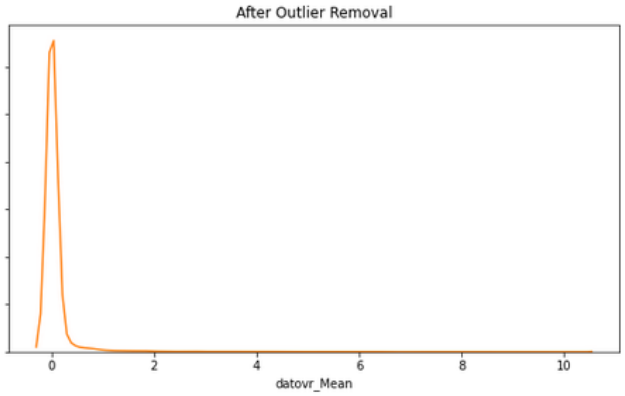


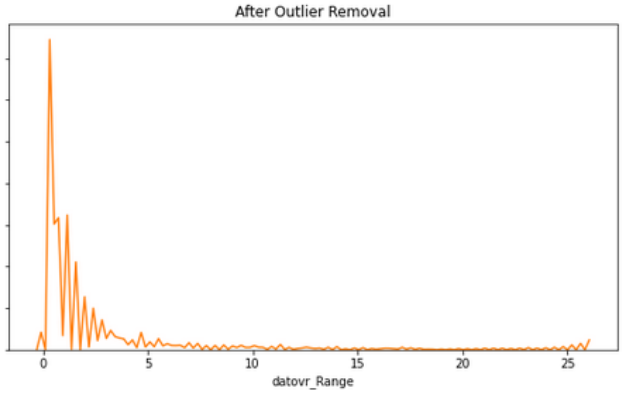


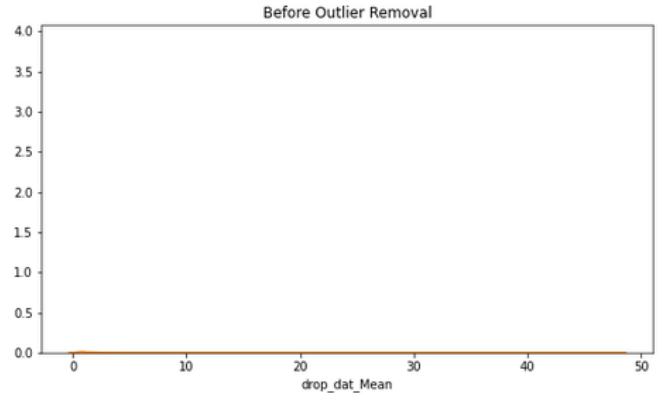
APPENDIX

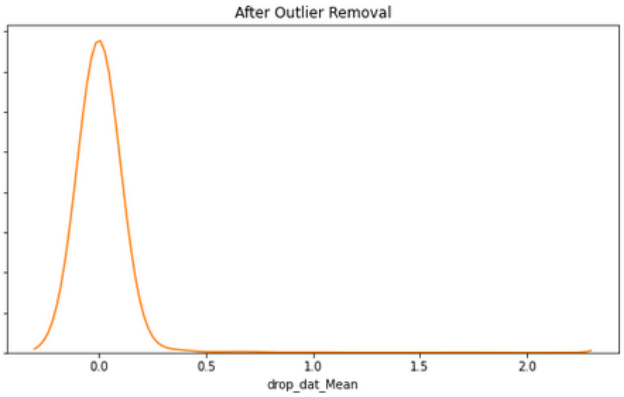
APPENDIX 3.2 - OUTLIER TREATMENT

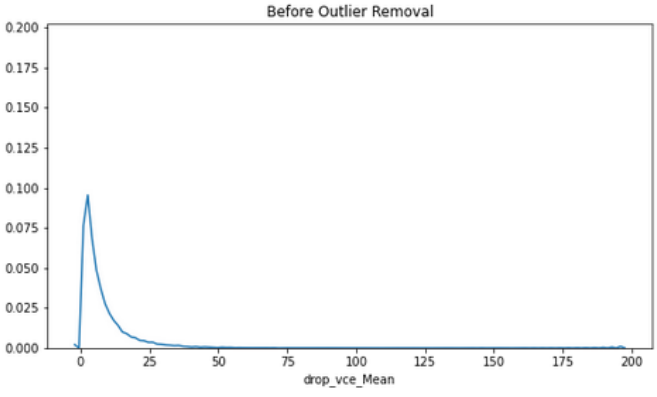


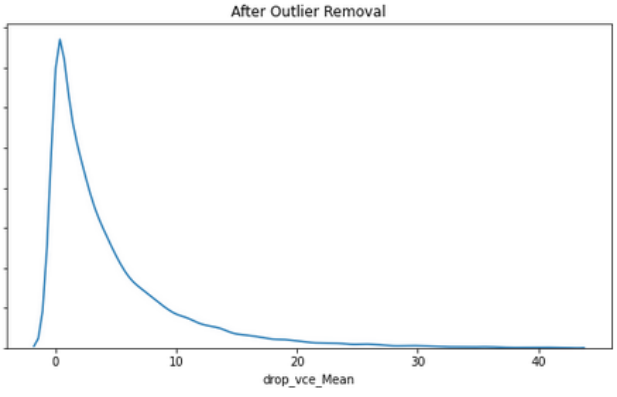






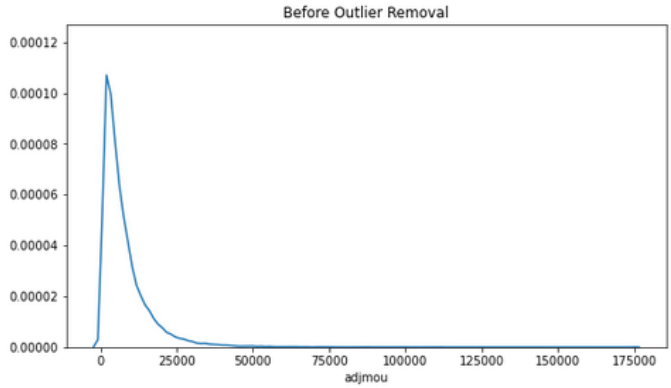


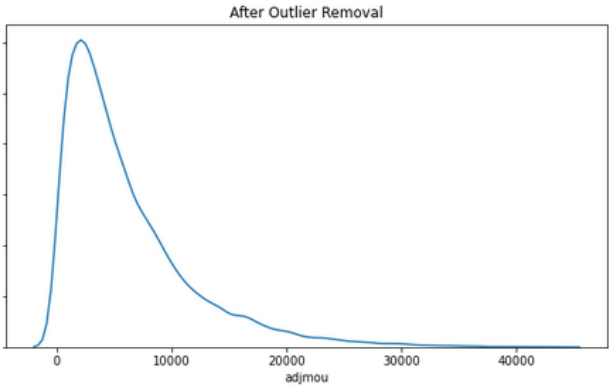


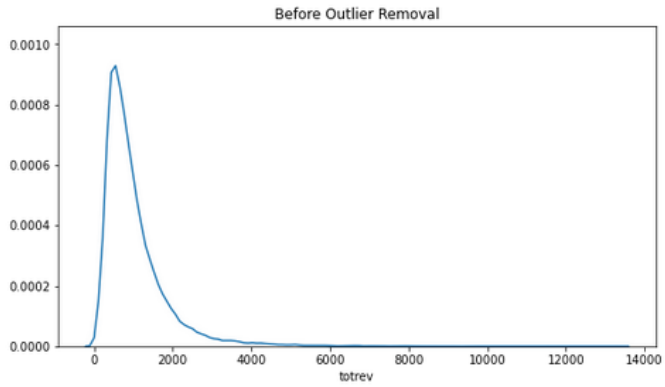


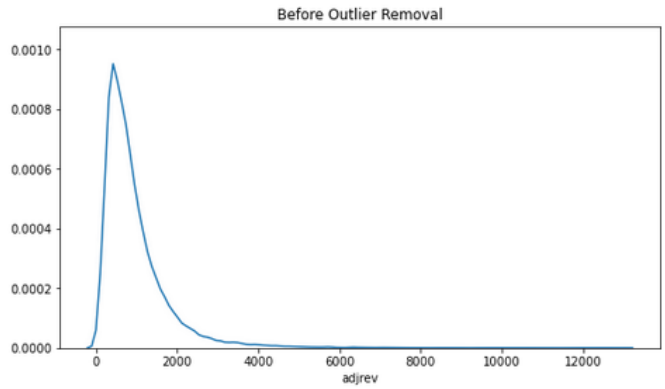
APPENDIX

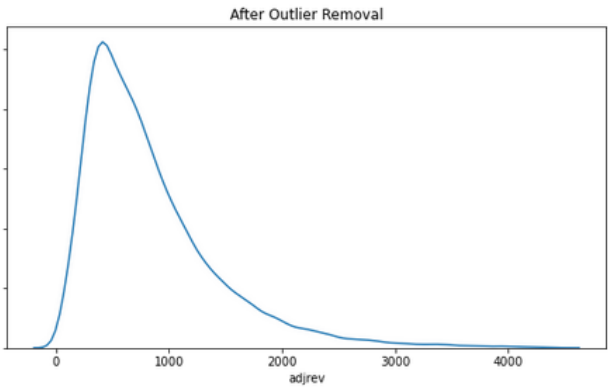
APPENDIX 3.2 - OUTLIER TREATMENT

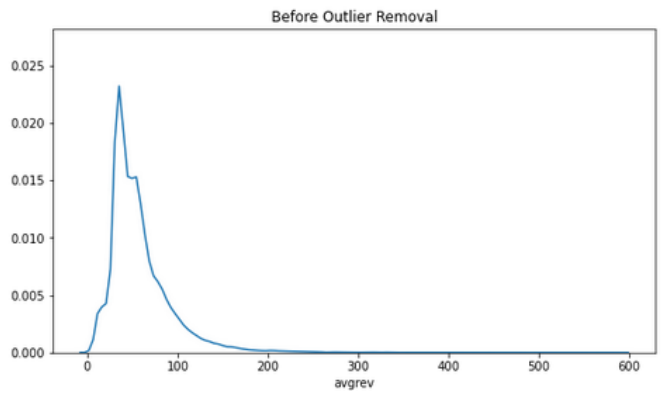


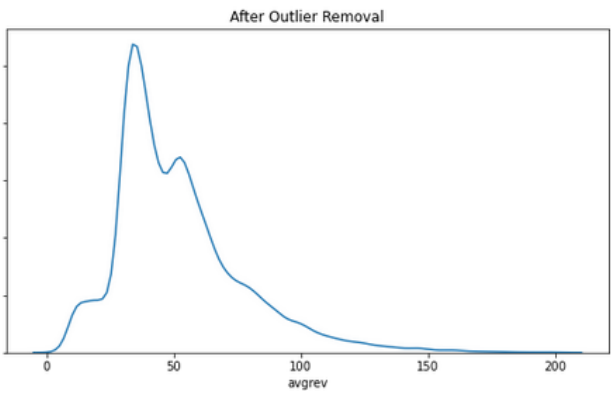






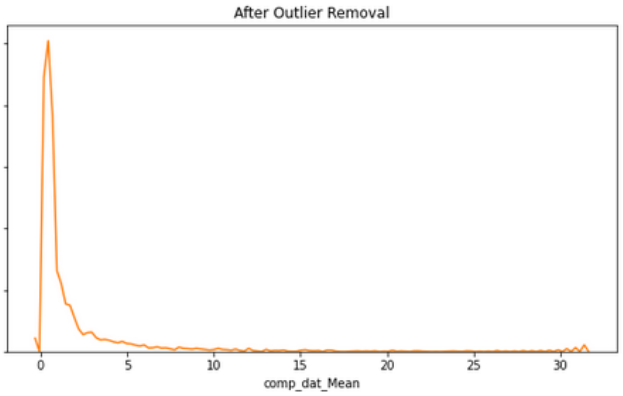


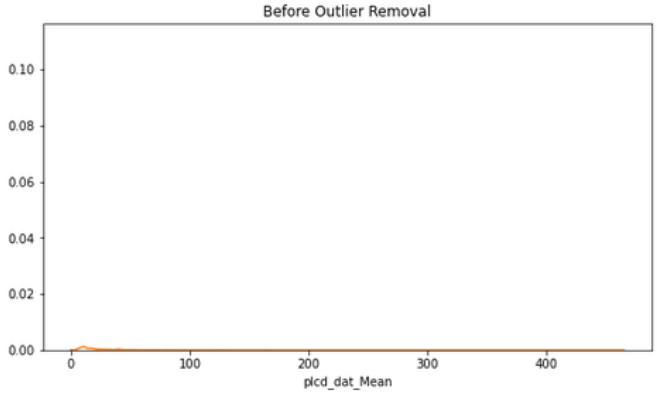


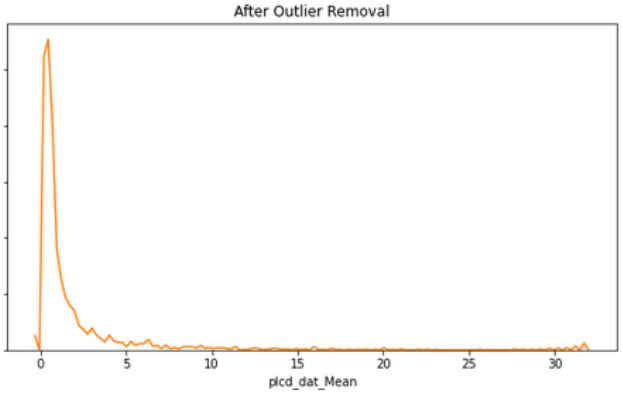


APPENDIX

APPENDIX 3.2 - OUTLIER TREATMENT







APPENDIX

APPENDIX 3.3 - VARIABLE TRANSFORMATION

All the columns which were bucketed together as part of variable transformation.

CSA

```
df['csa']=np.where(np.logical_not(df['csa'].isin(df['csa'].value_counts().head((df['csa'].value_counts()>100).sum()).index)),
                   'OTHERS',
                    df['csa'])
```

```
df['csa'].value_counts()
```

```
OTHERS        10096
NYCBRO917       759
HOUHOU281       684
DALDAL214       658
NYCMAN917       536
APCFCH703       361
DALFTW817       327
APCSIL301       317
SANSAN210       302
SANAUS512       290
SFRSFR415       251
OHICOL614       238
ATLATL678       236
NYCQUE917       235
STLSTL314       232
SFROAK510       230
ATLANE678       229
BOSBOS617       226
SFRSCL408       223
NEVLVS702       214
MINMIN612       213
CHINBK847       211
KCYKCM816       211
PHXPHX602       204
SANMCA210       204
APCBAL410       199
MIAMIA305       190
NYCNEW201       190
INDIND317       174
NYCNAS516       171
DENDEN303       170
KCYKCK913       170
LAXANA714       170
CHICHI773       164
DETDET313       161
PHIPHI215       160
NSHNSH615       159
NEVSDG619       154
APCWAS202       154
NYCSUF516       146
MILMIL414       144
DETPON248       140
NOLKEN504       138
SEASEA206       132
NYCWHI914       128
NNYBUF716       128
FLNORL407       127
OHICIN513       124
HARHAR860       123
NYCNEW973       122
CHILAG630       121
BOSMAN603       117
FLNTAM813       115
OMAOMA402       111
MINSTP612       109
MIAFTL954       109
SFRSMO650       106
HWIHON808       106
LOULOU502       105
LAXSAN714       102
FLNCLR813       102
Name: csa, dtype: int64
```

APPENDIX

APPENDIX 3.3 - VARIABLE TRANSFORMATION

ACTIVSUBS

```
df['actvsubs'].value_counts()
```

```
1     16047
2      5586
3       834
4       175
5        60
0        21
6         4
11        1
Name: actvsubs, dtype: int64
```

```
df['actvsubs']=np.where(df['actvsubs']>4,5,df['actvsubs'])
```

```
df['actvsubs'].dtype
```

```
dtype('int64')
```

```
df['actvsubs'].value_counts()
```

```
1    16047
2     5586
3      834
4      175
5       65
0       21
Name: actvsubs, dtype: int64
```

UNIQSUBS

```
df['uniqsubs'].value_counts()
```

```
1     14248
2      6295
3      1395
4       543
5       168
6        54
7        17
9         3
8         3
12        1
11        1
Name: uniqsubs, dtype: int64
```

```
df['uniqsubs']=np.where(df['uniqsubs']>5,6,df['uniqsubs'])
```

```
df['uniqsubs'].dtype
```

```
dtype('int64')
```

```
df['uniqsubs'].value_counts()
```

```
1    14248
2     6295
3     1395
4      543
5      168
6       79
Name: uniqsubs, dtype: int64
```

APPENDIX

APPENDIX 3.3 - VARIABLE TRANSFORMATION

ETHNIC

```
df['ethnic'].value_counts()
```

```
N     7909
S     3015
H     2913
U     2574
G     1424
Z     1044
O      900
I      861
J      637
F      510
B      300
R      213
D      195
P      124
C       54
M       30
X       25
Name: ethnic, dtype: int64
```

```
df['ethnic']=np.where(np.logical_not(df['ethnic'].isin(df['ethnic'].value_counts().head((
    df['ethnic'].value_counts()>500).sum()).index)),'OTHERS',df['ethnic'])
```

```
df['ethnic'].value_counts()
```

```
N         7909
S         3015
H         2913
U         2574
G         1424
Z         1044
OTHERS     941
O          900
I          861
J          637
F          510
Name: ethnic, dtype: int64
```

CRCLSCOD

```
df['crclscod']=np.where(np.logical_not(df['crclscod'].isin(df['crclscod'].value_counts().head(11).index)),'OTHERS',
                        df['crclscod'])
```

```
df['crclscod'].value_counts()
```

```
AA        8810
A         3824
BA        2791
CA        1908
EA        1459
OTHERS     880
B          867
DA         787
ZA         737
C          345
E4         161
A2         159
Name: crclscod, dtype: int64
```

APPENDIX

APPENDIX 4.1 - MODELLING APPROACH

Below is the code for **apply_eval** generic function.

APPLY_EVAL

```
def apply_evl(name,model,param_grid,X_train,X_test,y_train,y_test):
    test_metrics_dict={}
    train_metrics_dict={}
    from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,f1_score,precision_score,recall_score,roc_auc_score
    from sklearn.model_selection import GridSearchCV
    print(name)
    if param_grid!=None:
        grid_search = GridSearchCV(estimator = model, param_grid = param_grid, cv = 3,n_jobs=4)
        gs=grid_search.fit(X_train,y_train)
        print("--------------------Best Parameters--------------------")
        print(gs.best_params_)
        best_model=gs.best_estimator_
        print()
        print()
        print("--------------------Best Model Params--------------------")
        print(best_model)
        print()
        print()
        ytrain_predict = best_model.predict(X_train)
        ytest_predict = best_model.predict(X_test)
    elif param_grid == None:
        best_model=model
        best_model.fit(X_train,y_train)
        ytrain_predict = best_model.predict(X_train)
        ytest_predict = best_model.predict(X_test)

    print("Train Accuracy Score for model {} is {}".format(model,accuracy_score(y_train,ytrain_predict)))
    print()
    print()
    print("--------------------Classification Report - Train Data--------------------")
    print(classification_report(y_train,ytrain_predict))
    print("------------------------------------------------------------")
    print()
    print()
    sns.heatmap(confusion_matrix(y_train,ytrain_predict),annot=True, fmt='d',
            cbar=False,cmap='YlGnBu')
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')
    plt.title('Confusion Matrix - Train Data')
    plt.show()
    print()
    print()
    print("------------------------------------------------------------")

        # AUC and ROC for the training data

    # predict probabilities
    probs = best_model.predict_proba(X_train)
     # keep probabilities for the positive outcome only
     probs = probs[:, 1]
     # calculate AUC
     from sklearn.metrics import roc_auc_score
     auc = roc_auc_score(y_train, probs)
     print('AUC: %.3f' % auc)
     # calculate roc curve
     from sklearn.metrics import roc_curve
     fpr_train, tpr_train, thresholds = roc_curve(y_train, probs)
     plt.plot([0, 1], [0, 1], linestyle='--')
     # plot the roc curve for the model
     plt.plot(fpr_train, tpr_train, marker='.')
     plt.title('Train Data - ROC Curve')
     # show the plot
```

APPENDIX

APPENDIX 4.1 - MODELLING APPROACH

APPLY_EVAL

```
    plt.show()
    print()
    print()
    print("------------------------------------------------------------")


#     train_metrics_dict.update({"Accuracy":accuracy_score(y_train,ytrain_predict)})
#     train_metrics_dict.update({"Precision":precision_score(y_train,ytrain_predict)})
#     train_metrics_dict.update({"Recall":recall_score(y_train,ytrain_predict)})
#     train_metrics_dict.update({"F1":f1_score(y_train,ytrain_predict)})
#     train_metrics_dict.update({"AUC":roc_auc_score(y_train,probs)})
#     train_metrics_df=pd.DataFrame(train_metrics_dict,index=[name+"_Train"]).T

    train_metrics_dict.update({name+"_Train":[accuracy_score(y_train,ytrain_predict),precision_score(y_train,ytrain_predict),
                                              recall_score(y_train,ytrain_predict),f1_score(y_train,ytrain_predict),
                                              roc_auc_score(y_train,probs),fpr_train,tpr_train]})
    train_metrics_df=pd.DataFrame(train_metrics_dict,index=['Accuracy','Precision','Recall','F1','AUC','FPR',
                                                            'TPR'])


    print()
    print()

    print("Test Accuracy Score for model {} is {}".format(model,accuracy_score(y_test,ytest_predict)))
    print()
    print()
    print("--------------------Classification Report - Test Data--------------------")
    print(classification_report(y_test,ytest_predict))
    print("------------------------------------------------------------")
    print()
    print()
    sns.heatmap(confusion_matrix(y_test,ytest_predict),annot=True, fmt='d',
            cbar=False,cmap='YlGnBu')
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')
    plt.title('Confusion Matrix - Test Data')
    plt.show()
    print()
    print()
    print("------------------------------------------------------------")

        # AUC and ROC for the training data

    # predict probabilities
    probs = best_model.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    from sklearn.metrics import roc_auc_score
    auc = roc_auc_score(y_test, probs)
    print('AUC: %.3f' % auc)
    # calculate roc curve
    from sklearn.metrics import roc_curve
    fpr_test, tpr_test, thresholds = roc_curve(y_test, probs)
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr_test, tpr_test, marker='.')
    plt.title('Test Data - ROC Curve')
    # show the plot
    plt.show()
    print()
    print()
    print("------------------------------------------------------------")


    test_metrics_dict.update({name+"_Test":[accuracy_score(y_test,ytest_predict),precision_score(y_test,ytest_predict),
                                            recall_score(y_test,ytest_predict),f1_score(y_test,ytest_predict),
                                            roc_auc_score(y_test,probs),fpr_test,tpr_test]})
    test_metrics_df=pd.DataFrame(test_metrics_dict,index=['Accuracy','Precision','Recall','F1','AUC','FPR','TPR'])
    return pd.concat([train_metrics_df,test_metrics_df],axis=1),best_model
```

APPENDIX

APPENDIX 4.1 - MODELLING APPROACH

Below is the code for **tweak_threshold** generic function.

TWEAK_THRESHOLD

```
def tweak_threshold(model,X_train,X_test,y_train,y_test,step=0.1):

    dict_metrics={"AUC":[],
         "Accuracy":[],
         "Recall":[],
         "Precision":[],
         "F1_Score":[],
                 "Threshold":[],
         "Test/Train":[]}
    pred_prob_train = model.predict_proba(X_train)
    pred_prob_test = model.predict_proba(X_test)
    #print("--------------- TRAIN DATA-----------------")
    for j in np.arange(0.1,1,step):
        custom_prob = j #defining the cut-off value of our choice
        custom_cutoff_data=[]#defining an empty list
        for i in range(0,len(y_train)):#defining a loop for the length of the test data
            if np.array(pred_prob_train[:,1])[i] > custom_prob:#issuing a condition for our probability values to be
                #greater than the custom cutoff value
                a=1#if the probability values are greater than the custom cutoff then the value should be 1
            else:
                a=0#if the probability values are less than the custom cutoff then the value should be 0
            custom_cutoff_data.append(a)#adding either 1 or 0 based on the condition to the end of the list defined by us
        #print(round(j,3),'\n')
        #print('Accuracy Score',round(metrics.accuracy_score(y_train,custom_cutoff_data),4))
        #print('F1 Score',round(metrics.f1_score(y_train,custom_cutoff_data),4),'\n')
        #plt.figure(figsize=(6,4))
        #print('Confusion Matrix')
        #sns.heatmap(metrics.confusion_matrix(y_train,custom_cutoff_data),annot=True,fmt="d"),'\n\n'
        #plt.show()

        dict_metrics["AUC"].append(round(metrics.roc_auc_score(y_train,custom_cutoff_data),4))
        dict_metrics["Accuracy"].append(round(metrics.accuracy_score(y_train,custom_cutoff_data),4))
        dict_metrics["Recall"].append(round(metrics.recall_score(y_train,custom_cutoff_data),4))
        dict_metrics["Precision"].append(round(metrics.precision_score(y_train,custom_cutoff_data),4))
        dict_metrics["F1_Score"].append(round(metrics.f1_score(y_train,custom_cutoff_data),4))
        dict_metrics["Threshold"].append(j)
        dict_metrics["Test/Train"].append("Train")

    #print("--------------- TEST DATA-----------------")
    for j in np.arange(0.1,1,step):
        custom_prob = j #defining the cut-off value of our choice
        custom_cutoff_data=[]#defining an empty list
        for i in range(0,len(y_test)):#defining a loop for the length of the test data
            if np.array(pred_prob_test[:,1])[i] > custom_prob:#issuing a condition for our probability values to be
                #greater than the custom cutoff value
                a=1#if the probability values are greater than the custom cutoff then the value should be 1
            else:
                a=0#if the probability values are less than the custom cutoff then the value should be 0
            custom_cutoff_data.append(a)#adding either 1 or 0 based on the condition to the end of the list defined by us
        print(round(j,3),'\n')
        #print('Accuracy Score',round(metrics.accuracy_score(y_test,custom_cutoff_data),4))
        #print('F1 Score',round(metrics.f1_score(y_test,custom_cutoff_data),4),'\n')
        #plt.figure(figsize=(6,4))
        #print('Confusion Matrix')
        sns.heatmap(metrics.confusion_matrix(y_test,custom_cutoff_data),annot=True,fmt='d'),'\n\n'
        plt.show()
        dict_metrics["AUC"].append(round(metrics.roc_auc_score(y_test,custom_cutoff_data),4))
        dict_metrics["Accuracy"].append(round(metrics.accuracy_score(y_test,custom_cutoff_data),4))
        dict_metrics["Recall"].append(round(metrics.recall_score(y_test,custom_cutoff_data),4))
        dict_metrics["Precision"].append(round(metrics.precision_score(y_test,custom_cutoff_data),4))
        dict_metrics["F1_Score"].append(round(metrics.f1_score(y_test,custom_cutoff_data),4))
        dict_metrics["Threshold"].append(j)
        dict_metrics["Test/Train"].append("Test")
    temp_df=pd.DataFrame(dict_metrics)
    return temp_df
```

APPENDIX

APPENDIX 4.1 - MODELLING APPROACH

Below is the code for **tweak_threshold** generic function.

TWEAK_THRESHOLD

```
def tweak_threshold(model,X_train,X_test,y_train,y_test,step=0.1):

    dict_metrics={"AUC":[],
         "Accuracy":[],
         "Recall":[],
         "Precision":[],
         "F1_Score":[],
                 "Threshold":[],
         "Test/Train":[]}
    pred_prob_train = model.predict_proba(X_train)
    pred_prob_test = model.predict_proba(X_test)
    #print("--------------- TRAIN DATA-----------------")
    for j in np.arange(0.1,1,step):
        custom_prob = j #defining the cut-off value of our choice
        custom_cutoff_data=[]#defining an empty list
        for i in range(0,len(y_train)):#defining a loop for the length of the test data
            if np.array(pred_prob_train[:,1])[i] > custom_prob:#issuing a condition for our probability values to be
                #greater than the custom cutoff value
                a=1#if the probability values are greater than the custom cutoff then the value should be 1
            else:
                a=0#if the probability values are less than the custom cutoff then the value should be 0
            custom_cutoff_data.append(a)#adding either 1 or 0 based on the condition to the end of the list defined by us
        #print(round(j,3),'\n')
        #print('Accuracy Score',round(metrics.accuracy_score(y_train,custom_cutoff_data),4))
        #print('F1 Score',round(metrics.f1_score(y_train,custom_cutoff_data),4),'\n')
        #plt.figure(figsize=(6,4))
        #print('Confusion Matrix')
        #sns.heatmap(metrics.confusion_matrix(y_train,custom_cutoff_data),annot=True,fmt="d"),'\n\n'
        #plt.show()

        dict_metrics["AUC"].append(round(metrics.roc_auc_score(y_train,custom_cutoff_data),4))
        dict_metrics["Accuracy"].append(round(metrics.accuracy_score(y_train,custom_cutoff_data),4))
        dict_metrics["Recall"].append(round(metrics.recall_score(y_train,custom_cutoff_data),4))
        dict_metrics["Precision"].append(round(metrics.precision_score(y_train,custom_cutoff_data),4))
        dict_metrics["F1_Score"].append(round(metrics.f1_score(y_train,custom_cutoff_data),4))
        dict_metrics["Threshold"].append(j)
        dict_metrics["Test/Train"].append("Train")

    #print("--------------- TEST DATA-----------------")
    for j in np.arange(0.1,1,step):
        custom_prob = j #defining the cut-off value of our choice
        custom_cutoff_data=[]#defining an empty list
        for i in range(0,len(y_test)):#defining a loop for the length of the test data
            if np.array(pred_prob_test[:,1])[i] > custom_prob:#issuing a condition for our probability values to be
                #greater than the custom cutoff value
                a=1#if the probability values are greater than the custom cutoff then the value should be 1
            else:
                a=0#if the probability values are less than the custom cutoff then the value should be 0
            custom_cutoff_data.append(a)#adding either 1 or 0 based on the condition to the end of the list defined by us
        print(round(j,3),'\n')
        #print('Accuracy Score',round(metrics.accuracy_score(y_test,custom_cutoff_data),4))
        #print('F1 Score',round(metrics.f1_score(y_test,custom_cutoff_data),4),'\n')
        #plt.figure(figsize=(6,4))
        #print('Confusion Matrix')
        sns.heatmap(metrics.confusion_matrix(y_test,custom_cutoff_data),annot=True,fmt='d'),'\n\n'
        plt.show()
        dict_metrics["AUC"].append(round(metrics.roc_auc_score(y_test,custom_cutoff_data),4))
        dict_metrics["Accuracy"].append(round(metrics.accuracy_score(y_test,custom_cutoff_data),4))
        dict_metrics["Recall"].append(round(metrics.recall_score(y_test,custom_cutoff_data),4))
        dict_metrics["Precision"].append(round(metrics.precision_score(y_test,custom_cutoff_data),4))
        dict_metrics["F1_Score"].append(round(metrics.f1_score(y_test,custom_cutoff_data),4))
        dict_metrics["Threshold"].append(j)
        dict_metrics["Test/Train"].append("Test")
    temp_df=pd.DataFrame(dict_metrics)
    return temp_df
```

APPENDIX

APPENDIX 4.2 - KMEANS

Below is the code for KMeans model

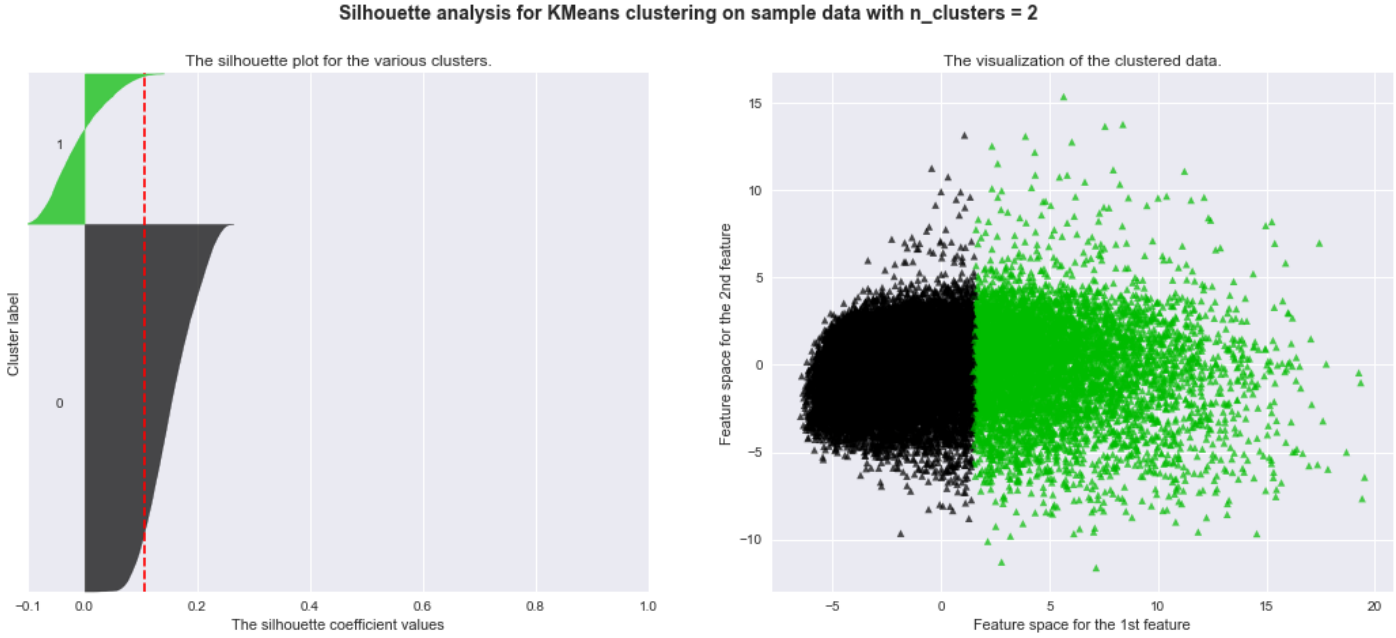
```
import matplotlib.cm as cm
range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(df_Linear_sc) + (n_clusters + 1) * 10])
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(df_Linear_sc)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(df_Linear_sc, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(df_Linear_sc, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        #Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

       # Compute the new y_lower for next plot
       y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)

    ax2.scatter(pca_2d[:,0],pca_2d[:,1],marker='^', s=30, lw=0, alpha=0.7,c=colors, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

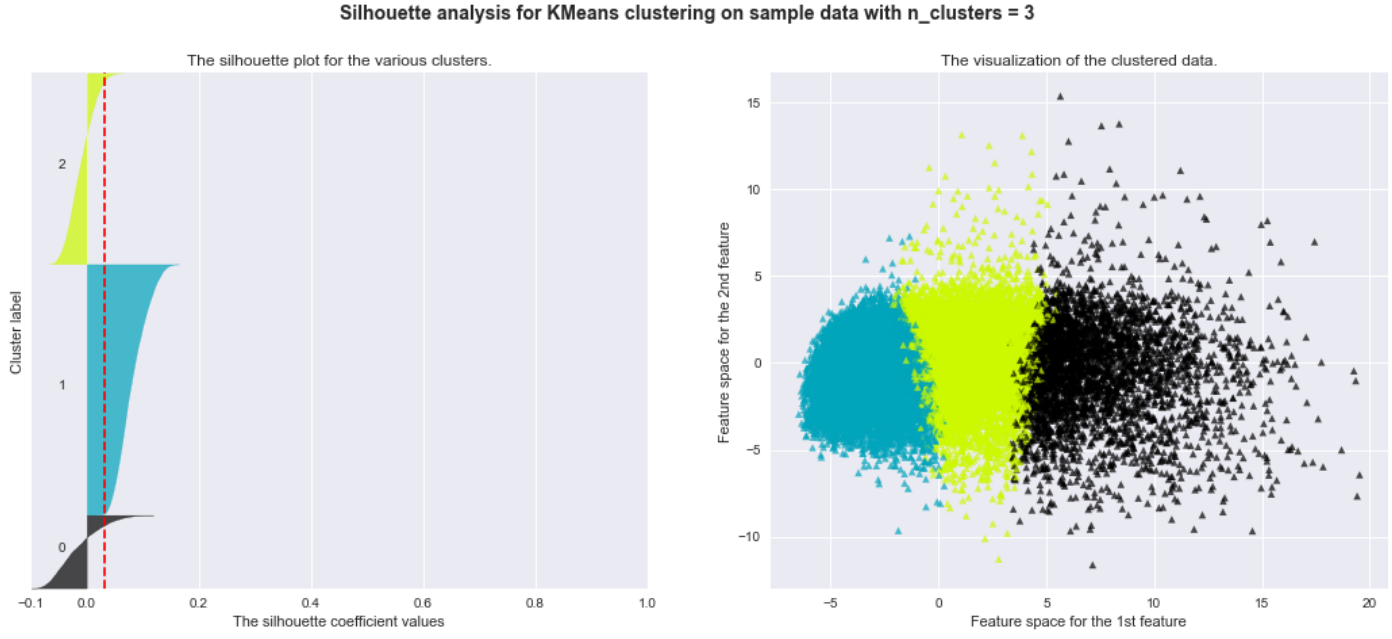
APPENDIX

APPENDIX 4.2 - KMEANS

Below is the raw output of the above code.

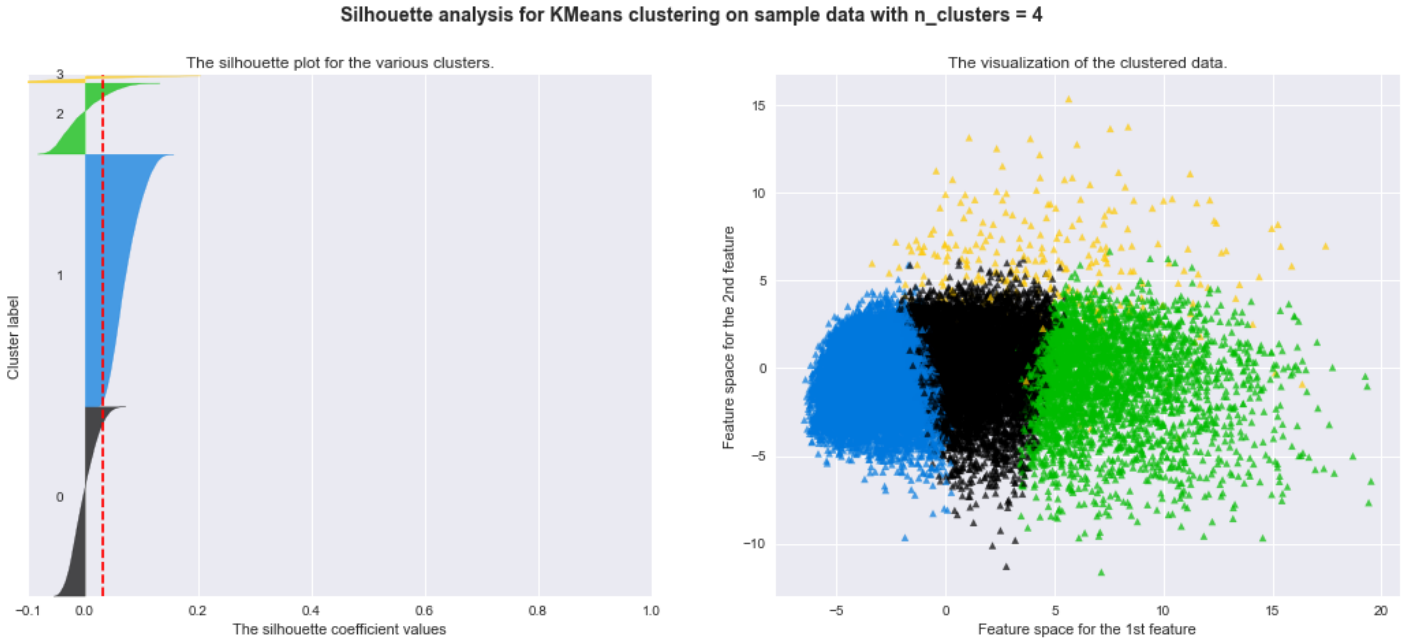
```
For n_clusters = 2 The average silhouette_score is : 0.10597881606454358
For n_clusters = 3 The average silhouette_score is : 0.0327066989584968
For n_clusters = 4 The average silhouette_score is : 0.033219032116542084
For n_clusters = 5 The average silhouette_score is : 0.020583715084196406
For n_clusters = 6 The average silhouette_score is : 0.022804775288251547
```

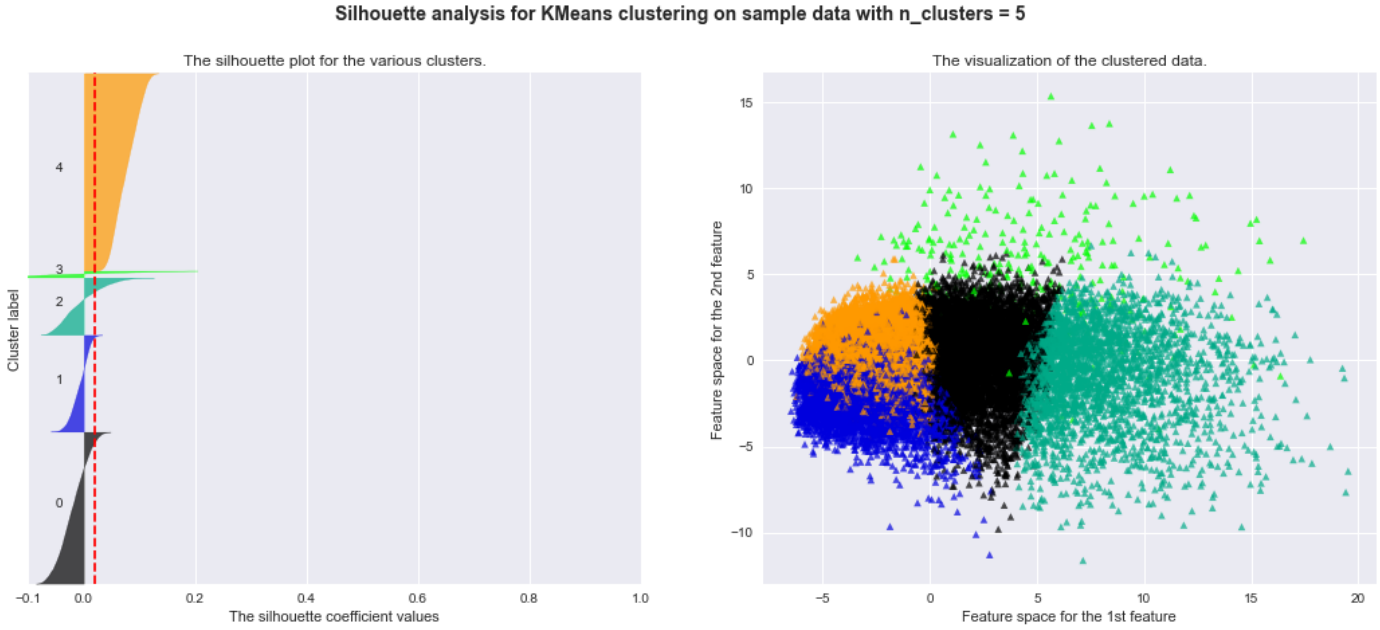




APPENDIX

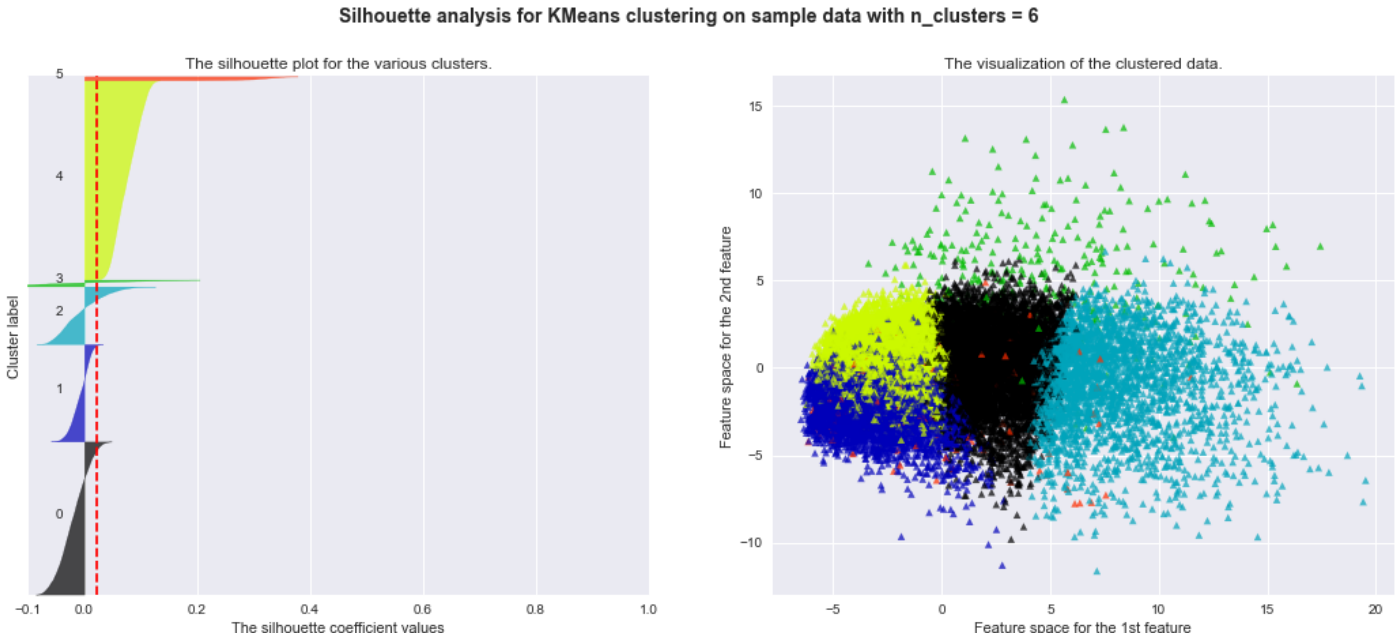
APPENDIX 4.2 - KMEANS





APPENDIX

APPENDIX 4.2 - KMEANS

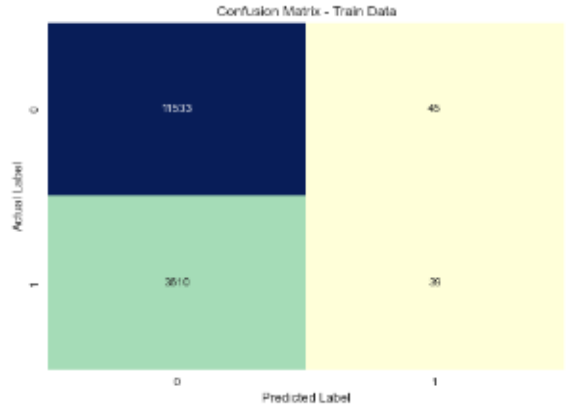


APPENDIX

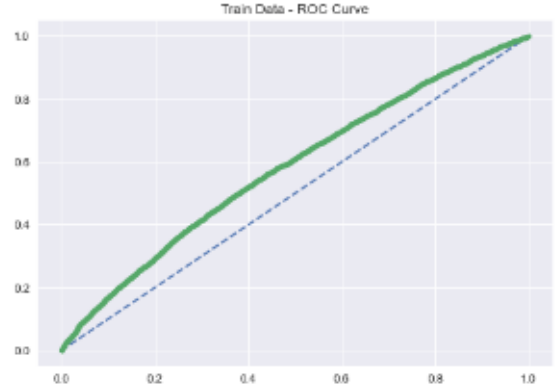
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear Unscaled

```
Logit_Linear_Unscaled
--------------------Best Parameters--------------------
{'max_iter': 100, 'random_state': 0, 'tol': 0.0001}

--------------------Best Model Params--------------------
LogisticRegression(random_state=0)

Train Accuracy Score for model LogisticRegression() is 0.7599658501346293

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86     11578
           1       0.46      0.01      0.02      3649

    accuracy                           0.76     15227
   macro avg       0.61      0.50      0.44     15227
weighted avg       0.69      0.76      0.66     15227

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




```
------------------------------------------------------------
AUC: 0.582
```



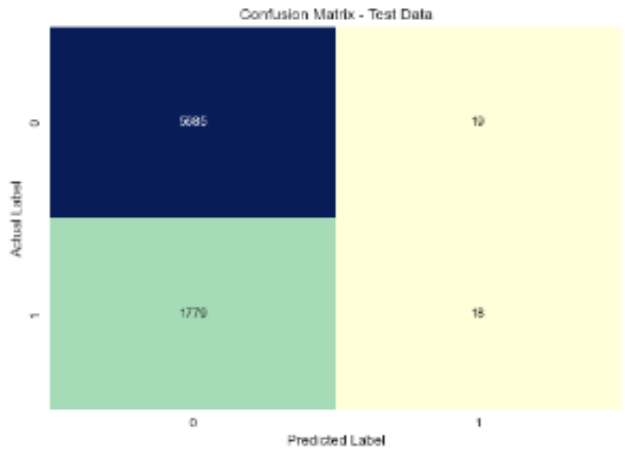

```
------------------------------------------------------------
```

APPENDIX

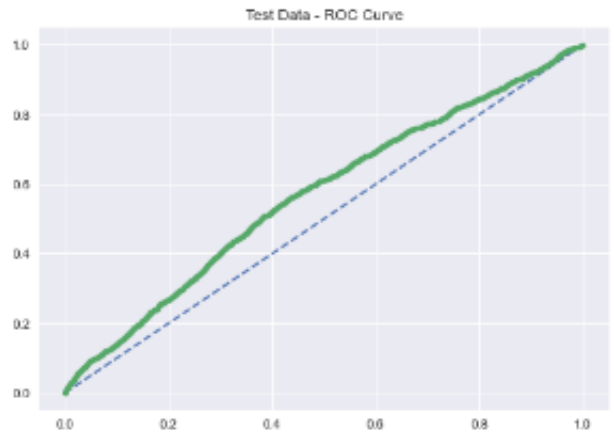
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear Unscaled

```
Test Accuracy Score for model LogisticRegression() is 0.7602986268497534

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86      5704
           1       0.49      0.01      0.02      1797

    accuracy                           0.76      7501
   macro avg       0.62      0.50      0.44      7501
weighted avg       0.70      0.76      0.66      7501

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




```
-----------------------------------------------------------------
AUC: 0.570
```



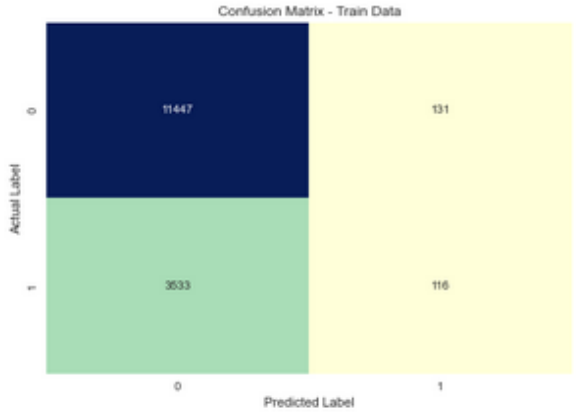

```
-----------------------------------------------------------------
```

APPENDIX

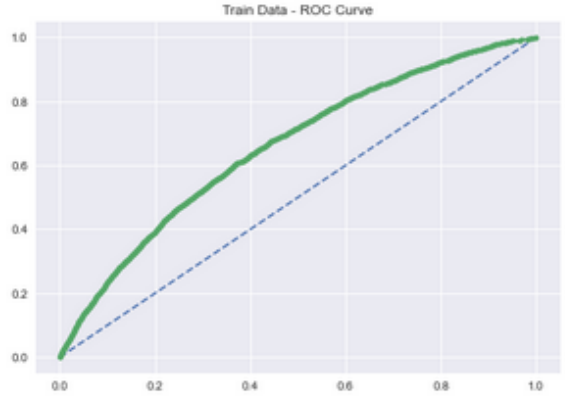
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear Scaled

```
Logit_Linear_scaled
--------------------Best Parameters--------------------
{'max_iter': 100, 'penalty': 'l1', 'random_state': 0, 'solver': 'liblinear', 'tol': 0.0001}


--------------------Best Model Params--------------------
LogisticRegression(penalty='l1', random_state=0, solver='liblinear')


Train Accuracy Score for model LogisticRegression() is 0.7593747947724437


--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      0.99      0.86     11578
           1       0.47      0.03      0.06      3649

    accuracy                           0.76     15227
   macro avg       0.62      0.51      0.46     15227
weighted avg       0.69      0.76      0.67     15227

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




```
------------------------------------------------------------
AUC: 0.659
```



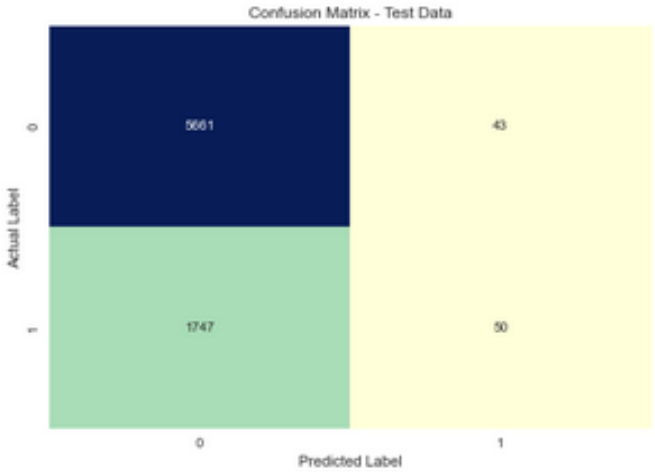

```
------------------------------------------------------------
```

APPENDIX

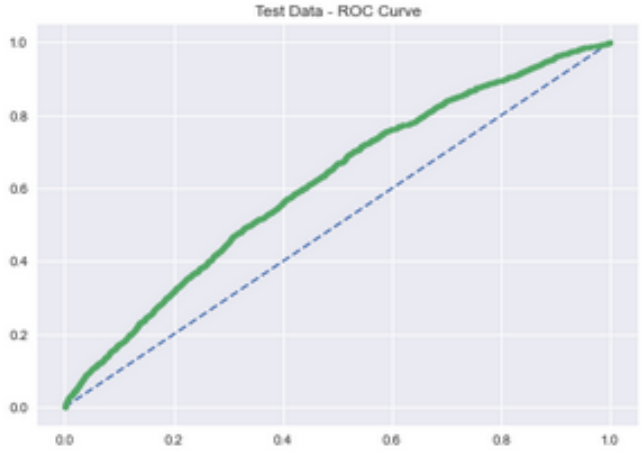
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear Scaled

```
Test Accuracy Score for model LogisticRegression() is 0.7613651513131583

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      5704
           1       0.54      0.03      0.05      1797

    accuracy                           0.76      7501
   macro avg       0.65      0.51      0.46      7501
weighted avg       0.71      0.76      0.67      7501

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




```
------------------------------------------------------------
AUC: 0.614
```

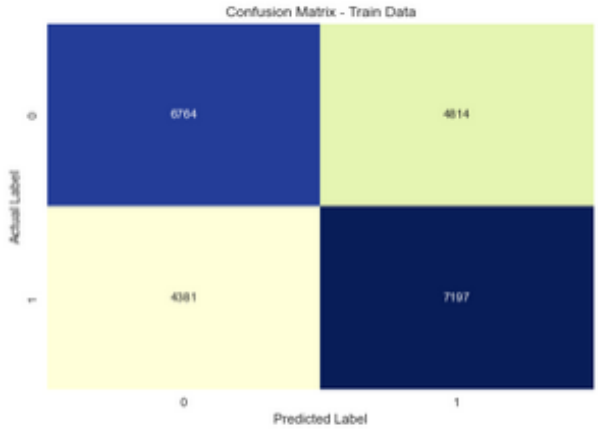


APPENDIX

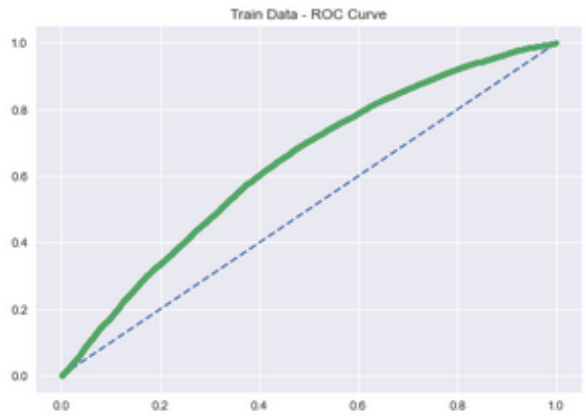
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear SMOTE Unscaled

```
Logit_Linear_SMOTE_Unscaled
--------------------Best Parameters--------------------
{'max_iter': 200, 'random_state': 0, 'tol': 0.0001}


--------------------Best Model Params--------------------
LogisticRegression(max_iter=200, random_state=0)


Train Accuracy Score for model LogisticRegression() is 0.6029106926930385


--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.61      0.58      0.60     11578
           1       0.60      0.62      0.61     11578

    accuracy                           0.60     23156
   macro avg       0.60      0.60      0.60     23156
weighted avg       0.60      0.60      0.60     23156

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




```
------------------------------------------------------------
AUC: 0.635
```

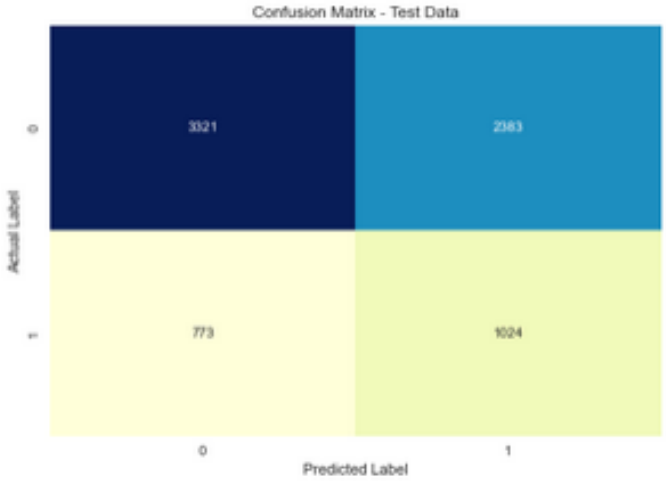


APPENDIX

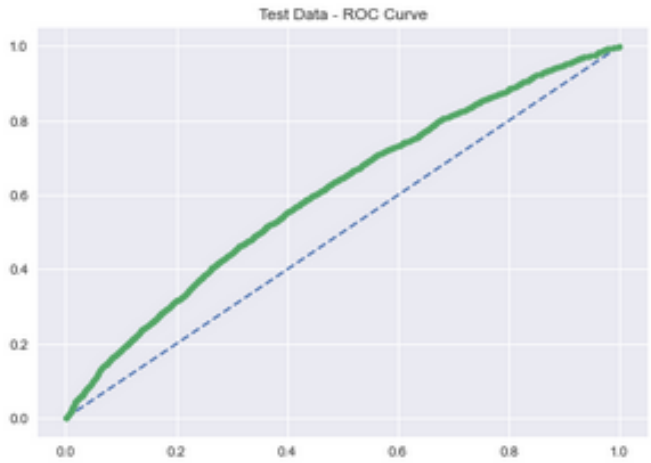
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear SMOTE Unscaled

```
Test Accuracy Score for model LogisticRegression() is 0.5792560991867751

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.81      0.58      0.68      5704
           1       0.30      0.57      0.39      1797

    accuracy                           0.58      7501
   macro avg       0.56      0.58      0.54      7501
weighted avg       0.69      0.58      0.61      7501

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




```
------------------------------------------------------------
AUC: 0.605
```

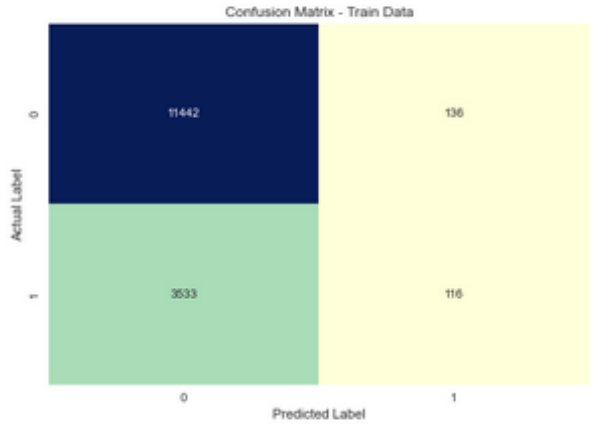


APPENDIX

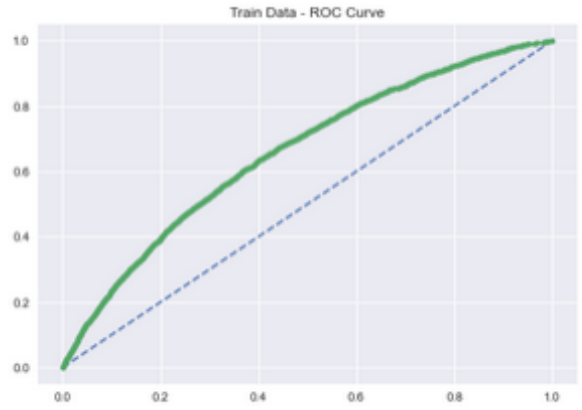
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear SMOTE Scaled

```
Logit_Linear_SMOTE_Scaled
--------------------Best Parameters--------------------
{'max_iter': 100, 'random_state': 0, 'tol': 0.0001}


--------------------Best Model Params--------------------
LogisticRegression(random_state=0)


Train Accuracy Score for model LogisticRegression() is 0.7590464306823406


--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      0.99      0.86     11578
           1       0.46      0.03      0.06      3649

    accuracy                           0.76     15227
   macro avg       0.61      0.51      0.46     15227
weighted avg       0.69      0.76      0.67     15227

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




```
------------------------------------------------------------
AUC: 0.659
```

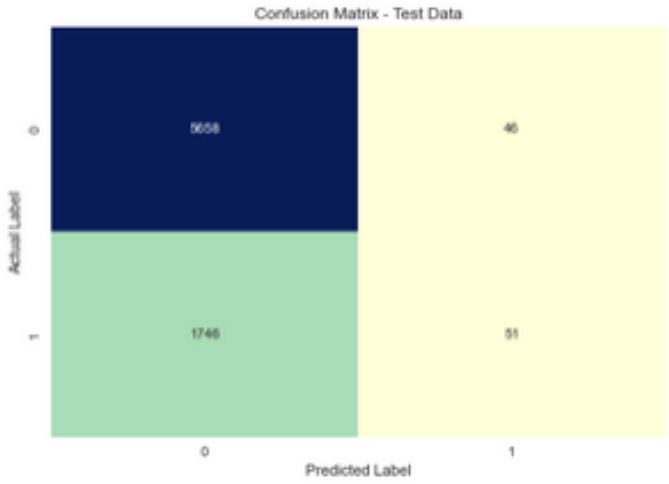


APPENDIX

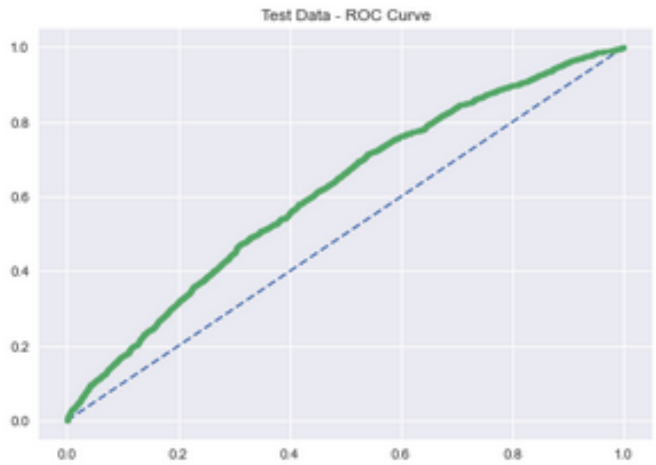
APPENDIX 5.1 - PERFORMANCE METRICS

Logit - Linear SMOTE Scaled

```
Test Accuracy Score for model LogisticRegression() is 0.7610985201973071

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      5704
           1       0.53      0.03      0.05      1797

    accuracy                           0.76      7501
   macro avg       0.64      0.51      0.46      7501
weighted avg       0.71      0.76      0.67      7501

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




```
-------------------------------------------------------------
AUC: 0.614
```

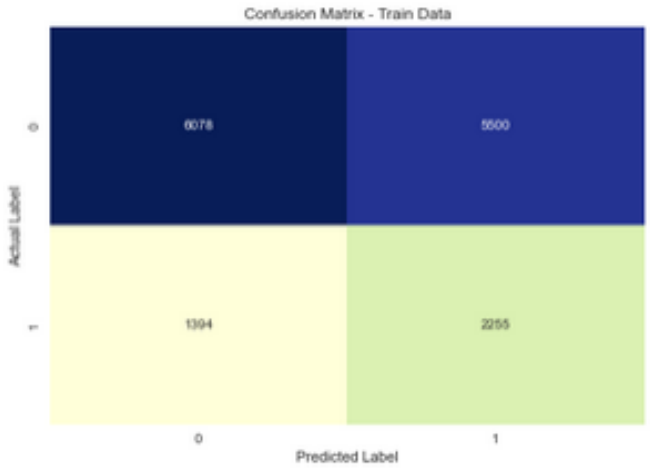


APPENDIX

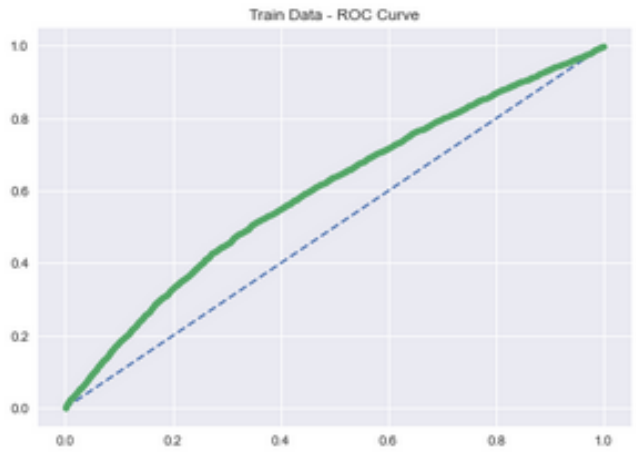
APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear Unscaled

```
NB_Linear_Unscaled
Train Accuracy Score for model GaussianNB() is 0.547251592565837

--------------------Classification Report - Train Data---------------------
              precision    recall  f1-score   support

           0       0.81      0.52      0.64     11578
           1       0.29      0.62      0.40      3649

    accuracy                           0.55     15227
   macro avg       0.55      0.57      0.52     15227
weighted avg       0.69      0.55      0.58     15227

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




```
------------------------------------------------------------
AUC: 0.599
```

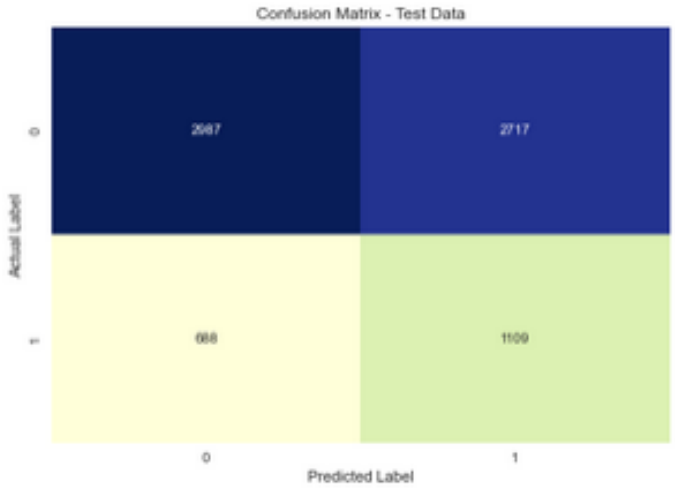


APPENDIX

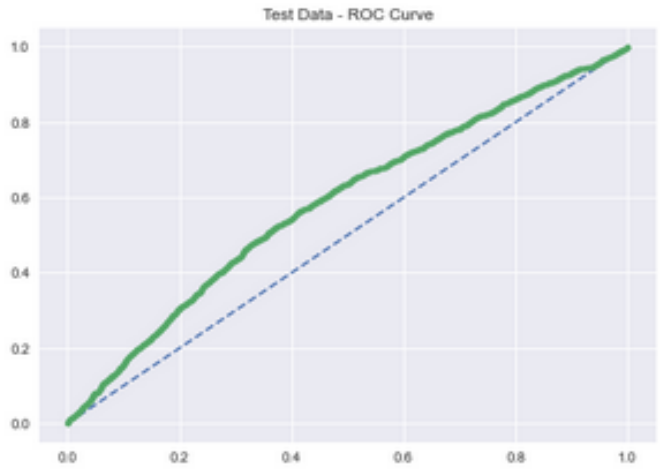
APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear Unscaled

```
Test Accuracy Score for model GaussianNB() is 0.5460605252632982

--------------------Classification Report - Test Data------------
              precision    recall  f1-score   support

           0       0.81      0.52      0.64      5704
           1       0.29      0.62      0.39      1797

    accuracy                           0.55      7501
   macro avg       0.55      0.57      0.52      7501
weighted avg       0.69      0.55      0.58      7501

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




```
------------------------------------------------------------
AUC: 0.587
```

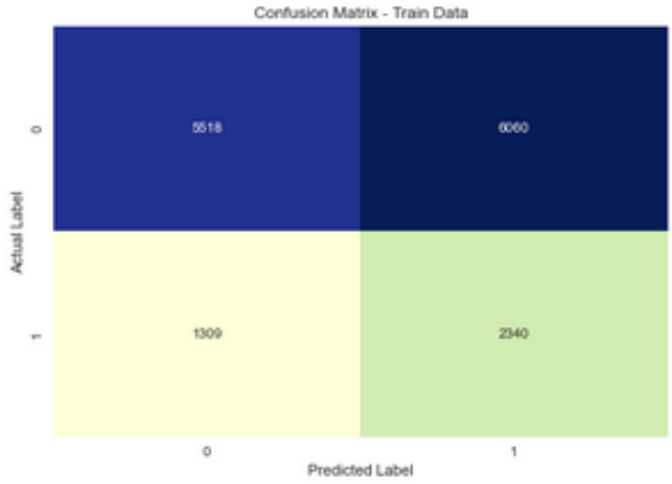


APPENDIX

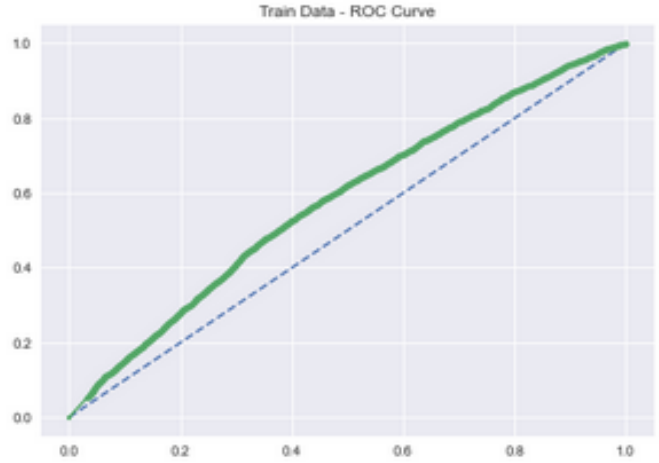
APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear Scaled

```
NB_Linear_Scaled
Train Accuracy Score for model GaussianNB() is 0.5160570040060419

--------------------Classification Report - Train Data-----------
              precision    recall  f1-score   support

           0       0.81      0.48      0.60     11578
           1       0.28      0.64      0.39      3649

    accuracy                           0.52     15227
   macro avg       0.54      0.56      0.49     15227
weighted avg       0.68      0.52      0.55     15227

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




```
------------------------------------------------------------
AUC: 0.580
```



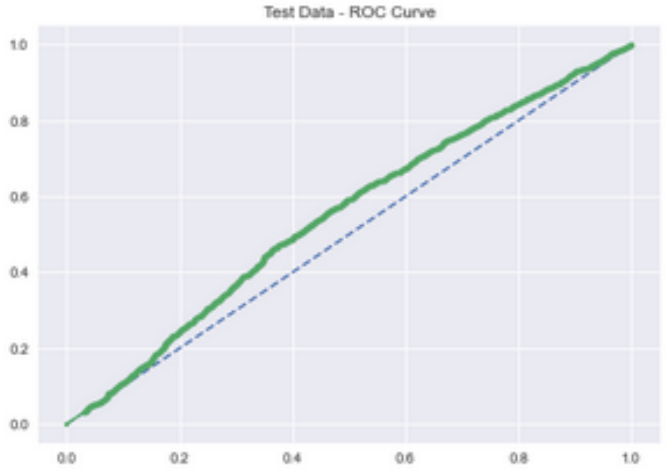
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear Scaled

```
Test Accuracy Score for model GaussianNB() is 0.5112651646447714

--------------------Classification Report - Test Data-----------
              precision    recall  f1-score   support

           0       0.80      0.48      0.60      5704
           1       0.27      0.61      0.38      1797

    accuracy                           0.51      7501
   macro avg       0.53      0.55      0.49      7501
weighted avg       0.67      0.51      0.55      7501

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


```
------------------------------------------------------------
AUC: 0.551
```

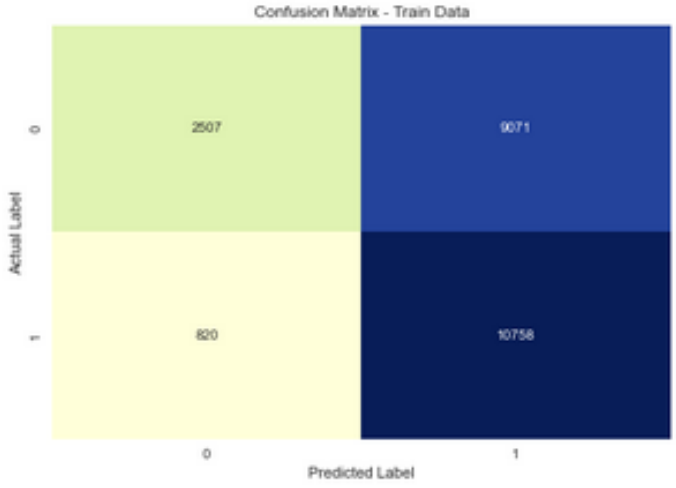


APPENDIX

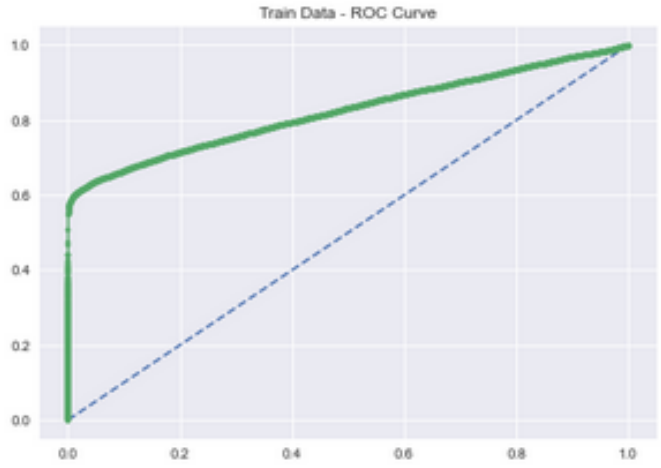
APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear SMOTE Unscaled

```
NB_Linear_SMOTE_Unscaled
Train Accuracy Score for model GaussianNB() is 0.5728536880290206

--------------------Classification Report - Train Data------------
              precision    recall  f1-score   support

           0       0.75      0.22      0.34     11578
           1       0.54      0.93      0.69     11578

    accuracy                           0.57     23156
   macro avg       0.65      0.57      0.51     23156
weighted avg       0.65      0.57      0.51     23156

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




```
------------------------------------------------------------
AUC: 0.823
```



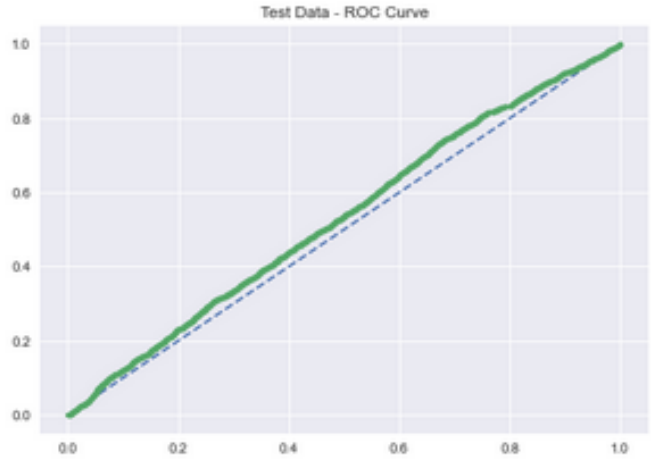
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear SMOTE Unscaled

```
Test Accuracy Score for model GaussianNB() is 0.3640847886948407

--------------------Classification Report - Test Data------------
              precision    recall  f1-score   support

           0       0.80      0.22      0.34      5704
           1       0.25      0.82      0.38      1797

    accuracy                           0.36      7501
   macro avg       0.52      0.52      0.36      7501
weighted avg       0.67      0.36      0.35      7501

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


```
-------------------------------------------------------------
AUC: 0.532
```



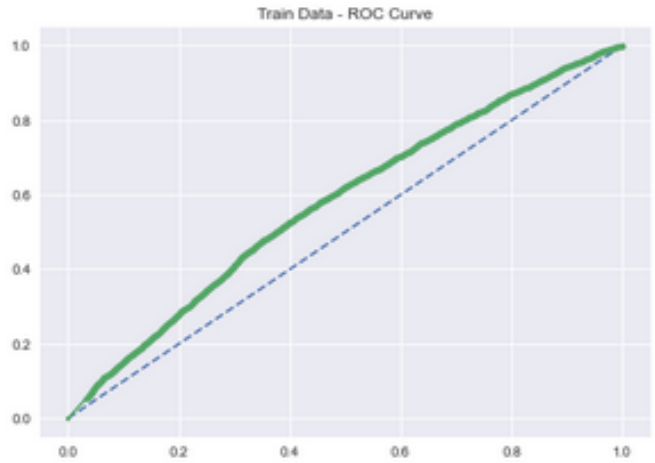
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear SMOTE Unscaled

```
NB_Linear_SMOTE_Scaled
Train Accuracy Score for model GaussianNB() is 0.5160570040060419

--------------------Classification Report - Train Data-----------
              precision    recall  f1-score   support

           0       0.81      0.48      0.60     11578
           1       0.28      0.64      0.39      3649

    accuracy                           0.52     15227
   macro avg       0.54      0.56      0.49     15227
weighted avg       0.68      0.52      0.55     15227

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


```
------------------------------------------------------------
AUC: 0.580
```



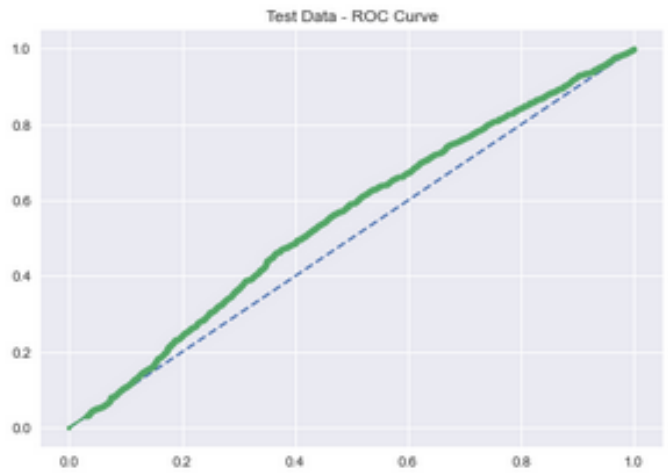
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

Gaussian Navie Bayes - Linear SMOTE Unscaled

```
Test Accuracy Score for model GaussianNB() is 0.5112651646447714

--------------------Classification Report - Test Data-----------
              precision    recall  f1-score   support

           0       0.80      0.48      0.60      5704
           1       0.27      0.61      0.38      1797

    accuracy                           0.51      7501
   macro avg       0.53      0.55      0.49      7501
weighted avg       0.67      0.51      0.55      7501

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


```
------------------------------------------------------------
AUC: 0.551
```

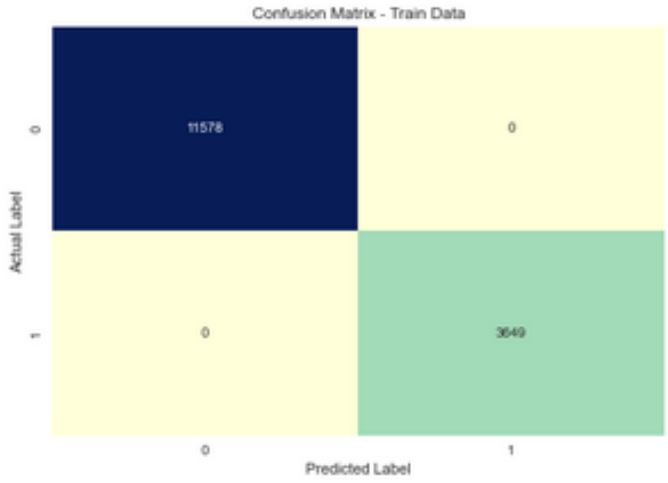


APPENDIX

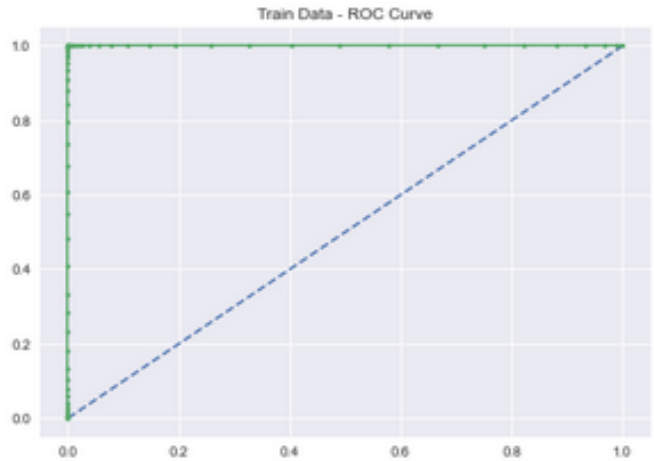
APPENDIX 5.1 - PERFORMANCE METRICS

Random Forest - Tree - Unscaled

```
RF_Tree_Unscaled
Train Accuracy Score for model RandomForestClassifier() is 1.0


--------------------Classification Report - Train Data---------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11578
           1       1.00      1.00      1.00      3649

    accuracy                           1.00     15227
   macro avg       1.00      1.00      1.00     15227
weighted avg       1.00      1.00      1.00     15227

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




```
-----------------------------------------------------------
AUC: 1.000
```

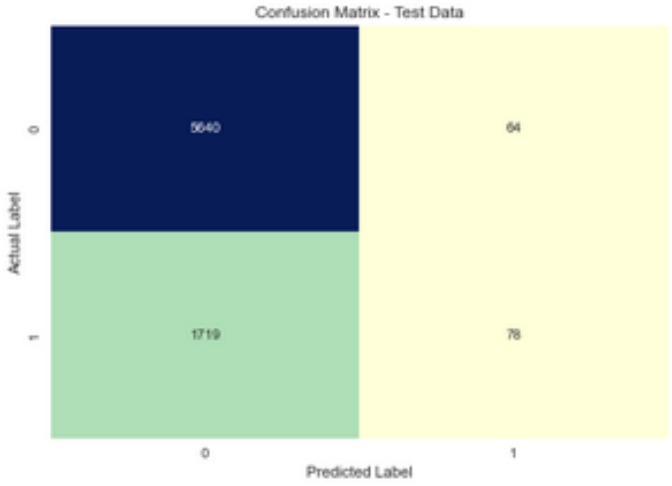


APPENDIX

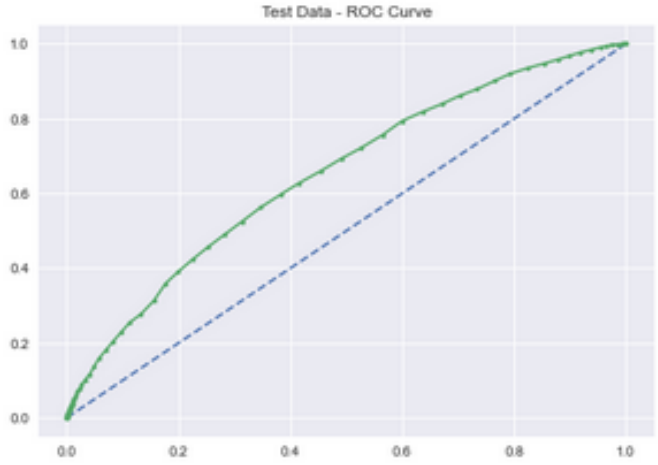
APPENDIX 5.1 - PERFORMANCE METRICS

Random Forest - Tree - Unscaled

```
Test Accuracy Score for model RandomForestClassifier() is 0.7622983602186375

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.77      0.99      0.86      5704
           1       0.55      0.04      0.08      1797

    accuracy                           0.76      7501
   macro avg       0.66      0.52      0.47      7501
weighted avg       0.71      0.76      0.68      7501

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




```
-----------------------------------------------------------------
AUC: 0.651
```

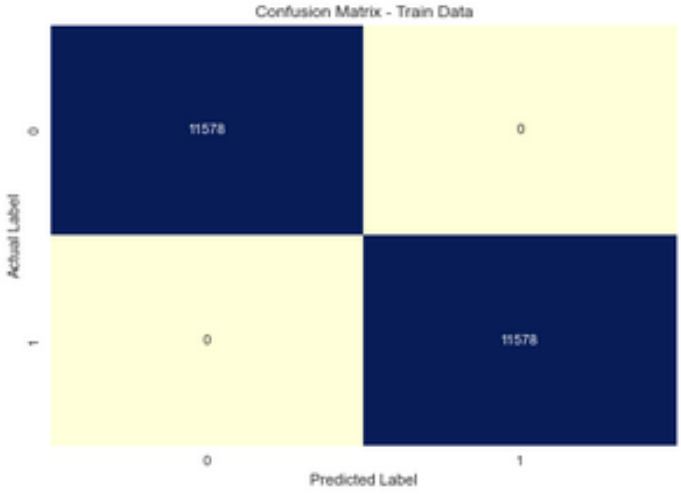


APPENDIX

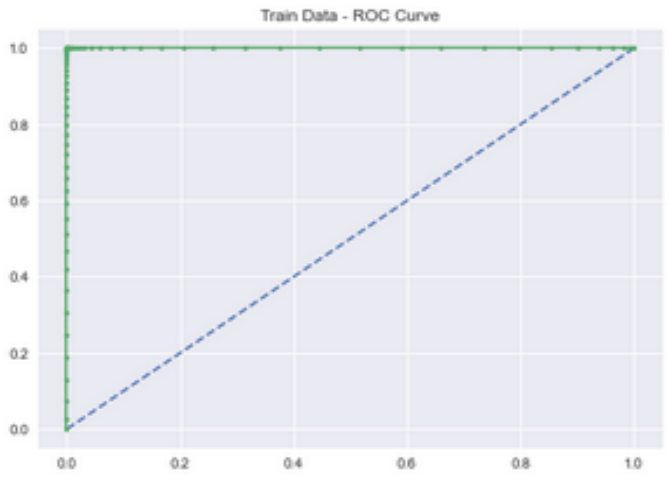
APPENDIX 5.1 - PERFORMANCE METRICS

Random Forest - Tree - SMOTE - Unscaled

```
RF_Tree_SMOTE_Unscaled
Train Accuracy Score for model RandomForestClassifier() is 1.0

--------------------Classification Report - Train Data--------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11578
           1       1.00      1.00      1.00     11578

    accuracy                           1.00     23156
   macro avg       1.00      1.00      1.00     23156
weighted avg       1.00      1.00      1.00     23156

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




```
----------------------------------------------------------
AUC: 1.000
```

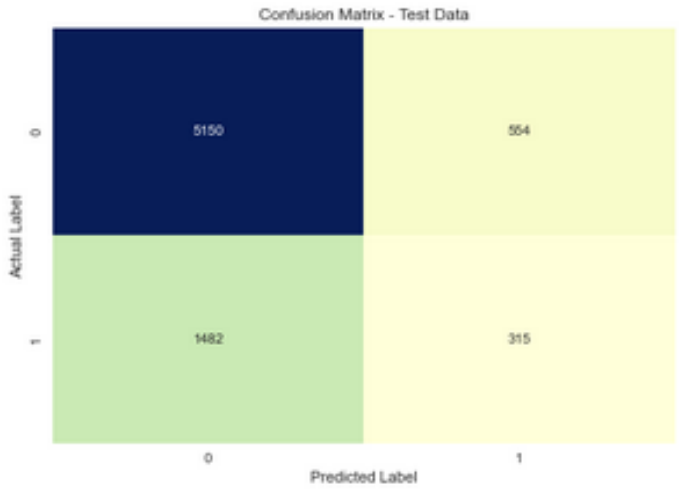


APPENDIX

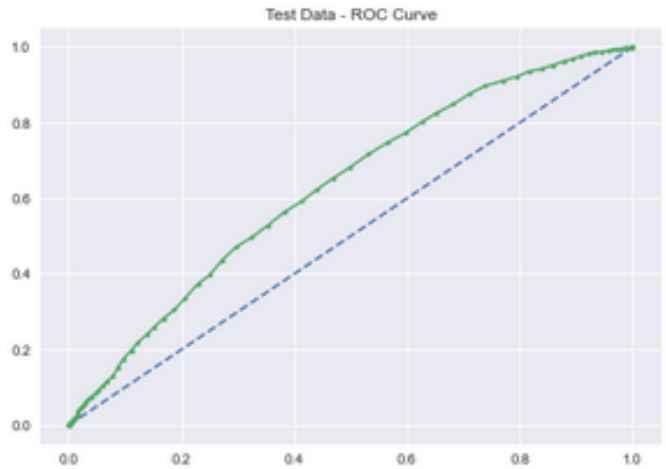
APPENDIX 5.1 - PERFORMANCE METRICS

Random Forest - Tree - SMOTE - Unscaled

```
Test Accuracy Score for model RandomForestClassifier() is 0.7285695240634582

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.78      0.90      0.83      5704
           1       0.36      0.18      0.24      1797

    accuracy                           0.73      7501
   macro avg       0.57      0.54      0.54      7501
weighted avg       0.68      0.73      0.69      7501

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




```
-------------------------------------------------------------
AUC: 0.629
```

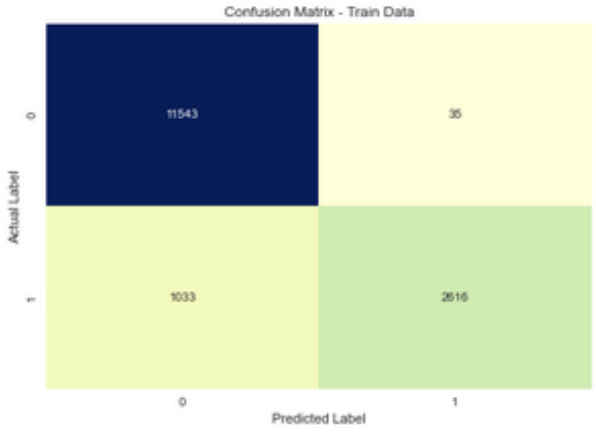


APPENDIX

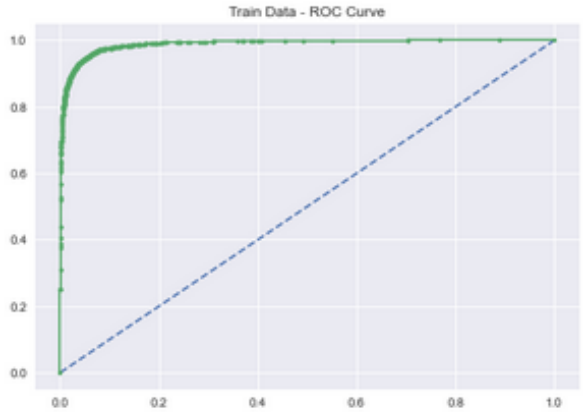
APPENDIX 5.1 - PERFORMANCE METRICS

XGBoost - Tree - Unscaled

```
XGB_Tree_Unscaled
Train Accuracy Score for model XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None) is 0.9298614303539765


--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     11578
           1       0.99      0.72      0.83      3649

    accuracy                           0.93     15227
   macro avg       0.95      0.86      0.89     15227
weighted avg       0.93      0.93      0.93     15227

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




```
------------------------------------------------------------
AUC: 0.989
```

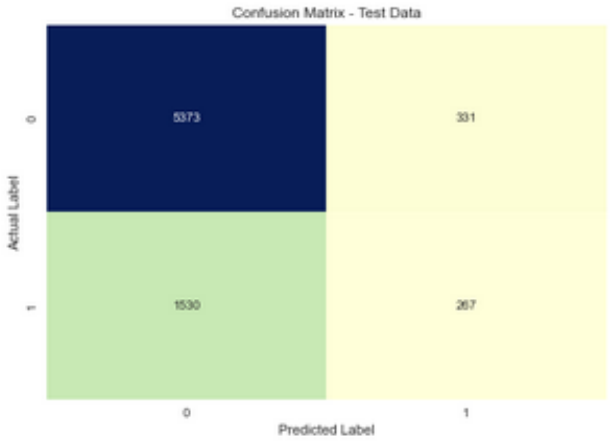


APPENDIX

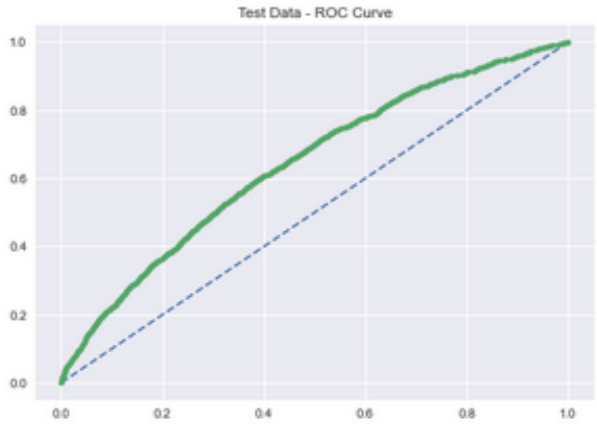
APPENDIX 5.1 - PERFORMANCE METRICS

XGBoost - Tree - Unscaled

```
Test Accuracy Score for model XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None) is 0.7518997467004399


--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.78      0.94      0.85      5704
           1       0.45      0.15      0.22      1797

    accuracy                           0.75      7501
   macro avg       0.61      0.55      0.54      7501
weighted avg       0.70      0.75      0.70      7501

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




```
------------------------------------------------------------
AUC: 0.642
```

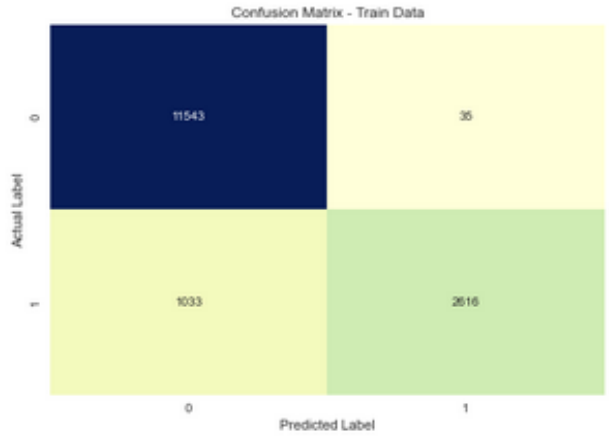


APPENDIX

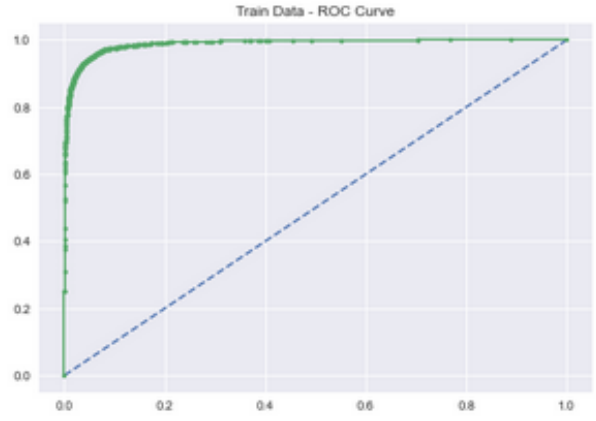
APPENDIX 5.1 - PERFORMANCE METRICS

XGBoost - Tree - Scaled

```
XGB_Tree_Scaled
Train Accuracy Score for model XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None) is 0.9298614303539765


--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     11578
           1       0.99      0.72      0.83      3649

    accuracy                           0.93     15227
   macro avg       0.95      0.86      0.89     15227
weighted avg       0.93      0.93      0.93     15227

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




```
----------------------------------------------------------------
AUC: 0.989
```

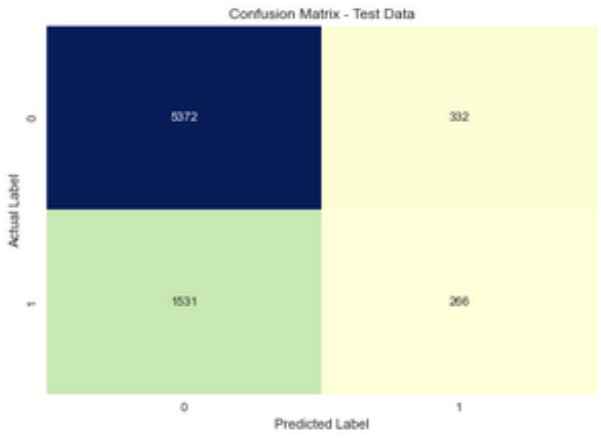


APPENDIX

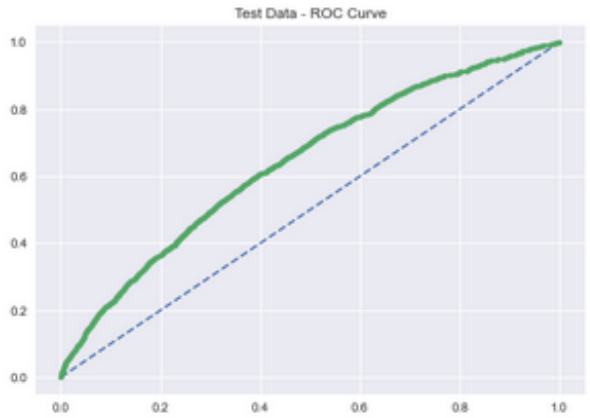
APPENDIX 5.1 - PERFORMANCE METRICS

XGBoost - Tree - Scaled

```
Test Accuracy Score for model XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None) is 0.7516331155845887

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.78      0.94      0.85      5704
           1       0.44      0.15      0.22      1797

    accuracy                           0.75      7501
   macro avg       0.61      0.54      0.54      7501
weighted avg       0.70      0.75      0.70      7501

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




```
------------------------------------------------------------
AUC: 0.642
```

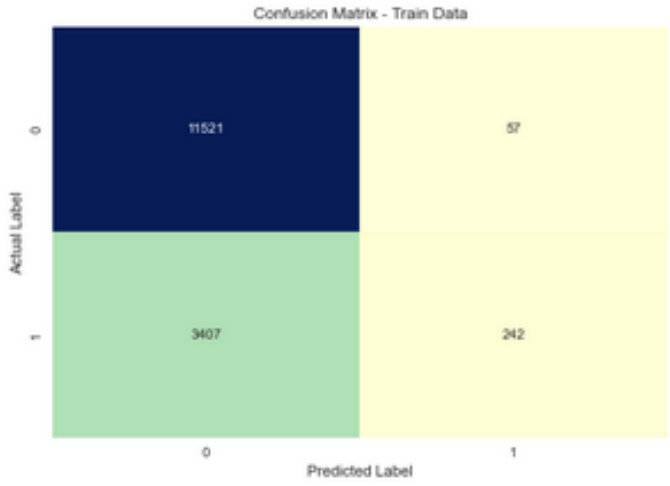


APPENDIX

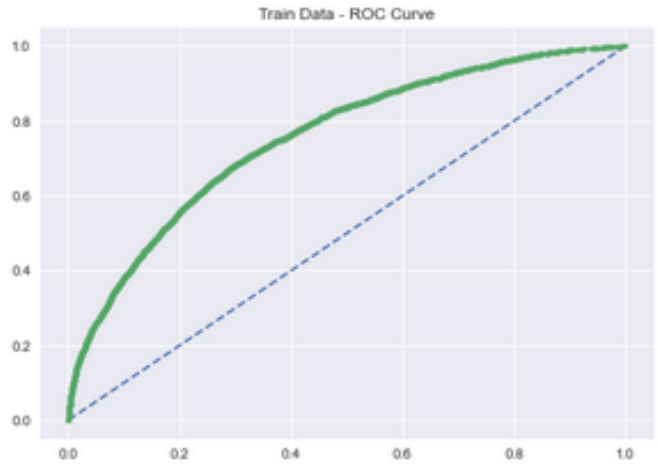
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - Unscaled

```
GBM_Tree_Unscaled
Train Accuracy Score for model GradientBoostingClassifier() is 0.772509358376568

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.77      1.00      0.87     11578
           1       0.81      0.07      0.12      3649

    accuracy                           0.77     15227
   macro avg       0.79      0.53      0.50     15227
weighted avg       0.78      0.77      0.69     15227

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




```
------------------------------------------------------------
AUC: 0.756
```

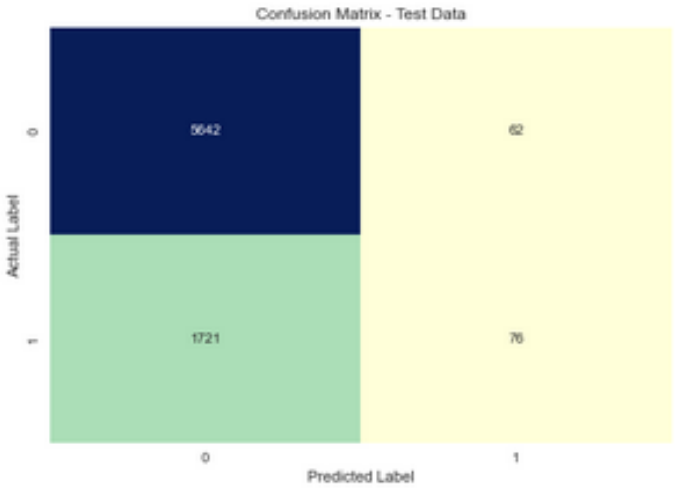


APPENDIX

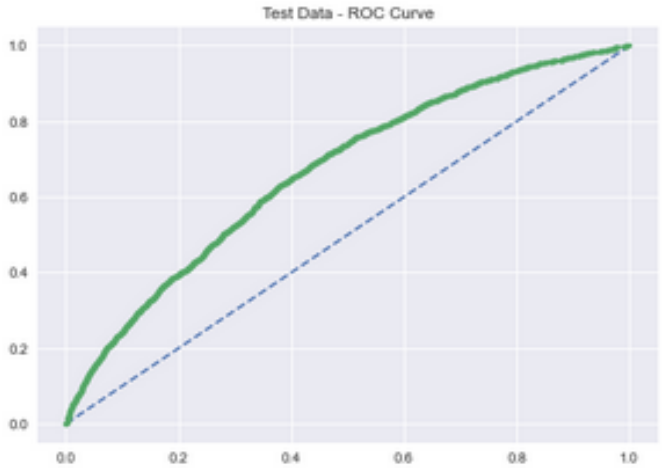
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - Unscaled

```
Test Accuracy Score for model GradientBoostingClassifier() is 0.7622983602186375

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.77      0.99      0.86      5704
           1       0.55      0.04      0.08      1797

    accuracy                           0.76      7501
   macro avg       0.66      0.52      0.47      7501
weighted avg       0.71      0.76      0.68      7501

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




```
------------------------------------------------------------
AUC: 0.668
```

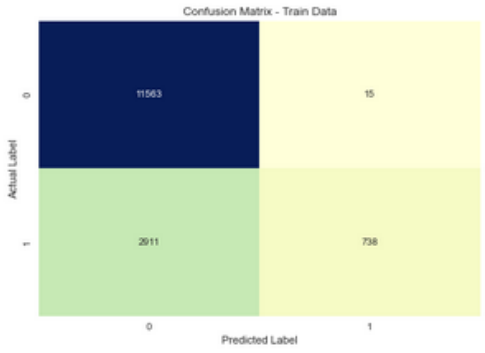


APPENDIX

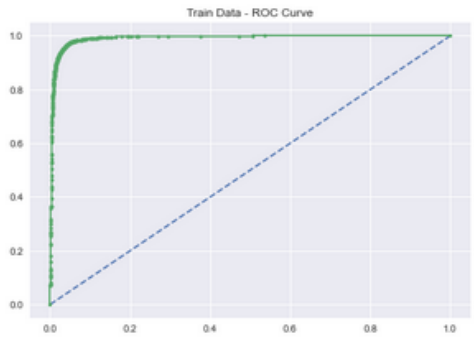
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - Scaled

```
GBM_Tree_Scaled
--------------------Best Parameters--------------------
{'criterion': 'mae', 'learning_rate': 0.01, 'loss': 'deviance', 'max_depth': 50, 'max_features': 20, 'min_samples_leaf': 10, 'min_samples_split': 100, 'n_estimators': 150}


--------------------Best Model Params--------------------
GradientBoostingClassifier(criterion='mae', learning_rate=0.01, max_depth=50,
                           max_features=20, min_samples_leaf=10,
                           min_samples_split=100, n_estimators=150)

Train Accuracy Score for model GradientBoostingClassifier() is 0.8078413344716622

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.80      1.00      0.89     11578
           1       0.98      0.20      0.34      3649

    accuracy                           0.81     15227
   macro avg       0.89      0.60      0.61     15227
weighted avg       0.84      0.81      0.76     15227

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




```
------------------------------------------------------------
AUC: 0.992
```

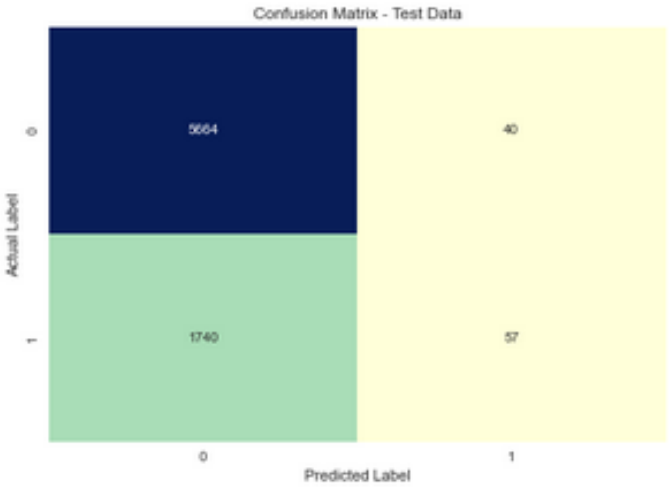


APPENDIX

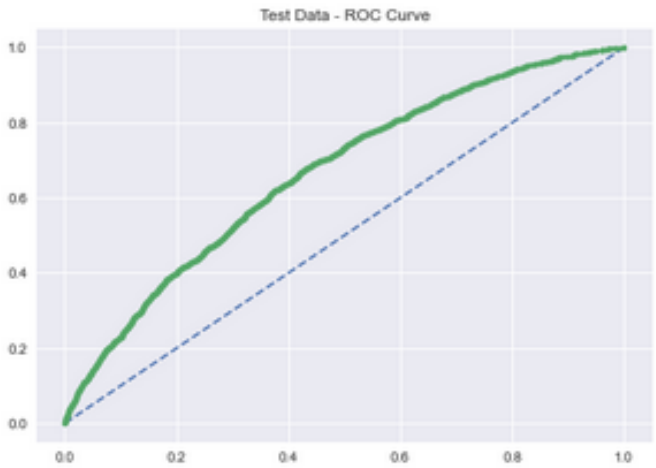
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - Scaled

```
Test Accuracy Score for model GradientBoostingClassifier() is 0.7626983068924144

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      5704
           1       0.59      0.03      0.06      1797

    accuracy                           0.76      7501
   macro avg       0.68      0.51      0.46      7501
weighted avg       0.72      0.76      0.67      7501

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




```
------------------------------------------------------------
AUC: 0.666
```



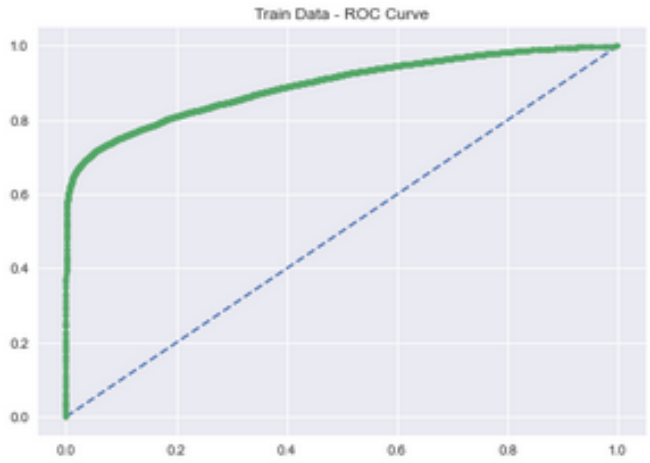
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - SMOTE - Unscaled

```
GBM_Tree_SMOTE_Unscaled
Train Accuracy Score for model GradientBoostingClassifier() is 0.8278200034548281

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.78      0.91      0.84     11578
           1       0.89      0.74      0.81     11578

    accuracy                           0.83     23156
   macro avg       0.84      0.83      0.83     23156
weighted avg       0.84      0.83      0.83     23156

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


```
------------------------------------------------------------
AUC: 0.894
```

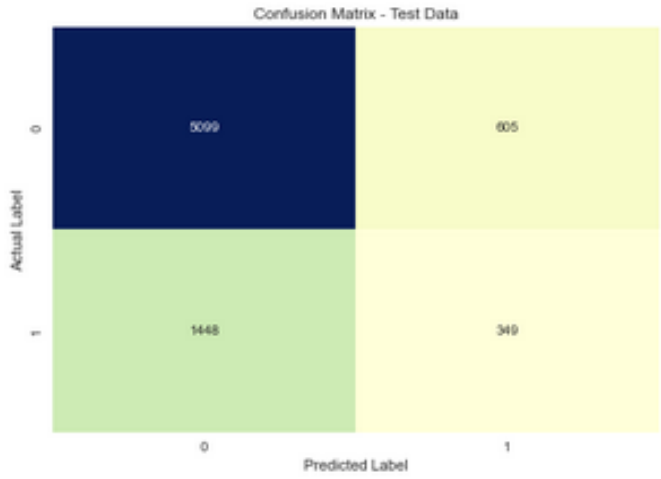


APPENDIX

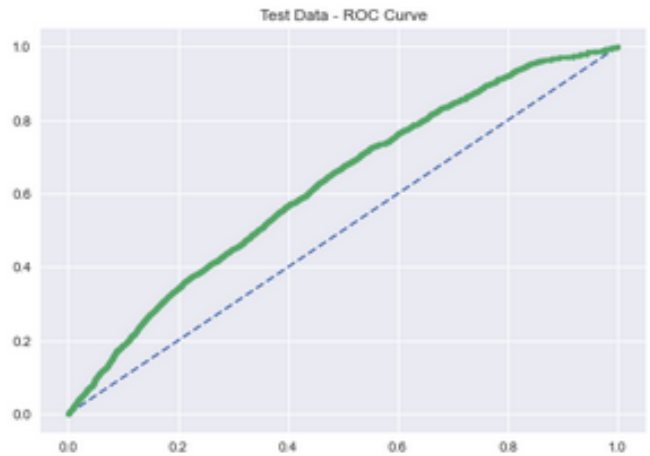
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - SMOTE - Unscaled

```
Test Accuracy Score for model GradientBoostingClassifier() is 0.7263031595787228

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.78      0.89      0.83      5704
           1       0.37      0.19      0.25      1797

    accuracy                           0.73      7501
   macro avg       0.57      0.54      0.54      7501
weighted avg       0.68      0.73      0.69      7501

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




```
------------------------------------------------------------
AUC: 0.623
```



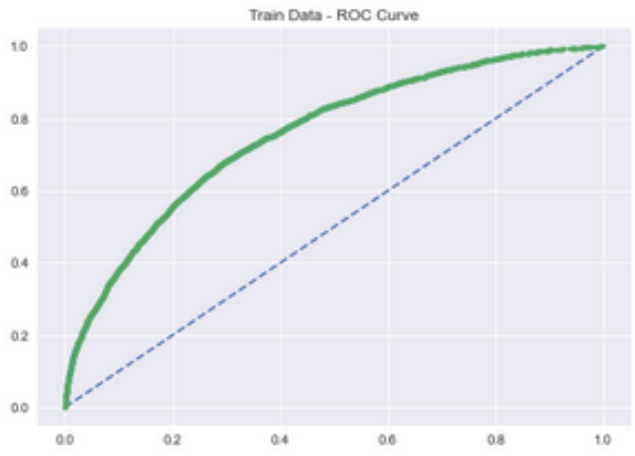
APPENDIX

APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - SMOTE - Scaled

```
GBM_Tree_SMOTE_Scaled
Train Accuracy Score for model GradientBoostingClassifier() is 0.772509358376568

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.77      1.00      0.87     11578
           1       0.81      0.07      0.12      3649

    accuracy                           0.77     15227
   macro avg       0.79      0.53      0.50     15227
weighted avg       0.78      0.77      0.69     15227

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


```
------------------------------------------------------------
AUC: 0.756
```

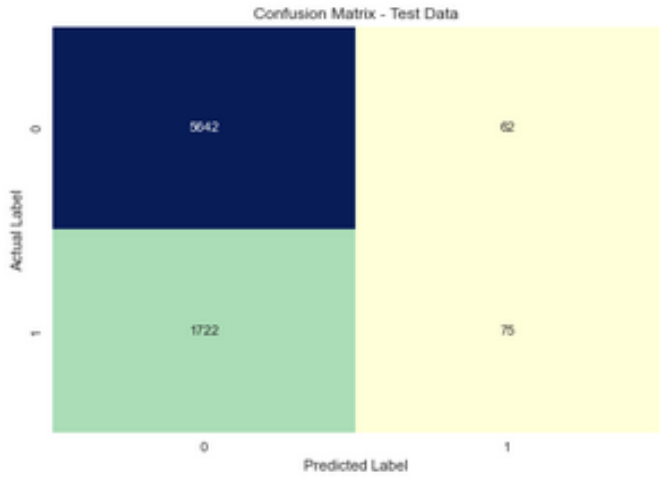


APPENDIX

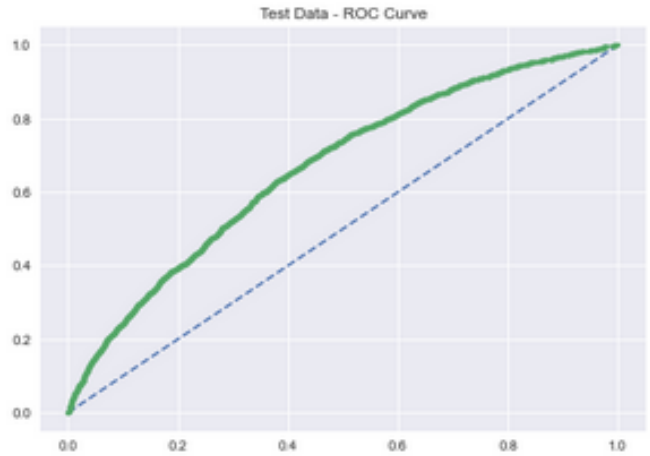
APPENDIX 5.1 - PERFORMANCE METRICS

GBM - Tree - SMOTE - Scaled

```
Test Accuracy Score for model GradientBoostingClassifier() is 0.7621650446607119

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.77      0.99      0.86      5704
           1       0.55      0.04      0.08      1797

    accuracy                           0.76      7501
   macro avg       0.66      0.52      0.47      7501
weighted avg       0.71      0.76      0.68      7501

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




```
------------------------------------------------------------
AUC: 0.668
```

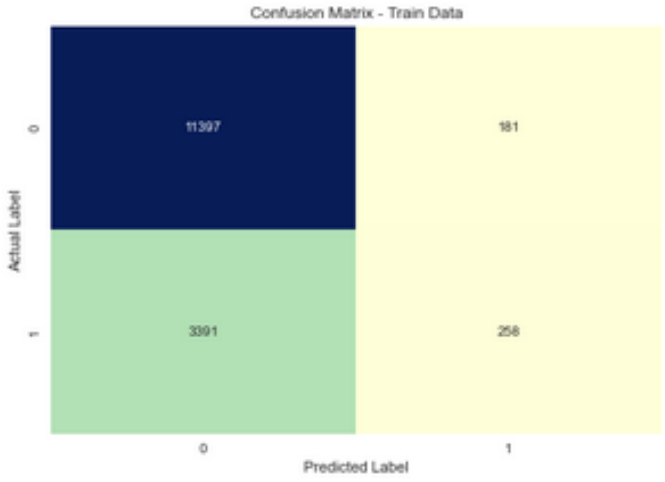


APPENDIX

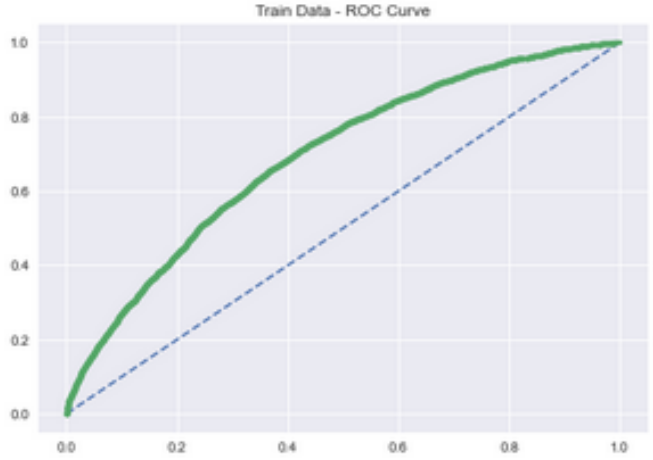
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - TREE - Unscaled

```
ADA_Tree_Unscaled
Train Accuracy Score for model AdaBoostClassifier() is 0.7654166940303408

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.77      0.98      0.86     11578
           1       0.59      0.07      0.13      3649

    accuracy                           0.77     15227
   macro avg       0.68      0.53      0.50     15227
weighted avg       0.73      0.77      0.69     15227

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




```
----------------------------------------------------------------
AUC: 0.695
```

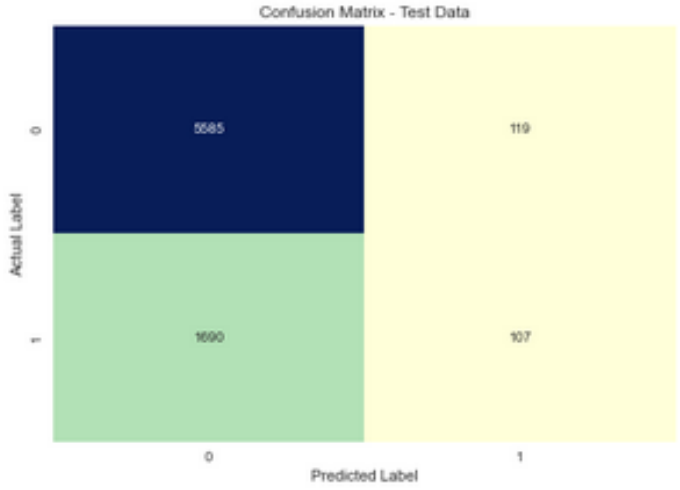


APPENDIX

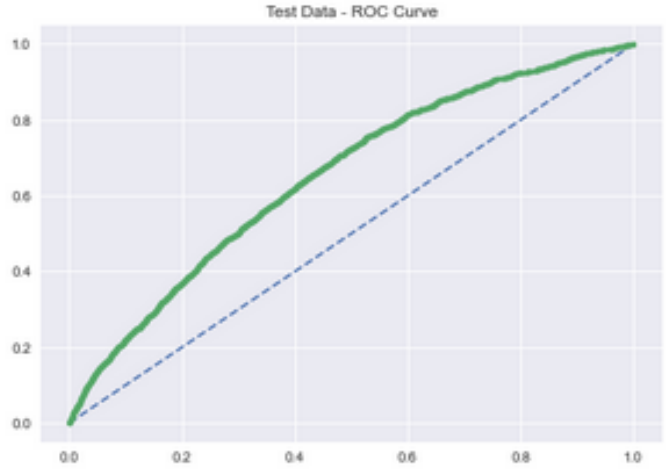
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - TREE - Unscaled

```
Test Accuracy Score for model AdaBoostClassifier() is 0.7588321557125717

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.77      0.98      0.86      5704
           1       0.47      0.06      0.11      1797

    accuracy                           0.76      7501
   macro avg       0.62      0.52      0.48      7501
weighted avg       0.70      0.76      0.68      7501

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




```
-------------------------------------------------------------
AUC: 0.654
```

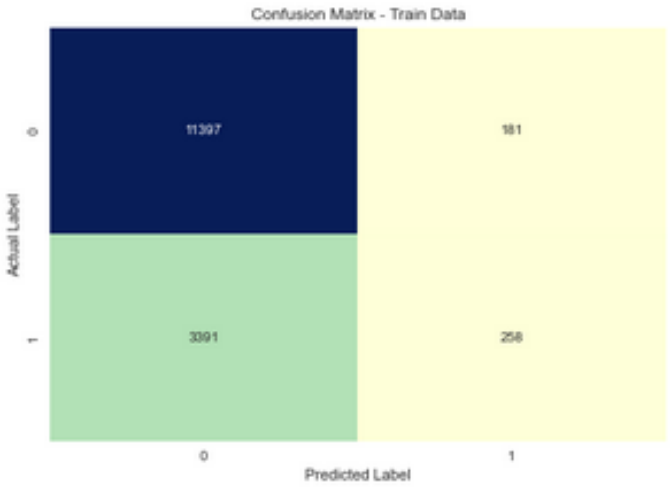


APPENDIX

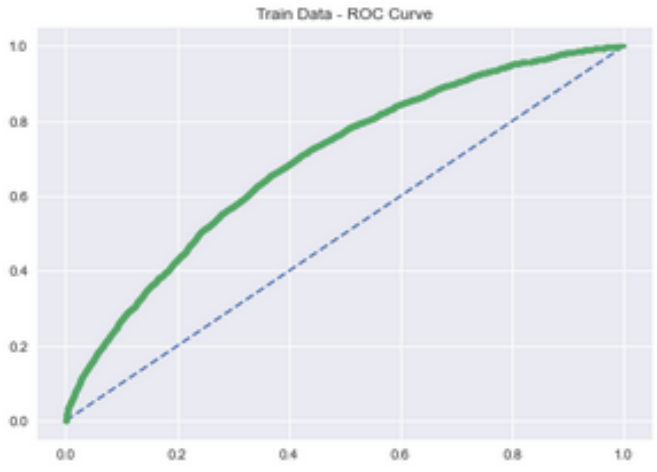
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - TREE - Scaled

```
ADA_Tree_Scaled
Train Accuracy Score for model AdaBoostClassifier() is 0.7654166940303408

--------------------Classification Report - Train Data---------------------
              precision    recall  f1-score   support

           0       0.77      0.98      0.86     11578
           1       0.59      0.07      0.13      3649

    accuracy                           0.77     15227
   macro avg       0.68      0.53      0.50     15227
weighted avg       0.73      0.77      0.69     15227

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




```
------------------------------------------------------------
AUC: 0.695
```

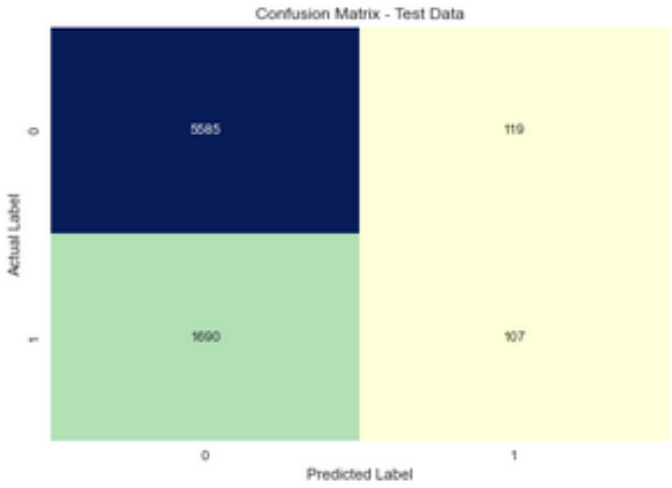


APPENDIX

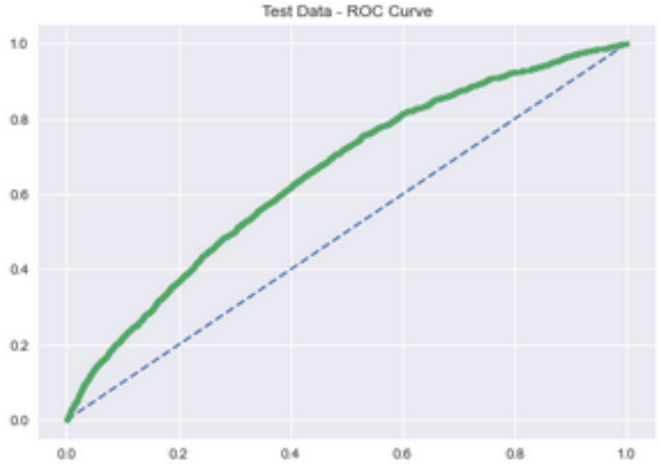
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - TREE - Scaled

```
Test Accuracy Score for model AdaBoostClassifier() is 0.7588321557125717

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.77      0.98      0.86      5704
           1       0.47      0.06      0.11      1797

    accuracy                           0.76      7501
   macro avg       0.62      0.52      0.48      7501
weighted avg       0.70      0.76      0.68      7501

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




```
-------------------------------------------------------------
AUC: 0.654
```

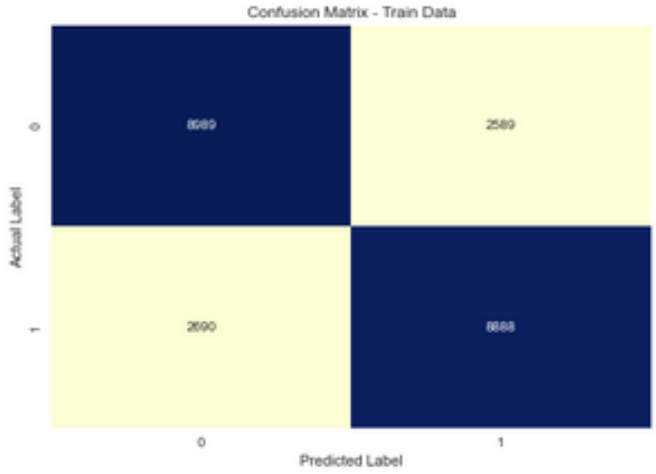


APPENDIX

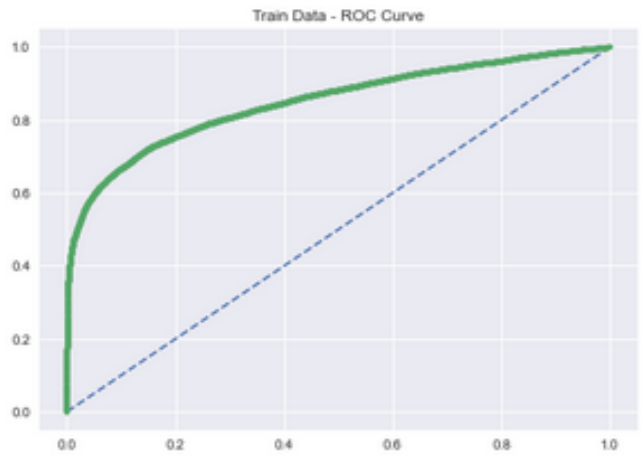
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - Tree - SMOTE - Unscaled

```
ADA_Tree_SMOTE_Unscaled
Train Accuracy Score for model AdaBoostClassifier() is 0.7720245292796684

--------------------Classification Report - Train Data---------------------
              precision    recall  f1-score   support

           0       0.77      0.78      0.77     11578
           1       0.77      0.77      0.77     11578

    accuracy                           0.77     23156
   macro avg       0.77      0.77      0.77     23156
weighted avg       0.77      0.77      0.77     23156

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




```
------------------------------------------------------------
AUC: 0.851
```

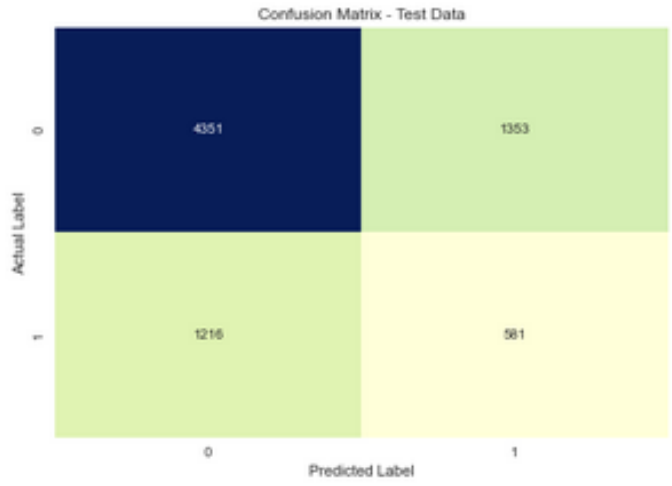


APPENDIX

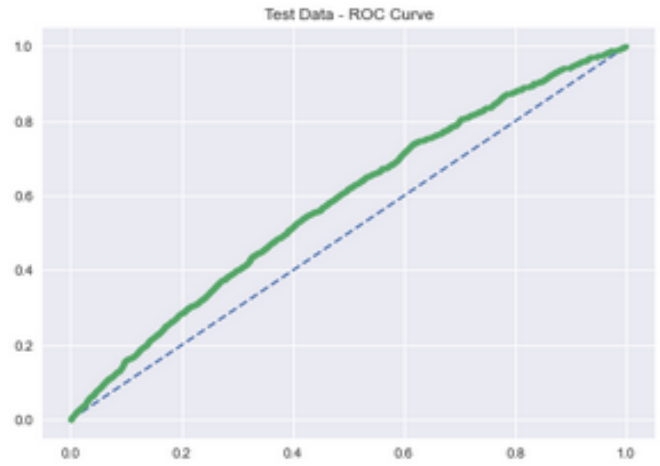
APPENDIX 5.1 - PERFORMANCE METRICS

ADABOOST - Tree - SMOTE - Unscaled

```
Test Accuracy Score for model AdaBoostClassifier() is 0.6575123316891082

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      5704
           1       0.30      0.32      0.31      1797

    accuracy                           0.66      7501
   macro avg       0.54      0.54      0.54      7501
weighted avg       0.67      0.66      0.66      7501

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




```
-------------------------------------------------------------
AUC: 0.582
```

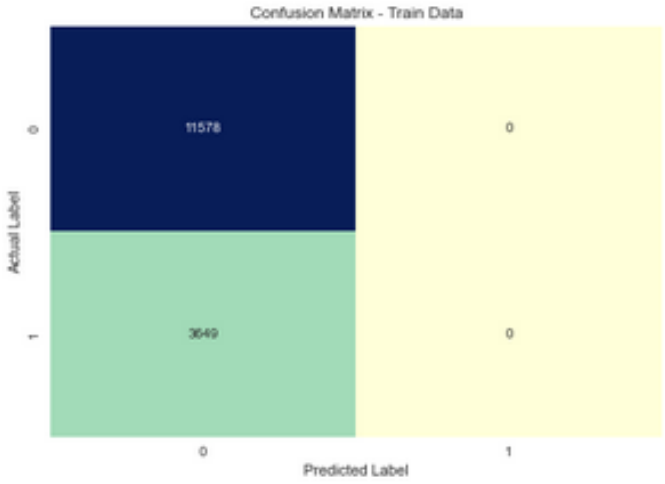


APPENDIX

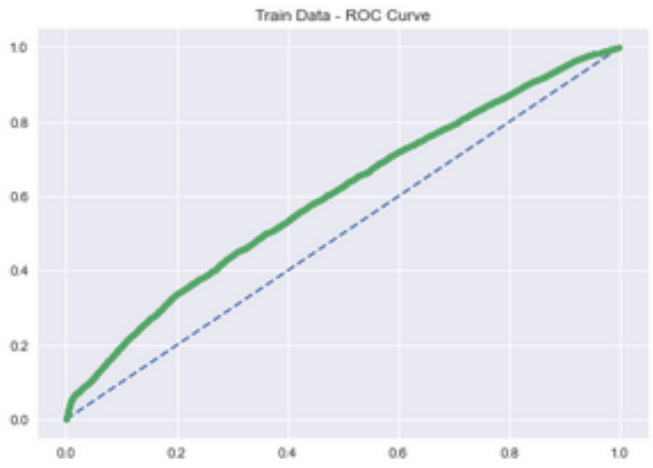
APPENDIX 5.1 - PERFORMANCE METRICS

SVM - Linear - Unscaled

```
SVM_Linear_Unscaled
Train Accuracy Score for model SVC(probability=True) is 0.760359887042753

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86     11578
           1       0.00      0.00      0.00      3649

    accuracy                           0.76     15227
   macro avg       0.38      0.50      0.43     15227
weighted avg       0.58      0.76      0.66     15227

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




```
------------------------------------------------------------
AUC: 0.600
```

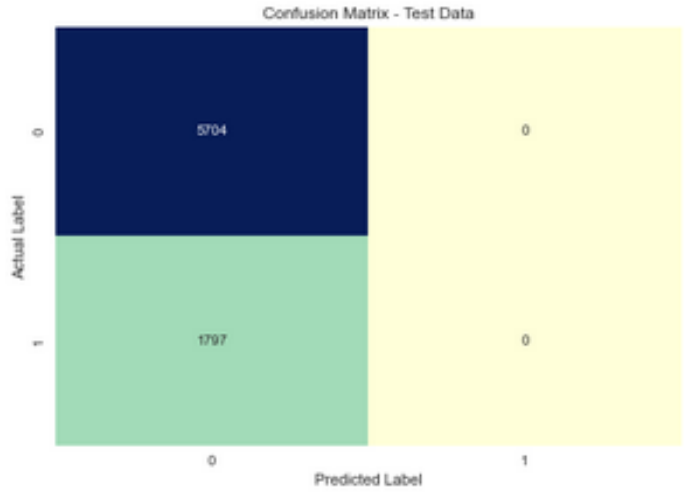


APPENDIX

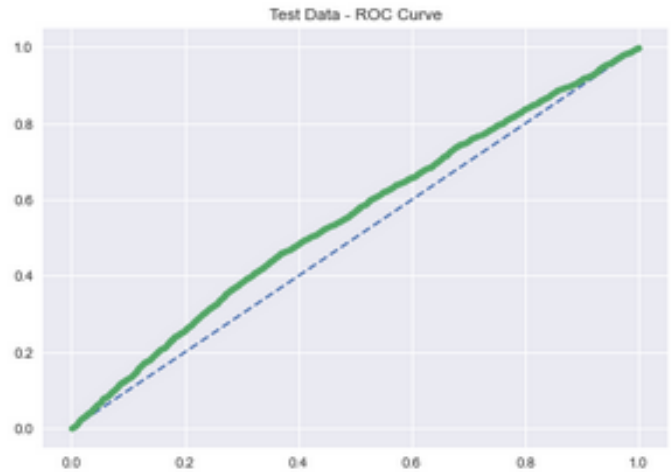
APPENDIX 5.1 - PERFORMANCE METRICS

SVM - Linear - Unscaled

```
Test Accuracy Score for model SVC(probability=True) is 0.760431942407679

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86      5704
           1       0.00      0.00      0.00      1797

    accuracy                           0.76      7501
   macro avg       0.38      0.50      0.43      7501
weighted avg       0.58      0.76      0.66      7501

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




```
------------------------------------------------------------
AUC: 0.551
```

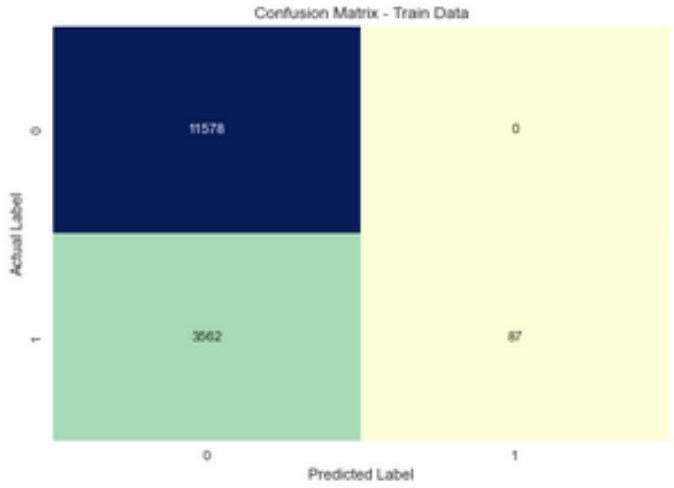


APPENDIX

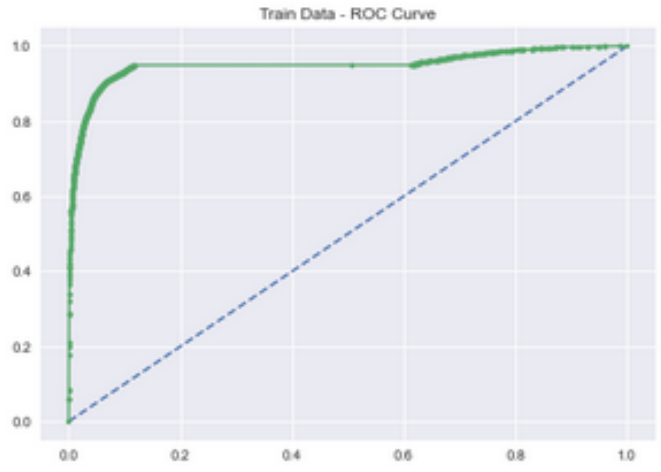
APPENDIX 5.1 - PERFORMANCE METRICS

SVM - Linear - Scaled

```
SVM_Linear_Scaled
Train Accuracy Score for model SVC(probability=True) is 0.7660734222105471

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.87     11578
           1       1.00      0.02      0.05      3649

    accuracy                           0.77     15227
   macro avg       0.88      0.51      0.46     15227
weighted avg       0.82      0.77      0.67     15227

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




```
------------------------------------------------------------
AUC: 0.950
```

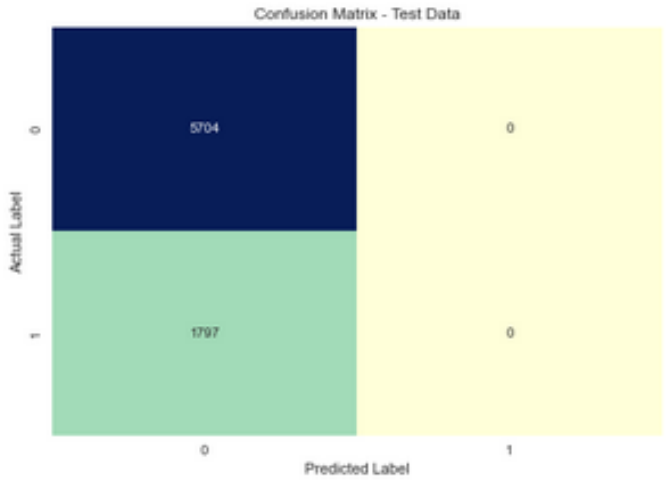


APPENDIX

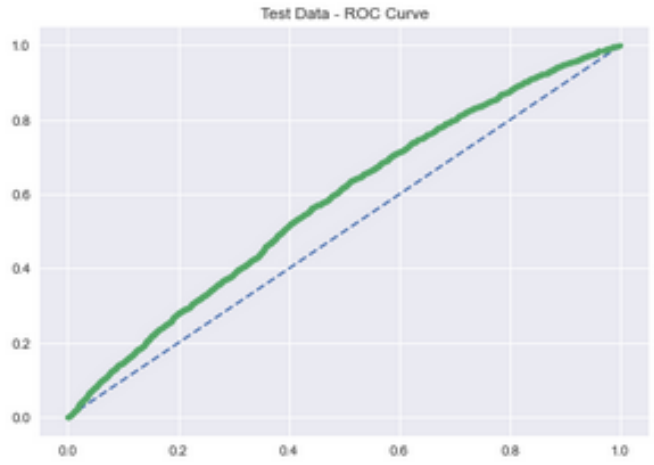
APPENDIX 5.1 - PERFORMANCE METRICS

SVM - Linear - Scaled

```
Test Accuracy Score for model SVC(probability=True) is 0.7604319424076795

--------------------Classification Report - Test Data---------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86      5704
           1       0.00      0.00      0.00      1797

    accuracy                           0.76      7501
   macro avg       0.38      0.50      0.43      7501
weighted avg       0.58      0.76      0.66      7501

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




```
------------------------------------------------------------
AUC: 0.580
```

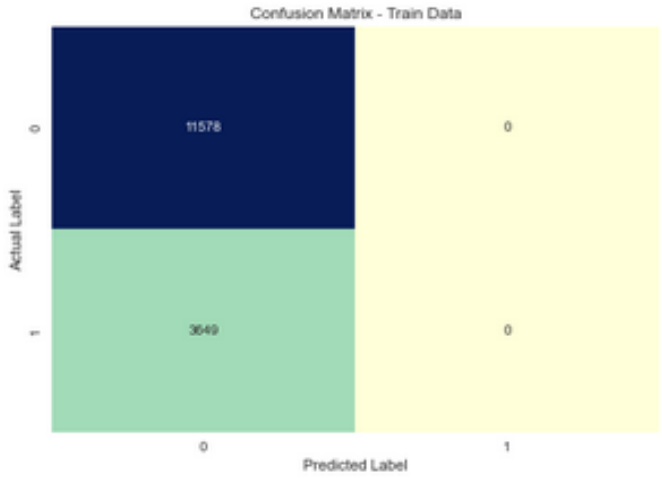


APPENDIX

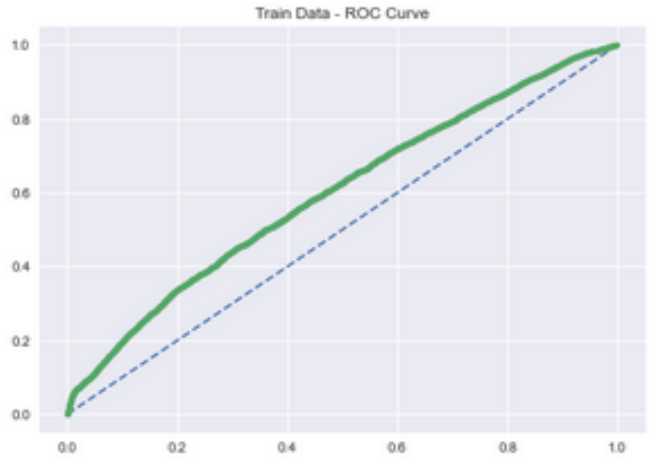
APPENDIX 5.1 - PERFORMANCE METRICS

LDA - Linear - Unscaled

```
LDA_Linear_Unscaled
Train Accuracy Score for model SVC(probability=True) is 0.760359887042753

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86     11578
           1       0.00      0.00      0.00      3649

    accuracy                           0.76     15227
   macro avg       0.38      0.50      0.43     15227
weighted avg       0.58      0.76      0.66     15227

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




```
------------------------------------------------------------
AUC: 0.600
```

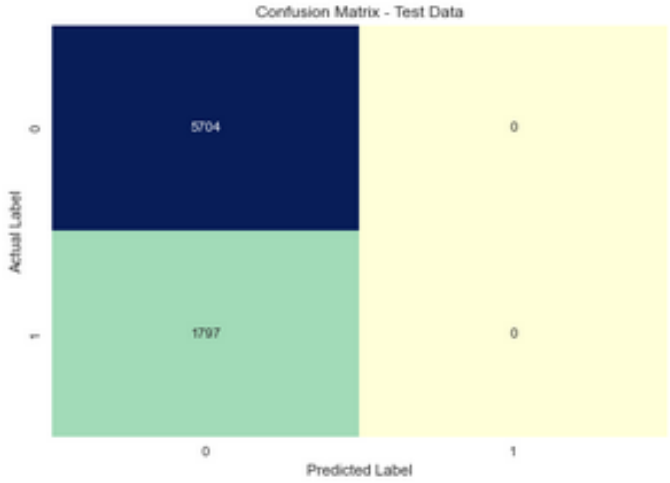


APPENDIX

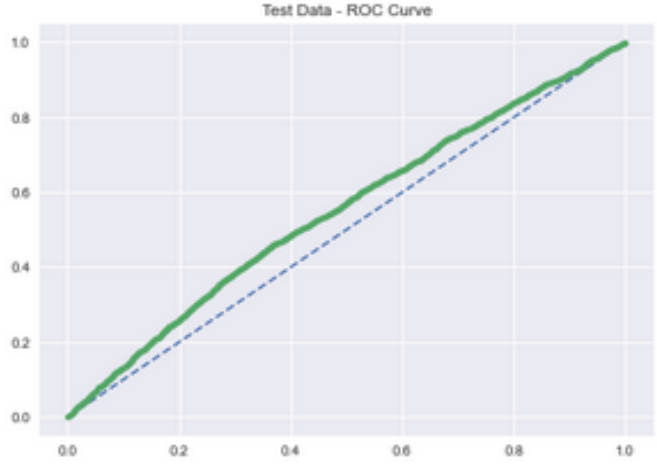
APPENDIX 5.1 - PERFORMANCE METRICS

LDA - Linear - Unscaled

```
Test Accuracy Score for model SVC(probability=True) is 0.760431942407679

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.76      1.00      0.86      5704
           1       0.00      0.00      0.00      1797

    accuracy                           0.76      7501
   macro avg       0.38      0.50      0.43      7501
weighted avg       0.58      0.76      0.66      7501

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




```
------------------------------------------------------------
AUC: 0.551
```




THANK
YOU!