Classification on Personal Loan Data

March 13, 2022

1 Classification on Personal Loan Data

1.1 Submitted By - Pramatosh Ray (QR2011)

This Notebook is about classification model on personal loan data and predicts whether a loan should be given or not

Following Models are used Here

1. SVM (Linear, Polynomial, Radial, Sigmoid)
2. Ensemble (Random Forest)
3. Ensemble (Bagging)
4. Ensemble (Boosting (Gradient boosting, Ada Boost, Stacking))
5. KNN
6. Logistic Regression
7. CART
8. Bayesian Learning (Naïve Bayes (Gaussian,Multinomial, Complement), Bayesian network)

List of dependency Libraries to run this File 1. Numpy 2. Pandas 3. Seaborn 4. SKLearn 5. Matplotlib

1.2 Importing common libraries

[595]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.cm as cm
import matplotlib as mpl
from matplotlib.gridspec import GridSpec

rounding_factor=4
```

1.3 Folder for saving the images

[596]:
```
# Where to save the figures

PROJECT_ROOT_DIR = "."
CHAPTER_ID = "Personal Loan"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
```

[925]:
```
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()

    plt.savefig(path, format=fig_extension, dpi=resolution)
```

1.4 Data Insights

[598]:
```
data=pd.read_csv('Personal Loan Data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 5000 non-null   int64
 1   Experience          5000 non-null   int64
 2   Income              5000 non-null   int64
 3   Family              5000 non-null   int64
 4   CCAvg               5000 non-null   float64
 5   Education           5000 non-null   int64
 6   Mortgage            5000 non-null   int64
 7   Securities Account  5000 non-null   int64
 8   CD Account          5000 non-null   int64
 9   Online              5000 non-null   int64
 10  CreditCard          5000 non-null   int64
 11  Personal Loan       5000 non-null   int64
dtypes: float64(1), int64(11)
memory usage: 468.9 KB
```

[599]:
```
data.head()
```

[599]:
```
   Age  Experience  Income  Family  CCAvg  Education  Mortgage  \
0   25           1      49       4    1.6          1         0
1   45          19      34       3    1.5          1         0
```

```
2   39            15       11         1     1.0           1           0
3   35             9      100         1     2.7           2           0
4   35             8       45         4     1.0           2           0

   Securities Account  CD Account  Online  CreditCard  Personal Loan
0                   1           0       0           0              0
1                   1           0       0           0              0
2                   0           0       0           0              0
3                   0           0       0           0              0
4                   0           0       0           1              0
```

[600]:
```python
for column in data:
    print(column ,end="  ")
    print(data[column].nunique())
    #print(data[column].value_counts())
```

```
Age  45
Experience  47
Income  162
Family  4
CCAvg  108
Education  3
Mortgage  347
Securities Account  2
CD Account  2
Online  2
CreditCard  2
Personal Loan  2
```

Categorical and numerical variables are seperated

[1265]:
```python
var1=['Age','Experience','Income','CCAvg','Mortgage']
var2=['Family','Education','Securities Account','CD␣
 ↪Account','Online','CreditCard']
var_all= ['Age','Experience','Income','CCAvg','Mortgage','Family','Education',
          'Securities Account','CD Account','Online','CreditCard']
```

1.5 Visualisation Plots

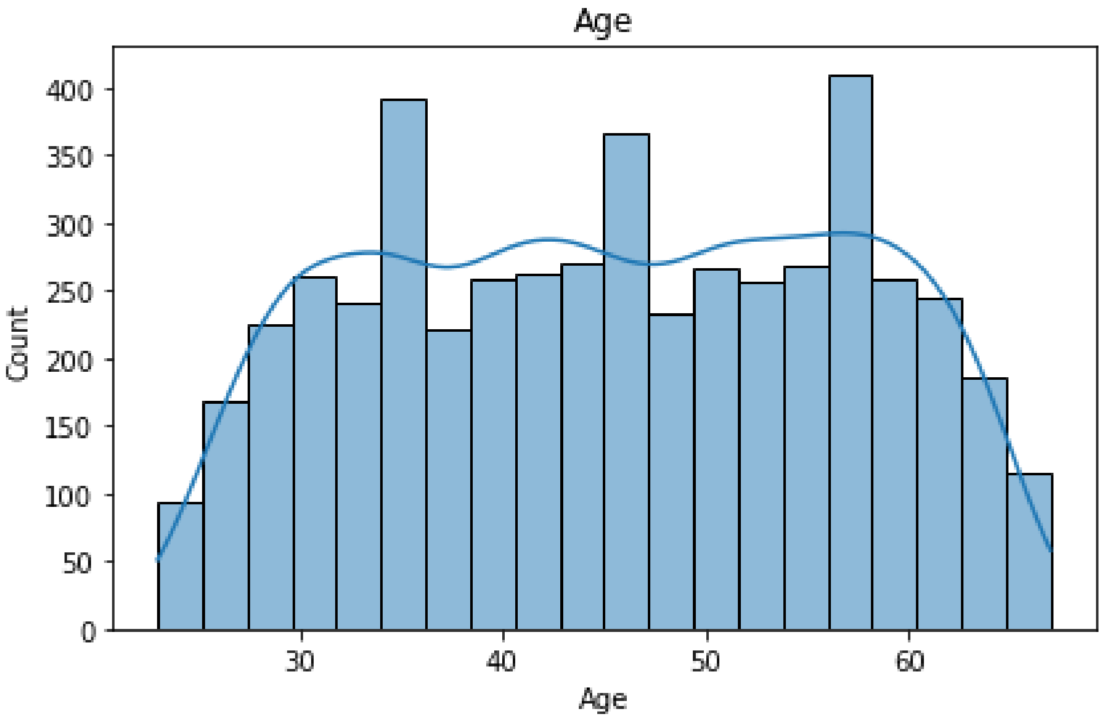
Histogram Plots

[602]:
```python
%matplotlib inline
```

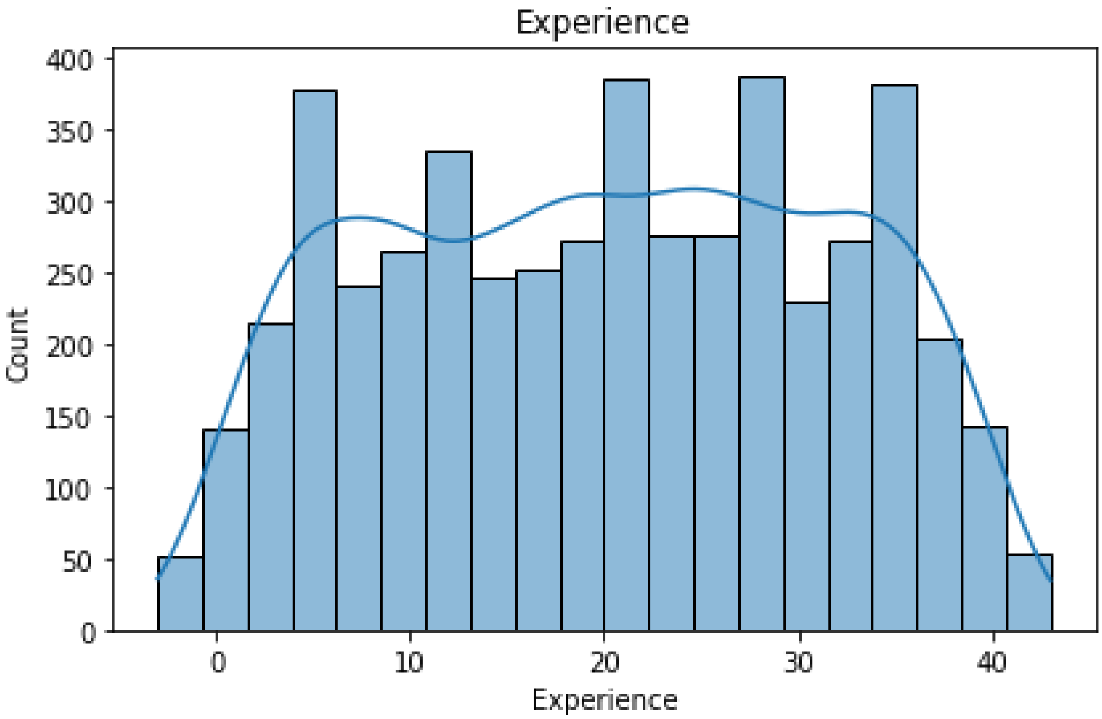
[603]:
```python
for var in var1:
    #plt.figure(dpi=300)
    sns.histplot(data[var],bins=20, kde=True)

    plt.title(var)
```

```
    #plt.legend()
    save_fig(f"Histogram of {var}")
    plt.show()
```

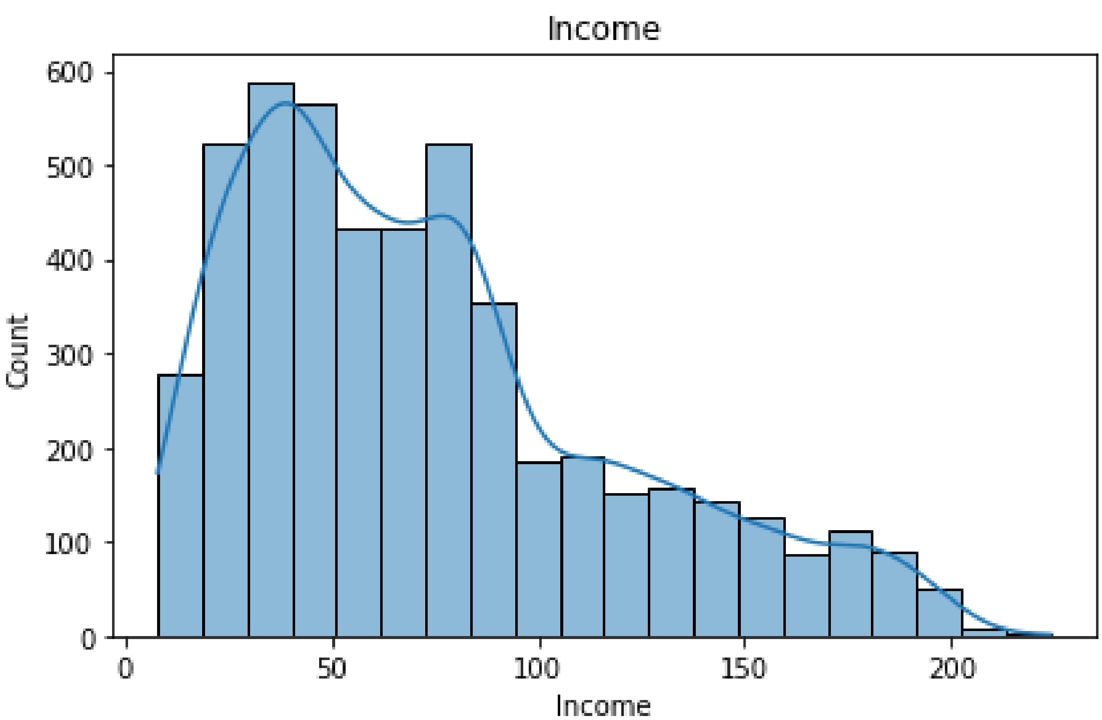
```
Saving figure Histogram of Age
```



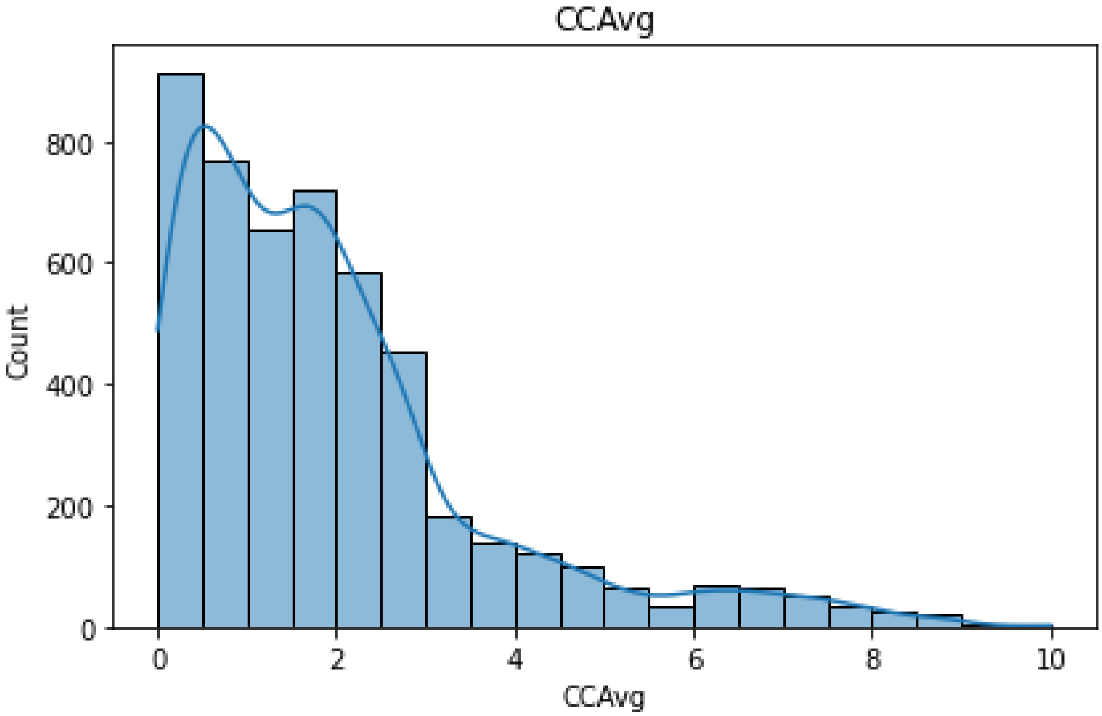

```
Saving figure Histogram of Experience
```



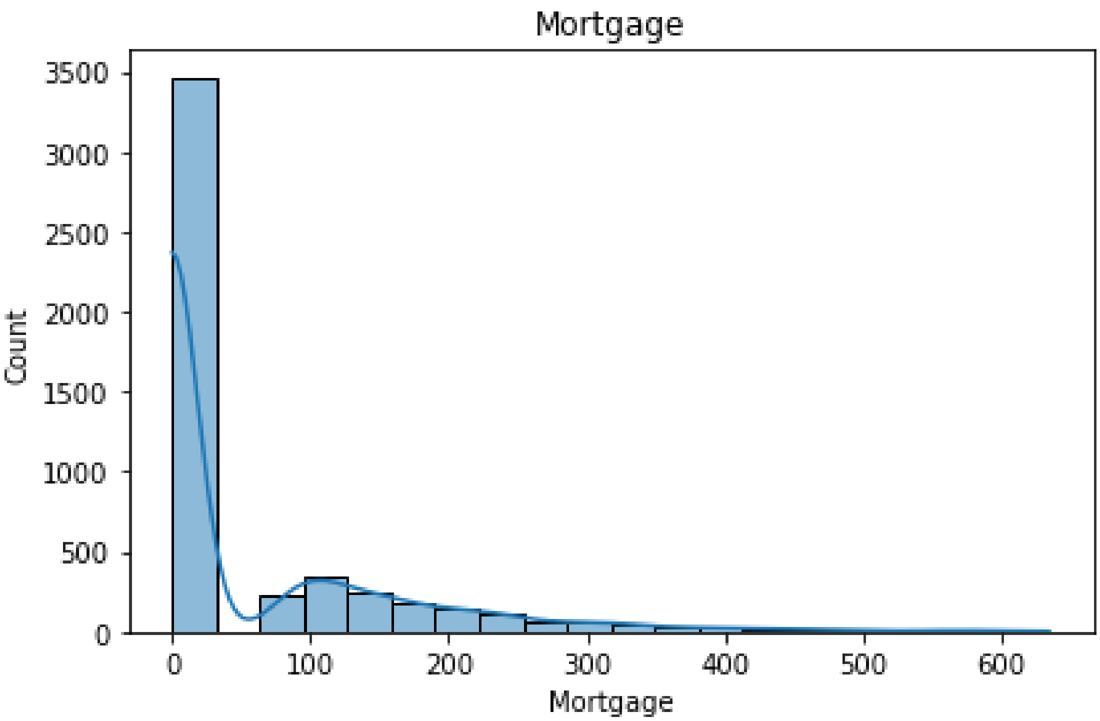
Saving figure Histogram of Income



```
Saving figure Histogram of CCAvg
```



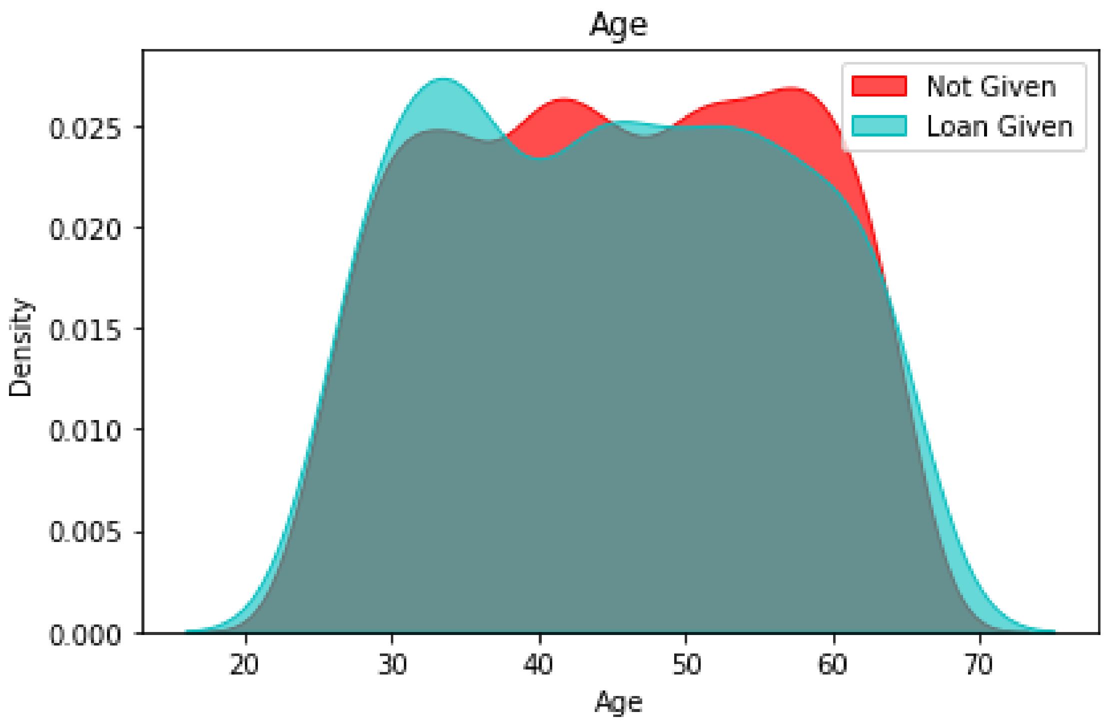

```
Saving figure Histogram of Mortgage
```



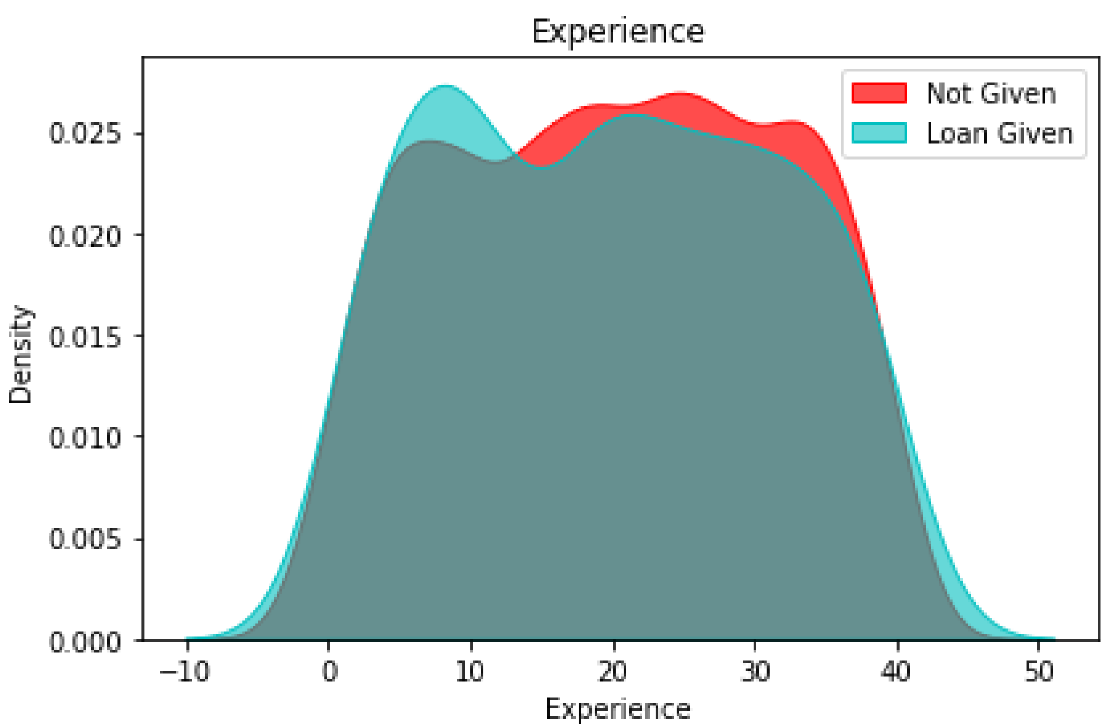
Density Plot Loan Status

[604]:
```
for var in var1:
    #plt.figure(dpi=300)
    sns.kdeplot(data.loc[data['Personal Loan']==0,var], shade=True,
                color="r", label='Not Given', alpha=.7)
    sns.kdeplot(data.loc[data['Personal Loan']==1,var], shade=True,
                color="c", label='Loan Given', alpha=.6)

    plt.title(var)
    plt.legend()
    save_fig(f"Density plot of {var}")
    plt.show()
```

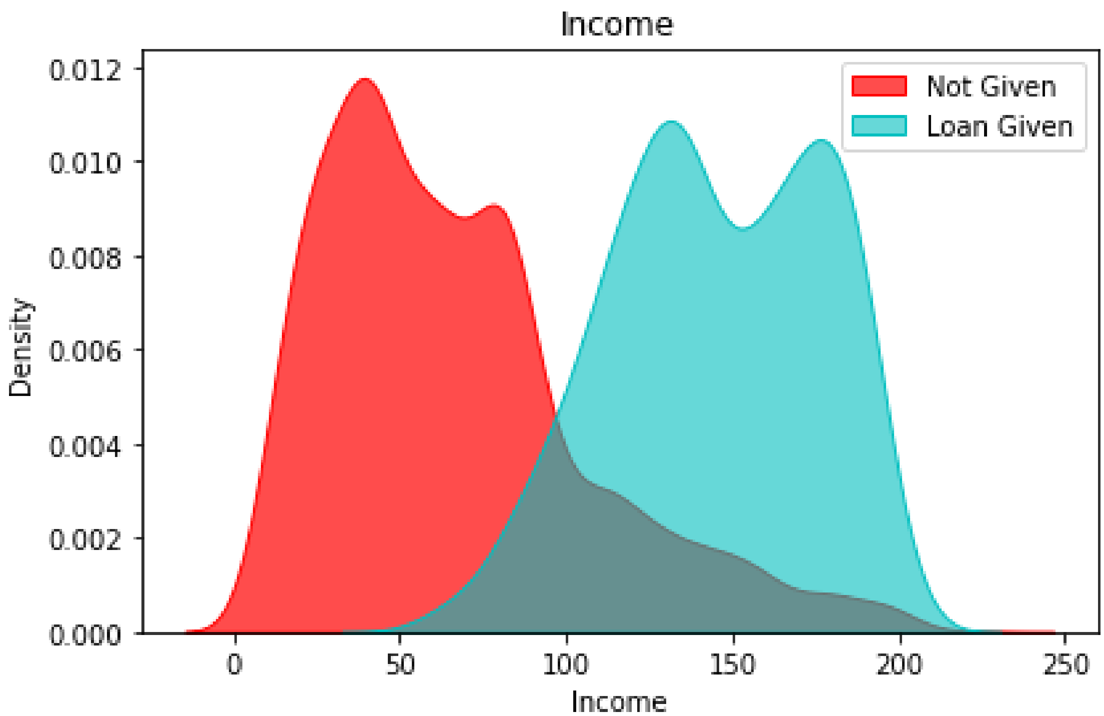
```
Saving figure Density plot of Age
```



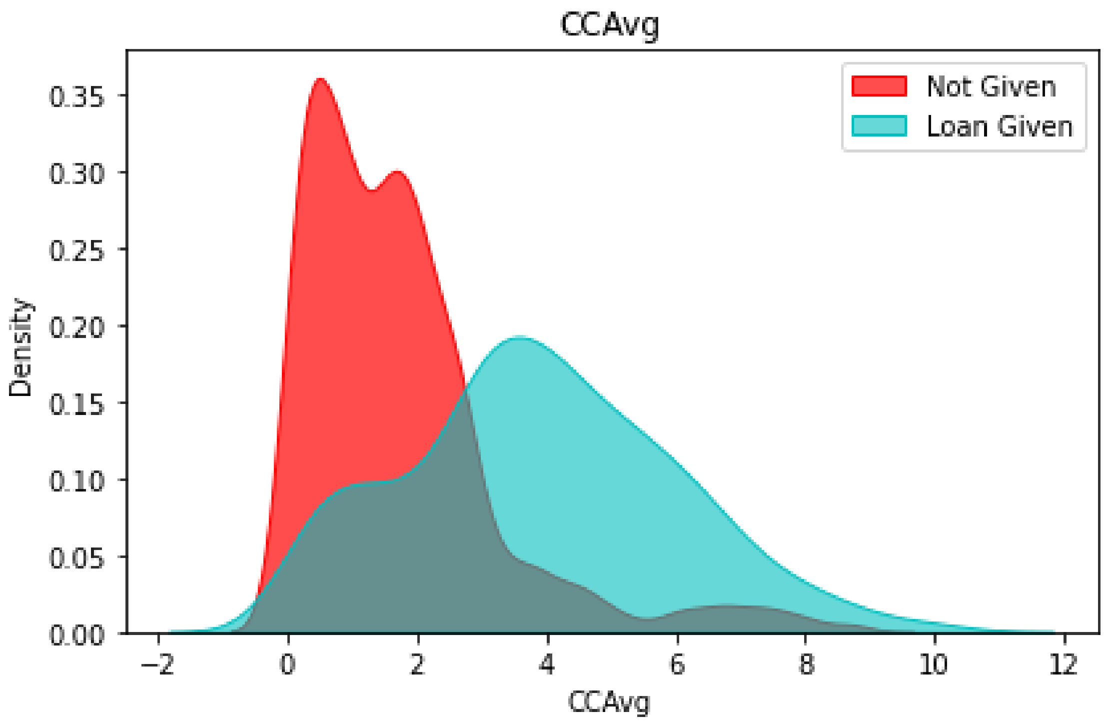

```
Saving figure Density plot of Experience
```



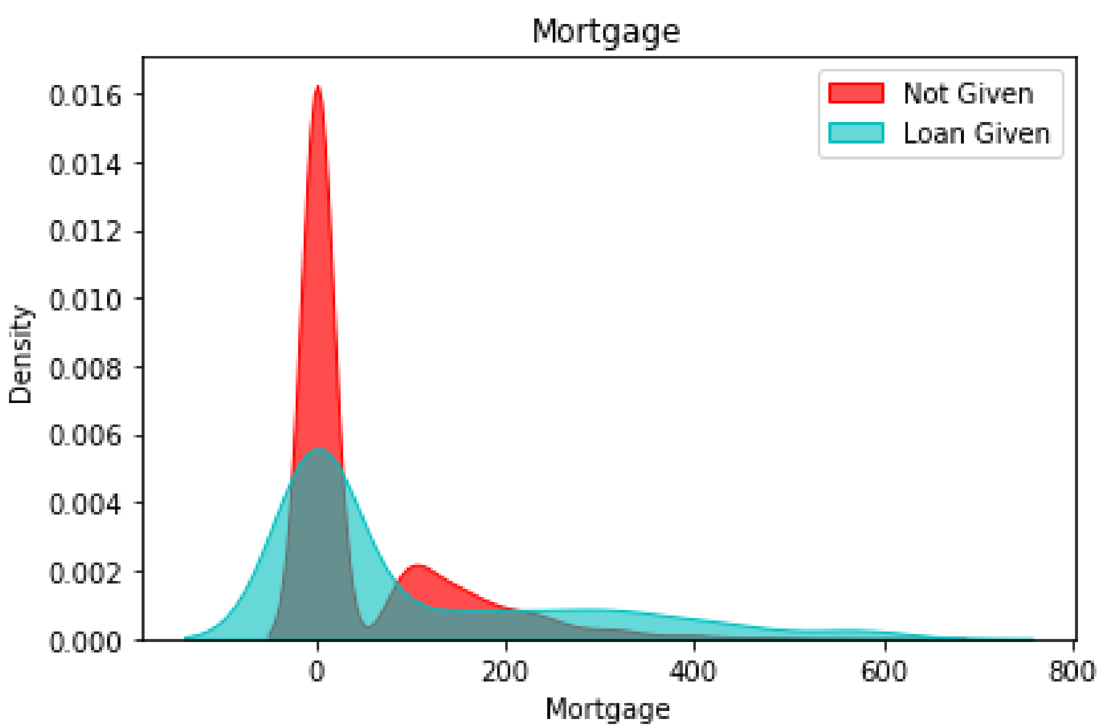
Saving figure Density plot of Income



Saving figure Density plot of CCAvg



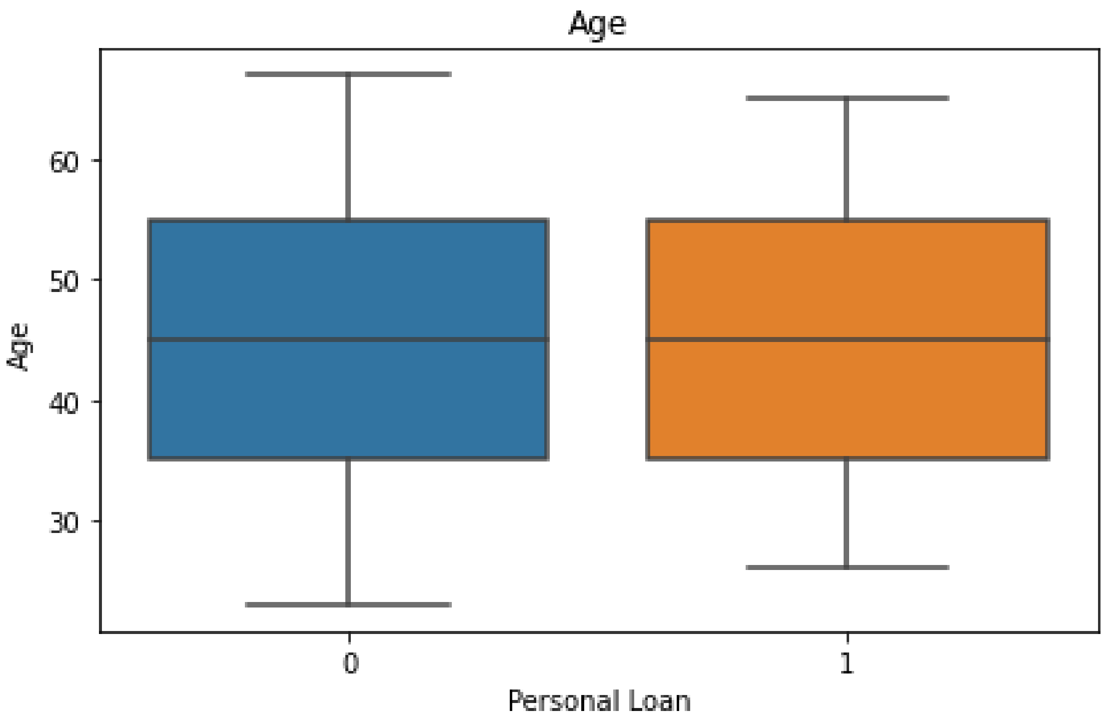

```
Saving figure Density plot of Mortgage
```



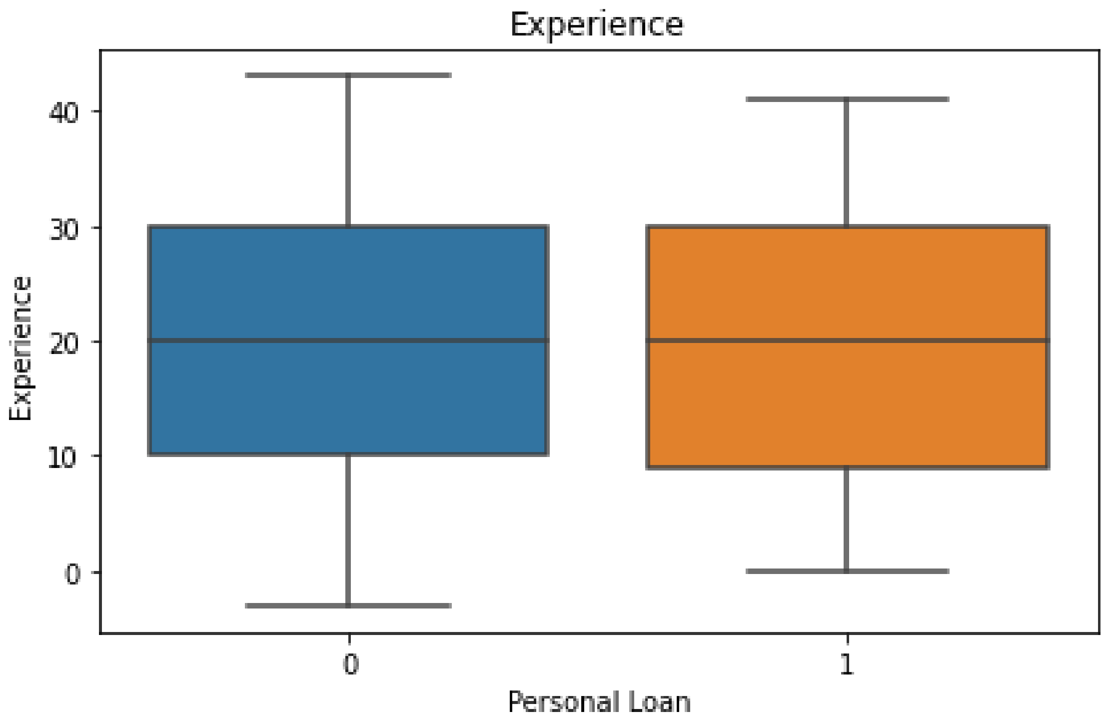
Box Plot Loan Status

[605]:
```
for var in var1:
    #plt.figure(dpi=300)
    sns.boxplot(x='Personal Loan',y=var,data=data)

    plt.title(var)
    #plt.legend()
    save_fig(f"Box plot of {var}")
    plt.show()
```

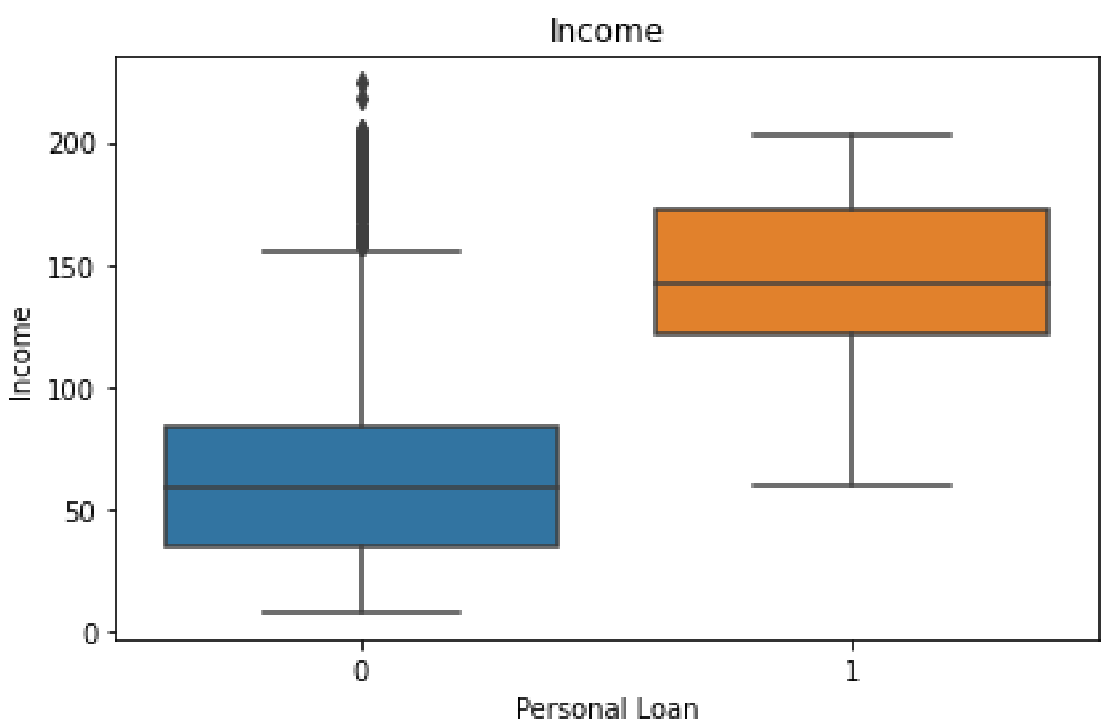
```
Saving figure Box plot of Age
```



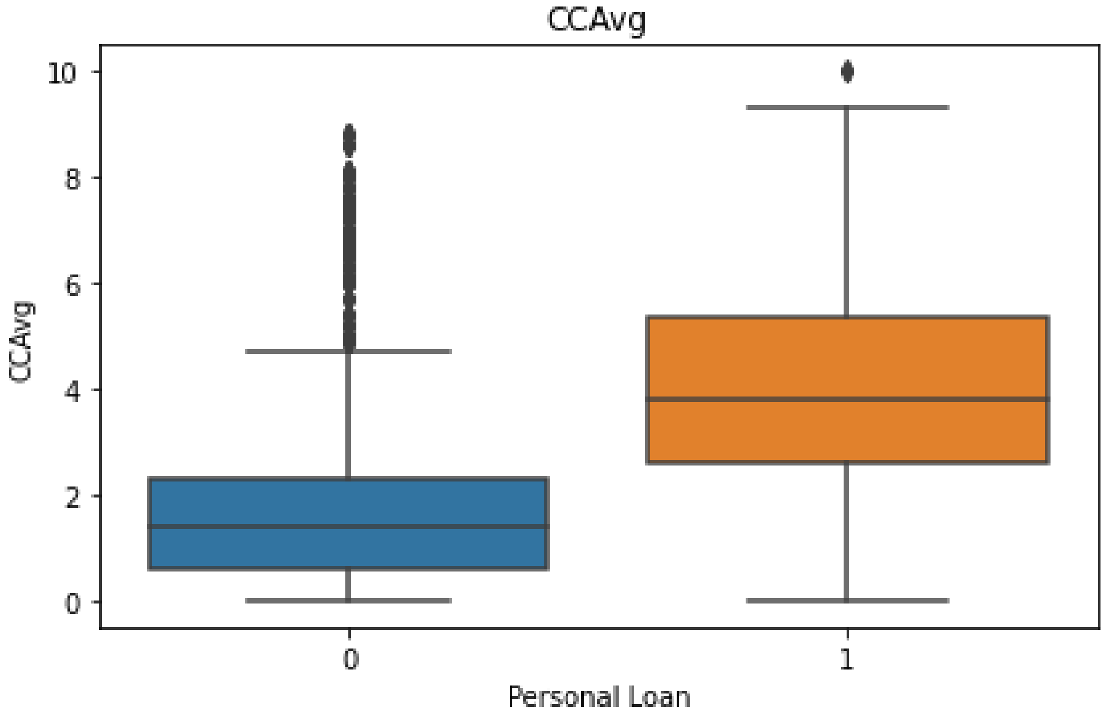

```
Saving figure Box plot of Experience
```



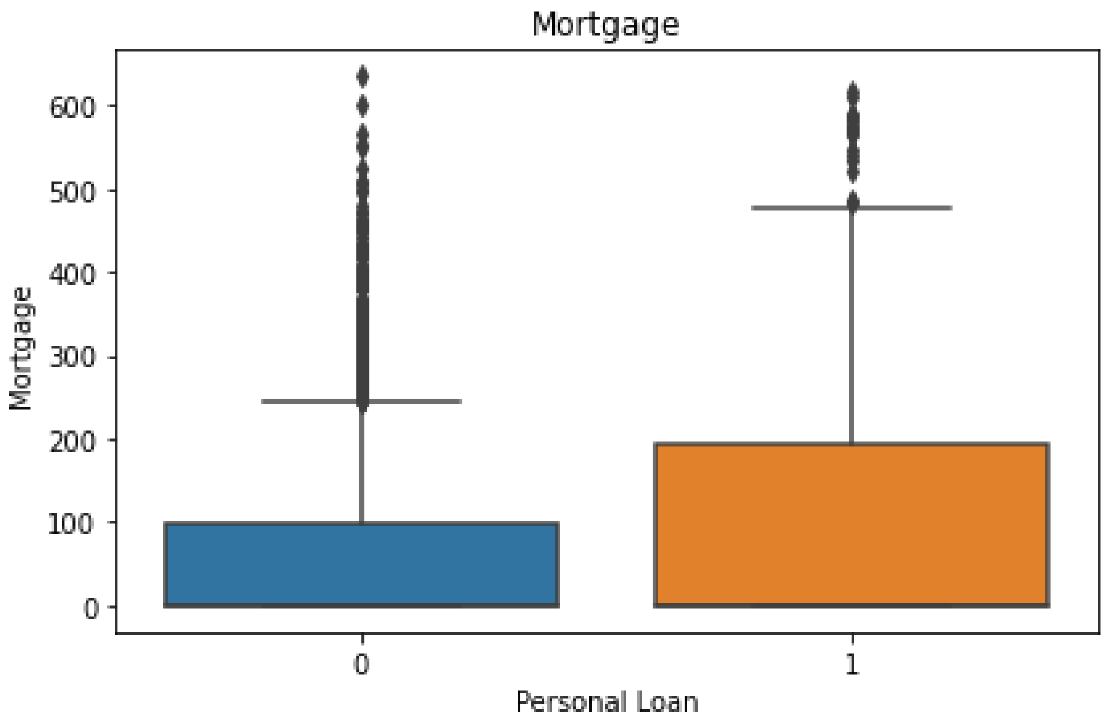
Saving figure Box plot of Income



```
Saving figure Box plot of CCAvg
```



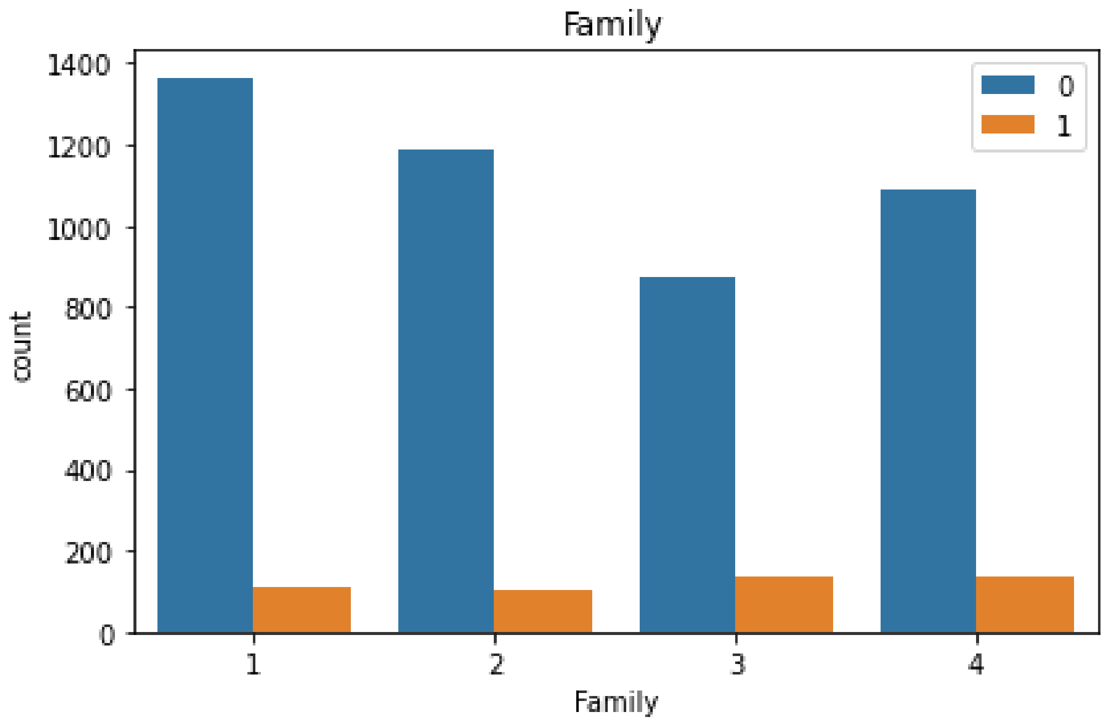

```
Saving figure Box plot of Mortgage
```



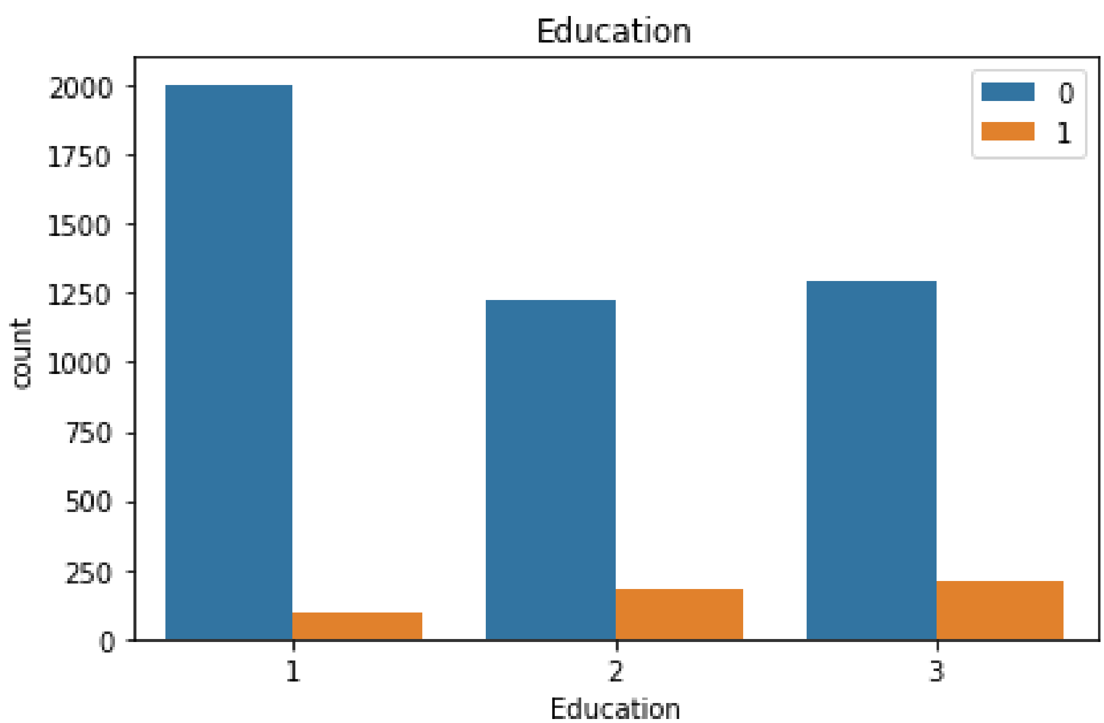
Count Plot Loan Status

[606]:
```
for var in var2:
    #plt.figure(dpi=300)
    sns.countplot(x=var,hue='Personal Loan',data=data)
    plt.title(var)
    plt.legend()
    save_fig(f"Count plot of {var}")
    plt.show()
```

```
Saving figure Count plot of Family
```

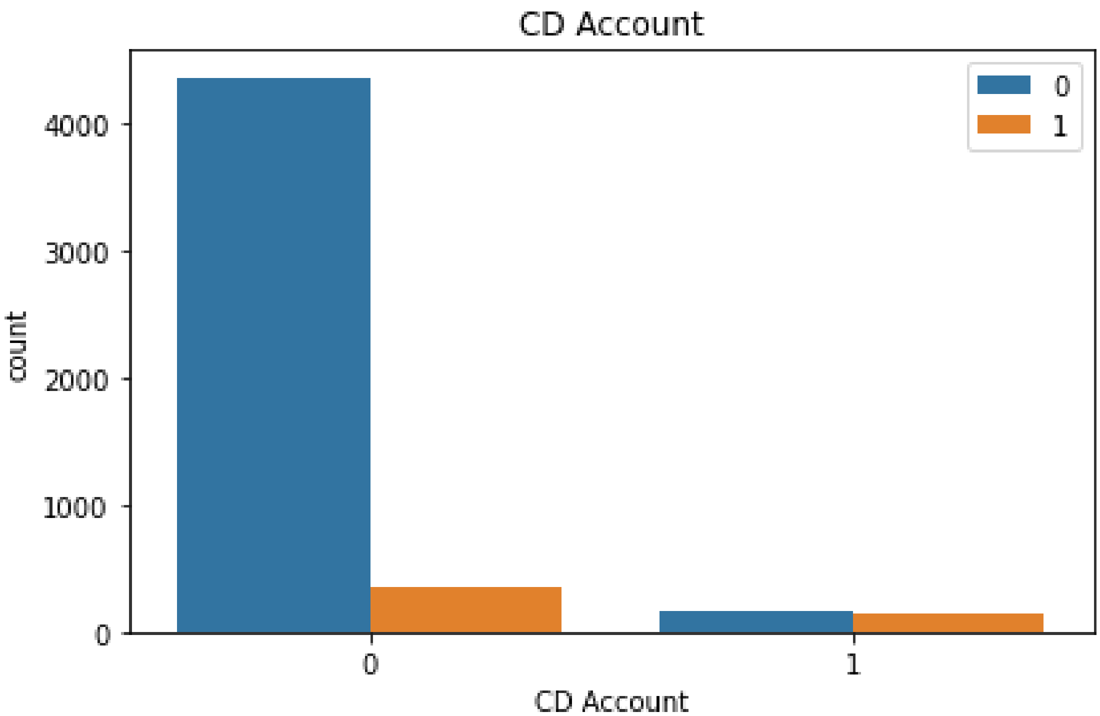



```
Saving figure Count plot of Education
```

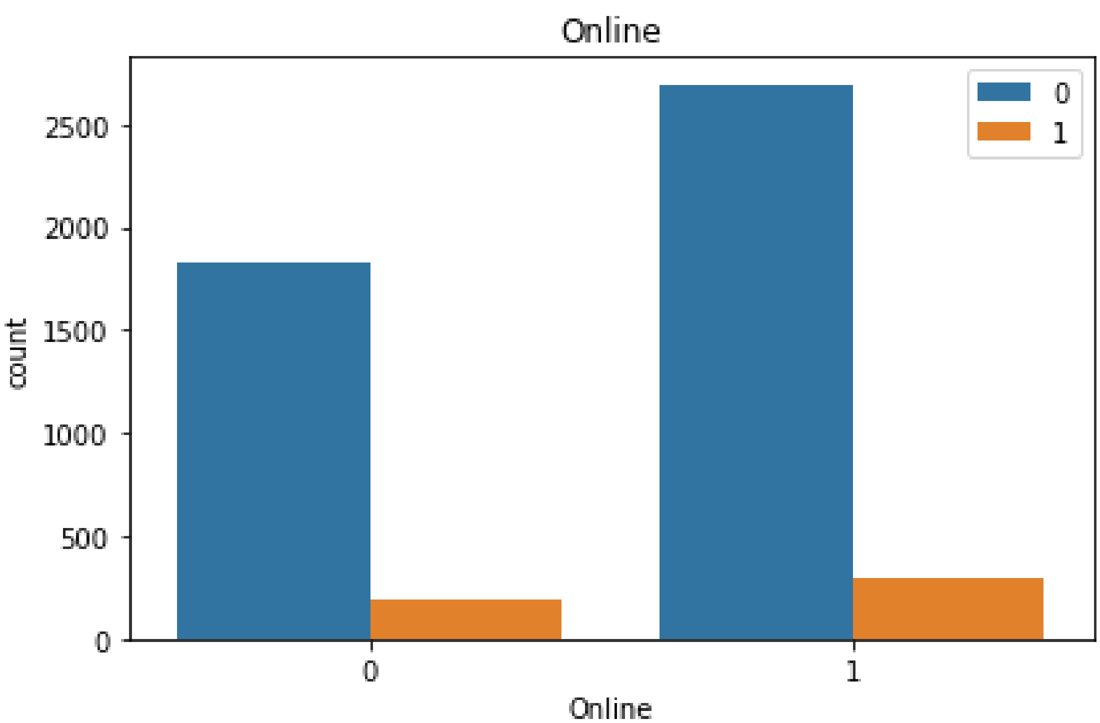


Saving figure Count plot of Securities Account

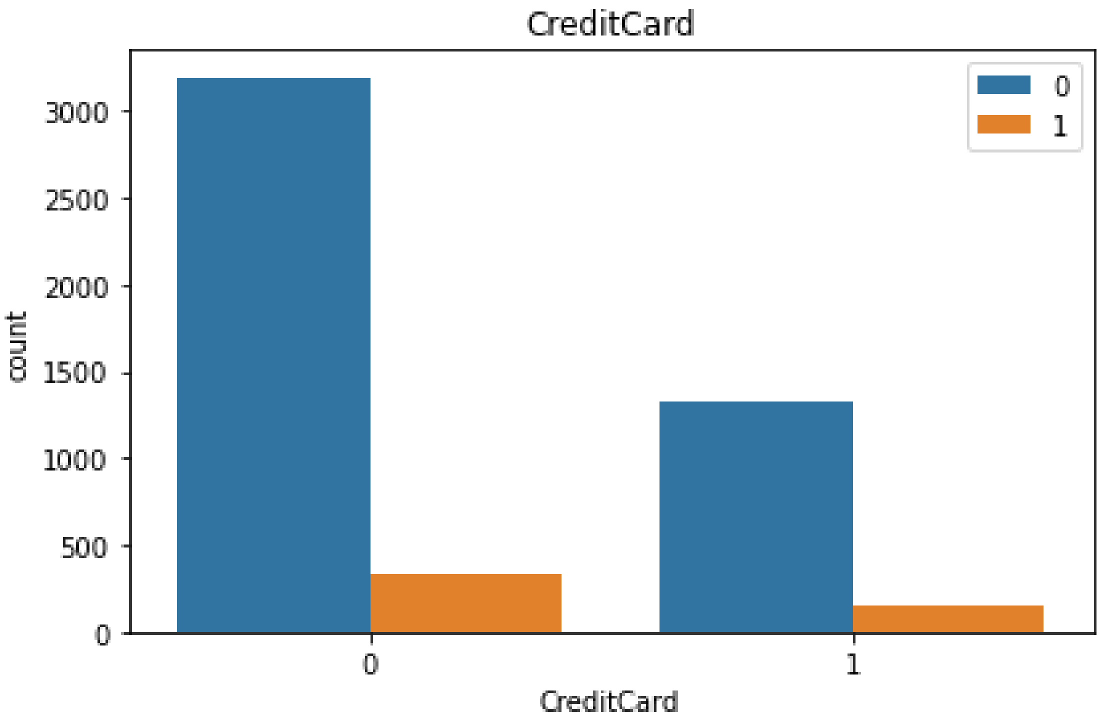
Saving figure Count plot of CD Account



Saving figure Count plot of Online



Saving figure Count plot of CreditCard



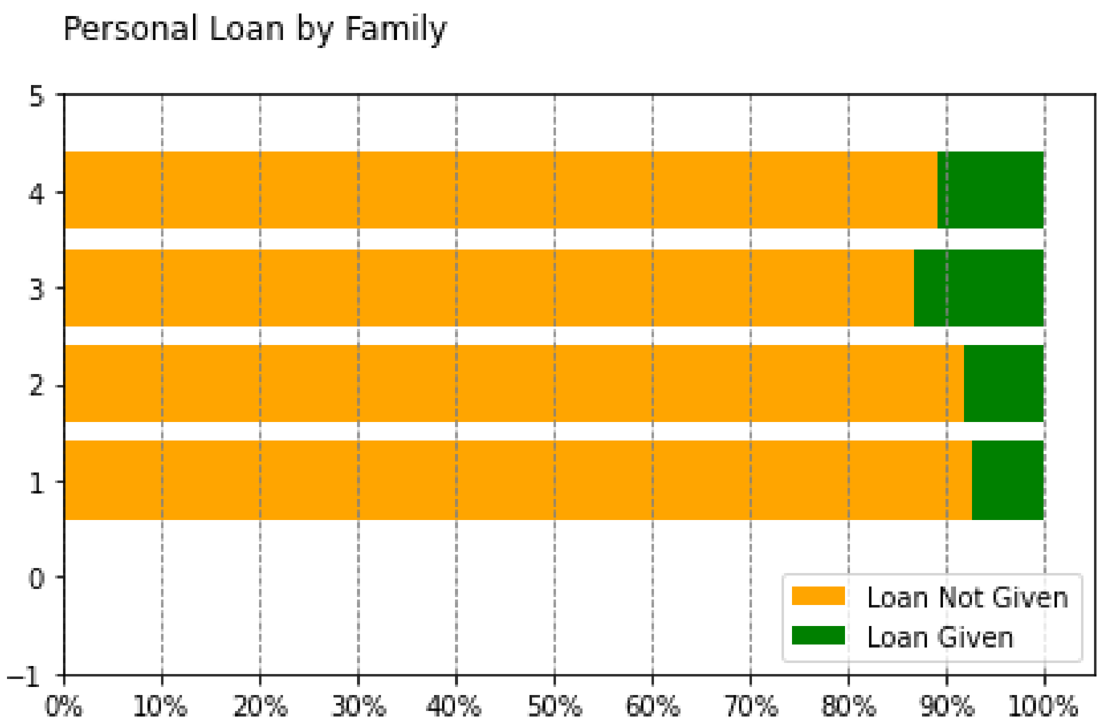
Divided Bar Diagram Loan Status

[607]:
```
def plot_stackedbar_p(df, labels, colors, title):

    fields = df.columns.tolist()
    fig, ax = plt.subplots(1)# plot bars
    left = len(df) * [0]

    for idx, name in enumerate(fields):
        plt.barh(df.index, df[name], left = left, color=colors[idx])
        left = left + df[name]# title and subtitle

    plt.title(title, loc='left')
    #plt.text(0, ax.get_yticks()[-1] + 0.3, subtitle)# legend
    plt.legend(labels,loc=4)# remove spines
    xticks = np.arange(0,1.1,0.1)
    xlabels = ['{}%'.format(i) for i in np.arange(0,101,10)]
    plt.xticks(xticks, xlabels)# adjust limits and draw grid lines
    plt.ylim(-1, ax.get_yticks()[-1])
    ax.xaxis.grid(color='gray', linestyle='dashed')
    save_fig(f"Divided Bar Diagram of {var}")
```

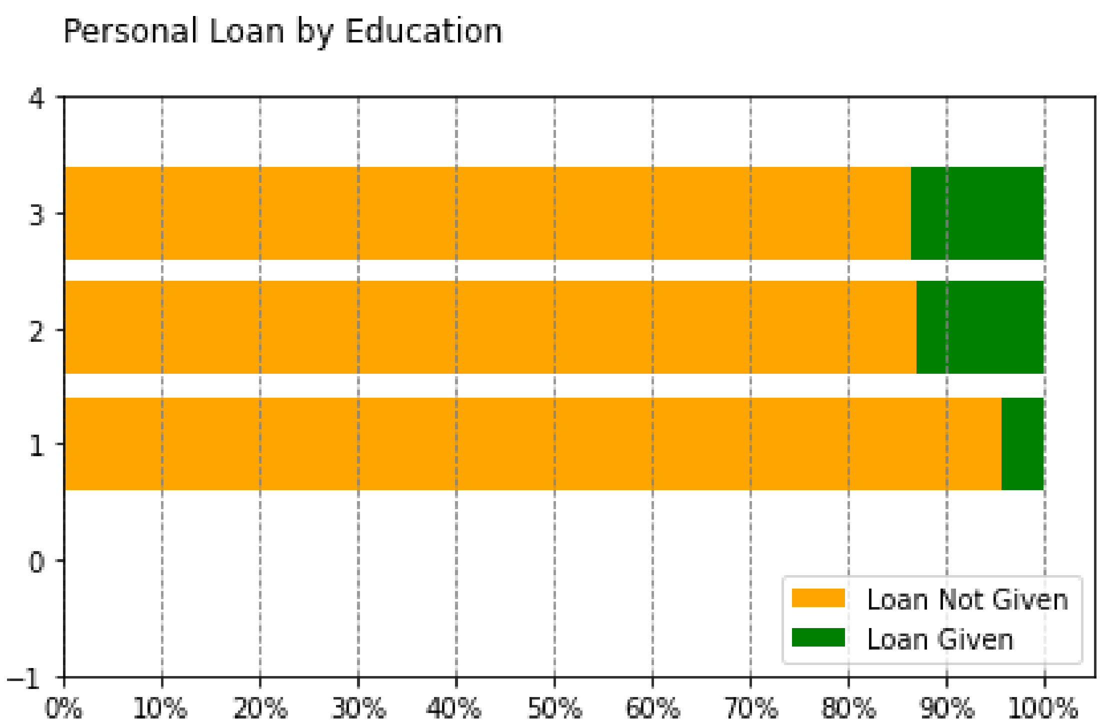
```
    plt.show()
```

[608]:
```
for var in var2:
    df_agg=data[[var,'Personal Loan']].copy()

    for types in df_agg['Personal Loan'].unique():
        df_agg[types]=df_agg['Personal Loan'].map(lambda x : 1 if x==types else␣
 ↪0)

    df_agg.drop(['Personal Loan'],axis=1,inplace=True)
    df_agg=df_agg.groupby(var).sum()
    fields=[0,1]

    df_agg['Total'] = df_agg[fields].sum(axis=1)


    for i in fields:
        df_agg['{}_Percent'.format(i)] = df_agg[i] / df_agg['Total']


    df_agg.drop([0,1,'Total'],axis=1,inplace=True)

    # variables
    labels = ['Loan Not Given', 'Loan Given']
    colors = ['orange', 'green']
    title = f'Personal Loan by {var}\n'
   #subtitle = 'Proportion of loan Status'
    plot_stackedbar_p(df_agg, labels, colors, title)
```

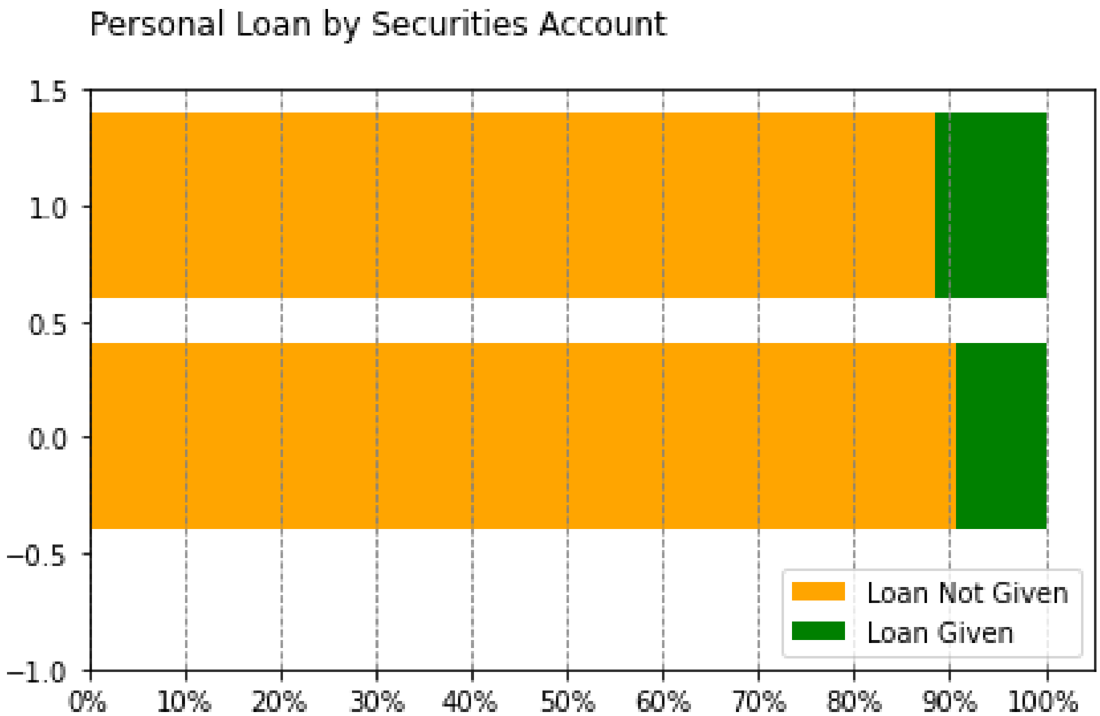
```
Saving figure Divided Bar Diagram of Family
```



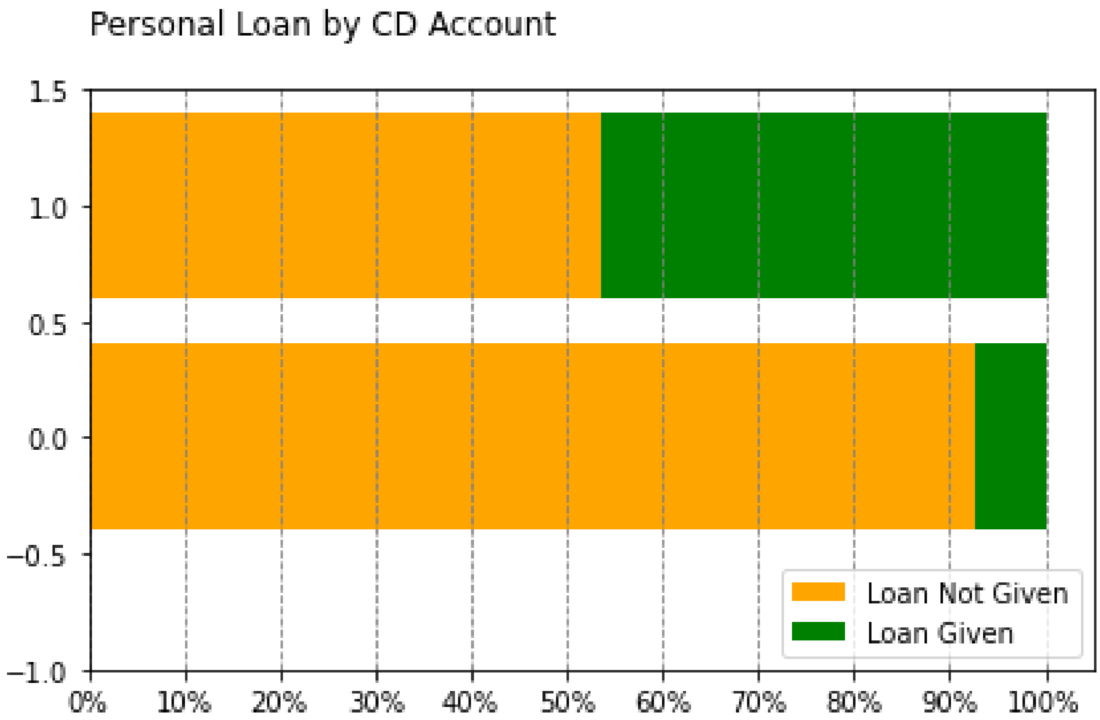

```
Saving figure Divided Bar Diagram of Education
```



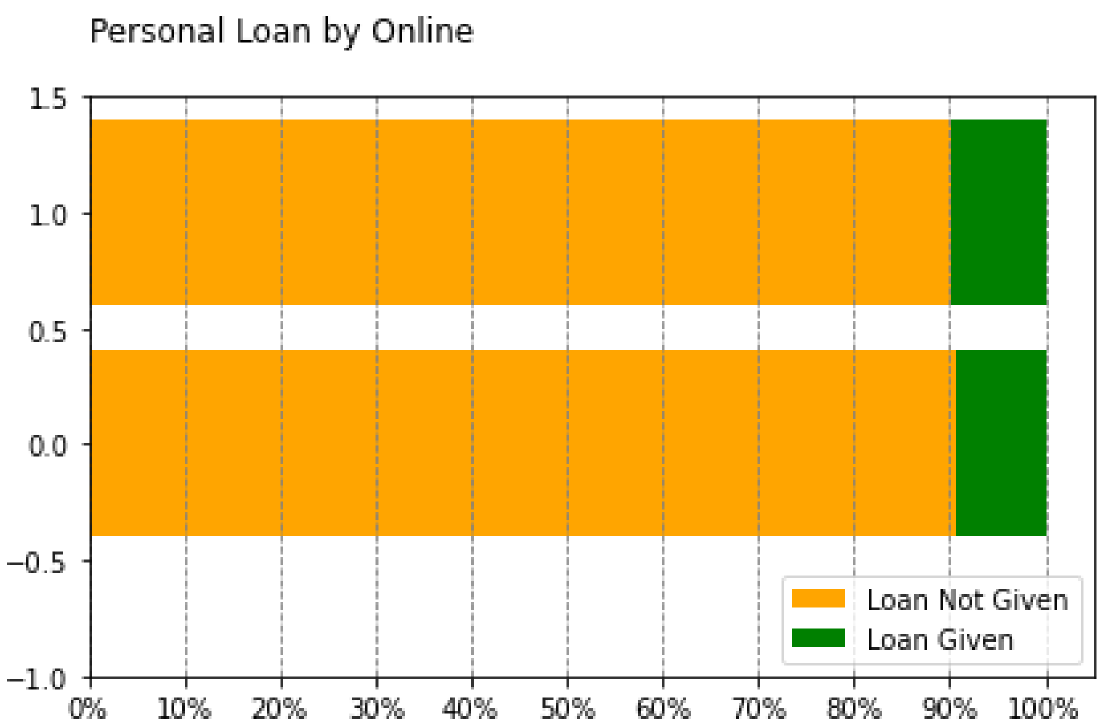
Saving figure Divided Bar Diagram of Securities Account



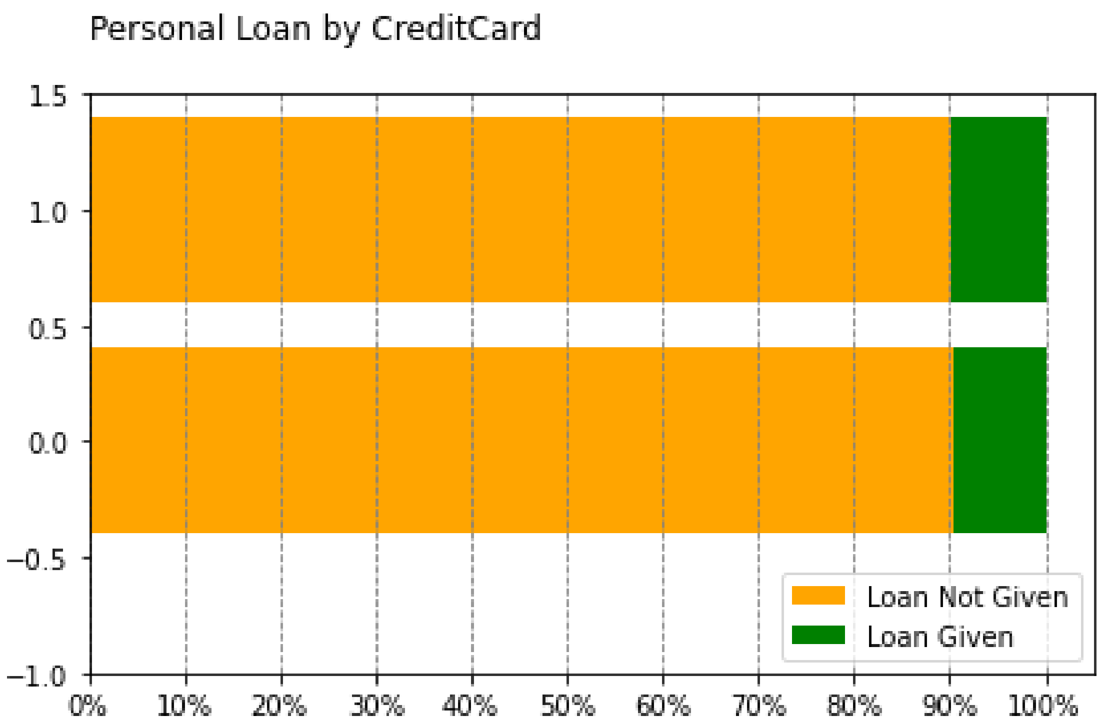
Saving figure Divided Bar Diagram of CD Account



Saving figure Divided Bar Diagram of Online



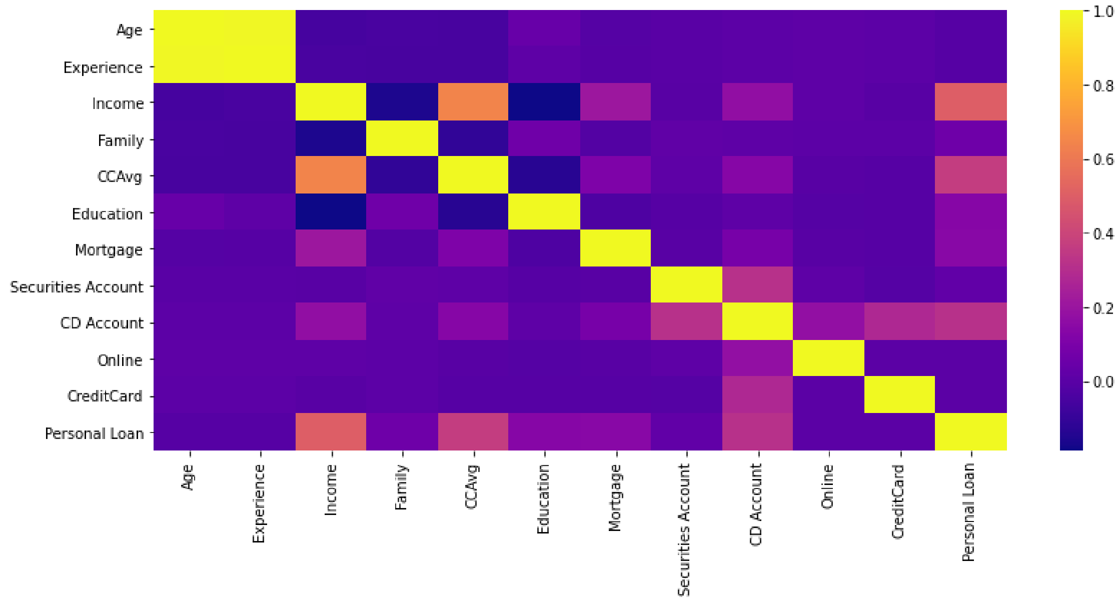
Saving figure Divided Bar Diagram of CreditCard



Correlation Matrix

[609]:
```
corr_matrix=data.corr()
plt.figure(figsize=(12,6))
sns.heatmap(corr_matrix,cmap='plasma')
save_fig("Correlation Matrix Plot")
plt.show()
```

Saving figure Correlation Matrix Plot



Kolmogorom Smirnov Test

[610]:
```
from scipy import stats

for var in var1:
    df_1=data.loc[data['Personal Loan']==0,var]
    df_2=data.loc[data['Personal Loan']==1,var]
    test=stats.ks_2samp(df_1, df_2)
    p_value=round(test[1],6)
    print(p_value)
    if (p_value<0.01):
        print(f"Personal Loan depends on {var}")
    else:
        print(f"Personal Loan does not depend on {var}")
```

```
0.473165
Personal Loan does not depend on Age
0.480255
Personal Loan does not depend on Experience
0.0
Personal Loan depends on Income
0.0
Personal Loan depends on CCAvg
0.0
Personal Loan depends on Mortgage
```

1.6 Data Preprocessing

[973]:
```python
unique_data=data.copy()
```

[974]:
```python
unique_data[(unique_data["Experience"]< 0)].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 52 entries, 89 to 4957
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 52 non-null     int64
 1   Experience          52 non-null     int64
 2   Income              52 non-null     int64
 3   Family              52 non-null     int64
 4   CCAvg               52 non-null     float64
 5   Education           52 non-null     int64
 6   Mortgage            52 non-null     int64
 7   Securities Account  52 non-null     int64
 8   CD Account          52 non-null     int64
 9   Online              52 non-null     int64
 10  CreditCard          52 non-null     int64
 11  Personal Loan       52 non-null     int64
dtypes: float64(1), int64(11)
memory usage: 5.3 KB
```

[976]:
```python
unique_data = unique_data [(unique_data>=0).all(axis=1)]
```

[977]:
```python
unique_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4948 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 4948 non-null   int64
 1   Experience          4948 non-null   int64
 2   Income              4948 non-null   int64
 3   Family              4948 non-null   int64
 4   CCAvg               4948 non-null   float64
 5   Education           4948 non-null   int64
 6   Mortgage            4948 non-null   int64
 7   Securities Account  4948 non-null   int64
 8   CD Account          4948 non-null   int64
 9   Online              4948 non-null   int64
 10  CreditCard          4948 non-null   int64
 11  Personal Loan       4948 non-null   int64
dtypes: float64(1), int64(11)
```

```
memory usage: 502.5 KB
```

[978]:
```python
unique_data.drop_duplicates(keep='first',inplace=True)
```

[979]:
```python
unique_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4935 entries, 0 to 4999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 4935 non-null   int64
 1   Experience          4935 non-null   int64
 2   Income              4935 non-null   int64
 3   Family              4935 non-null   int64
 4   CCAvg               4935 non-null   float64
 5   Education           4935 non-null   int64
 6   Mortgage            4935 non-null   int64
 7   Securities Account  4935 non-null   int64
 8   CD Account          4935 non-null   int64
 9   Online              4935 non-null   int64
 10  CreditCard          4935 non-null   int64
 11  Personal Loan       4935 non-null   int64
dtypes: float64(1), int64(11)
memory usage: 501.2 KB
```

[980]:
```python
unique_data.reset_index(inplace = True)
```

[1001]:
```python
unique_data.head()
```

[1001]:
```
   index  Age  Experience  Income  Family  CCAvg  Education  Mortgage  \
0      0   25           1      49       4    1.6          1         0
1      1   45          19      34       3    1.5          1         0
2      2   39          15      11       1    1.0          1         0
3      3   35           9     100       1    2.7          2         0
4      4   35           8      45       4    1.0          2         0

   Securities Account  CD Account  Online  CreditCard  Personal Loan
0                   1           0       0           0              0
1                   1           0       0           0              0
2                   0           0       0           0              0
3                   0           0       0           0              0
4                   0           0       0           1              0
```

[1002]:
```python
filtered_data=unique_data.copy()
```

[1003]:
```python
filtered_data['Personal Loan'].value_counts()
```

[1003]:
```
0    4455
1     480
Name: Personal Loan, dtype: int64
```

[1004]:
```
filtered_data['Personal Loan']=filtered_data['Personal Loan'].apply(lambda x:
                                                                    "Loan␣
↪Given" if x==1 else "Loan Not Given" )
```

[1005]:
```
filtered_data['Personal Loan'].value_counts()
```

[1005]:
```
Loan Not Given    4455
Loan Given         480
Name: Personal Loan, dtype: int64
```

1.7 Data Transformation

[1006]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names]
```

[1007]:
```
class MostFrequentImputer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.most_frequent_ = pd.Series([X[c].value_counts().index[0] for c in␣
↪X],
                                        index=X.columns)
        return self
    def transform(self, X, y=None):
        return X.fillna(self.most_frequent_)
```

[1008]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
```

1.8 Test Train Segmentation

[1009]:
```
final_data=unique_data.copy()
```

Here we divite the processed dataset into train and test dataset. we use 1. 70% data for training 2. 15% data for validation 3. 15% data for testing

[1010]:

```
from sklearn.model_selection import StratifiedShuffleSplit
split=StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=42)


for train_index,test_index in split.split(final_data,final_data["Personal␣
↪Loan"]):
        strat_train_set=final_data.loc[train_index]
        strat_test_set=final_data.loc[test_index]

train_data=strat_train_set.copy()
remain_data=strat_test_set.copy()
remain_data.reset_index(inplace = True)


print("Train\n",strat_train_set["Personal Loan"].value_counts()/
↪len(strat_train_set))


split=StratifiedShuffleSplit(n_splits=1,test_size=0.5,random_state=42)
for valid_index,test_index in split.split(remain_data,remain_data["Personal␣
↪Loan"]):
        strat_valid_set = remain_data.loc[valid_index]
        strat_test_set = remain_data.loc[test_index]

print("Valid\n",strat_valid_set["Personal Loan"].value_counts()/
↪len(strat_valid_set))
print("Test\n",strat_test_set["Personal Loan"].value_counts()/
↪len(strat_test_set))
print("Actual\n",final_data["Personal Loan"].value_counts()/len(final_data))


valid_data=strat_valid_set.copy()
test_data=strat_test_set.copy()
```

```
Train
 0    0.902721
1    0.097279
Name: Personal Loan, dtype: float64
Valid
 0    0.902703
1    0.097297
Name: Personal Loan, dtype: float64
Test
 0    0.902834
1    0.097166
Name: Personal Loan, dtype: float64
Actual
```

 0   0.902736
1   0.097264
Name: Personal Loan, dtype: float64

[1011]:
```
train_data.shape
```

[1011]: (3454, 13)

[1012]:
```
test_data.shape
```

[1012]: (741, 14)

[1013]:
```
valid_data.shape
```

[1013]: (740, 14)

1.9 Pipeline with All Attributes

[1014]:
```
num_pipeline_all = Pipeline([
        ("select_numeric", DataFrameSelector(var1)),
        ("imputer", SimpleImputer(strategy="median")),
    ])
```

[1015]:
```
cat_pipeline_all = Pipeline([
        ("select_cat", DataFrameSelector(var2)),
        ("imputer", MostFrequentImputer()),
        ("cat_encoder", OneHotEncoder(sparse=False)),
    ])
```

[1016]:
```
from sklearn.pipeline import FeatureUnion
preprocess_pipeline_all = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline_all),
        ("cat_pipeline", cat_pipeline_all),
    ])
```

[1017]:
```
X_train_all = preprocess_pipeline_all.fit_transform(train_data)
y_train_all = train_data["Personal Loan"]
```

[1018]:
```
X_train_all.shape
```

[1018]: (3454, 20)

[1019]:
```
y_train_all.shape
```

[1019]: (3454,)

[1020]:
```
X_test_all = preprocess_pipeline_all.transform(test_data)
y_test_all = test_data["Personal Loan"]
```

[1021]:
```
X_test_all.shape
```

[1021]: (741, 20)

[1022]:
```
y_test_all.shape
```

[1022]: (741,)

[1023]:
```
X_valid_all = preprocess_pipeline_all.transform(valid_data)
y_valid_all = valid_data["Personal Loan"]
```

[1024]:
```
X_valid_all.shape
```

[1024]: (740, 20)

[1025]:
```
y_valid_all.shape
```

[1025]: (740,)

1.10 Pipeline with Selected Attributes

[1026]:
```
num_pipeline_selected = Pipeline([
        ("select_numeric", DataFrameSelector(["Income", "CCAvg", "Mortgage"])),
        ("imputer", SimpleImputer(strategy="median")),
    ])
```

[1027]:
```
cat_pipeline_selected = Pipeline([
        ("select_cat", DataFrameSelector(["Education", "CD Account",␣
 ↪"Family"])),
        ("imputer", MostFrequentImputer()),
        ("cat_encoder", OneHotEncoder(sparse=False)),
    ])
```

[1028]:
```
from sklearn.pipeline import FeatureUnion
preprocess_pipeline_selected = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline_selected),
        ("cat_pipeline", cat_pipeline_selected),
    ])
```

[1029]:
```
X_train_selected = preprocess_pipeline_selected.fit_transform(train_data)
y_train_selected = train_data["Personal Loan"]
```

[1030]:
```
X_train_selected.shape
```

[1030]: (3454, 12)

```
[1031]: y_train_selected.shape
```

[1031]: (3454,)

```
[1032]: X_test_selected = preprocess_pipeline_selected.transform(test_data)
        y_test_selected = test_data["Personal Loan"]
```

```
[1033]: X_test_selected.shape
```

[1033]: (741, 12)

```
[1034]: y_test_selected.shape
```

[1034]: (741,)

```
[1035]: X_valid_selected = preprocess_pipeline_selected.transform(valid_data)
        y_valid_selected = valid_data["Personal Loan"]
```

```
[1036]: X_valid_selected.shape
```

[1036]: (740, 12)

```
[1037]: y_valid_selected.shape
```

[1037]: (740,)

2 Classifier Training

```
[1038]: from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import cross_val_predict
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import precision_score, recall_score
        from sklearn.metrics import f1_score
        from sklearn.metrics import roc_curve
        from sklearn.metrics import roc_auc_score
```

```
[1039]: def plot_roc_curve(y_train,y_scores, label=None):

            fpr, tpr, thresholds = roc_curve(y_train,y_scores)
            print(round((roc_auc_score(y_train, y_scores)),rounding_factor))

            plt.plot(fpr, tpr, linewidth=2, label=label)
            plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
            plt.axis([0, 1, 0, 1])
```

```
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16)
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)
    plt.grid(True)
    plt.show()
```

[1040]:

```
def plot_cf_matrix(cf_matrix):
    group_names = ['True Neg','False Pos','False Neg','True Pos']
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.
 ↪flatten()/np.sum(cf_matrix)]

    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
    ax.set_title('Confusion Matrix')
    ax.set_xlabel('Predicted Values')
    ax.set_ylabel('Actual Values ')
    ## Ticket labels - List must be in alphabetical order
    ax.xaxis.set_ticklabels(['False','True'])
    ax.yaxis.set_ticklabels(['False','True'])
    ## Display the visualization of the Confusion Matrix.
    plt.show()
```

[1041]:

```
def print_classification_report(y_train,y_train_pred):
    print()
    print('=============Confusion Matrix =============')
    print(confusion_matrix(y_train, y_train_pred))


    y_train_perfect_predictions = y_train #Perfect Prediction
    print()
    print('Perfect Prediction If Done')
    print(confusion_matrix(y_train, y_train_perfect_predictions))


    print()
    print("=============Sumarry Measures=============")
    print('Precision Score = ',round((precision_score(y_train,␣
 ↪y_train_pred)),rounding_factor))
    print('Recall = ', round((recall_score(y_train,␣
 ↪y_train_pred)),rounding_factor))
    print('F1 Value = ', round((f1_score(y_train,␣
 ↪y_train_pred)),rounding_factor))
```

3 Support Vector Machine

3.1 SVM (Polynomial Kernel)

[1042]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

svm_clf_poly =␣
↪make_pipeline(StandardScaler(),SVC(gamma="auto",class_weight="balanced",C=3,
                                            kernel="poly",probability=True))
```

3.2 Full Model

3.3 Training

[1043]:

```
svm_clf_poly.fit(X_train_all, y_train_all)

svm_scores = cross_val_score(svm_clf_poly, X_train_all, y_train_all, cv=6)
print(svm_scores.mean())
```

```
0.9739437399355877
```

Confusion Matrix for SVM: Train Data

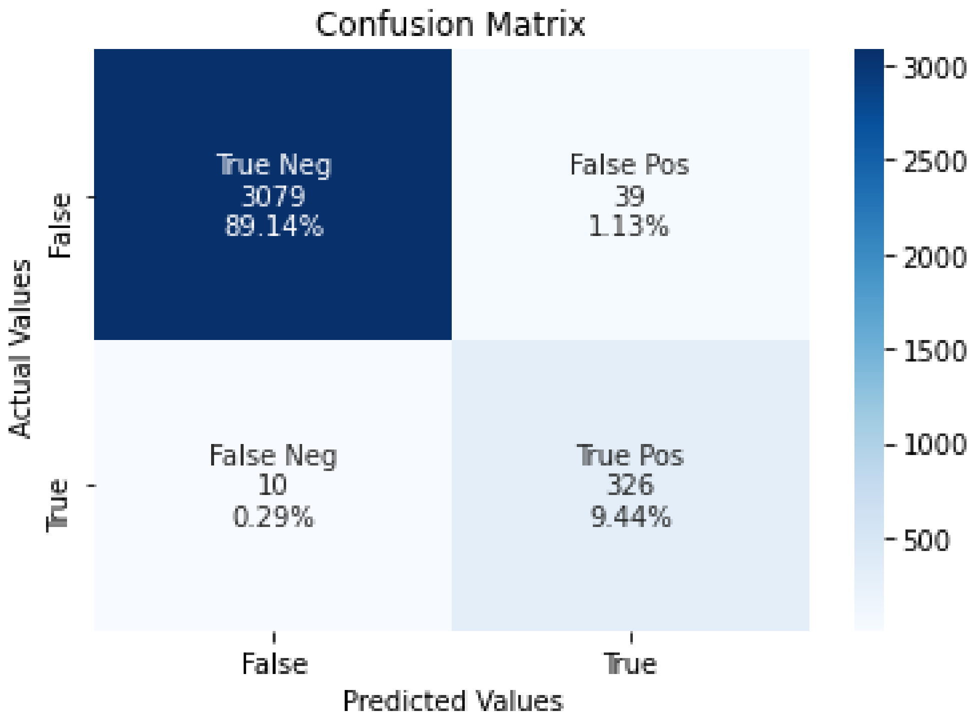
[1044]:

```
y_train_pred = svm_clf_poly.predict( X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3079   39]
 [  10  326]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8932
Recall =  0.9702
F1 Value =  0.9301
```

[1045]:

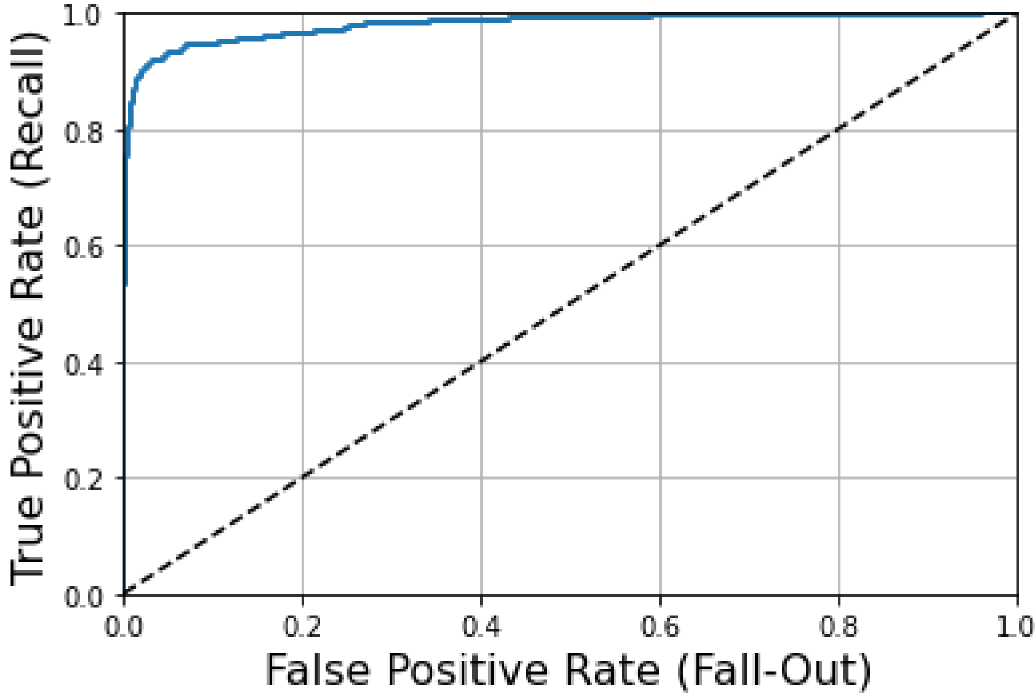
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1046]:
```
y_probas_svm = cross_val_predict(svm_clf_poly, X_train_all,␣
↪y_train_all,method="predict_proba")
y_scores_svm = y_probas_svm[:, 1] # score = proba of positive class          ␣
↪

plot_roc_curve(y_train_all ,y_scores_svm)
save_fig("ROC for SVM Full Model Poly Kernel")
```

```
0.9799
```




```
Saving figure ROC for SVM Full Model Poly Kernel
```

```
<Figure size 432x288 with 0 Axes>
```

3.4 Performance on Validation Set

[1047]:

```
y_valid_pred = svm_clf_poly.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[656  12]
 [ 12  60]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8333
Recall =  0.8333
F1 Value =  0.8333
```

3.5 Model with Selected Attributes

3.6 Training

[1048]:
```
svm_clf_poly.fit(X_train_selected, y_train_selected)

svm_scores = cross_val_score(svm_clf_poly, X_train_selected, y_train_selected,␣
↪cv=6)
print(svm_scores.mean())
```

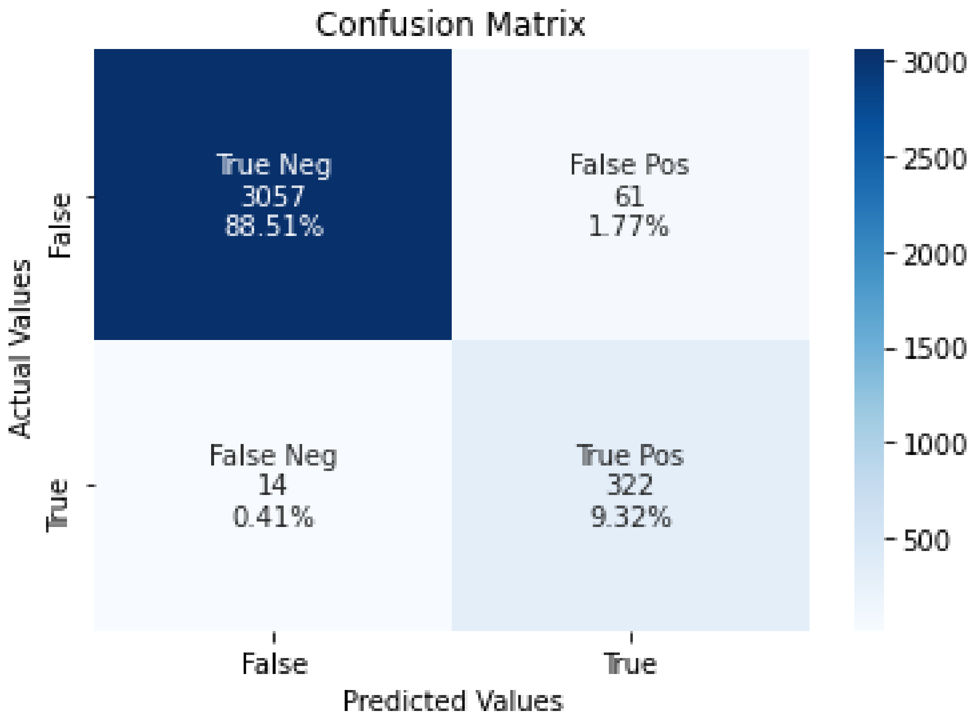
```
0.9695974235104671
```

Confusion Matrix

[1049]:
```
y_train_pred = svm_clf_poly.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3057   61]
 [  14  322]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8407
Recall =  0.9583
F1 Value =  0.8957
```

[1050]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```

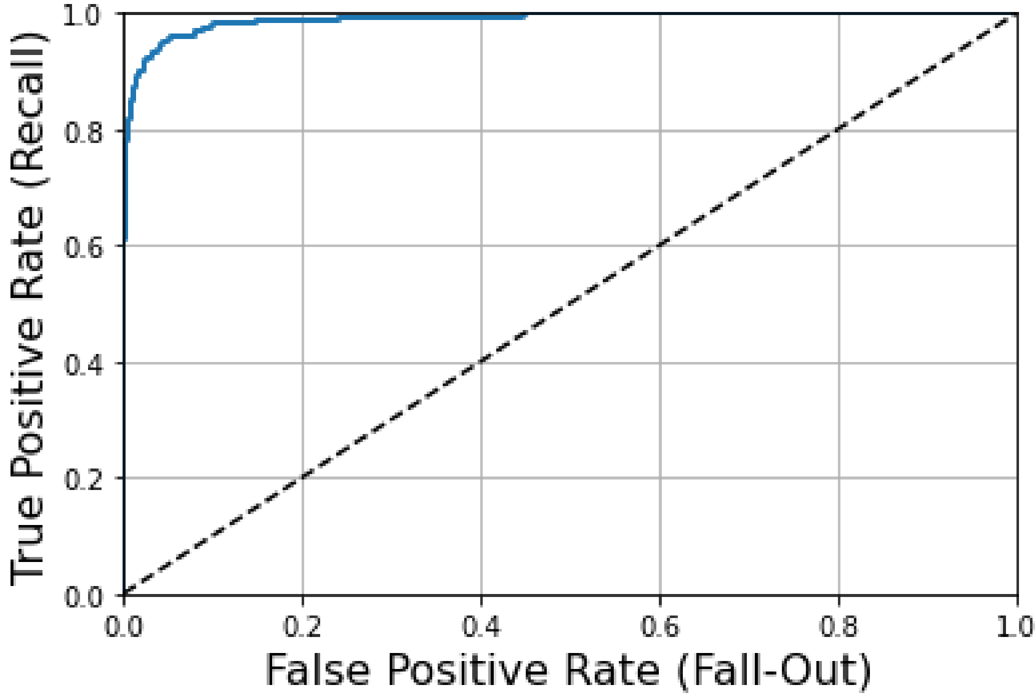


ROC Curve

[1051]:

```
y_probas_svm = cross_val_predict(svm_clf_poly, X_train_selected,␣
↪y_train_selected,method="predict_proba")
y_scores_svm = y_probas_svm[:, 1] # score = proba of positive class      ␣
↪

plot_roc_curve(y_train_selected,y_scores_svm)
save_fig("ROC for SVM Partial Model Poly Kernel")
```

0.9895




```
Saving figure ROC for SVM Partial Model Poly Kernel

<Figure size 432x288 with 0 Axes>
```

3.7 Performance on Validation Set

[1052]:
```
y_valid_pred = svm_clf_poly.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[655  13]
 [  5  67]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8375
Recall =  0.9306
F1 Value =  0.8816
```

3.8 SVM (Linear Kernel)

[1053]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

svm_clf_lin = make_pipeline(StandardScaler(),SVC(gamma="auto",class_weight={0:
↪1,1:2},C=2,
                                            kernel="linear",probability=True))
```

3.9 Full Model

3.10 Training

[1054]:
```
svm_clf_lin.fit(X_train_all, y_train_all)

svm_scores = cross_val_score(svm_clf_lin, X_train_all, y_train_all, cv=6)
print(svm_scores.mean())
```

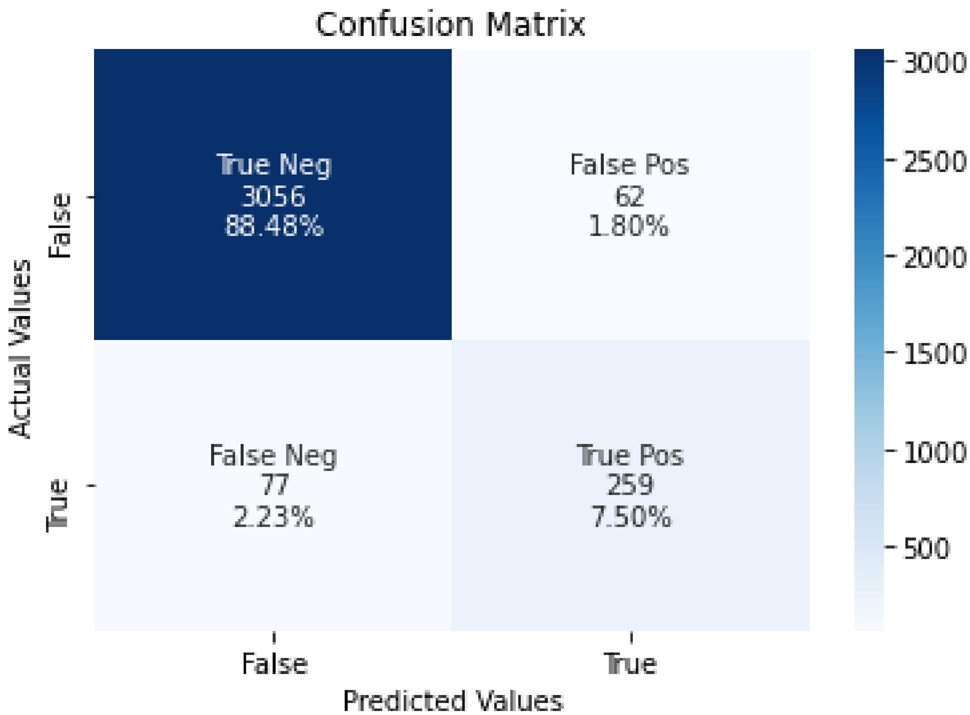
```
0.9571568035426732
```

Confusion Matrix

[1055]:
```
y_train_pred = svm_clf_lin.predict( X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3056   62]
 [  77  259]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8069
Recall =  0.7708
F1 Value =  0.7884
```

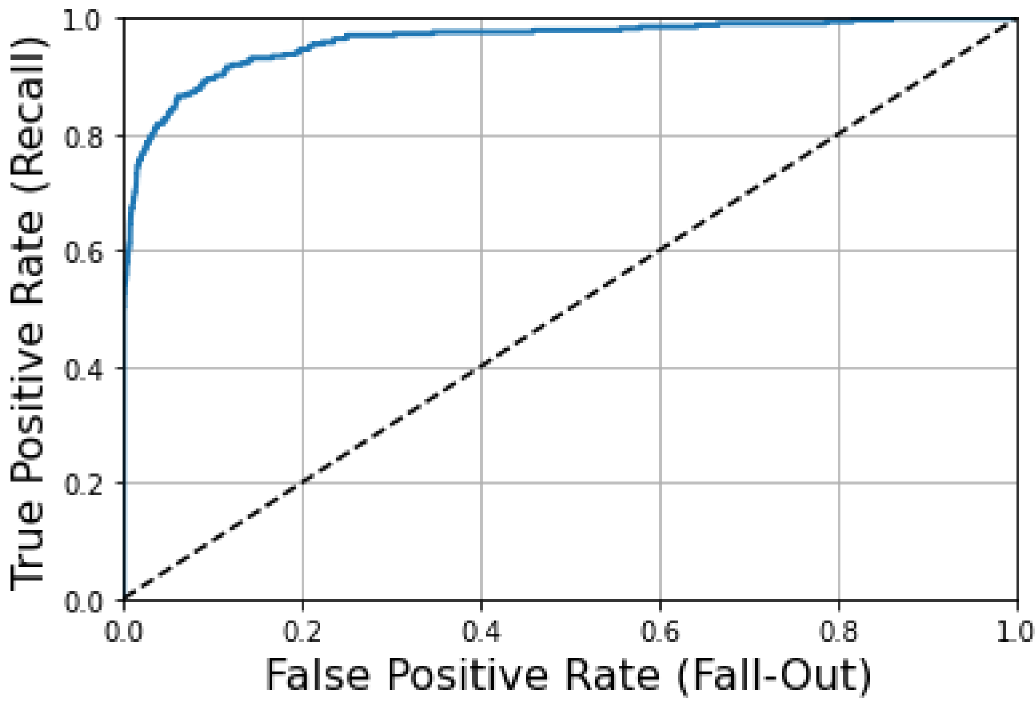
[1056]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1057]:
```
y_probas_svm = cross_val_predict(svm_clf_lin, X_train_all,␣
↪y_train_all,method="predict_proba")
y_scores_svm = y_probas_svm[:, 1] # score = proba of positive class          ␣
↪

plot_roc_curve(y_train_all ,y_scores_svm)
save_fig("ROC for SVM Full Model Linear Kernel")
```

```
0.9607
```



Saving figure ROC for SVM Full Model Linear Kernel

<Figure size 432x288 with 0 Axes>

3.11 Performance on Validation Set

[1058]:
```
y_valid_pred = svm_clf_lin.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[654  14]
 [ 21  51]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7846
Recall =  0.7083
F1 Value =  0.7445
```

3.12 SVM (RBF Kernel)

[1059]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

svm_clf_rbf =␣
↪make_pipeline(StandardScaler(),SVC(gamma="auto",class_weight="balanced",C=2,
                                     kernel="rbf",probability=True))
```

3.13 Full Model

3.14 Training

[1060]:
```
svm_clf_rbf.fit(X_train_all, y_train_all)

svm_scores = cross_val_score(svm_clf_rbf, X_train_all, y_train_all, cv=6)
print(svm_scores.mean())
```

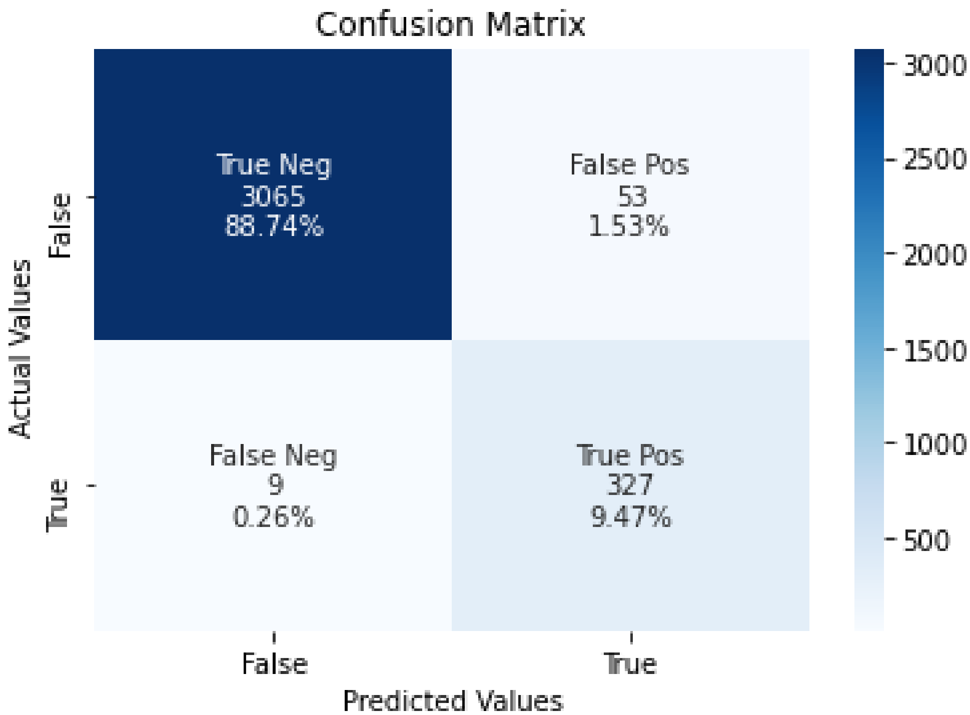
```
0.9745204307568437
```

Confusion Matrix

[1061]:
```
y_train_pred = svm_clf_rbf.predict( X_train_all)
print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3065   53]
 [   9  327]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8605
Recall =  0.9732
F1 Value =  0.9134
```

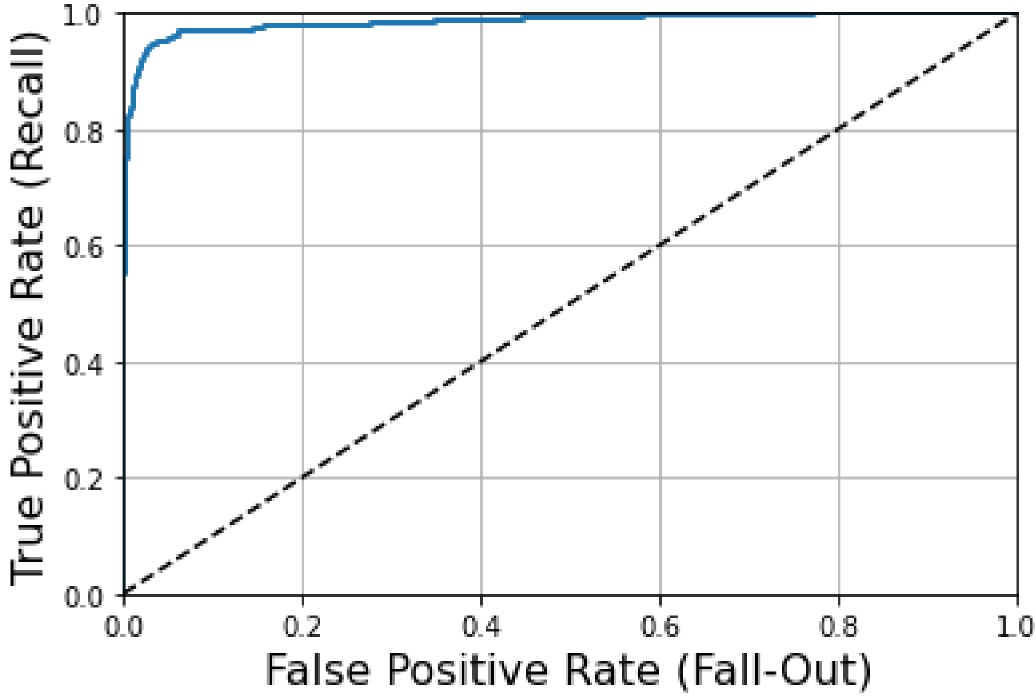
[1062]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1063]:
```
y_probas_svm = cross_val_predict(svm_clf_rbf, X_train_all,␣
↪y_train_all,method="predict_proba")
y_scores_svm = y_probas_svm[:, 1] # score = proba of positive class       ␣
↪

plot_roc_curve(y_train_all ,y_scores_svm)
save_fig("ROC for SVM Full Model RBF Kernel")
```

0.9849




```
Saving figure ROC for SVM Full Model RBF Kernel
```

```
<Figure size 432x288 with 0 Axes>
```

3.15 Performance on Validation Set

[1064]:
```
y_valid_pred = svm_clf_rbf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[647  21]
 [  8  64]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7529
Recall =  0.8889
F1 Value =  0.8153
```

3.16 SVM (Sigmoid Kernel)

[1065]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

svm_clf_sig =␣
↪make_pipeline(StandardScaler(),SVC(gamma="auto",class_weight="balanced",C=2,
                                                    kernel="sigmoid",probability=True))
```

3.17 Full Model

3.18 Training

[1066]:
```
svm_clf_sig.fit(X_train_all, y_train_all)

svm_scores = cross_val_score(svm_clf_sig , X_train_all, y_train_all, cv=10)
print(svm_scores.mean())
```

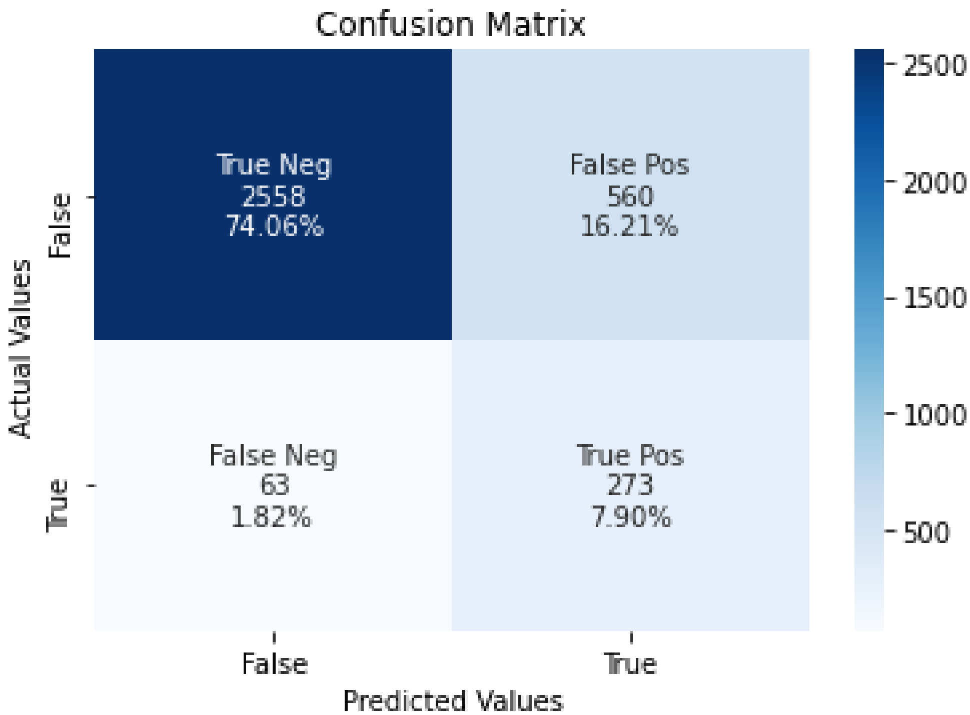
```
0.8196439641450951
```

Confusion Matrix

[1067]:
```
y_train_pred = svm_clf_sig.predict( X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[2558  560]
 [  63  273]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.3277
Recall =  0.8125
F1 Value =  0.4671
```

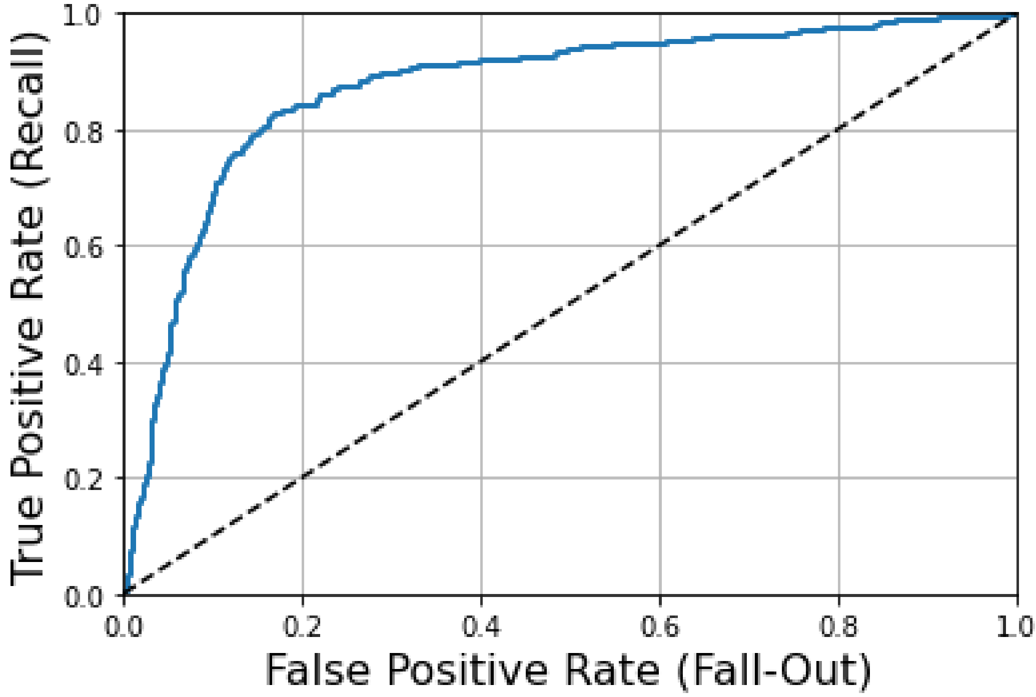
[1068]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1069]:
```
y_probas_svm = cross_val_predict(svm_clf_sig, X_train_all,␣
↪y_train_all,method="predict_proba")

y_scores_svm = y_probas_svm[:, 1] # score = proba of positive class           ␣
↪

plot_roc_curve(y_train_all ,y_scores_svm)
save_fig("ROC for SVM Full Model RBF Kernel")
```

```
0.8705
```




```
Saving figure ROC for SVM Full Model RBF Kernel
```

```
<Figure size 432x288 with 0 Axes>
```

3.19 Performance on Validation Set

[1070]:
```
y_valid_pred = svm_clf_sig.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[556 112]
 [  9  63]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.36
Recall =  0.875
F1 Value =  0.5101
```

3.20 Ensemble (Random Forest)

[1071]:
```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=100,␣
↪random_state=42,min_samples_split=8,
                                              ␣
↪min_samples_leaf=4,class_weight="balanced",oob_score=True)
```

3.21 Full Model

3.22 Training

[1072]:
```
forest_clf.fit(X_train_all, y_train_all)
forest_scores = cross_val_score(forest_clf, X_train_all, y_train_all, cv=6)
print(forest_scores.mean())
```

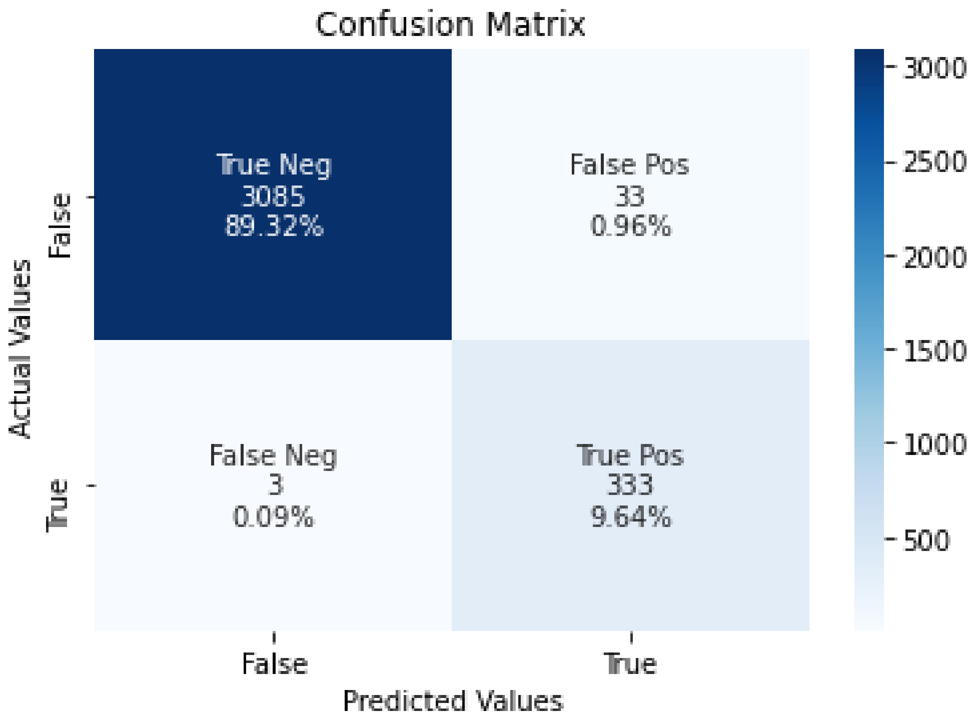
```
0.9803099838969405
```

Confusion Matrix for Random Forest: Train Data

[1073]:
```
y_train_pred = forest_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3085   33]
 [   3  333]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9098
Recall =  0.9911
F1 Value =  0.9487
```

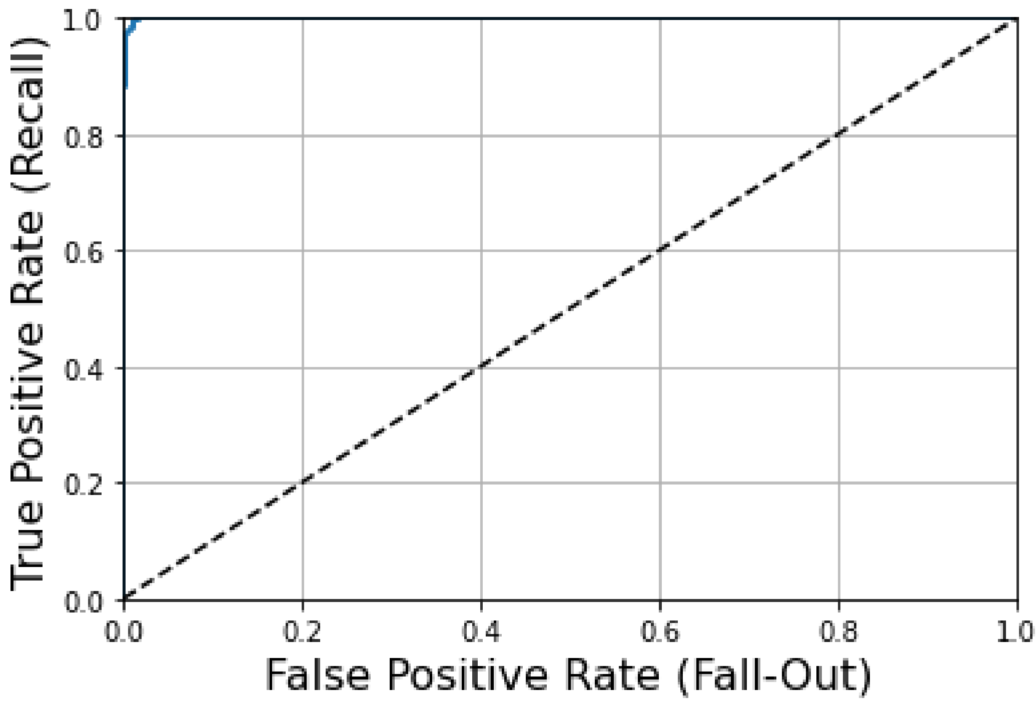
[1074]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1075]:
```
y_probas_forest = forest_clf.predict_proba( X_train_all)

y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class

plot_roc_curve(y_train_all ,y_scores_forest)
save_fig("ROC for Random Forest Full Model")
```

```
0.9995
```




```
Saving figure ROC for Random Forest Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.23 Performance on Validation Set

[1076]:
```
y_valid_pred = forest_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[660   8]
 [  2  70]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8974
Recall =  0.9722
F1 Value =  0.9333
```

3.24 Model with Selected Attributes

3.25 Training

[1077]:
```
forest_clf.fit(X_train_selected, y_train_selected)
forest_scores = cross_val_score(forest_clf, X_train_selected, y_train_selected,
    cv=6)
print(forest_scores.mean())
```

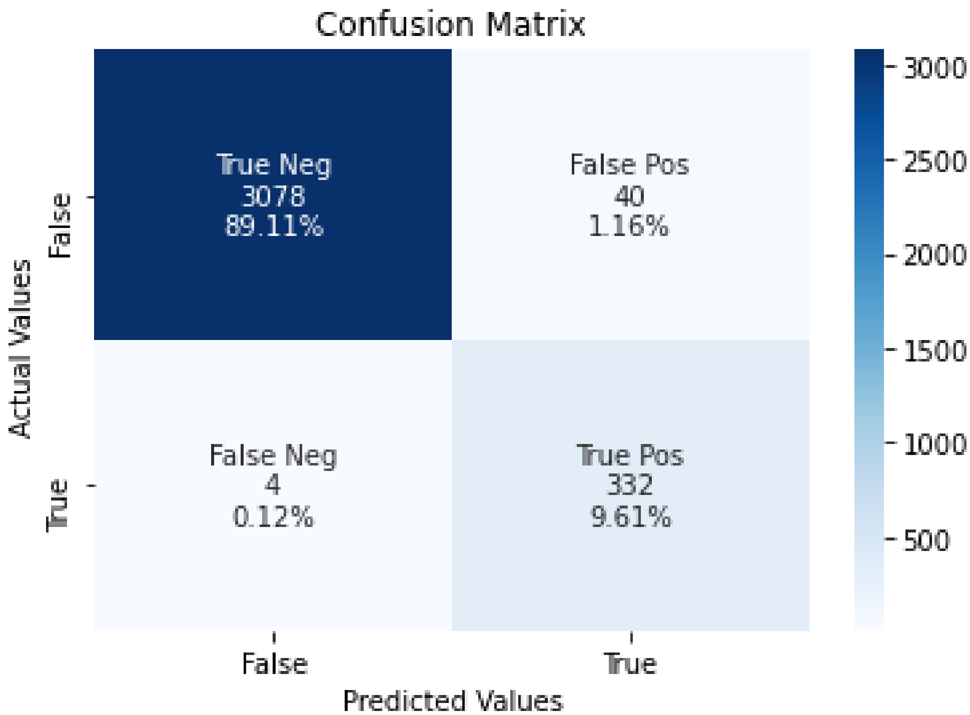
```
0.977992652979066
```

Confusion Matrix

[1078]:
```
y_train_pred = forest_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3078   40]
 [   4  332]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8925
Recall =  0.9881
F1 Value =  0.9379
```

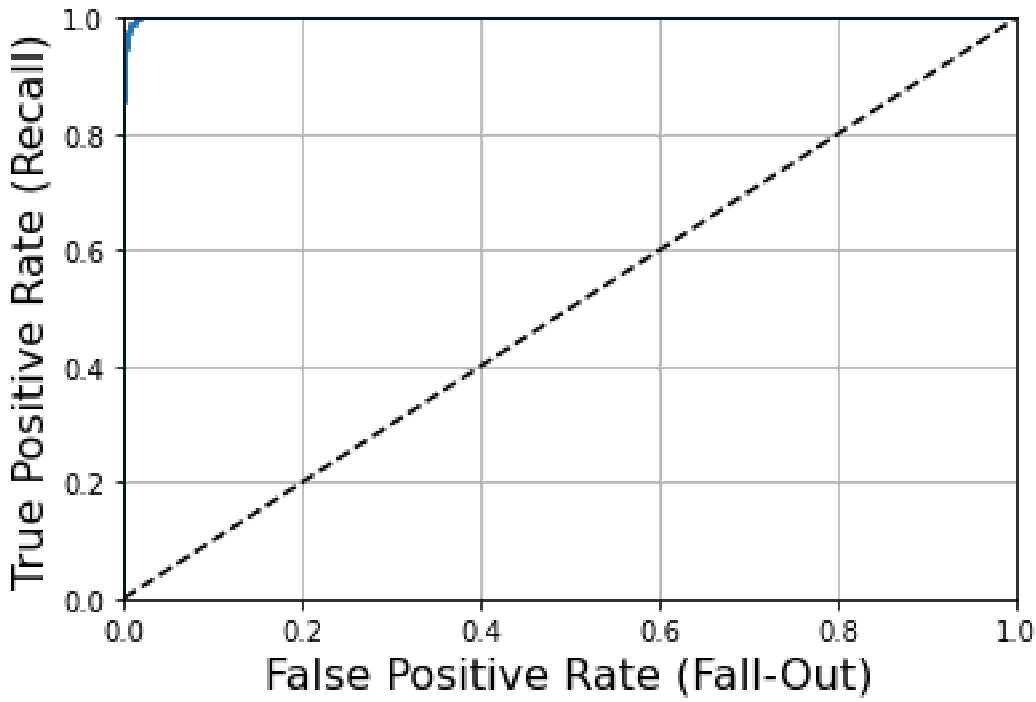
[1079]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1080]:
```
y_probas_forest = forest_clf.predict_proba(X_train_selected)

y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class

plot_roc_curve(y_train_selected ,y_scores_forest)
save_fig("ROC for Random Forest Partial Model")
```

0.9993




```
Saving figure ROC for Random Forest Partial Model

<Figure size 432x288 with 0 Axes>
```

3.26 Performance on Validation Set

[1081]:
```
y_valid_pred = forest_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[657  11]
 [  1  71]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8659
Recall =  0.9861
F1 Value =  0.9221
```

3.27 Ensemble (Bagging)

[1082]:
```
from sklearn.ensemble import BaggingClassifier

bagging_clf = BaggingClassifier(n_estimators=5, random_state=42)
```

3.28 Full Model

3.29 Training

[1083]:
```
bagging_clf.fit(X_train_all, y_train_all)
bagging_scores = cross_val_score(bagging_clf, X_train_all, y_train_all, cv=6)
print(bagging_scores.mean())
```

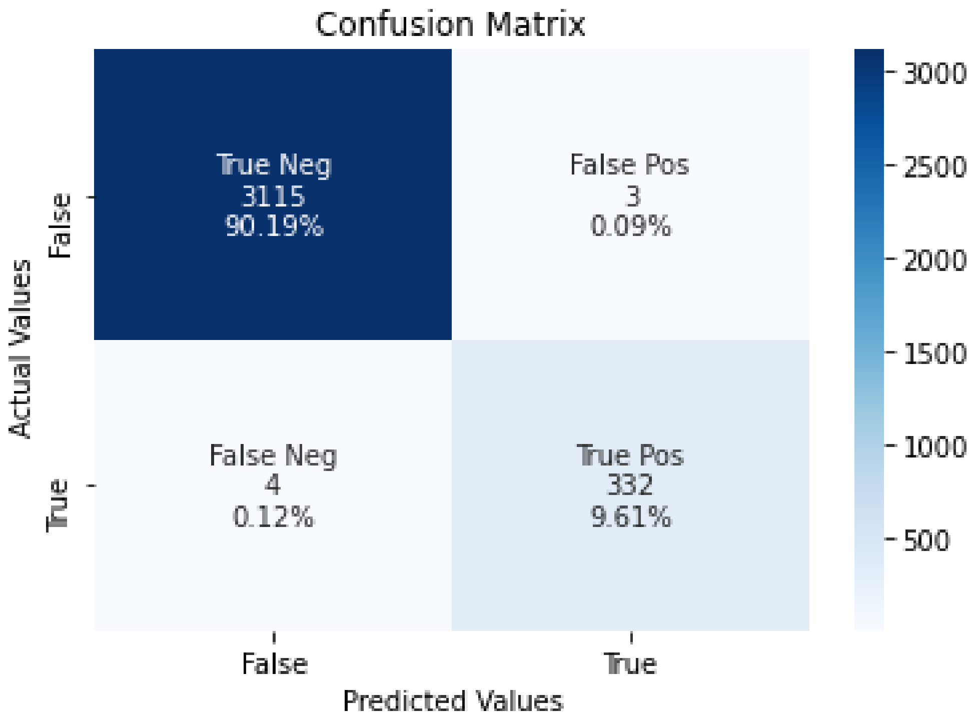
```
0.9823399758454107
```

Confusion Matrix

[1084]:
```
y_train_pred = bagging_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3115    3]
 [   4  332]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.991
Recall =  0.9881
F1 Value =  0.9896
```

[1085]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1086]:
```
y_probas_bagging = bagging_clf.predict_proba( X_train_all)
y_scores_bagging = y_probas_bagging [:,-1]

plot_roc_curve(y_train_all ,y_scores_bagging)
save_fig("ROC for Bagging Full Model")
```

```
0.9999
```


```
Saving figure ROC for Bagging Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.30 Performance on Validation Set

[1087]:
```
y_valid_pred = bagging_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[666   2]
 [  6  66]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9706
Recall =  0.9167
F1 Value =  0.9429
```

3.31 Model with Selected Attributes

3.32 Training

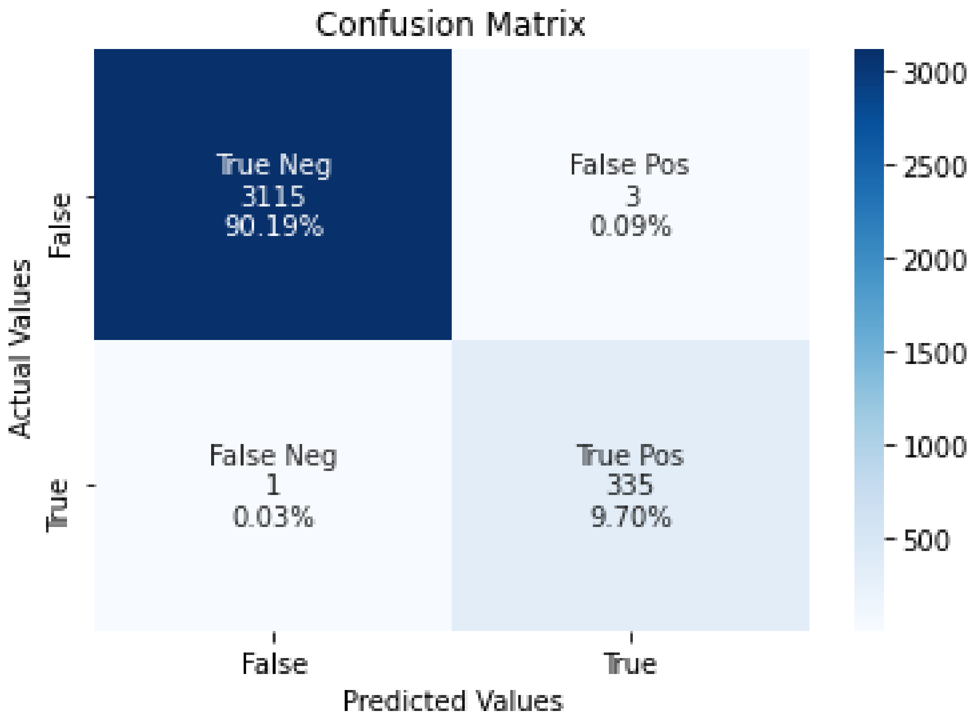
[1088]:
```
bagging_clf.fit(X_train_selected, y_train_selected)
bagging_scores = cross_val_score(bagging_clf, X_train_selected,␣
↪y_train_selected, cv=6)
print(bagging_scores.mean())
```

```
0.9829176731078905
```

[1089]:
```
y_train_pred = bagging_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3115    3]
 [   1  335]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9911
Recall =  0.997
F1 Value =  0.9941
```

[1090]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1091]:
```
y_probas_bagging = bagging_clf.predict_proba( X_train_selected)
y_scores_bagging = y_probas_bagging [:,-1]

plot_roc_curve(y_train_selected ,y_scores_bagging)
save_fig("ROC for Bagging Partial Model")
```

```
0.9999
```


```
Saving figure ROC for Bagging Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.33 Performance on Validation Set

[1092]:
```python
y_valid_pred = bagging_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[664   4]
 [  7  65]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.942
Recall =  0.9028
F1 Value =  0.922
```

3.34 Ensemble (Boosting)

3.35 Gradient Boosting

[1093]:
```
from sklearn.ensemble import GradientBoostingClassifier

gradient_boosting_clf = GradientBoostingClassifier(n_estimators=50,␣
↪random_state=42,learning_rate = 1,
                                                    ␣
↪min_samples_split=10,min_samples_leaf=4,max_depth=2)
```

3.36 Full Model

3.37 Training

[1094]:
```
gradient_boosting_clf.fit(X_train_all, y_train_all)
gradient_boosting_scores = cross_val_score(gradient_boosting_clf, X_train_all,␣
↪y_train_all, cv=6)
print(gradient_boosting_scores.mean())
```

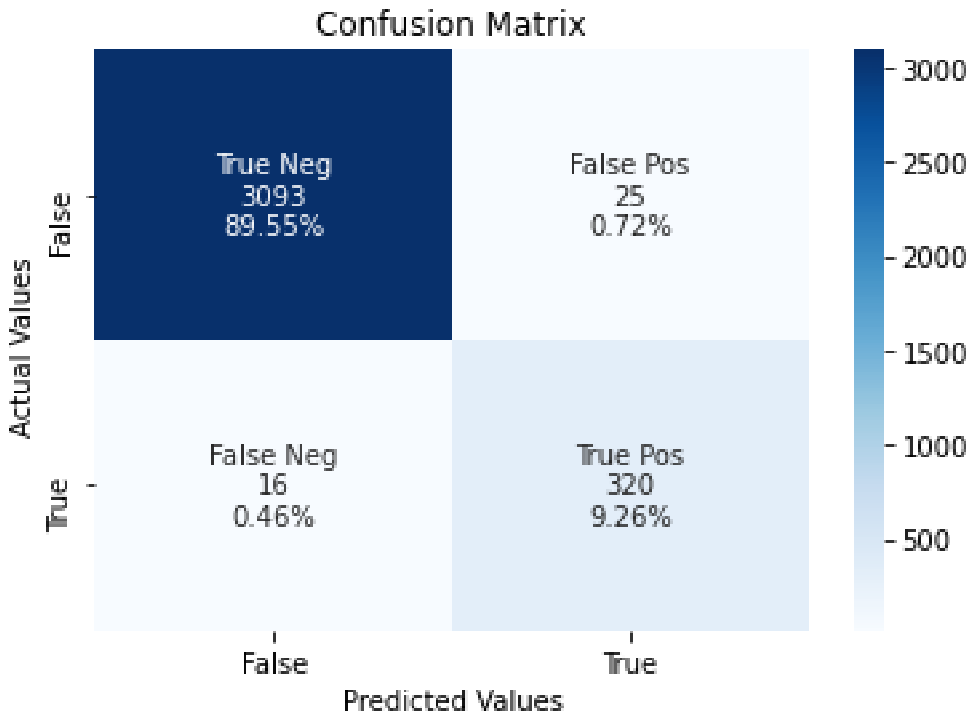
```
0.9785738727858293
```

Confusion Matrix

[1095]:
```
y_train_pred = gradient_boosting_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3093   25]
 [  16  320]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9275
Recall =  0.9524
F1 Value =  0.9398
```

[1096]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1097]:
```
y_probas_boosting = gradient_boosting_clf.predict_proba( X_train_all)
y_scores_boosting = y_probas_boosting [:,-1]

plot_roc_curve(y_train_all ,y_scores_boosting)
save_fig("ROC for Gradient Boosting Full Model")
```

0.9928


```
Saving figure ROC for Gradient Boosting Full Model

<Figure size 432x288 with 0 Axes>
```

3.38 Performance on Validation Set

[1098]:
```
y_valid_pred = gradient_boosting_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[656  12]
 [  6  66]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8462
Recall =  0.9167
F1 Value =  0.88
```

3.39 Model with Selected Attributes

3.40 Training

[1099]:
```
gradient_boosting_clf.fit(X_train_selected, y_train_selected)
gradient_boosting_scores = cross_val_score(gradient_boosting_clf,␣
 ↪X_train_selected, y_train_selected, cv=6)
print(gradient_boosting_scores.mean())
```

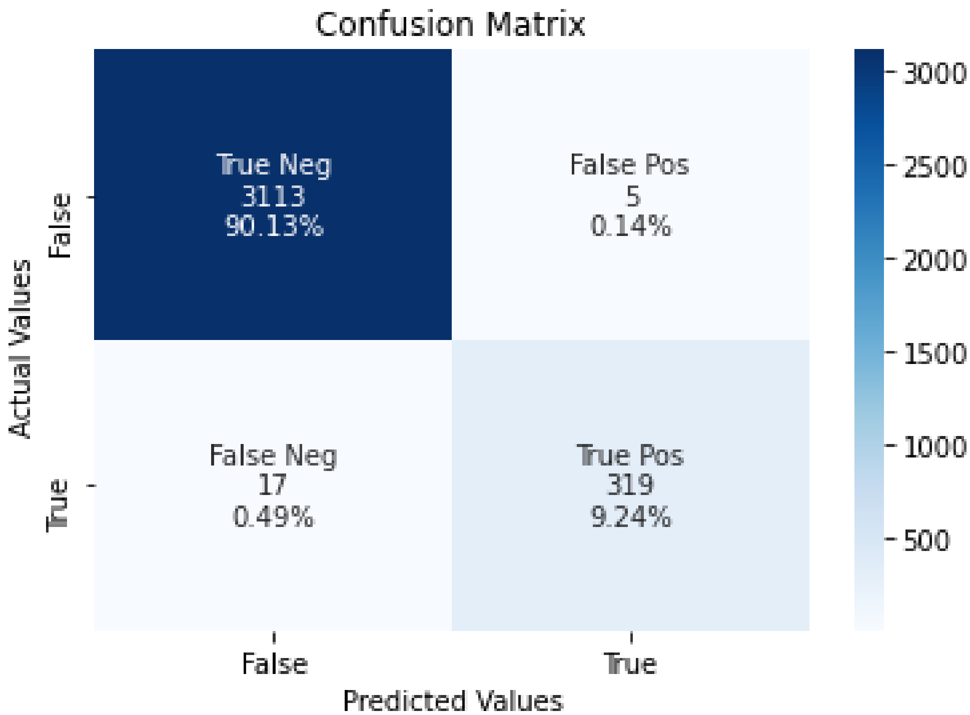
```
0.9785688405797103
```

Confusion Matrix

[1100]:
```
y_train_pred = gradient_boosting_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3113    5]
 [  17  319]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9846
Recall =  0.9494
F1 Value =  0.9667
```

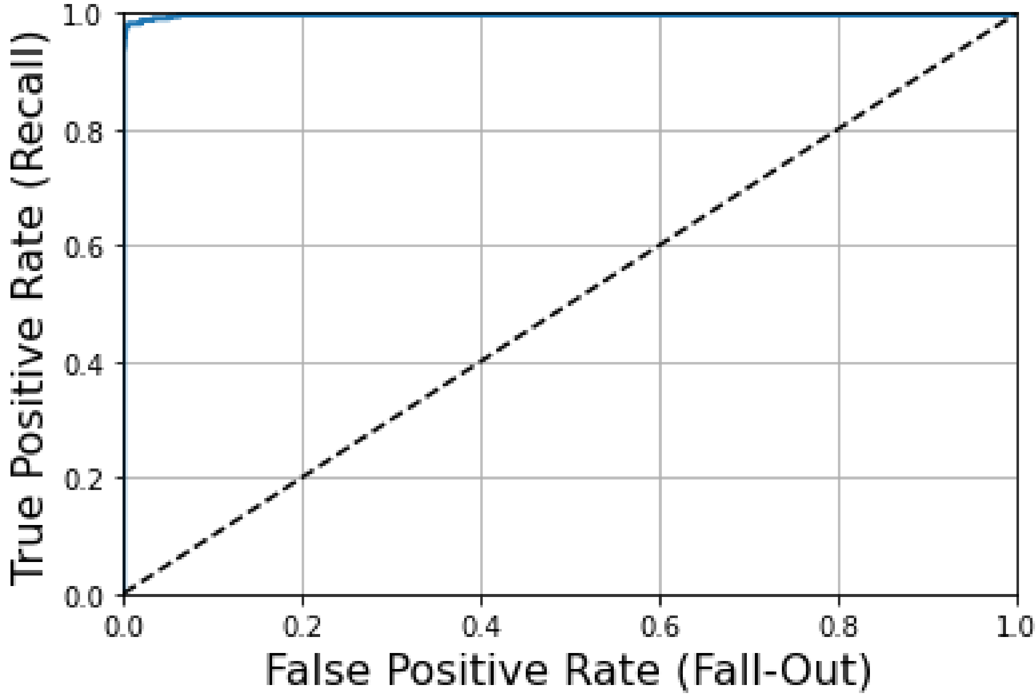
[1101]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1102]:
```
y_probas_boosting = gradient_boosting_clf.predict_proba( X_train_selected)
y_scores_boosting = y_probas_boosting [:,-1]

plot_roc_curve(y_train_all ,y_scores_boosting)
save_fig("ROC for Gradient Boosting Partial Model")
```

0.9964




```
Saving figure ROC for Gradient Boosting Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.41 Performance on Validation Set

[1103]:
```
y_valid_pred = gradient_boosting_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[664   4]
 [  5  67]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9437
Recall =  0.9306
F1 Value =  0.9371
```

3.42 Ada Boost

[1104]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_boosting_clf = AdaBoostClassifier(n_estimators=50,␣
 ↪random_state=42,learning_rate = 1)
```

3.43 Full Model

3.44 Training

[1105]:
```
ada_boosting_clf.fit(X_train_all, y_train_all)
ada_boosting_scores = cross_val_score(ada_boosting_clf, X_train_all,␣
 ↪y_train_all, cv=6)
print(ada_boosting_scores.mean())
```

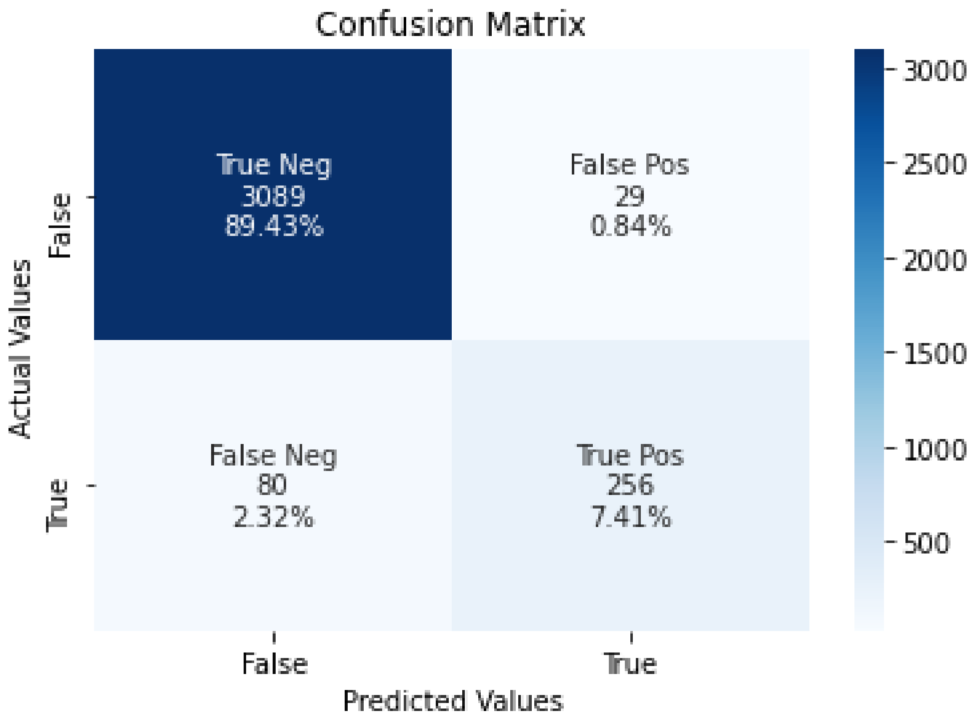
```
0.9646794484702094
```

Confusion Matrix

[1106]:
```
y_train_pred = ada_boosting_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3089   29]
 [  80  256]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8982
Recall =  0.7619
F1 Value =  0.8245
```

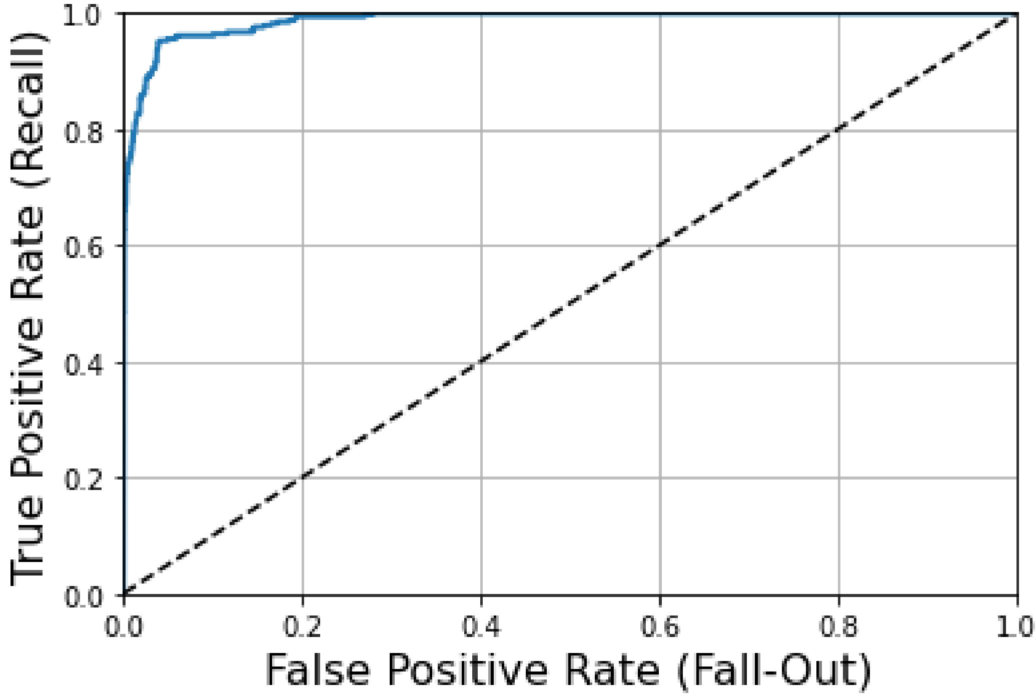
[1107]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1108]:
```
y_probas_boosting = ada_boosting_clf.predict_proba( X_train_all)
y_scores_boosting = y_probas_boosting [:,-1]

plot_roc_curve(y_train_all ,y_scores_boosting)
save_fig("ROC for Ada Boosting Full Model")
```

```
0.9877
```




```
Saving figure ROC for Ada Boosting Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.45 Performance on Validation Set

[1109]:
```
y_valid_pred = ada_boosting_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[665   3]
 [ 18  54]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9474
Recall =  0.75
F1 Value =  0.8372
```

3.46 Model with Selected Attributes

3.47 Training

[1110]:
```
ada_boosting_clf.fit(X_train_selected, y_train_selected)
ada_boosting_scores = cross_val_score(ada_boosting_clf, X_train_selected,␣
 ↪y_train_selected, cv=6)
print(ada_boosting_scores.mean())
```

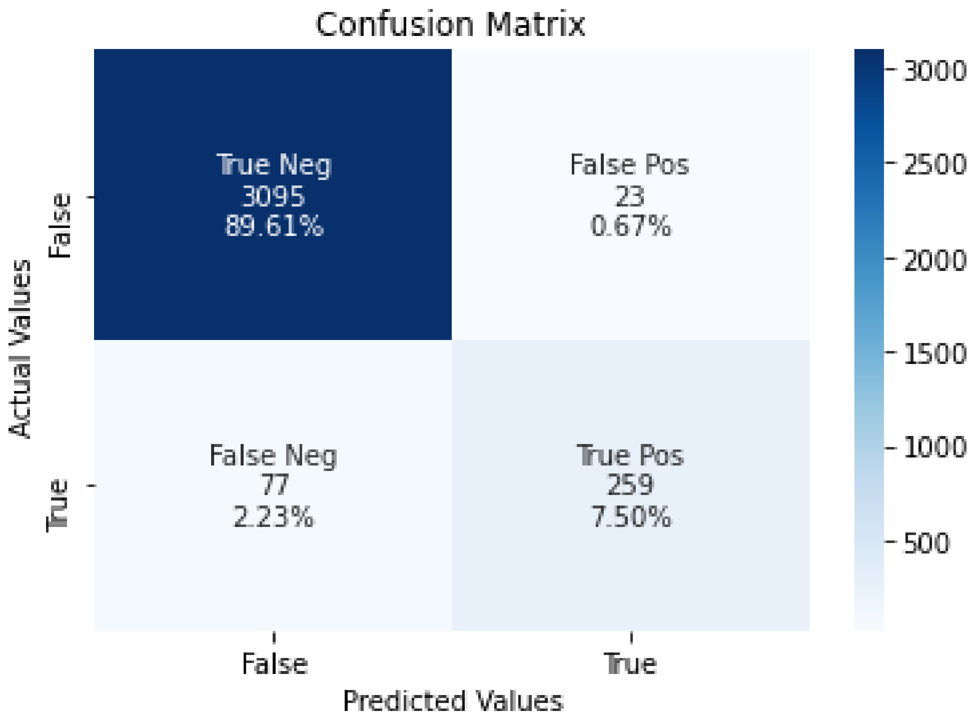
```
0.9652601650563608
```

Confusion Matrix

[1111]:
```
y_train_pred = ada_boosting_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3095   23]
 [  77  259]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9184
Recall =  0.7708
F1 Value =  0.8382
```

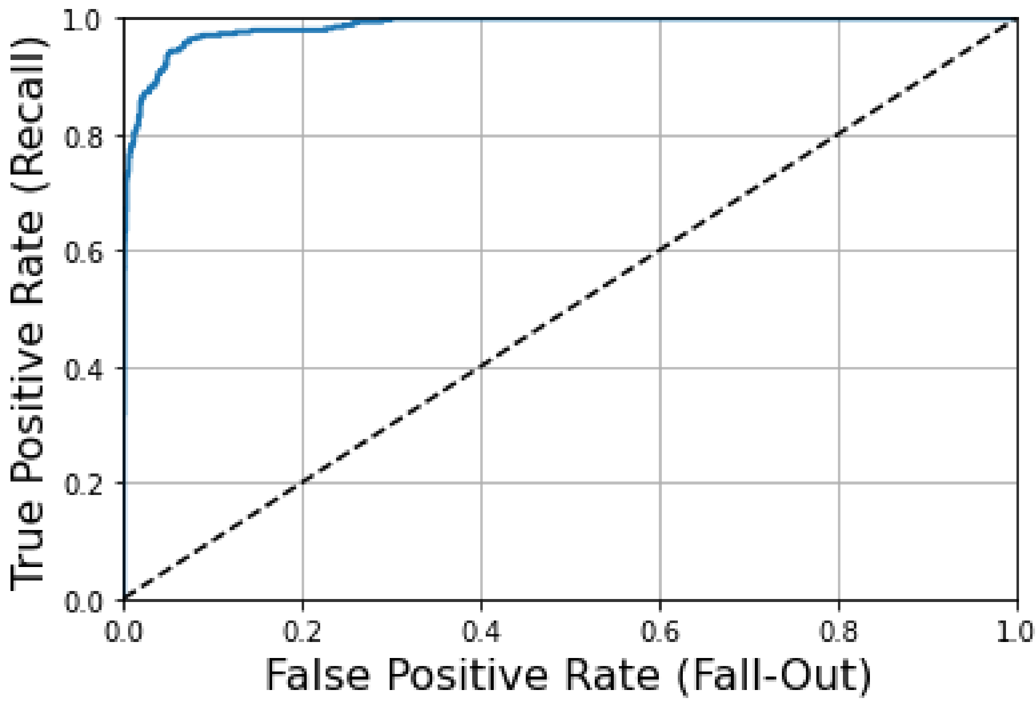
[1112]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1113]:
```
y_probas_boosting = ada_boosting_clf.predict_proba( X_train_selected)
y_scores_boosting = y_probas_boosting [:,-1]

plot_roc_curve(y_train_all ,y_scores_boosting)
save_fig("ROC for Ada Boosting Partial Model")
```

0.9861




```
Saving figure ROC for Ada Boosting Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.48 Performance on Validation Set

[1114]:
```
y_valid_pred = ada_boosting_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[663   5]
 [ 16  56]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.918
Recall =  0.7778
F1 Value =  0.8421
```

3.49 Stacking

[1115]:
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline

estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
 ↪('svr',make_pipeline(StandardScaler(),SVC(gamma="auto",class_weight="balanced",C=1,
 ↪kernel="poly",probability=True)))]


Stacking_clf = StackingClassifier(estimators=estimators,
 ↪final_estimator=LogisticRegression(),
                                  stack_method="predict_proba")
```

3.50 Full Model

3.51 Training

[1116]:
```
Stacking_clf.fit(X_train_all, y_train_all)
Stacking_scores = cross_val_score(Stacking_clf, X_train_all, y_train_all, cv=6)
print(Stacking_scores.mean())
```

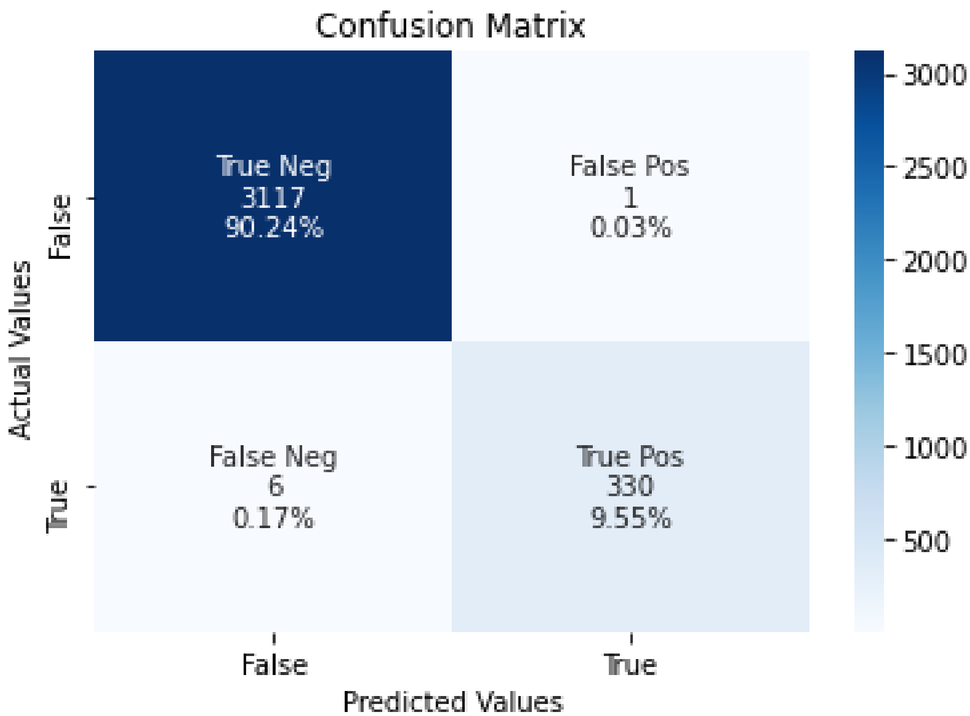
```
0.9834953703703704
```

Confusion Matrix

[1117]:
```
y_train_pred = Stacking_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3117    1]
 [   6  330]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.997
Recall =  0.9821
F1 Value =  0.9895
```

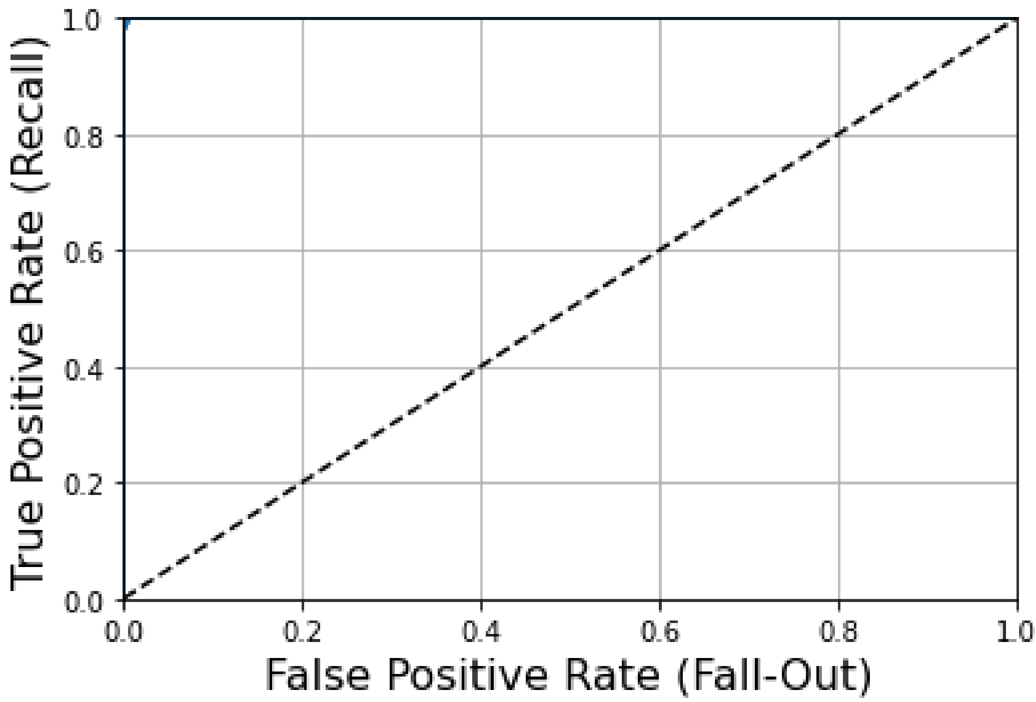
[1118]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1119]:
```
y_probas_stacking = Stacking_clf.predict_proba( X_train_all)
y_scores_stacking = y_probas_stacking [:,-1]

plot_roc_curve(y_train_all ,y_scores_stacking)
save_fig("ROC for stacking Full Model")
```

```
1.0
```




```
Saving figure ROC for stacking Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.52 Performance on Validation Set

[1120]:
```
y_valid_pred = Stacking_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[667   1]
 [  9  63]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9844
Recall =  0.875
F1 Value =  0.9265
```

3.53 Model with Selected Attributes

3.54 Training

[1121]:
```
Stacking_clf.fit(X_train_selected, y_train_selected)
stacking_scores = cross_val_score(Stacking_clf, X_train_selected,␣
 ↪y_train_selected, cv=6)
print(stacking_scores.mean())
```

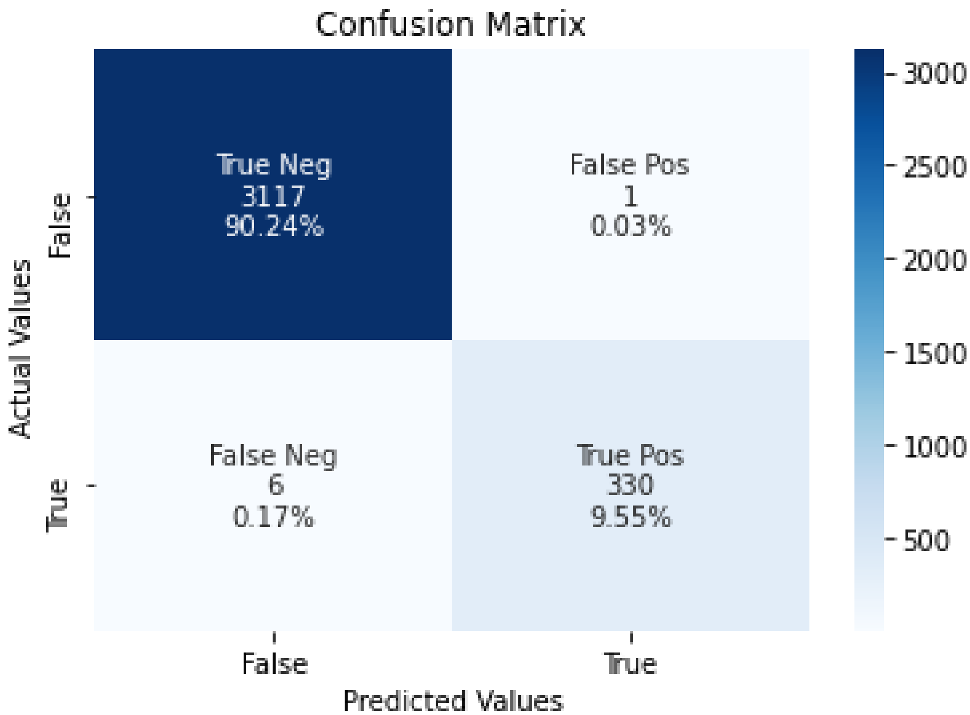
```
0.9814678945249597
```

Confusion Matrix

[1122]:
```
y_train_pred = Stacking_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3117    1]
 [   6  330]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.997
Recall =  0.9821
F1 Value =  0.9895
```

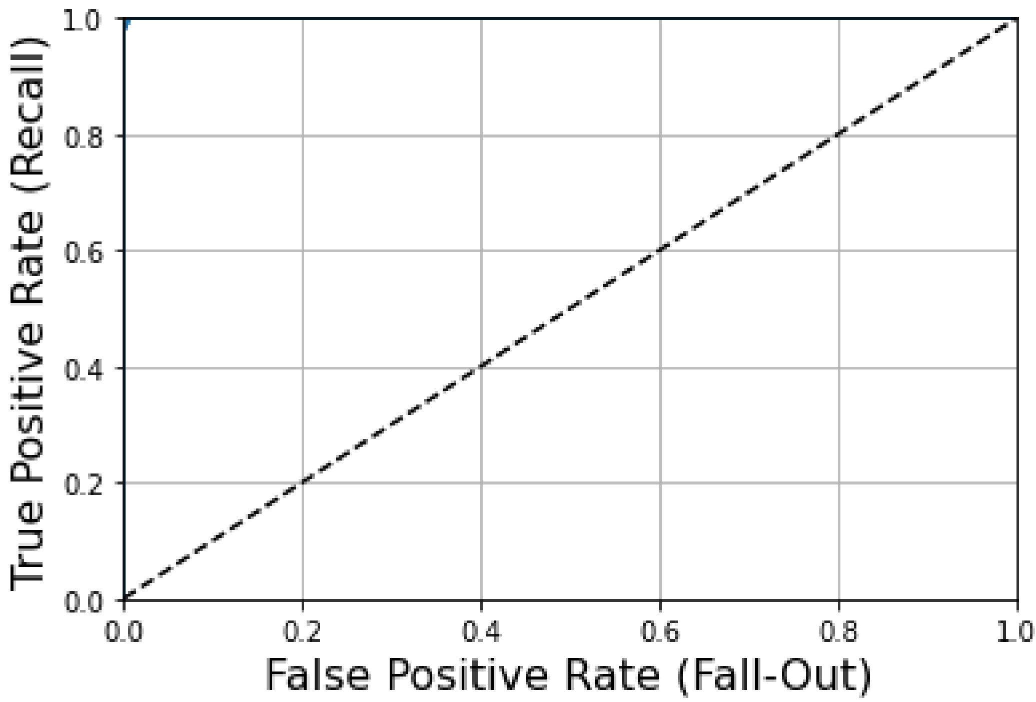
[1123]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1124]:
```
y_probas_stacking = Stacking_clf.predict_proba( X_train_selected)
y_scores_stacking = y_probas_stacking [:,-1]

plot_roc_curve(y_train_selected ,y_scores_stacking)
save_fig("ROC for Stacking Partial Model")
```

```
0.9999
```




```
Saving figure ROC for Stacking Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.55 Performance on Validation Set

[1125]:
```python
y_valid_pred = Stacking_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[667   1]
 [  7  65]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9848
Recall =  0.9028
F1 Value =  0.942
```

3.56 KNN

[1139]:
```
from sklearn.neighbors import KNeighborsClassifier

neigh_clf = KNeighborsClassifier(n_neighbors=3)
```

3.57 Full Model

3.58 Training

[1140]:
```
neigh_clf.fit(X_train_all, y_train_all)
neigh_scores = cross_val_score(neigh_clf, X_train_all, y_train_all, cv=6)
print(neigh_scores.mean())
```

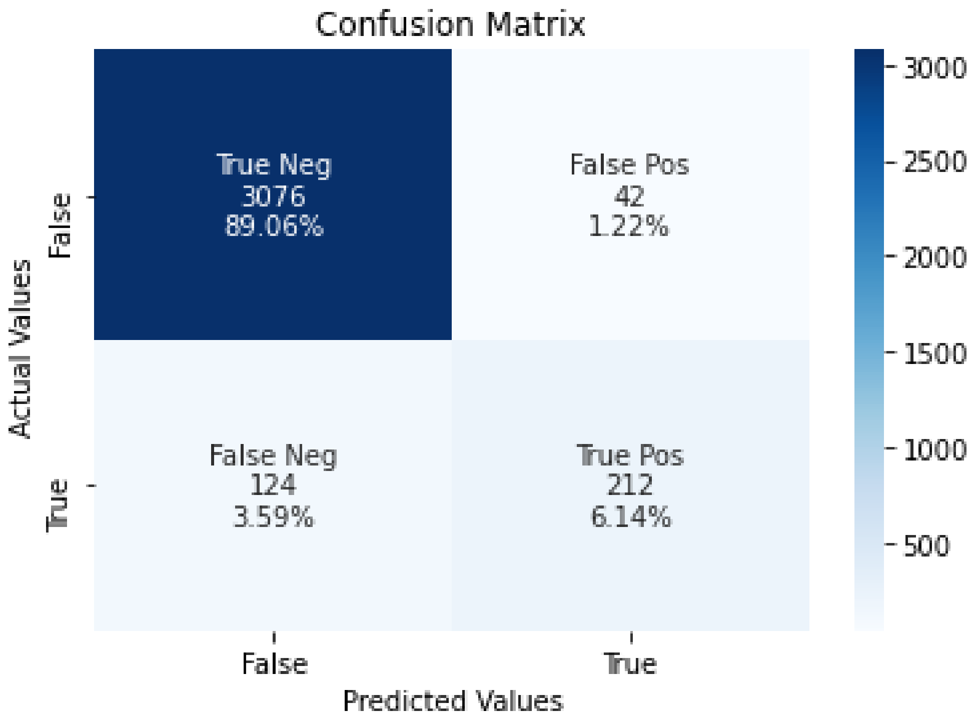
```
0.903010768921095
```

Confusion Matrix

[1141]:
```
y_train_pred = neigh_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3076   42]
 [ 124  212]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8346
Recall =  0.631
F1 Value =  0.7186
```

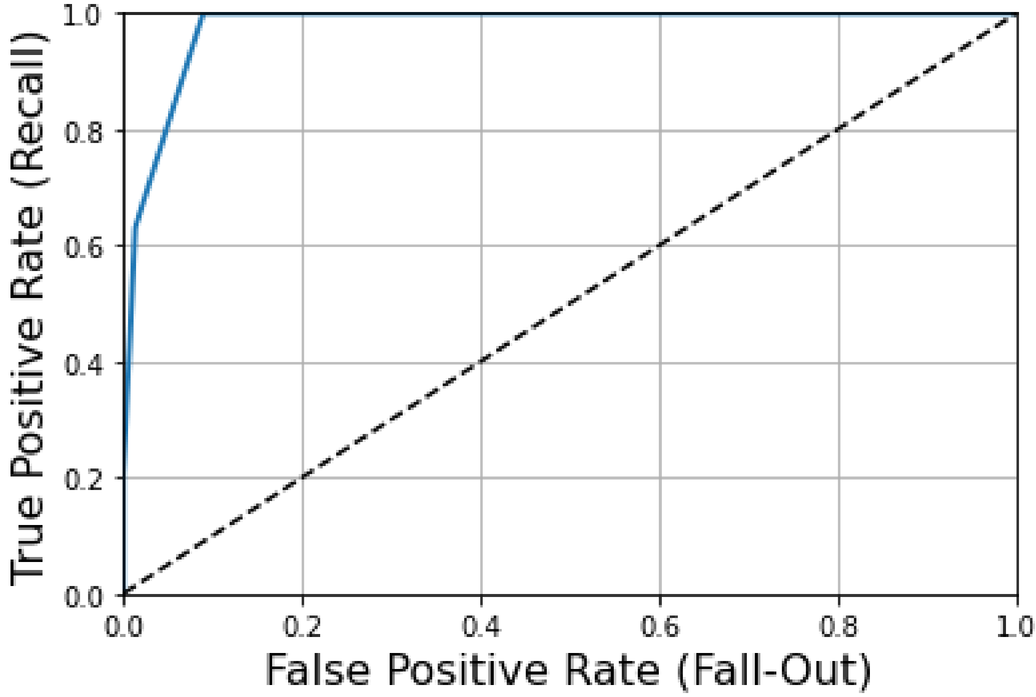
[1142]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1143]:
```
y_probas_knn = neigh_clf.predict_proba( X_train_all)
y_scores_knn = y_probas_knn [:,-1]

plot_roc_curve(y_train_all ,y_scores_knn)
save_fig("ROC for KNN Full Model")
```

0.9781




```
Saving figure ROC for KNN Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.59 Performance on Validation Set

[1144]:
```python
y_valid_pred = neigh_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[643  25]
 [ 48  24]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.4898
Recall =  0.3333
F1 Value =  0.3967
```

3.60 Model with Selected Attributes

3.61 Training

[1145]:
```
neigh_clf.fit(X_train_selected, y_train_selected)
neigh_scores = cross_val_score(neigh_clf, X_train_selected, y_train_selected,␣
↪cv=6)
print(neigh_scores.mean())
```

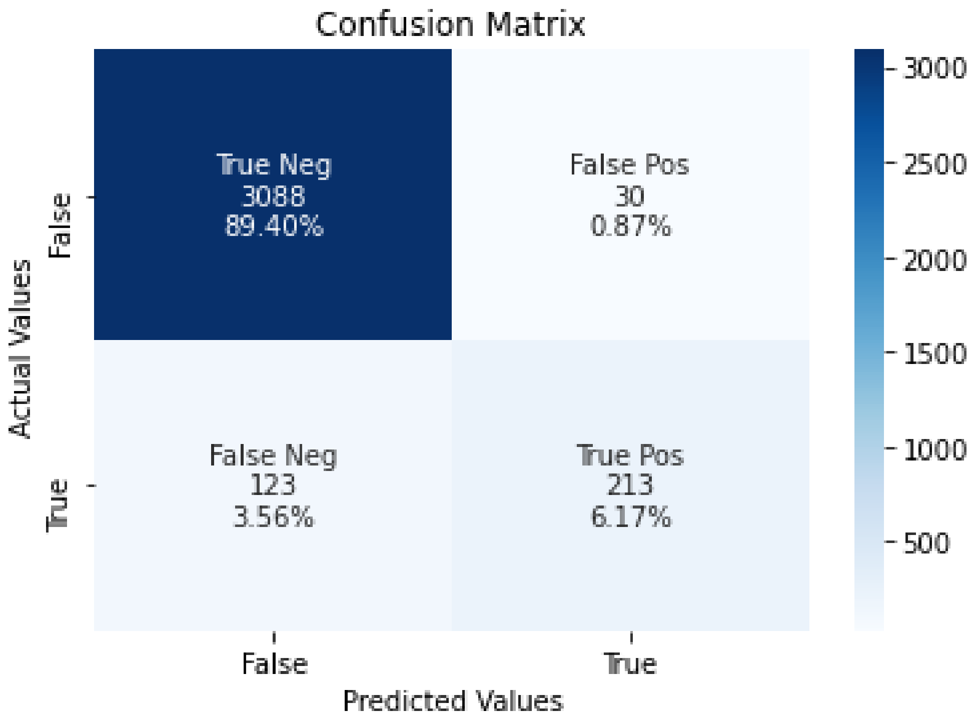
```
0.9241440217391305
```

Confusion Matrix

[1146]:
```
y_train_pred = neigh_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3088   30]
 [ 123  213]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8765
Recall =  0.6339
F1 Value =  0.7358
```

[1147]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1148]:
```
y_probas_knn = neigh_clf.predict_proba( X_train_selected)
y_scores_knn = y_probas_knn [:,-1]

plot_roc_curve(y_train_selected ,y_scores_knn)
save_fig("ROC for KNN Partial Model")
```

```
0.9865
```


```
Saving figure ROC for KNN Partial Model

<Figure size 432x288 with 0 Axes>
```

3.62 Performance on Validation Set

[1149]:
```
y_valid_pred = neigh_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[654  14]
 [ 50  22]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.6111
Recall =  0.3056
F1 Value =  0.4074
```

3.63 Logistic Regression

[1150]:
```
from sklearn.linear_model import LogisticRegression

logit_clf = LogisticRegression(random_state=42,max_iter=10000,penalty="l2",
                               solver="liblinear")
```

3.64 Full Model

3.65 Training

[1151]:
```
logit_clf.fit(X_train_all, y_train_all)
```

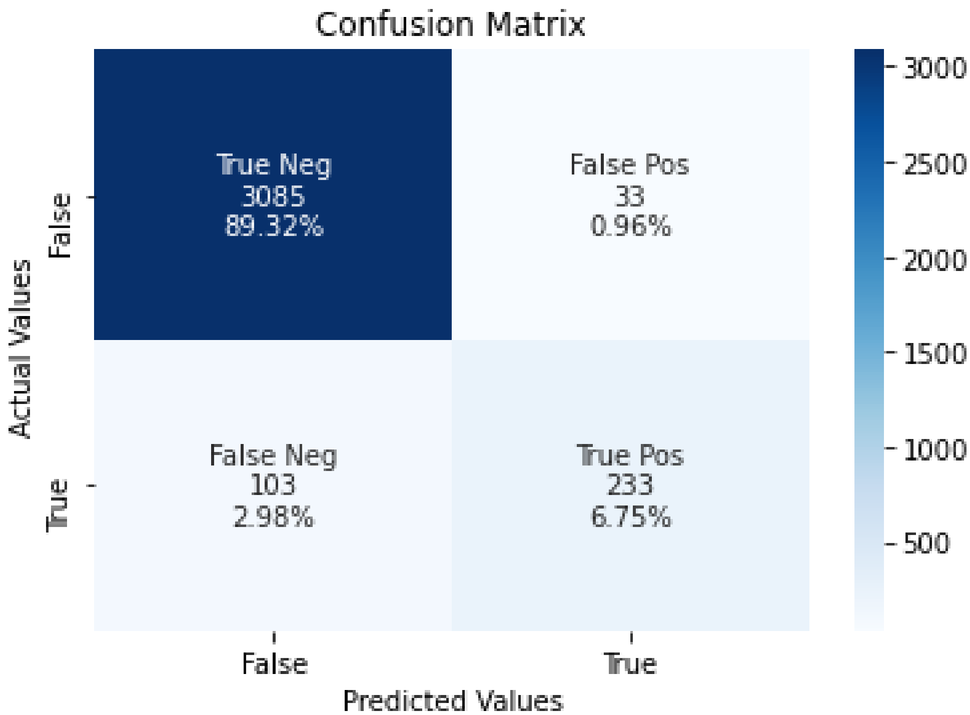
[1151]:
```
LogisticRegression(max_iter=10000, random_state=42, solver='liblinear')
```

Confusion Matrix

[1152]:
```
y_train_pred = logit_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3085   33]
 [ 103  233]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8759
Recall =  0.6935
F1 Value =  0.7741
```

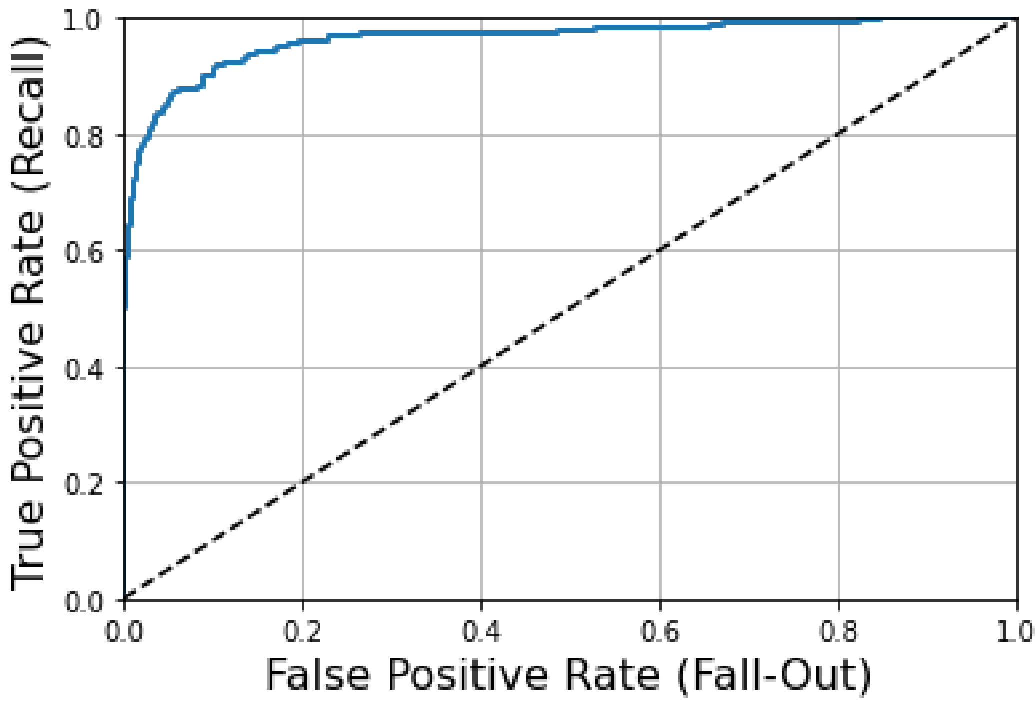
[1153]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1154]:
```
y_probas_logit = logit_clf.predict_proba( X_train_all)
y_scores_logit = y_probas_logit [:,-1]

plot_roc_curve(y_train_all ,y_scores_logit)
save_fig("ROC for Logistic Full Model")
```

```
0.9636
```




```
Saving figure ROC for Logistic Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.66 Performance on Validation Set

[1155]:
```
y_valid_pred = logit_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[659   9]
 [ 23  49]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8448
Recall =  0.6806
F1 Value =  0.7538
```

3.67 Model with Selected Attributes

3.68 Training

[1156]:
```
logit_clf.fit(X_train_selected, y_train_selected)
logit_scores = cross_val_score(logit_clf, X_train_selected, y_train_selected,␣
↪cv=6)
print(logit_scores.mean())
```

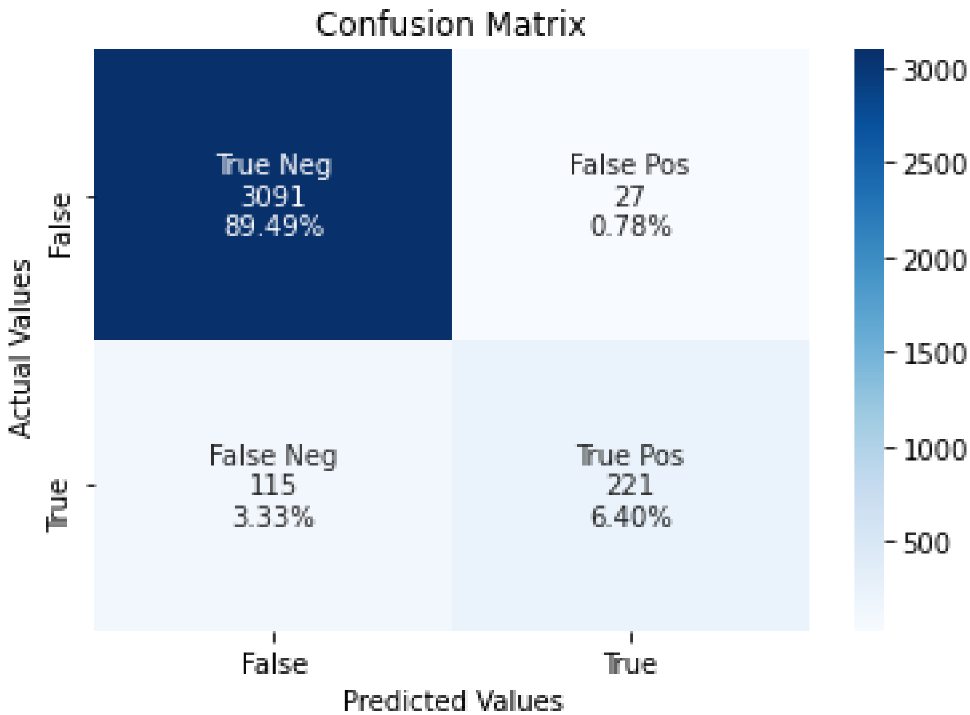
```
0.9583137077294687
```

Confusion Matrix

[1157]:
```
y_train_pred = logit_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3091   27]
 [ 115  221]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.8911
Recall =  0.6577
F1 Value =  0.7568
```

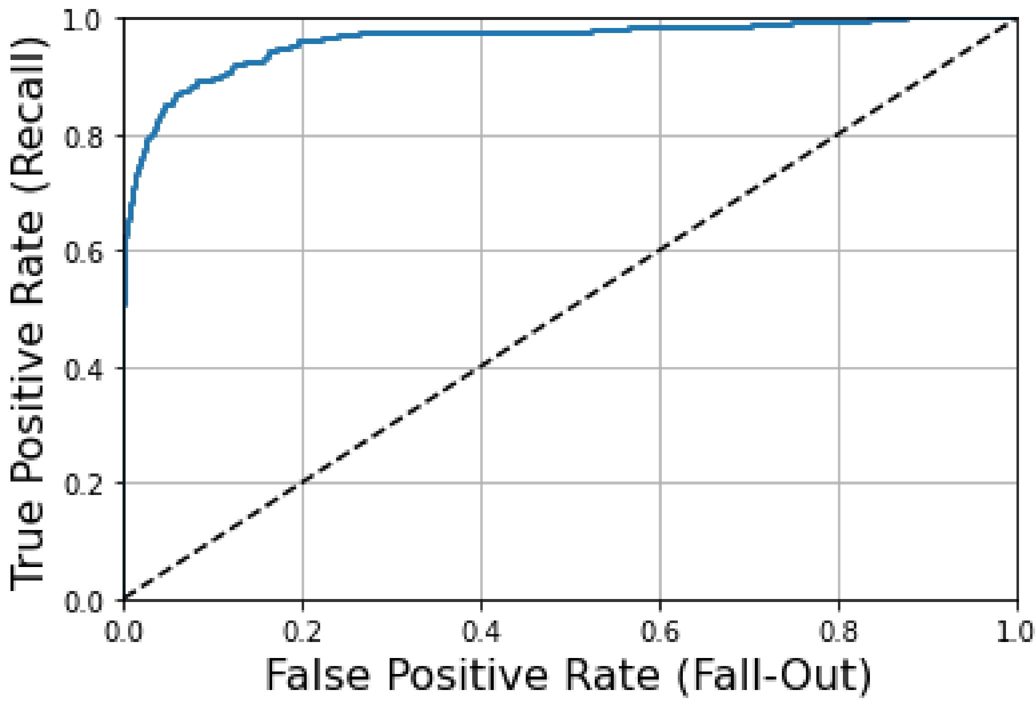
[1158]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1159]:
```
y_probas_logit = logit_clf.predict_proba( X_train_selected)
y_scores_logit = y_probas_logit [:,-1]

plot_roc_curve(y_train_selected ,y_scores_logit)
save_fig("ROC for Logistic Partial Model")
```

```
0.9611
```




```
Saving figure ROC for Logistic Partial Model

<Figure size 432x288 with 0 Axes>
```

3.69 Performance on Validation Set

[1160]:
```
y_valid_pred = logit_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[659   9]
 [ 29  43]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8269
Recall =  0.5972
F1 Value =  0.6935
```

3.70 CART

[1267]:
```
from sklearn.tree import DecisionTreeClassifier

cart_clf = DecisionTreeClassifier(random_state=42,min_samples_split=6,
                                  min_samples_leaf =␣
 ↪3,max_features="sqrt",class_weight={0:2,1:3})
```

3.71 Full Model

3.72 Training

[1268]:
```
cart_clf.fit(X_train_all, y_train_all)
cart_scores = cross_val_score(cart_clf, X_train_all, y_train_all, cv=6)
print(cart_scores.mean())
```

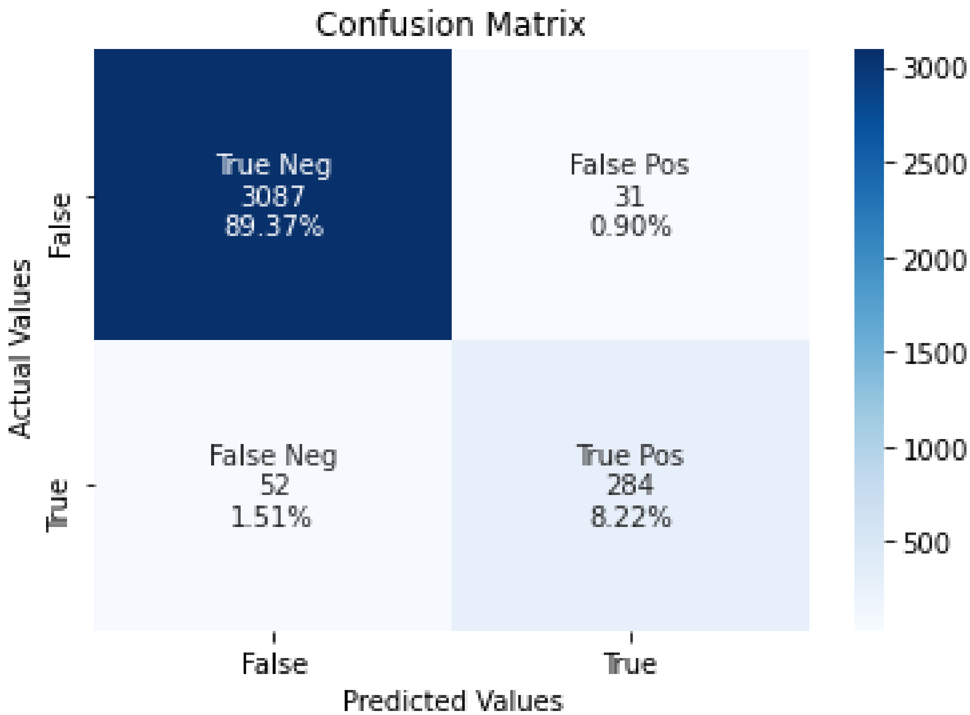
```
0.958311191626409
```

Confusion Matrix

[1269]:
```
y_train_pred = cart_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[3087   31]
 [  52  284]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9016
Recall =  0.8452
F1 Value =  0.8725
```

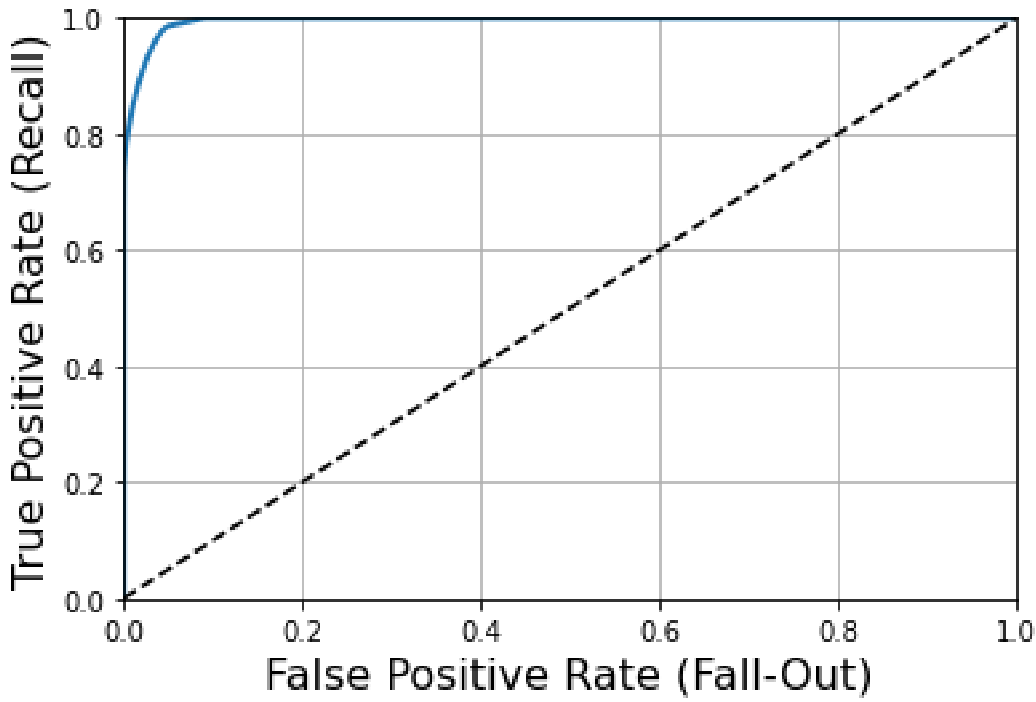
[1270]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1171]:
```
y_probas_cart = cart_clf.predict_proba( X_train_all)
y_scores_cart = y_probas_cart [:,-1]

plot_roc_curve(y_train_all ,y_scores_cart)
save_fig("ROC for CART Full Model")
```

```
0.9951
```




```
Saving figure ROC for CART Full Model

<Figure size 432x288 with 0 Axes>
```

3.73 Performance on Validation Set

[1172]:
```python
y_valid_pred = cart_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[655  13]
 [ 14  58]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8169
Recall =  0.8056
F1 Value =  0.8112
```

3.74 Model with Selected Attributes

3.75 Training

[1173]:
```
cart_clf.fit(X_train_selected, y_train_selected)
cart_scores = cross_val_score(cart_clf, X_train_selected, y_train_selected,␣
 ↪cv=6)
print(cart_scores.mean())
```

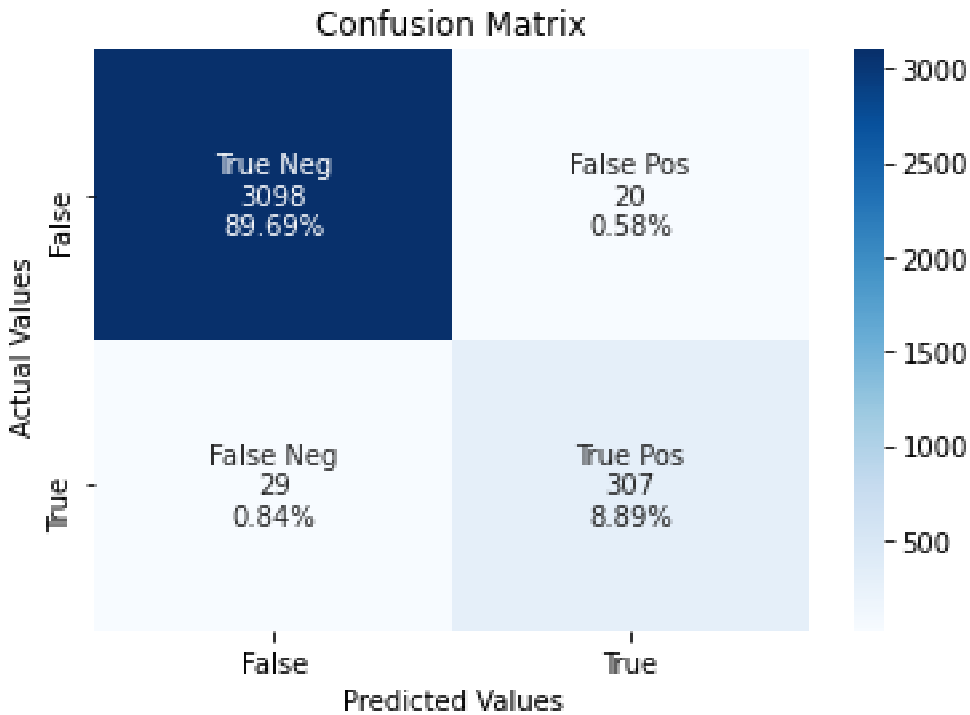
```
0.9649677938808373
```

Confusion Matrix

[1174]:
```
y_train_pred = cart_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[3098   20]
 [  29  307]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.9388
Recall =  0.9137
F1 Value =  0.9261
```

[1175]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1176]:
```
y_probas_cart = cart_clf.predict_proba( X_train_selected)
y_scores_cart = y_probas_cart [:,-1]

plot_roc_curve(y_train_selected ,y_scores_cart)
save_fig("ROC for CART Partial Model")
```

```
0.9983
```


```
Saving figure ROC for CART Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.76 Performance on Validation Set

[1177]:
```
y_valid_pred = cart_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[665   3]
 [ 12  60]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9524
Recall =  0.8333
F1 Value =  0.8889
```

3.77 Bayesian Learning

3.78 Naïve Bayes (Gaussian)

[1230]:
```
from sklearn.naive_bayes import GaussianNB

gnb_clf = GaussianNB()
```

3.79 Full Model

3.80 Training

[1231]:
```
gnb_clf.fit(X_train_all, y_train_all)
gnb_scores = cross_val_score(gnb_clf, X_train_all, y_train_all, cv=6)
print(gnb_scores.mean())
```

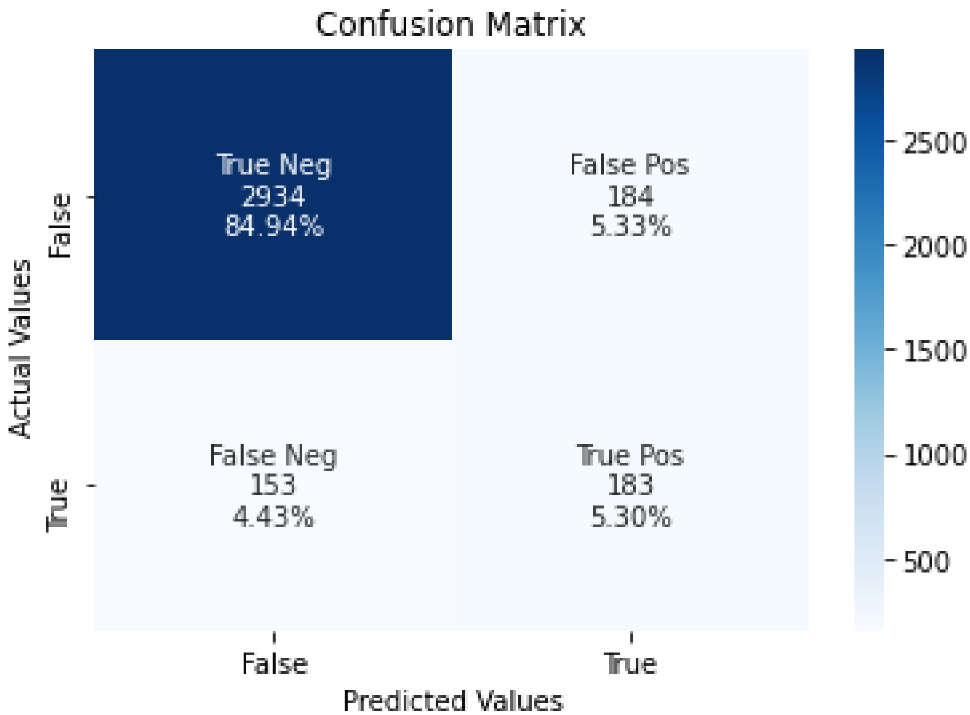
```
0.8992567431561995
```

Confusion Matrix

[1232]:
```
y_train_pred = gnb_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[2934  184]
 [ 153  183]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.4986
Recall =  0.5446
F1 Value =  0.5206
```

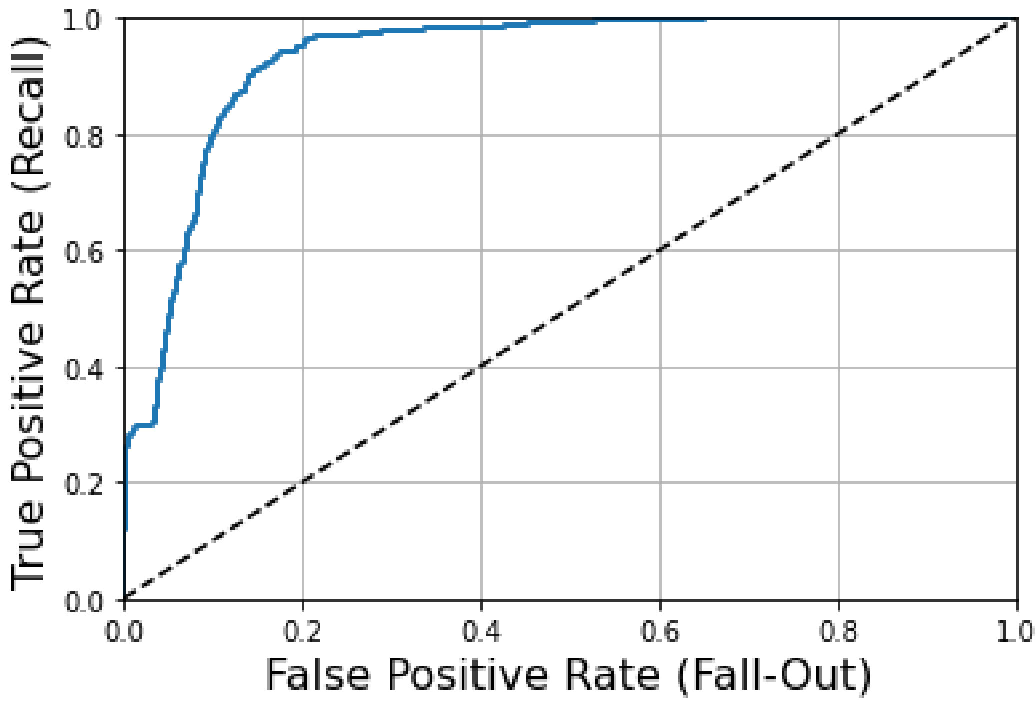
[1233]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1234]:
```
y_probas_gnb = gnb_clf.predict_proba( X_train_all)
y_scores_gnb = y_probas_gnb [:,-1]

plot_roc_curve(y_train_all ,y_scores_gnb)
save_fig("ROC for Gaussian Naive Bayes Full Model")
```

```
0.9306
```




```
Saving figure ROC for Gaussian Naive Bayes Full Model

<Figure size 432x288 with 0 Axes>
```

3.81 Performance on Validation Set

[1235]:
```
y_valid_pred = gnb_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[624  44]
 [ 37  35]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.443
Recall =  0.4861
F1 Value =  0.4636
```

3.82 Model with Selected Attributes

3.83 Training

[1236]:
```
gnb_clf.fit(X_train_selected, y_train_selected)
gnb_scores = cross_val_score(gnb_clf, X_train_selected, y_train_selected, cv=6)
print(gnb_scores.mean())
```

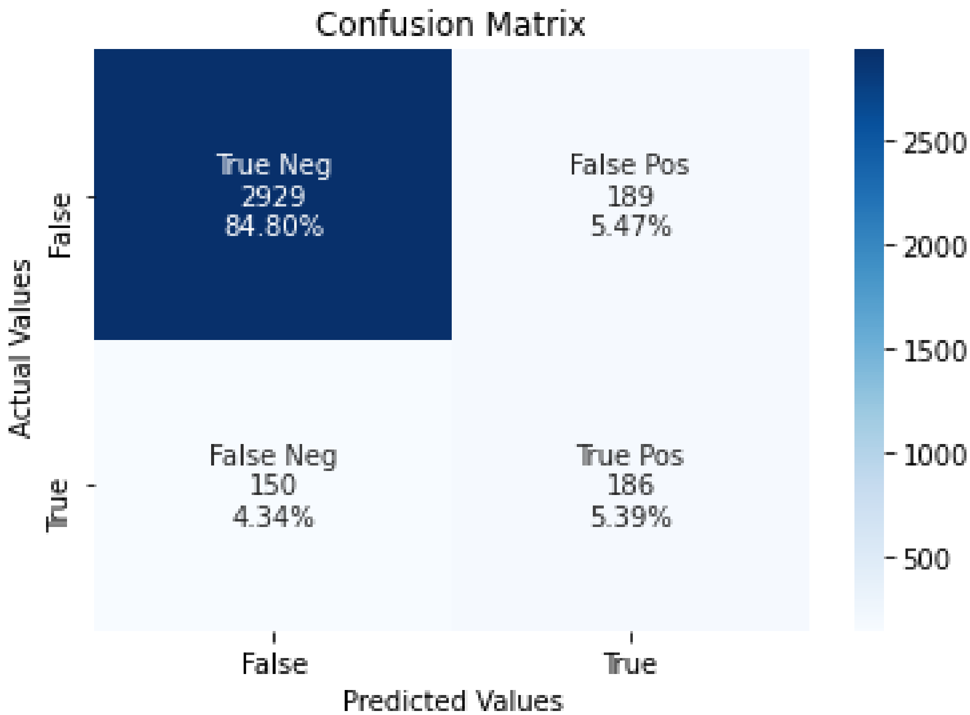
```
0.9009938607085345
```

Confusion Matrix

[1237]:
```
y_train_pred = gnb_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[2929  189]
 [ 150  186]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.496
Recall =  0.5536
F1 Value =  0.5232
```

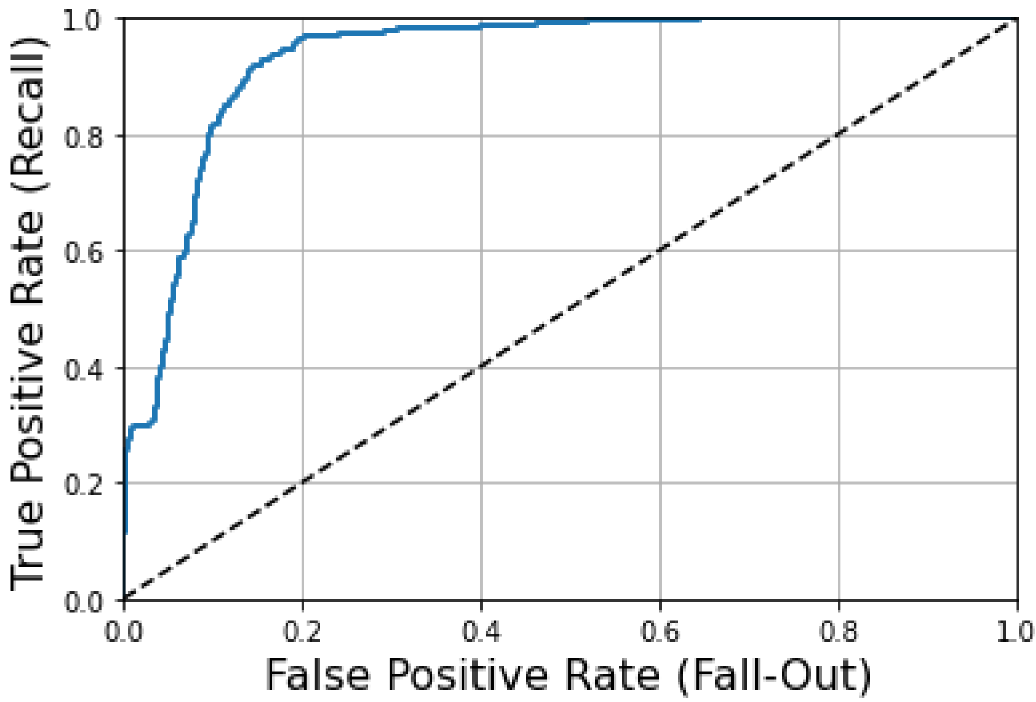
[1238]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1239]:
```
y_probas_gnb = gnb_clf.predict_proba( X_train_selected)
y_scores_gnb = y_probas_gnb [:,-1]

plot_roc_curve(y_train_selected ,y_scores_gnb)
save_fig("ROC for Gaussian Naive Bayse Partial Model")
```

```
0.9326
```




```
Saving figure ROC for Gaussian Naive Bayse Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.84 Performance on Validation Set

[1240]:
```
y_valid_pred = gnb_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[624  44]
 [ 36  36]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.45
Recall =  0.5
F1 Value =  0.4737
```

3.85 Naïve Bayes (Multinomial)

[1241]:
```
from sklearn.naive_bayes import MultinomialNB

gnb_multi_clf = MultinomialNB()
```

3.86 Full Model

3.87 Training

[1242]:
```
gnb_multi_clf.fit(X_train_all, y_train_all)
gnb_multi_scores = cross_val_score(gnb_multi_clf, X_train_all, y_train_all,␣
↪cv=6)
print(gnb_multi_scores.mean())
```

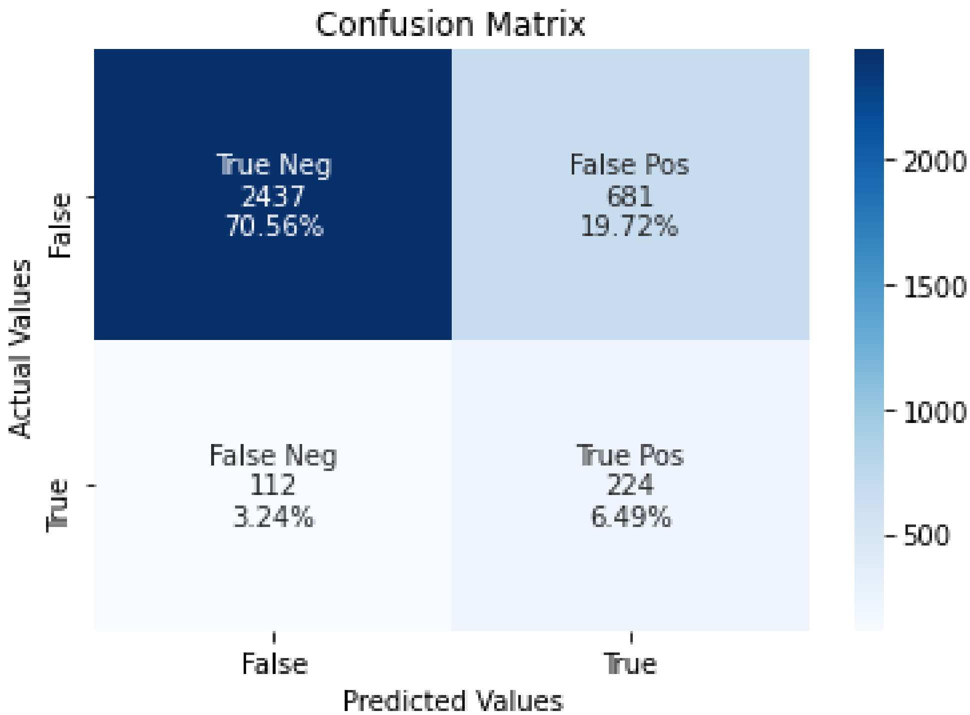
```
0.7678170289855073
```

Confusion Matrix

[1243]:
```
y_train_pred = gnb_multi_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[2437  681]
 [ 112  224]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.2475
Recall =  0.6667
F1 Value =  0.361
```

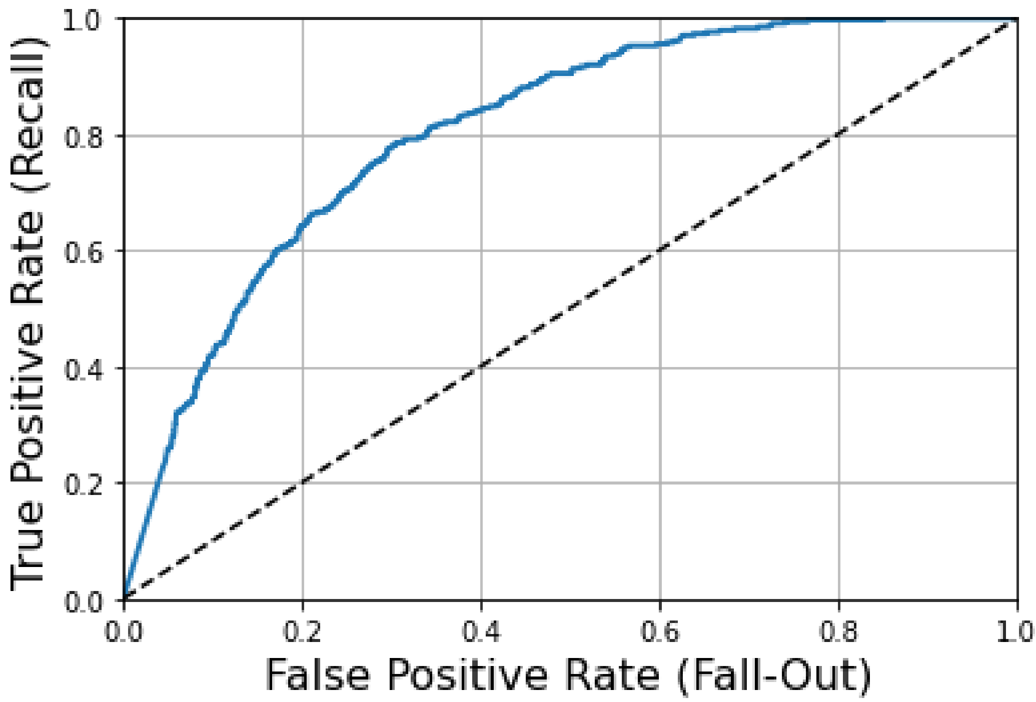
[1244]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1245]:
```
y_probas_gnb_multi = gnb_multi_clf.predict_proba( X_train_all)
y_scores_gnb_multi = y_probas_gnb_multi [:,-1]

plot_roc_curve(y_train_all ,y_scores_gnb_multi)
save_fig("ROC for Multinomial Naive Bayes Full Model")
```

```
0.807
```




```
Saving figure ROC for Multinomial Naive Bayes Full Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.88 Performance on Validation Set

[1246]:
```
y_valid_pred = gnb_multi_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[528 140]
 [ 19  53]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.2746
Recall =  0.7361
F1 Value =  0.4
```

3.89 Model with Selected Attributes

3.90 Training

[1247]:
```
gnb_multi_clf.fit(X_train_selected, y_train_selected)
gnb_multi_scores = cross_val_score(gnb_multi_clf, X_train_selected,␣
 ↪y_train_selected, cv=6)
print(gnb_multi_scores.mean())
```

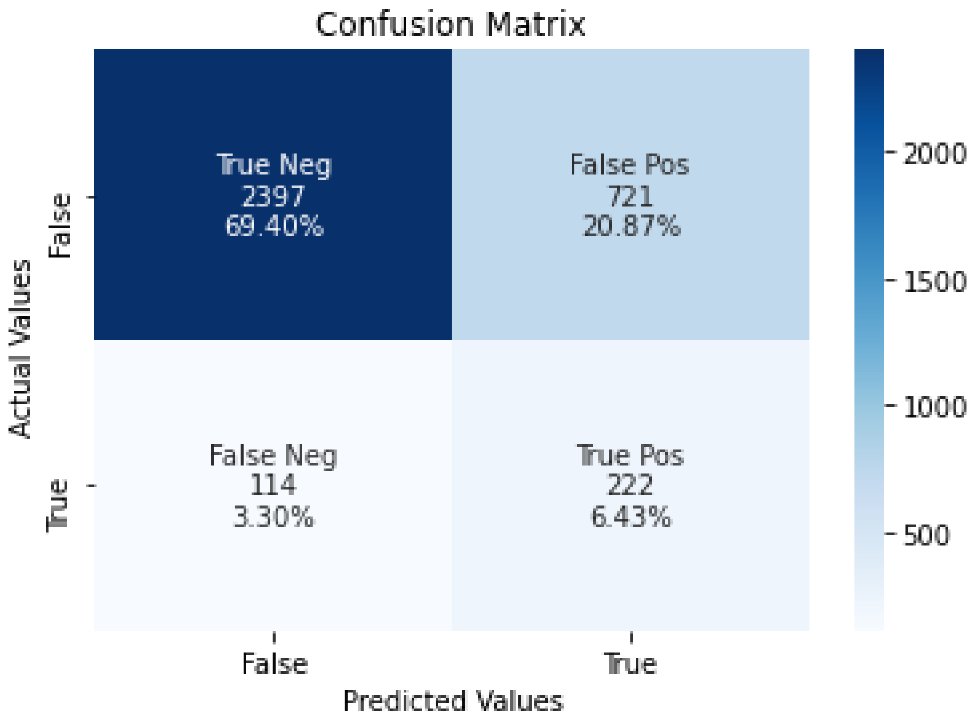
```
0.773257347020934
```

Confusion Matrix

[1248]:
```
y_train_pred = gnb_multi_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[2397  721]
 [ 114  222]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.2354
Recall =  0.6607
F1 Value =  0.3471
```

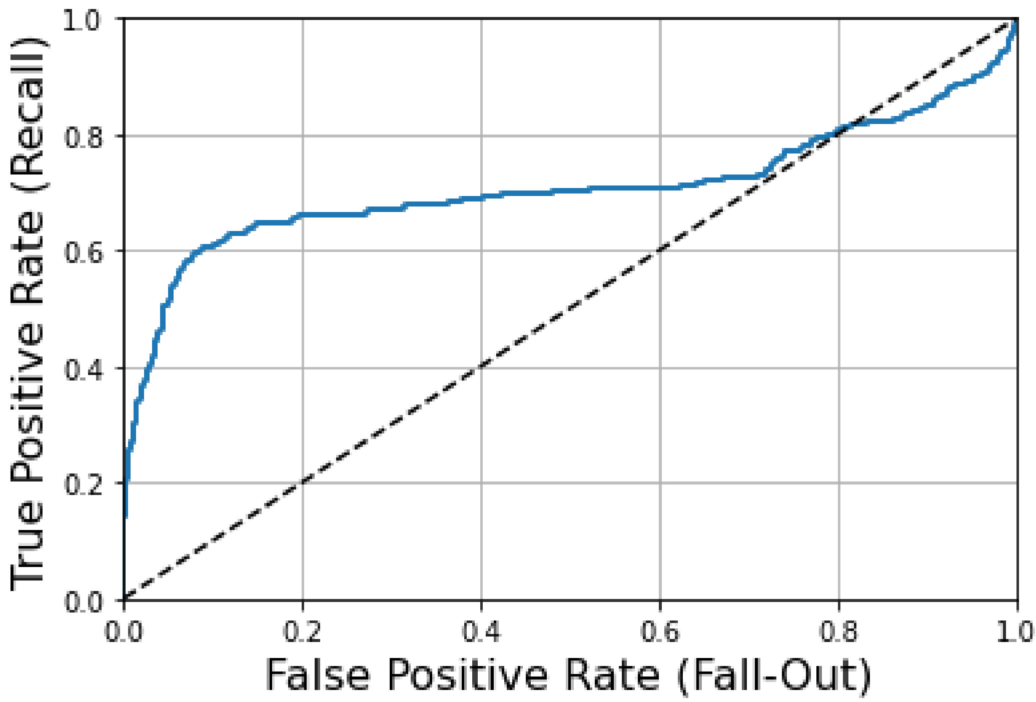
[1249]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1250]:
```
y_probas_gnb_multi = gnb_multi_clf.predict_proba( X_train_selected)
y_scores_gnb_multi = y_probas_gnb_multi [:,-1]

plot_roc_curve(y_train_selected ,y_scores_gnb_multi)
save_fig("ROC for Multinomial Naive Bayse Partial Model")
```

```
0.7067
```




```
Saving figure ROC for Multinomial Naive Bayse Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.91 Performance on Validation Set

[1251]:
```
y_valid_pred = gnb_multi_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[526 142]
 [ 28  44]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.2366
Recall =  0.6111
F1 Value =  0.3411
```

3.92 Naïve Bayes (Complement)

[1252]:
```
from sklearn.naive_bayes import ComplementNB

gnb_comple_clf = ComplementNB()
```

3.93 Full Model

3.94 Training

[1253]:
```
gnb_comple_clf.fit(X_train_all, y_train_all)
gnb_comple_scores = cross_val_score(gnb_comple_clf, X_train_all, y_train_all,␣
↪cv=6)
print(gnb_comple_scores.mean())
```

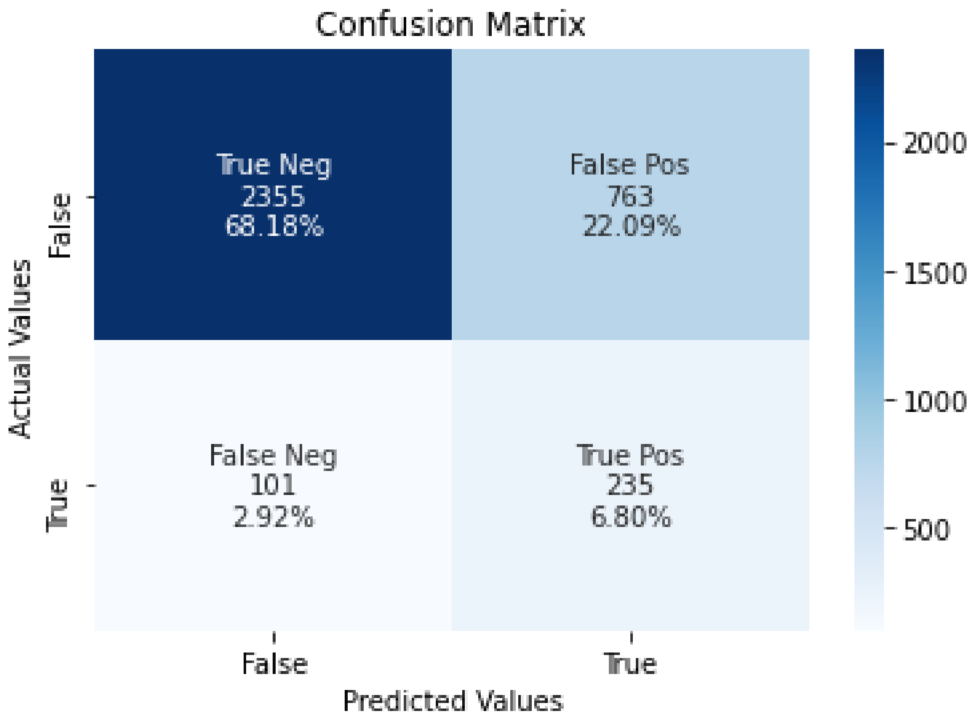
```
0.7504463566827697
```

Confusion Matrix

[1254]:
```
y_train_pred = gnb_comple_clf.predict(X_train_all)

print_classification_report(y_train_all,y_train_pred)
```

```
=============Confusion Matrix =============
[[2355  763]
 [ 101  235]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.2355
Recall =  0.6994
F1 Value =  0.3523
```

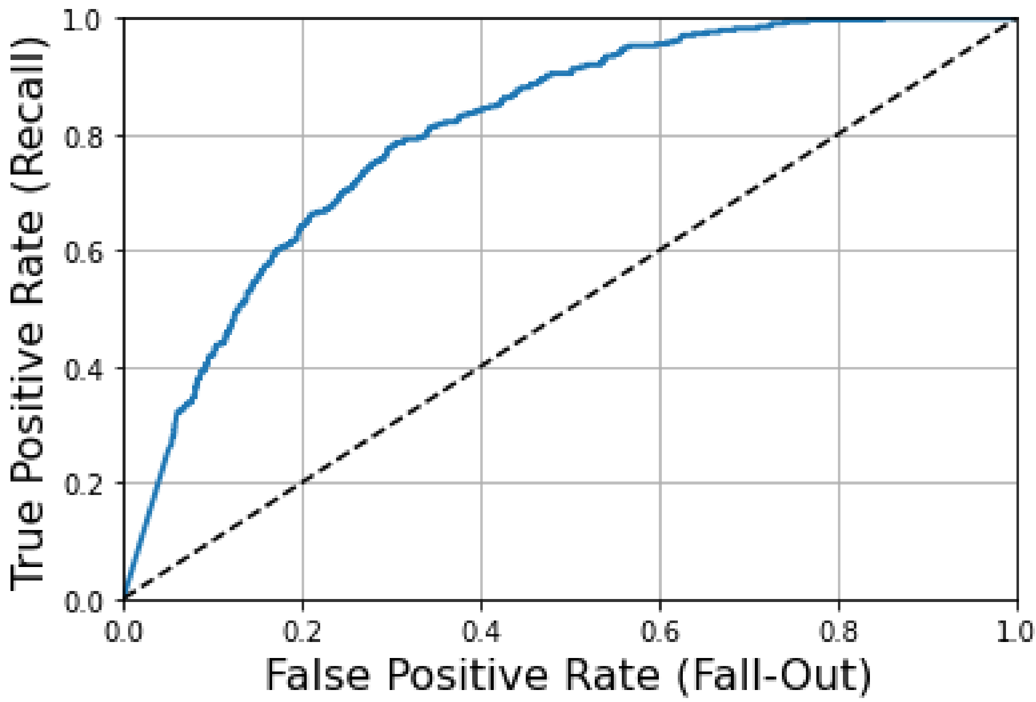
[1255]:
```
cf_matrix=confusion_matrix(y_train_all, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1257]:
```
y_probas_gnb_comple = gnb_comple_clf.predict_proba( X_train_all)
y_scores_gnb_comple = y_probas_gnb_comple [:,-1]

plot_roc_curve(y_train_all ,y_scores_gnb_comple)
save_fig("ROC for Complement Naive Bayes Full Model")
```

0.807




```
Saving figure ROC for Complement Naive Bayes Full Model

<Figure size 432x288 with 0 Axes>
```

3.95 Performance on Validation Set

[1258]:
```
y_valid_pred = gnb_comple_clf.predict(X_valid_all)

print_classification_report(y_valid_all,y_valid_pred)
```

```
=============Confusion Matrix =============
[[505 163]
 [ 16  56]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.2557
Recall =  0.7778
F1 Value =  0.3849
```

3.96 Model with Selected Attributes

3.97 Training

[1259]:
```
gnb_comple_clf.fit(X_train_selected, y_train_selected)
gnb_comple_scores = cross_val_score(gnb_comple_clf, X_train_selected,␣
↪y_train_selected, cv=6)
print(gnb_comple_scores.mean())
```

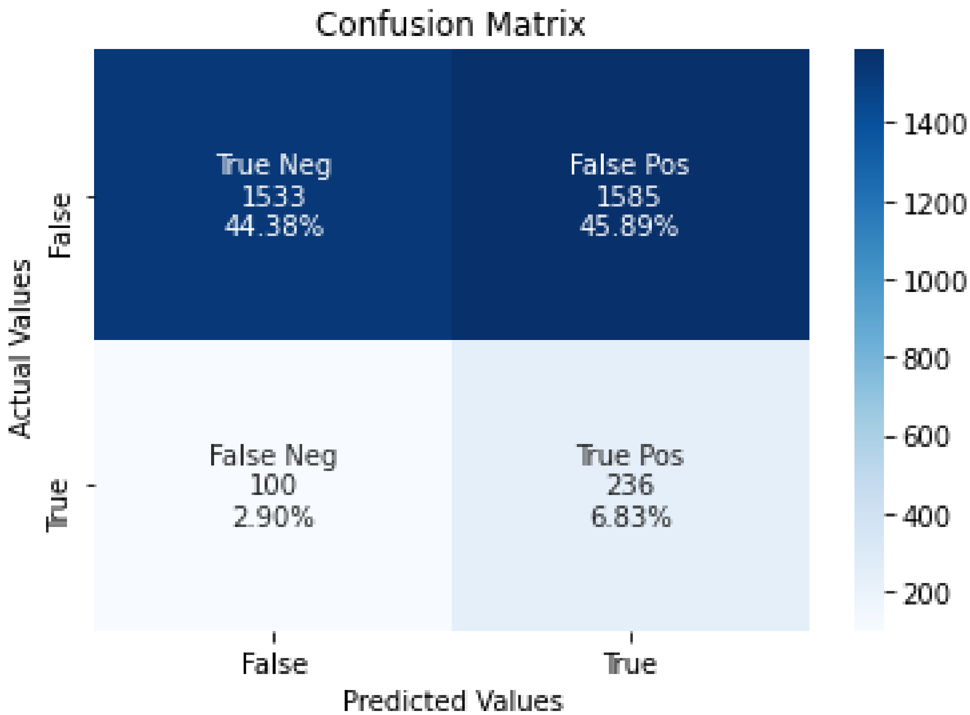
```
0.546285728663446
```

Confusion Matrix

[1260]:
```
y_train_pred = gnb_comple_clf.predict(X_train_selected)

print_classification_report(y_train_selected,y_train_pred)
```

```
=============Confusion Matrix =============
[[1533 1585]
 [ 100  236]]

Perfect Prediction If Done
[[3118    0]
 [   0  336]]

=============Sumarry Measures==============
Precision Score =  0.1296
Recall =  0.7024
F1 Value =  0.2188
```

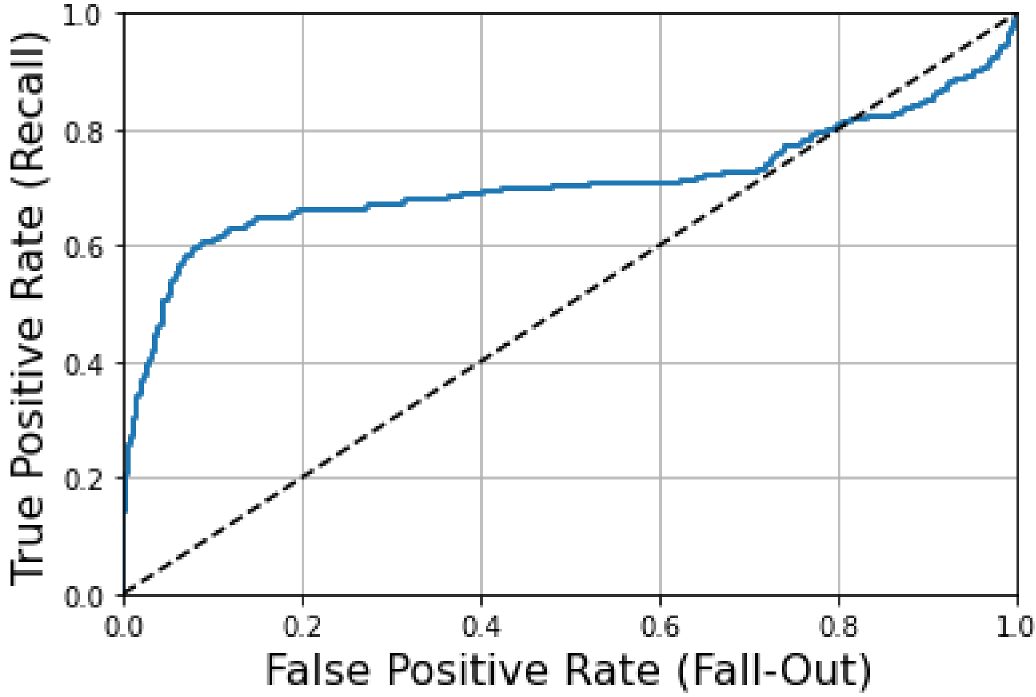
[1261]:
```
cf_matrix=confusion_matrix(y_train_selected, y_train_pred)

plot_cf_matrix(cf_matrix)
```



ROC Curve

[1262]:
```
y_probas_gnb_comple = gnb_comple_clf.predict_proba( X_train_selected)
y_scores_gnb_comple = y_probas_gnb_comple [:,-1]

plot_roc_curve(y_train_selected ,y_scores_gnb_comple)
save_fig("ROC for Complement Naive Bayse Partial Model")
```

```
0.7067
```




```
Saving figure ROC for Complement Naive Bayse Partial Model
```

```
<Figure size 432x288 with 0 Axes>
```

3.98 Performance on Validation Set

[1263]:
```
y_valid_pred = gnb_comple_clf.predict(X_valid_selected)

print_classification_report(y_valid_selected,y_valid_pred)
```

```
=============Confusion Matrix =============
[[352 316]
 [ 26  46]]

Perfect Prediction If Done
[[668   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.1271
Recall =  0.6389
F1 Value =  0.212
```

4 Performance of different classifiers on Test Data

4.1 List of Classifiers :

Clasification Algorithm————————**Alias** * SVM (Polynomial Kernel)——————— svm_clf_poly * SVM (Linear Kernel)————————svm_clf_lin * SVM (RBF Kernel)——— ————————svm_clf_rbf * SVM (Sigmoid Kernel)————————svm_clf_sig * Ensemble (Random Forest)——————-forest_clf * Ensemble (Bagging)————————-bagging_clf * Gradient Boosting————————gradient_boosting_clf * Ada Boost———————————— -ada_boosting_clf * Stacking————————————Stacking_clf * KNN——————————— ——————-neigh_clf * Logistic Regression————————logit_clf * CART———————————— —————cart_clf * Naïve Bayes (Gaussian)————————gnb_clf * Naïve Bayes (Multinomial)— ——————gnb_multi_clf * Naïve Bayes (Complement)——————-gnb_comple_clf

4.2 Test Data : Performance

4.3 SVM (Polynomial Kernel)

4.4 Full Model

[1341]:
```
svm_clf_poly.fit(X_train_all, y_train_all)

y_test_pred = svm_clf_poly.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[648  21]
 [ 11  61]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7439
Recall =  0.8472
F1 Value =  0.7922
```

4.5 Model with Selected Attributes

[1342]:
```
svm_clf_poly.fit(X_train_selected, y_train_selected)

y_test_pred = svm_clf_poly.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
```

```
[[641  28]
 [  7  65]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.6989
Recall =  0.9028
F1 Value =  0.7879
```

4.6 SVM (Linear Kernel)

4.7 Full Model

[1282]:
```
svm_clf_lin.fit(X_train_all, y_train_all)

y_test_pred = svm_clf_lin.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[653  16]
 [ 16  56]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7778
Recall =  0.7778
F1 Value =  0.7778
```

4.8 Model with Selected Attributes

[1297]:
```
svm_clf_lin.fit(X_train_selected, y_train_selected)

y_test_pred = svm_clf_lin.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[654  15]
 [ 19  53]]
```

```
Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7794
Recall =  0.7361
F1 Value =  0.7571
```

4.9 SVM (RBF Kernel)

4.10 Full Model

[1283]:
```
svm_clf_rbf.fit(X_train_all, y_train_all)

y_test_pred = svm_clf_rbf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[644  25]
 [  6  66]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7253
Recall =  0.9167
F1 Value =  0.8098
```

4.11 Model with Selected Attributes

[1298]:
```
svm_clf_rbf.fit(X_train_selected, y_train_selected)

y_test_pred = svm_clf_rbf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[639  30]
 [  4  68]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]
```

```
=============Sumarry Measures==============
Precision Score =  0.6939
Recall =  0.9444
F1 Value =  0.8
```

4.12 SVM (Sigmoid Kernel)

4.13 Full Model

[1284]:
```
svm_clf_sig.fit(X_train_all, y_train_all)

y_test_pred = svm_clf_sig.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[555 114]
 [ 14  58]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.3372
Recall =  0.8056
F1 Value =  0.4754
```

4.14 Model with Selected Attributes

[1299]:
```
svm_clf_sig.fit(X_train_selected, y_train_selected)

y_test_pred = svm_clf_sig.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[550 119]
 [ 12  60]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.3352
```

```
Recall =  0.8333
F1 Value =  0.4781
```

4.15 Ensemble (Random Forest)

4.16 Full Model

[1285]:
```
forest_clf.fit(X_train_all, y_train_all)

y_test_pred = forest_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[656  13]
 [  3  69]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8415
Recall =  0.9583
F1 Value =  0.8961
```

4.17 Model with Selected Attributes

[1300]:
```
forest_clf.fit(X_train_selected, y_train_selected)

y_test_pred = forest_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[652  17]
 [  4  68]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8
Recall =  0.9444
F1 Value =  0.8662
```

4.18 Ensemble (Bagging)

4.19 Full Model

[1286]:
```
bagging_clf.fit(X_train_all, y_train_all)

y_test_pred = bagging_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[666   3]
 [  9  63]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9545
Recall =  0.875
F1 Value =  0.913
```

4.20 Model with Selected Attributes

[1301]:
```
bagging_clf.fit(X_train_selected, y_train_selected)

y_test_pred = bagging_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[666   3]
 [  8  64]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9552
Recall =  0.8889
F1 Value =  0.9209
```

4.21 Gradient Boosting

4.22 Full Model

[1287]:
```
gradient_boosting_clf.fit(X_train_all, y_train_all)

y_test_pred = gradient_boosting_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[658  11]
 [  8  64]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8533
Recall =  0.8889
F1 Value =  0.8707
```

4.23 Model with Selected Attributes

[1302]:
```
gradient_boosting_clf.fit(X_train_selected, y_train_selected)

y_test_pred = gradient_boosting_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[664   5]
 [  8  64]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9275
Recall =  0.8889
F1 Value =  0.9078
```

4.24 Ada Boost

4.25 Full Model

[1288]:
```
ada_boosting_clf.fit(X_train_all, y_train_all)

y_test_pred = ada_boosting_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[657  12]
 [ 20  52]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8125
Recall =  0.7222
F1 Value =  0.7647
```

4.26 Model with Selected Attributes

[1303]:
```
ada_boosting_clf.fit(X_train_selected, y_train_selected)

y_test_pred = ada_boosting_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[659  10]
 [ 15  57]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8507
Recall =  0.7917
F1 Value =  0.8201
```

4.27 Stacking

4.28 Full Model

[1289]:
```
Stacking_clf.fit(X_train_all, y_train_all)

y_test_pred = Stacking_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[666   3]
 [ 10  62]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9538
Recall =  0.8611
F1 Value =  0.9051
```

4.29 Model with Selected Attributes

[1304]:
```
Stacking_clf.fit(X_train_selected, y_train_selected)

y_test_pred = Stacking_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[666   3]
 [ 10  62]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.9538
Recall =  0.8611
F1 Value =  0.9051
```

4.30 KNN

4.31 Full Model

[1290]:
```
neigh_clf.fit(X_train_all, y_train_all)

y_test_pred = neigh_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[651  18]
 [ 45  27]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.6
Recall =  0.375
F1 Value =  0.4615
```

4.32 Model with Selected Attributes

[1305]:
```
neigh_clf.fit(X_train_selected, y_train_selected)

y_test_pred = neigh_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[648  21]
 [ 41  31]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.5962
Recall =  0.4306
F1 Value =  0.5
```

4.33 Logistic Regression

4.34 Full Model

[1291]:
```
logit_clf.fit(X_train_all, y_train_all)

y_test_pred = logit_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[660   9]
 [ 22  50]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8475
Recall =  0.6944
F1 Value =  0.7634
```

4.35 Model with Selected Attributes

[1306]:
```
logit_clf.fit(X_train_selected, y_train_selected)

y_test_pred = logit_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[659  10]
 [ 27  45]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8182
Recall =  0.625
F1 Value =  0.7087
```

4.36 CART

4.37 Full Model

[1292]:
```
cart_clf.fit(X_train_all, y_train_all)

y_test_pred = cart_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[658  11]
 [ 19  53]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.8281
Recall =  0.7361
F1 Value =  0.7794
```

4.38 Model with Selected Attributes

[1307]:
```
cart_clf.fit(X_train_selected, y_train_selected)

y_test_pred = cart_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[654  15]
 [ 14  58]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.7945
Recall =  0.8056
F1 Value =  0.8
```

4.39 Naïve Bayes (Gaussian)

4.40 Full Model

[1293]:
```
gnb_clf.fit(X_train_all, y_train_all)

y_test_pred = gnb_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[630  39]
 [ 41  31]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.4429
Recall =  0.4306
F1 Value =  0.4366
```

4.41 Model with Selected Attributes

[1308]:
```
gnb_clf.fit(X_train_selected, y_train_selected)

y_test_pred = gnb_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[630  39]
 [ 41  31]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.4429
Recall =  0.4306
F1 Value =  0.4366
```

4.42 Naïve Bayes (Multinomial)

4.43 Full Model

[1294]:
```
gnb_multi_clf.fit(X_train_all, y_train_all)

y_test_pred = gnb_multi_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[529 140]
 [ 29  43]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.235
Recall =  0.5972
F1 Value =  0.3373
```

4.44 Model with Selected Attributes

[1309]:
```
gnb_multi_clf.fit(X_train_selected, y_train_selected)

y_test_pred = gnb_multi_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[518 151]
 [ 20  52]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.2562
Recall =  0.7222
F1 Value =  0.3782
```

4.45 Naïve Bayes (Complement)

4.46 Full Model

[1295]:
```
gnb_comple_clf.fit(X_train_all, y_train_all)

y_test_pred = gnb_comple_clf.predict(X_test_all)

print_classification_report(y_test_all,y_test_pred)
```

```
=============Confusion Matrix =============
[[502 167]
 [ 27  45]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.2123
Recall =  0.625
F1 Value =  0.3169
```

4.47 Model with Selected Attributes

[1310]:
```
gnb_comple_clf.fit(X_train_selected, y_train_selected)

y_test_pred = gnb_comple_clf.predict(X_test_selected)

print_classification_report(y_test_selected,y_test_pred)
```

```
=============Confusion Matrix =============
[[344 325]
 [ 13  59]]

Perfect Prediction If Done
[[669   0]
 [  0  72]]

=============Sumarry Measures==============
Precision Score =  0.1536
Recall =  0.8194
F1 Value =  0.2588
```

5 Prediction of new data

[1418]:
```
def print_class(y_predict):
    y_predict=np.ndarray.flatten(y_predict)
    for i in range(len(y_predict)):
        if y_predict[i]==1:
            print (f"Prediction Against {i}th row is = Loan Given")
        else:
            print (f"Prediction Against {i}th row is = Loan Not Given")
```

5.1 Input data for prediction Here

Please enter the file name for prediction data set here

[1412]:
```
prediction_data = pd.read_csv('prediction.csv')
prediction_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Age                 10 non-null     int64  
 1   Experience          10 non-null     int64  
 2   Income              10 non-null     int64  
 3   Family              10 non-null     int64  
 4   CCAvg               10 non-null     float64
 5   Education           10 non-null     int64  
 6   Mortgage            10 non-null     int64  
 7   Securities Account  10 non-null     int64  
 8   CD Account          10 non-null     int64  
 9   Online              10 non-null     int64  
 10  CreditCard          10 non-null     int64  
dtypes: float64(1), int64(10)
memory usage: 1008.0 bytes
```

5.2 PreProcess Prediction data

We have seen full model performs very well. Hence Here prediction will assume only the full models

[1413]:
```
X_predict_all = preprocess_pipeline_all.transform(prediction_data)
```

[1414]:
```
X_predict_all.shape
```

[1414]:
```
(10, 20)
```

[1415]:
```
X_predict_selected = preprocess_pipeline_selected.transform(prediction_data)
```

[1416]:
```
X_predict_selected.shape
```

[1416]: (10, 12)

5.3 Full Model

[1417]:
```
svm_clf_poly.fit(X_train_all, y_train_all)
svm_clf_lin.fit(X_train_all, y_train_all)
svm_clf_rbf.fit(X_train_all, y_train_all)
svm_clf_sig.fit(X_train_all, y_train_all)

forest_clf.fit(X_train_all, y_train_all)
bagging_clf.fit(X_train_all, y_train_all)
gradient_boosting_clf.fit(X_train_all, y_train_all)
ada_boosting_clf.fit(X_train_all, y_train_all)

Stacking_clf.fit(X_train_all, y_train_all)
neigh_clf.fit(X_train_all, y_train_all)
logit_clf.fit(X_train_all, y_train_all)
cart_clf.fit(X_train_all, y_train_all)

gnb_clf.fit(X_train_all, y_train_all)
gnb_multi_clf.fit(X_train_all, y_train_all)
gnb_comple_clf.fit(X_train_all, y_train_all)
```

[1417]: ComplementNB()

5.4 SVM (Polynomial)

[1419]:
```
y_predict = svm_clf_poly.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

[1420]:
```
y_predict
```

[1420]: array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0], dtype=int64)

5.5 SVM (Linear)

[1422]:
```
y_predict = svm_clf_lin.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.6 SVM (RBF)

[1423]:
```
y_predict = svm_clf_rbf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.7 SVM (Sigmoid)

[1424]:
```
y_predict = svm_clf_sig.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
```

```
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.8 Random Forest

[1425]:
```
y_predict = forest_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.9 Bagging

[1426]:
```
y_predict = bagging_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.10 Gradient Boosting

[1427]:
```
y_predict=gradient_boosting_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
```

```
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.11 Ada Boost

[1428]:
```
y_predict=ada_boosting_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.12 Stacking

[1429]:
```
y_predict=Stacking_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.13 KNN

[1430]:
```
y_predict=neigh_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
```

```
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.14 Logistic

[1431]:
```
y_predict=logit_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.15 CART

[1432]:
```
y_predict=cart_clf.predict(X_predict_all)


print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.16 Naive Bayes Gaussian

[1433]:
```
y_predict=gnb_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Not Given
Prediction Against 9th row is = Loan Not Given
```

5.17 Naive Bayes Multinomial

[1434]:
```
y_predict=gnb_multi_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Not Given
Prediction Against 9th row is = Loan Not Given
```

5.18 Naive Bayes Complement

[1435]:
```
y_predict=gnb_comple_clf.predict(X_predict_all)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Given
Prediction Against 7th row is = Loan Not Given
```

```
Prediction Against 8th row is = Loan Not Given
Prediction Against 9th row is = Loan Not Given
```

5.19 Model with Selected Attributes

[1436]:
```
svm_clf_poly.fit(X_train_selected, y_train_selected)
svm_clf_lin.fit(X_train_selected, y_train_selected)
svm_clf_rbf.fit(X_train_selected, y_train_selected)
svm_clf_sig.fit(X_train_selected, y_train_selected)

forest_clf.fit(X_train_selected, y_train_selected)
bagging_clf.fit(X_train_selected, y_train_selected)
gradient_boosting_clf.fit(X_train_selected, y_train_selected)
ada_boosting_clf.fit(X_train_selected, y_train_selected)

Stacking_clf.fit(X_train_selected, y_train_selected)
neigh_clf.fit(X_train_selected, y_train_selected)
logit_clf.fit(X_train_selected, y_train_selected)
cart_clf.fit(X_train_selected, y_train_selected)

gnb_clf.fit(X_train_selected, y_train_selected)
gnb_multi_clf.fit(X_train_selected, y_train_selected)
gnb_comple_clf.fit(X_train_selected, y_train_selected)
```

[1436]:
```
ComplementNB()
```

5.20 SVM (Polynomial)

[1437]:
```
y_predict = svm_clf_poly.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.21 SVM (Linear)

[1438]:
```
y_predict = svm_clf_lin.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.22 SVM (RBF)

[1439]:
```
y_predict = svm_clf_rbf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.23 SVM (Sigmoid)

[1440]:
```
y_predict = svm_clf_sig.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Given
Prediction Against 7th row is = Loan Not Given
```

```
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.24 Random Forest

[1441]:
```python
y_predict = forest_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.25 Bagging

[1442]:
```python
y_predict = bagging_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.26 Gradient Boosting

[1443]:
```python
y_predict = gradient_boosting_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
```

```
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.27 Ada Boost

[1444]:
```
y_predict = ada_boosting_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.28 Stacking

[1445]:
```
y_predict = Stacking_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.29 KNN

[1446]:
```
y_predict = neigh_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
```

```
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Not Given
Prediction Against 9th row is = Loan Not Given
```

5.30 Logistic

[1447]:
```
y_predict = logit_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.31 CART

[1448]:
```
y_predict = cart_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Not Given
```

5.32 Naive Bayes Gaussian

[1449]:
```
y_predict = gnb_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Not Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Not Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Not Given
Prediction Against 9th row is = Loan Not Given
```

5.33 Naive Bayes Multinomial

[1450]:
```
y_predict = gnb_multi_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Not Given
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Given
```

5.34 Naive Bayes Complement

[1451]:
```
y_predict = gnb_comple_clf.predict(X_predict_selected)

print_class(y_predict)
```

```
Prediction Against 0th row is = Loan Given
Prediction Against 1th row is = Loan Not Given
Prediction Against 2th row is = Loan Not Given
Prediction Against 3th row is = Loan Not Given
Prediction Against 4th row is = Loan Given
Prediction Against 5th row is = Loan Not Given
Prediction Against 6th row is = Loan Not Given
Prediction Against 7th row is = Loan Given
```

```
Prediction Against 8th row is = Loan Given
Prediction Against 9th row is = Loan Given
```

6 Conclusion

The Full model works well. If data collection is not expensive, Full model should be preferred

6.1 Thank you