Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from scipy.stats import zscore
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from scipy.stats import zscore
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,BaggingClassifier,AdaBoostClassifier,GradientBoostingCl

from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import scipy.stats as stats

from sklearn import model_selection
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix,accuracy_score, classification_repo
```

Importing Data

In [2]:
```
df = pd.read_csv('sample_data_intw.csv')
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209593 entries, 0 to 209592
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   label                 209593 non-null  int64  
 1   msisdn                209593 non-null  object 
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64  
 10  cnt_ma_rech30         209593 non-null  int64  
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64  
 16  fr_ma_rech90          209593 non-null  int64  
 17  sumamnt_ma_rech90     209593 non-null  int64  
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64  
 23  fr_da_rech90          209593 non-null  int64  
 24  cnt_loans30           209593 non-null  int64  
 25  amnt_loans30          209593 non-null  int64  
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64  
 30  maxamnt_loans90       209593 non-null  int64  
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
 34  pcircle               209593 non-null  object 
 35  pdate                 209593 non-null  object 
```

```
dtypes: float64(21), int64(12), object(3)
memory usage: 57.6+ MB
```

In [4]:
```
df.head(10)
```

Out[4]:

	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_
0	0	21408I70789	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	
1	1	76462I70374	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	
2	1	17943I70372	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	
3	1	55773I70781	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	
4	1	03813I82730	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	
5	1	35819I70783	568.0	2257.362667	2261.460000	368.13	380.13	2.0	0.0	
6	1	96759I84459	545.0	2876.641667	2883.970000	335.75	402.90	13.0	0.0	
7	1	09832I90846	768.0	12905.000000	17804.150000	900.35	2549.11	4.0	55.0	
8	1	59772I84450	1191.0	90.695000	90.695000	2287.50	2287.50	1.0	0.0	
9	1	56331I70783	536.0	29.357333	29.357333	612.96	612.96	11.0	0.0	

10 rows × 36 columns

In [5]:
```
# Getting the count of unique values in each column
unique_value_counts = df.nunique().to_frame(name='Unique Value Count')
unique_value_counts
```

Out[5]:

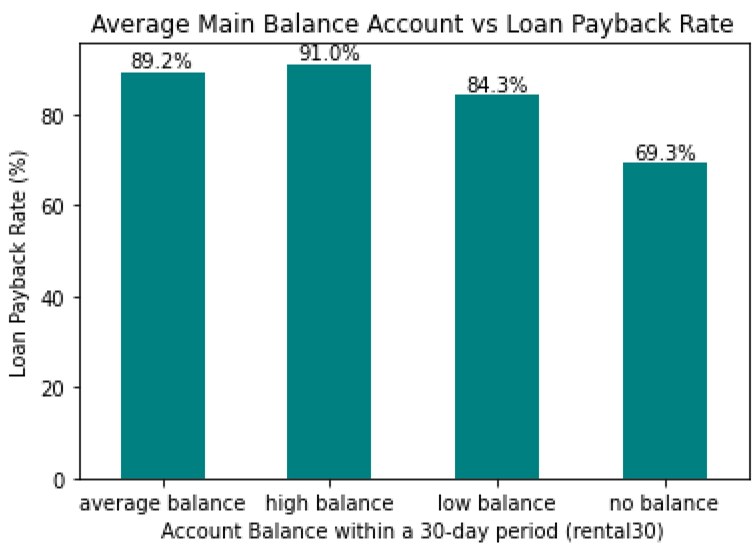
	Unique Value Count
label	2
msisdn	186243
aon	4507
daily_decr30	146328
daily_decr90	155483
rental30	131338
rental90	139036
last_rech_date_ma	1186
last_rech_date_da	1174
last_rech_amt_ma	70
cnt_ma_rech30	71
fr_ma_rech30	1083
sumamnt_ma_rech30	15141
medianamnt_ma_rech30	510
medianmarechprebal30	23907
cnt_ma_rech90	110
fr_ma_rech90	89
sumamnt_ma_rech90	31771
medianamnt_ma_rech90	608
medianmarechprebal90	22694
cnt_da_rech30	1066
fr_da_rech30	1072
cnt_da_rech90	27
fr_da_rech90	46
cnt_loans30	40

	Unique Value Count
amnt_loans30	48
maxamnt_loans30	1050
medianamnt_loans30	6
cnt_loans90	1110
amnt_loans90	69
maxamnt_loans90	3
medianamnt_loans90	6
payback30	1363
payback90	2381
pcircle	1
pdate	82

Columns that have only one unique value (low variance) across all rows are not useful for analysis as they do not contribute any information that can differentiate between rows. For example, pcircle has only 1 unique value, so it likely won't be useful in any analysis.

In [6]:
```
import matplotlib.pyplot as plt

# thresholds based on percentiles
low_balance_threshold = df['rental30'].quantile(0.25)
high_balance_threshold = df['rental30'].quantile(0.75)

# categorize balance based on thresholds
def categorize_balance(value):
    if value <= 0:
        return 'no balance'
    elif value <= low_balance_threshold:
        return 'low balance'
    elif value <= high_balance_threshold:
        return 'average balance'
    else:
        return 'high balance'
```

```
df['balance_group'] = df['rental30'].apply(categorize_balance)

#calculate the percentage of labels for each balance group
percentage = pd.crosstab(df['label'], df['balance_group'], normalize='columns').apply(lambda x: x * 100)
percentage = percentage.transpose()

plot_balance = percentage[1].plot(kind='bar', color='teal', figsize=(6, 4))

plt.title('Average Main Balance Account vs Loan Payback Rate')
plt.ylabel('Loan Payback Rate (%)')
plt.xlabel('Account Balance within a 30-day period (rental30)')
plt.xticks(rotation='horizontal')

for rec, label in zip(plot_balance.patches, percentage[1].round(1).astype(str)):
    plot_balance.text(rec.get_x() + rec.get_width() / 2,
                      rec.get_height() + 1,
                      label + '%',
                      ha='center',
                      color='black')

plt.show()
```

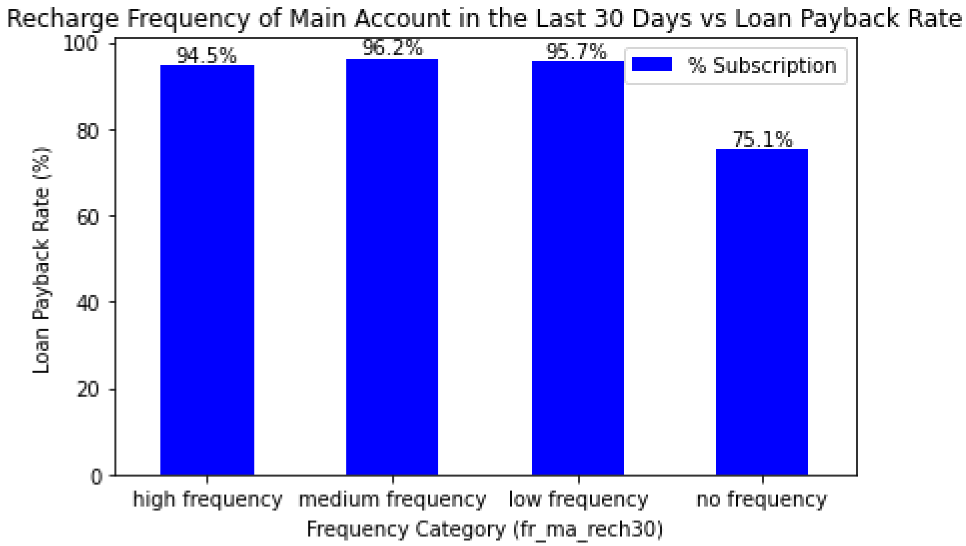


High balance leads to higher payback rate. Accounts with an average balance have an 89.2% payback rate, while those with a low balance have a significantly lower payback rate at 84.3%. This indicates that as the account balance decreases, so does the likelihood of loan repayment. No balance has the lowest payback rate. The graph illustrates a clear trend that the higher the account balance, the higher the loan payback rate

In [7]:

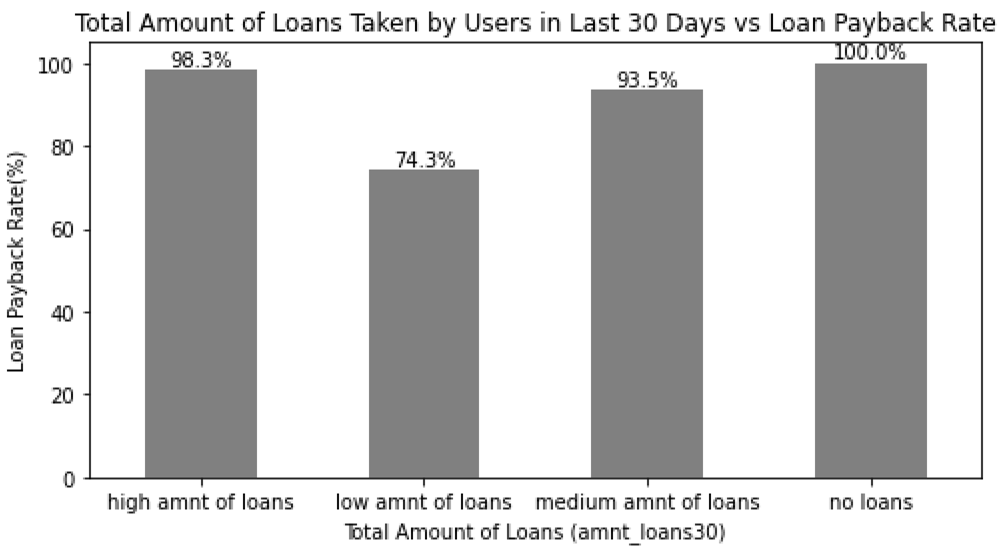
```
import matplotlib.pyplot as plt
import pandas as pd

# Define the frequency groups based on the 'fr_ma_rech30' column
df['frequency_group'] = pd.cut(
    df['fr_ma_rech30'],
    bins=[-1, 0, 1, 2, float('inf')],
    labels=['no frequency', 'low frequency', 'medium frequency', 'high frequency']
)
category_order = ['high frequency', 'medium frequency', 'low frequency', 'no frequency']
frequency_counts = df['frequency_group'].value_counts().reindex(category_order)
subscription_counts = df[df['label'] == 1]['frequency_group'].value_counts().reindex(category_order)
percent_subscriptions = (subscription_counts / frequency_counts) * 100

fre = pd.DataFrame({
    'Frequency Group': category_order,
    'Frequency Count': frequency_counts,
    'Subscription Count': subscription_counts,
    '% Subscription': percent_subscriptions
}).reset_index(drop=True)

ax = fre.plot(
    x='Frequency Group',
    y='% Subscription',
    kind='bar',
    color='blue',
    figsize=(6, 4)
)

plt.title('Recharge Frequency of Main Account in the Last 30 Days vs Loan Payback Rate')
plt.ylabel('Loan Payback Rate (%)')
plt.xlabel('Frequency Category (fr_ma_rech30)')
plt.xticks(rotation='horizontal')

for rec, label in zip(ax.patches, fre['% Subscription'].round(1).astype(str)):
```

```
    ax.text(
        rec.get_x() + rec.get_width() / 2,
        rec.get_height() + 1,
        label + '%',
        ha='center',
        color='black'
    )
plt.tight_layout()
plt.show()
```



In [8]:
```
# categorize loan amounts based on thresholds
low_loanamnt_threshold = df['amnt_loans30'].quantile(0.25)
high_loanamnt_threshold = df['amnt_loans30'].quantile(0.75)

def categorize_loanamnt(value):
    if value <= 0:
        return 'no loans'
    elif value <= low_loanamnt_threshold:
        return 'low amnt of loans'
    elif value <= high_loanamnt_threshold:
        return 'medium amnt of loans'
    else:
        return 'high amnt of loans'

df['loanamnt_frequency_group'] = df['amnt_loans30'].apply(categorize_loanamnt)
```

```
# Calculate the percentage of labels for each loan amount group
percentage = pd.crosstab(df['label'], df['loanamnt_frequency_group'], normalize='columns').apply(lambda x:
percentage = percentage.transpose()

plot_loanamnt = percentage[1].plot(kind='bar', color='gray', figsize=(8, 4))
plt.title('Total Amount of Loans Taken by Users in Last 30 Days vs Loan Payback Rate')
plt.ylabel('Loan Payback Rate(%)')
plt.xlabel('Total Amount of Loans (amnt_loans30)')
plt.xticks(rotation='horizontal')

for rec, label in zip(plot_loanamnt.patches, percentage[1].round(1).astype(str)):
    plot_loanamnt.text(
        rec.get_x() + rec.get_width() / 2,
        rec.get_height() + 1,
        label + '%',
        ha='center',
        color='black'
    )
plt.show()
```



In [9]:
```
df.shape
```

Out[9]:
```
(209593, 39)
```

In [10]:
```
# Separate categorical columns, we are going to drop non-numeric columns, and the columns we added earlier
non_num = [col for col in df.columns if df[col].dtype == "object"]
print(non_num)
```

```
['msisdn', 'pcircle', 'pdate', 'balance_group', 'loanamnt_frequency_group']
```

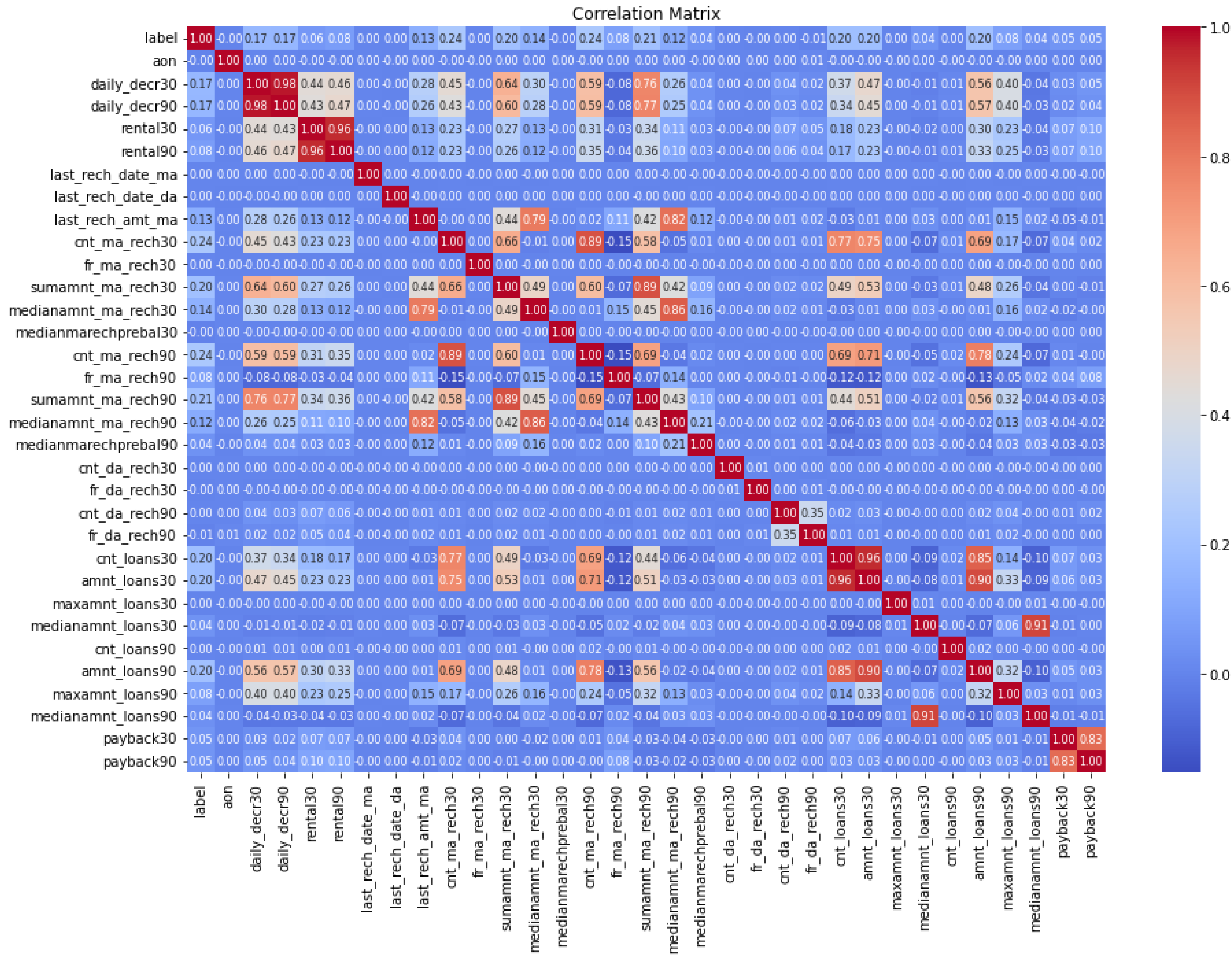
In [11]:
```
# dropping the column ['balance_group', 'frequency_group',      'loanamnt_frequency_group'that we added abo
# the features that we can ignore, 'pcircle', 'pdate', 'msisdn']
df = df.drop(columns=['balance_group', 'frequency_group',       'loanamnt_frequency_group', 'pcircle', 'pda
df.head(3)
```

Out[11]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cnt_m
0	0	272.0	3055.05	3065.15	220.13	260.13	2.0	0.0	1539	
1	1	712.0	12122.00	12124.75	3691.26	3691.26	20.0	0.0	5787	
2	1	535.0	1398.00	1398.00	900.13	900.13	3.0	0.0	1539	

3 rows × 33 columns

In [12]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix, considering only numeric columns
corr_matrix = df.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', annot_kws={"size": 8})
plt.title('Correlation Matrix')
plt.show()
```



In [13]:

```
example = df.copy()
# example['rental30'] = zscore(example['rental30'])
```

```
example

# outliers = (example['rental30'] > 3) | (example['rental30'] < -3)
# # df.loc[outliers, column] = df[column].median()
# outliers
```

Out[13]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma
0	0	272.0	3055.050000	3065.150000	220.13	260.13	2.0	0.0	1539
1	1	712.0	12122.000000	12124.750000	3691.26	3691.26	20.0	0.0	5787
2	1	535.0	1398.000000	1398.000000	900.13	900.13	3.0	0.0	1539
3	1	241.0	21.228000	21.228000	159.42	159.42	41.0	0.0	947
4	1	947.0	150.619333	150.619333	1098.90	1098.90	4.0	0.0	2309
...	...	...	...	...	...	...	...	...	..
209588	1	404.0	151.872333	151.872333	1089.19	1089.19	1.0	0.0	4048
209589	1	1075.0	36.936000	36.936000	1728.36	1728.36	4.0	0.0	773
209590	1	1013.0	11843.111670	11904.350000	5861.83	8893.20	3.0	0.0	1539
209591	1	1732.0	12488.228330	12574.370000	411.83	984.58	2.0	38.0	773
209592	1	1581.0	4489.362000	4534.820000	483.92	631.20	13.0	0.0	7526

209593 rows × 33 columns

In [14]:

```
# Data Standardization and Outlier Replacement
from scipy.stats import zscore
from scipy.stats import zscore
import re  # Import the regular expressions module

# Select columns that end with a number (30 or 90, etc.), excluding non-relevant columns like 'msisdn', 'p
pattern = re.compile(r'.*\d+$')
columns = [col for col in df.columns if pattern.match(col)]

# Standardize the columns, handle outliers, and apply cube root transformation
for column in columns:
    # The z-score normalization is applied to standardize the data, which centers the data around the mean
```

```
    # and scales it according to the standard deviation.
    df[column] = zscore(df[column])

    # Replace values that are outliers with the median of the column
    outliers = (df[column] > 3) | (df[column] < -3)
    print(sum(outliers))
    print('/n')
    df.loc[outliers, column] = df[column].median()

    # Apply cube root transformation
    df[column] = np.cbrt(df[column])
```

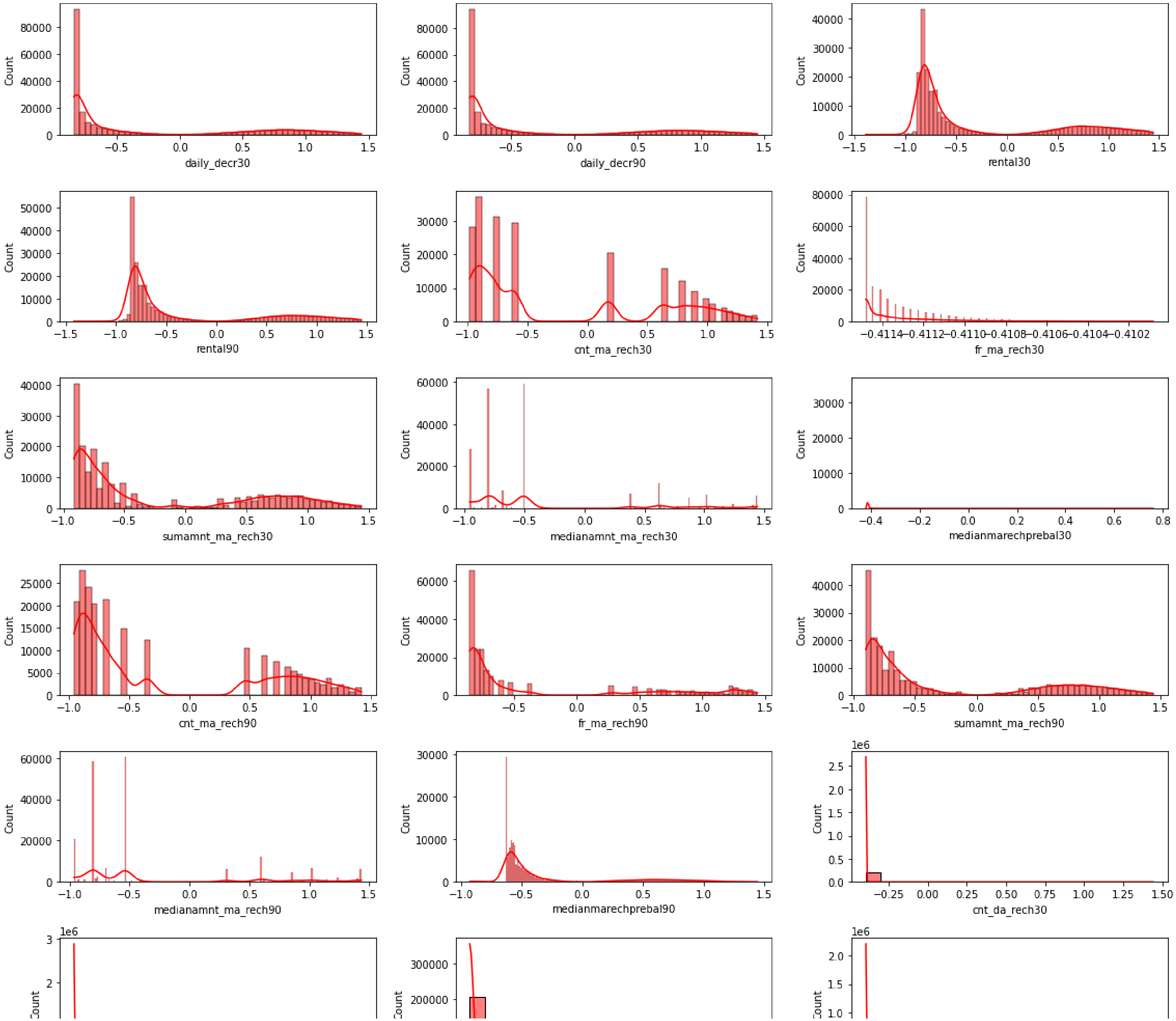
```
4168
/n
4263
/n
4471
/n
4605
/n
3766
/n
1047
/n
3617
/n
2973
/n
1047
/n
4047
/n
4707
/n
3916
/n
3020
/n
1185
/n
958
/n
1047
/n
1194
/n
733
/n
3973
/n
4311
/n
963
/n
```

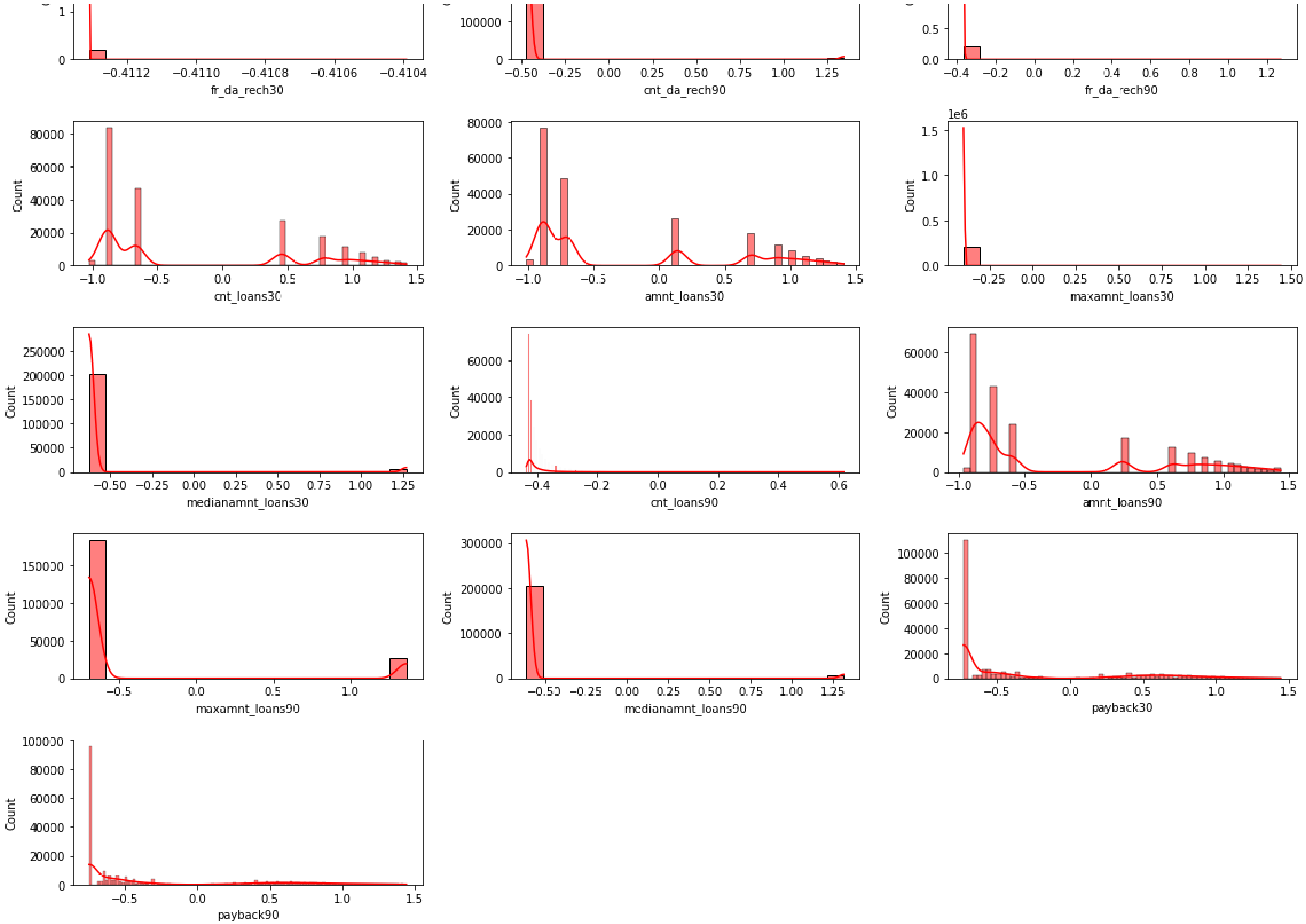
```
7610
/n
1047
/n
4164
/n
2043
/n
6501
/n
3152
/n
3657
/n
```

In [15]:
```
sum(outliers)
```

Out[15]:
```
3657
```

In [16]:
```
# The columns used for plotting are those in the DataFrame df that end with numbers
num_plots = len(columns)
# This ensures we have enough rows for all columns
num_rows = num_plots // 3 + (num_plots % 3 > 0)

fig, ax = plt.subplots(num_rows, 3, figsize=(16, num_rows * 2.5))
ax = ax.flatten()

for i, col in enumerate(columns):
    sns.histplot(df[col], ax=ax[i], color='red', kde=True)

# If the number of plots is not a multiple of 3, we hide the last few subplots which are not needed
for j in range(num_plots, len(ax)):
    fig.delaxes(ax[j])
plt.tight_layout()
plt.show()
```


daily_decr30
daily_decr90
rental30
rental90
cnt_ma_rech30
fr_ma_rech30
sumamnt_ma_rech30
medianamnt_ma_rech30
medianmarechprebal30
cnt_ma_rech90
fr_ma_rech90
sumamnt_ma_rech90
medianamnt_ma_rech90
medianmarechprebal90
cnt_da_rech30
Count


fr_da_rech30
cnt_da_rech90
fr_da_rech90
Count
cnt_loans30
amnt_loans30
1e6
maxamnt_loans30
medianamnt_loans30
cnt_loans90
amnt_loans90
maxamnt_loans90
medianamnt_loans90
payback30
payback90

In [17]:
```
df
```

Out[17]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_r
0	0	272.0	-0.631886	-0.651355	-0.830994	-0.823561	2.0	0.0	15
1	1	712.0	0.900837	0.820996	0.614277	0.330255	20.0	0.0	57
2	1	535.0	-0.755959	-0.754218	-0.746514	-0.764987	3.0	0.0	15
3	1	241.0	-0.834590	-0.821858	-0.837741	-0.832051	41.0	0.0	9
4	1	947.0	-0.827820	-0.815968	-0.717832	-0.744840	4.0	0.0	23
...	...	...	...	...	...	...	...	...	
209588	1	404.0	-0.827754	-0.815910	-0.719287	-0.745850	1.0	0.0	40
209589	1	1075.0	-0.833774	-0.821147	-0.607129	-0.672501	4.0	0.0	7
209590	1	1013.0	0.888238	0.810890	0.902695	0.978717	3.0	0.0	15
209591	1	1732.0	0.916865	0.840875	-0.808937	-0.756558	2.0	38.0	7
209592	1	1581.0	-0.459066	-0.521399	-0.800323	-0.790661	13.0	0.0	75

209593 rows × 33 columns

In [18]:
```
feature_columns = ['aon', 'daily_decr30', 'daily_decr90', 'rental30', 'rental90',
       'last_rech_date_ma', 'last_rech_date_da', 'last_rech_amt_ma',
       'cnt_ma_rech30', 'fr_ma_rech30', 'sumamnt_ma_rech30',
       'medianamnt_ma_rech30', 'medianmarechprebal30', 'cnt_ma_rech90',
       'fr_ma_rech90', 'sumamnt_ma_rech90', 'medianamnt_ma_rech90',
       'medianmarechprebal90', 'cnt_da_rech30', 'fr_da_rech30',
       'cnt_da_rech90', 'fr_da_rech90', 'cnt_loans30', 'amnt_loans30',
       'maxamnt_loans30', 'medianamnt_loans30', 'cnt_loans90', 'amnt_loans90',
       'maxamnt_loans90', 'medianamnt_loans90', 'payback30', 'payback90']
feature_data = df[feature_columns]
```

In [19]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MaxAbsScaler

# Exclude non-numeric columns from scaling
numeric_df = feature_data.select_dtypes(include=[np.number])
```

```python
scaler = MaxAbsScaler()
scaled_numeric_df = scaler.fit_transform(numeric_df)

# Apply PCA
pca = PCA(n_components=15)
principal_components = pca.fit_transform(numeric_df)
scaled_numeric_df
```

Out[19]:
```
array([[ 2.72037880e-04, -4.38131664e-01, -4.51638957e-01, ...,
        -4.66482524e-01,  9.89379453e-01,  9.28013310e-01],
       [ 7.12099156e-04,  6.24614818e-01,  5.69265213e-01, ...,
        -4.66482524e-01, -5.04722414e-01, -5.19189072e-01],
       [ 5.35074506e-04, -5.24160395e-01, -5.22962229e-01, ...,
        -4.66482524e-01, -5.04722414e-01, -5.19189072e-01],
       ...,
       [ 1.01314107e-03,  6.15878846e-01,  5.62257692e-01, ...,
        -4.66482524e-01,  2.83319339e-01, -2.50976384e-01],
       [ 1.73224121e-03,  6.35728051e-01,  5.83048988e-01, ...,
        -4.66482524e-01, -5.04722414e-01,  5.84892289e-01],
       [ 1.58122018e-03, -3.18303174e-01, -3.61529641e-01, ...,
        -4.66482524e-01, -5.04722414e-01, -5.19189072e-01]])
```

In [20]:
```python
len(scaled_numeric_df[0])
```

Out[20]:
```
32
```

In [21]:
```python
numeric_df.shape
```

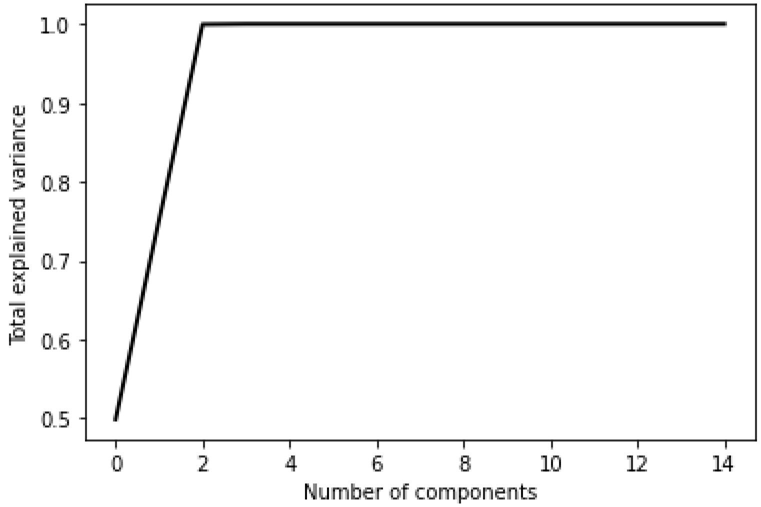
Out[21]:
```
(209593, 32)
```

In [22]:
```python
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_sum = np.cumsum(pca.explained_variance_ratio_)
cumulative_sum
```

Out[22]:
```
array([0.49867821, 0.75161412, 0.99951084, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ])
```

In [23]:
```python
# Plot the cumulative sum of explained variance
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(6, 4))
plt.plot(cumulative_sum , color='k', lw=2)
plt.xlabel('Number of components')
plt.ylabel('Total explained variance')
plt.show()
```



In [24]:
```
# Apply PCA
pca = PCA(n_components=3)
pca.fit(scaled_numeric_df)
df_transformed = pca.transform(scaled_numeric_df)
```

In [25]:
```
df_transformed.shape
```

Out[25]: (209593, 3)

Training Data

In [26]:
```
X1=df_transformed
y1=df['label']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3, random_state=1)

print('X_train', X_train.shape)
```

```
print('X_test', X_test.shape)
print('y_train', y_train.shape)
print('y_test', y_test.shape)
```

```
X_train (146715, 3)
X_test (62878, 3)
y_train (146715,)
y_test (62878,)
```

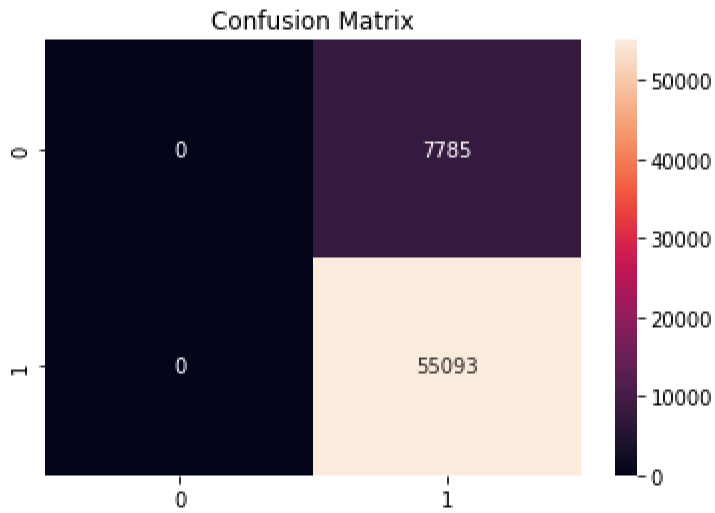
Accuracy

Logistic Regression

In [27]:
```
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
model1.fit(X_train, y_train)

print('Training score =', model1.score(X_train, y_train))
print('Test score =', model1.score(X_test, y_test))
```

```
Training score = 0.874743550420884
Test score = 0.8761888100766564
```

In [28]:
```
from sklearn.metrics import confusion_matrix
ypred = model1.predict(X_test)
cm = confusion_matrix(y_test, ypred)
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.show()
```



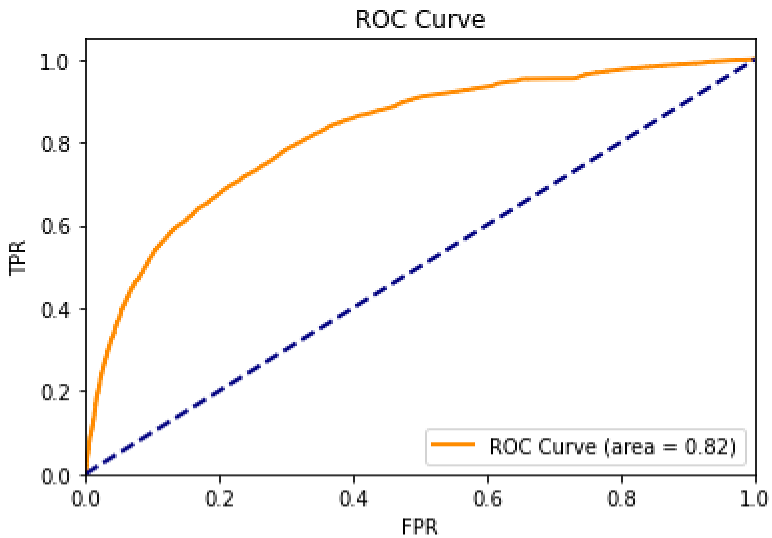
In [29]:
```
tn = cm[0,0]  #True Negative
tp = cm[1,1]  #True Positives
fp = cm[0,1]  #False Positives
fn = cm[1,0]  #False Negatives

accuracy = (tp+tn)/(tp+fn+fp+tn)
precision = tp / (tp+fp)
recall = tp / (tp+fn)
f1 = 2*precision*recall / (precision+recall)

print('Accuracy =',accuracy)
print('Precision =', precision)
print('Recall =', recall)
print('F1 Score =', f1)
```

```
Accuracy = 0.8761888100766564
Precision = 0.8761888100766564
Recall = 1.0
F1 Score = 0.9340092056522366
```

In [30]:
```
from sklearn.metrics import roc_curve,roc_auc_score
ypred = model1.predict_proba(X_test)
fpr,tpr,threshold = roc_curve(y_test,ypred[:,1])
roc_auc = roc_auc_score(y_test,ypred[:,1])

print('ROC AUC =', roc_auc)
```

```
plt.figure()
lw = 2
plt.plot(fpr,tpr,color='darkorange',lw=lw,label='ROC Curve (area = %0.2f)'%roc_auc)
plt.plot([0,1],[0,1],color='navy',lw=lw,linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
ROC AUC = 0.8197284614824416
```



Random Forest Classifier

In [31]:
```
RFCtest = RandomForestClassifier().fit(X_train,y_train)
rfc_predictions = RFCtest.predict(X_test)
acc_rfc = RFCtest.score(X_test, y_test)
print('The Random Forest Algorithm has an accuracy of', acc_rfc)
f1 = f1_score(y_test, rfc_predictions, average='weighted')  # Use 'binary' for binary classification
print("F1 Score:", f1)
```

```
The Random Forest Algorithm has an accuracy of 0.8932695060275454
F1 Score: 0.8800145997807787
```

K Nearest Neighbors Model

In [32]:
```
KNCtest = KNeighborsClassifier().fit(X_train,y_train)
knc_predictions = KNCtest.predict(X_test)
acc_knc = KNCtest.score(X_test, y_test)
print('The K Neighbors Algorithm has an accuracy of', acc_knc)
f1 = f1_score(y_test, knc_predictions, average='weighted')  # Use 'binary' for binary classification
print("F1 Score:", f1)
```

```
The K Neighbors Algorithm has an accuracy of 0.8891822258977703
F1 Score: 0.8782227242343663
```

Decision Tree Classifier

In [33]:
```
DTCtest = DecisionTreeClassifier().fit(X_train,y_train)
dtc_predictions = DTCtest.predict(X_test)
acc_dtc = DTCtest.score(X_test, y_test)
print('The Decision Tree Algorithm has an accuracy of', acc_dtc)
f1 = f1_score(y_test, dtc_predictions, average='weighted')  # Use 'binary' for binary classification
print("F1 Score:", f1)
```

```
The Decision Tree Algorithm has an accuracy of 0.8461146983046535
F1 Score: 0.8476895538986121
```

In [34]:
```
NB_Model = MultinomialNB()
RFC_Model = RandomForestClassifier()
SVC_Model = SVC()
KNC_Model = KNeighborsClassifier()
DTC_Model = DecisionTreeClassifier()
```

In [35]:
```
vector = CountVectorizer()
```

In [36]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
knn=KNeighborsClassifier()
param={'n_neighbors':np.arange(5,30),'weights':['uniform','distance']}
GS=RandomizedSearchCV(knn,param,cv=3,scoring='f1_weighted',n_jobs=-1)
GS.fit(X_train,y_train)
```

Out[36]:
```
▸ RandomizedSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

In [37]:
```
GS.best_params_
```

Out[37]: {'weights': 'distance', 'n_neighbors': 24}

In [38]:
```
dt=DecisionTreeClassifier(random_state=0)
```

In [39]:
```
param={'max_depth':np.arange(3,50),'criterion':['entropy','gini'],'min_samples_leaf':np.arange(3,20)}
GS=RandomizedSearchCV(dt,param,cv=3,scoring='f1_weighted')
GS.fit(X_train,y_train)
```

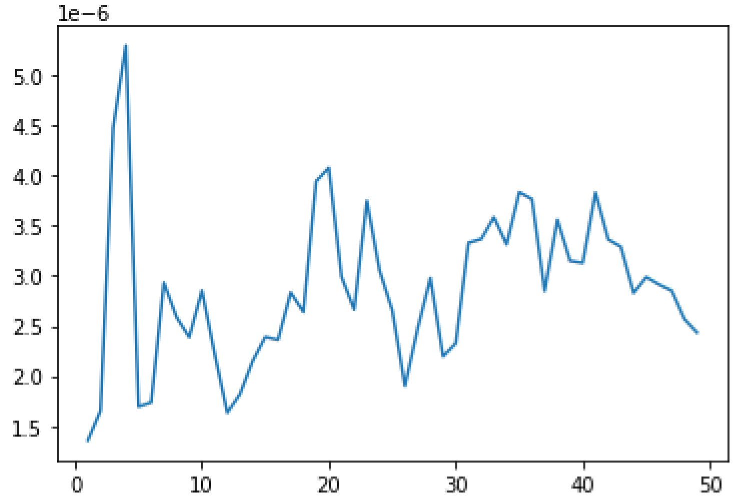
Out[39]:
```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [40]:
```
LR=LogisticRegression()
NB=GaussianNB()
KNN=KNeighborsClassifier(n_neighbors=5,weights='distance')
DT=DecisionTreeClassifier(criterion='gini',max_depth=14,min_samples_leaf=19,random_state=0)
RF=RandomForestClassifier(criterion='entropy',n_estimators=7,random_state=0)
```

In [41]:
```
RF_var=[]
for val in np.arange(1,50):
  RF=RandomForestClassifier(criterion='gini',n_estimators=val,random_state=0)
  kfold = model_selection.KFold(shuffle=True,n_splits=3,random_state=0)
  cv_results = model_selection.cross_val_score(RF, X_train,y_train,cv=kfold, scoring='f1_weighted',n_jobs=-
  RF_var.append(np.var(cv_results,ddof=1))
```

In [42]:
```
x_axis=np.arange(1,50)
plt.plot(x_axis,RF_var)
```

Out[42]: [<matplotlib.lines.Line2D at 0x1399d2160>]



In [43]:
```
param={'max_depth':np.arange(3,50),'criterion':['entropy','gini'],'min_samples_leaf':np.arange(3,20)}
GS=RandomizedSearchCV(dt,param,cv=3,scoring='f1_weighted')
GS.fit(X_train,y_train)
```

Out[43]:
```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [44]:
```
GS.best_params_
```

Out[44]: {'min_samples_leaf': 17, 'max_depth': 38, 'criterion': 'entropy'}

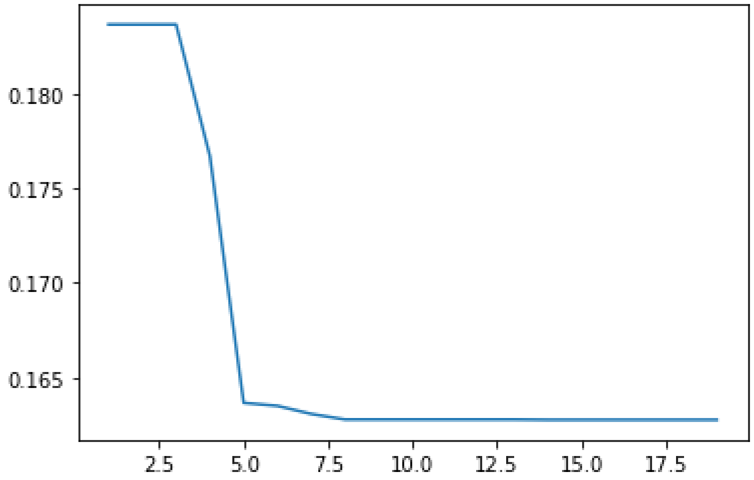
Ada Boost Classifier

In [45]:
```
Ada_bias=[]
for val in np.arange(1,20):
  Ada=AdaBoostClassifier(n_estimators=val,random_state=0)
  kfold = model_selection.KFold(shuffle=True,n_splits=3,random_state=0)
  cv_results = model_selection.cross_val_score(Ada, X_train, y_train,cv=kfold, scoring='f1_weighted',n_job
```

```
  Ada_bias.append(1-np.mean(cv_results))
  #print(val,1-np.mean(cv_results))
```

In [46]:
```
x_axis=np.arange(1,20)
plt.plot(x_axis,Ada_bias)
```

Out[46]: [<matplotlib.lines.Line2D at 0x13939e5b0>]



In [47]:
```
np.argmin(Ada_bias)
```

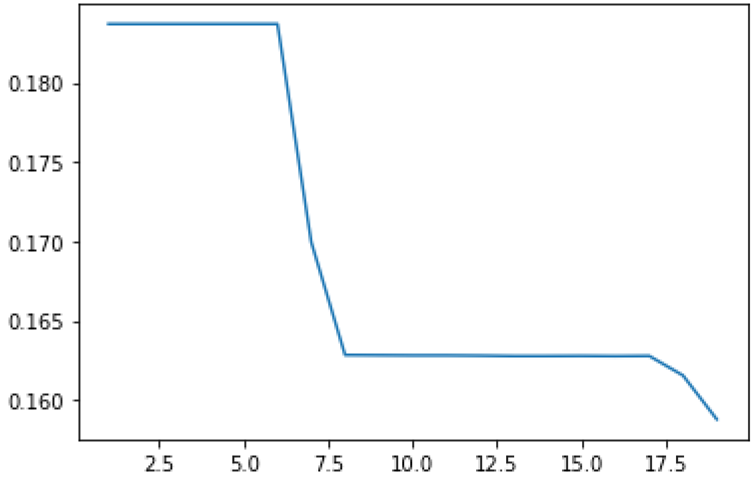
Out[47]: 13

Gradient Boosting Classifier

In [48]:
```
GB_bias=[]
for val in np.arange(1,20):
  gb=GradientBoostingClassifier(n_estimators=val)
  kfold = model_selection.KFold(shuffle=True,n_splits=3,random_state=0)
  cv_results = model_selection.cross_val_score(gb, X_train, y_train,cv=kfold, scoring='f1_weighted',n_jobs=
  GB_bias.append(1-np.mean(cv_results))
  #print(val,1-np.mean(cv_results))
```

In [49]:
```
x_axis=np.arange(1,20)
plt.plot(x_axis,GB_bias)
```

Out[49]: [<matplotlib.lines.Line2D at 0x139401be0>]



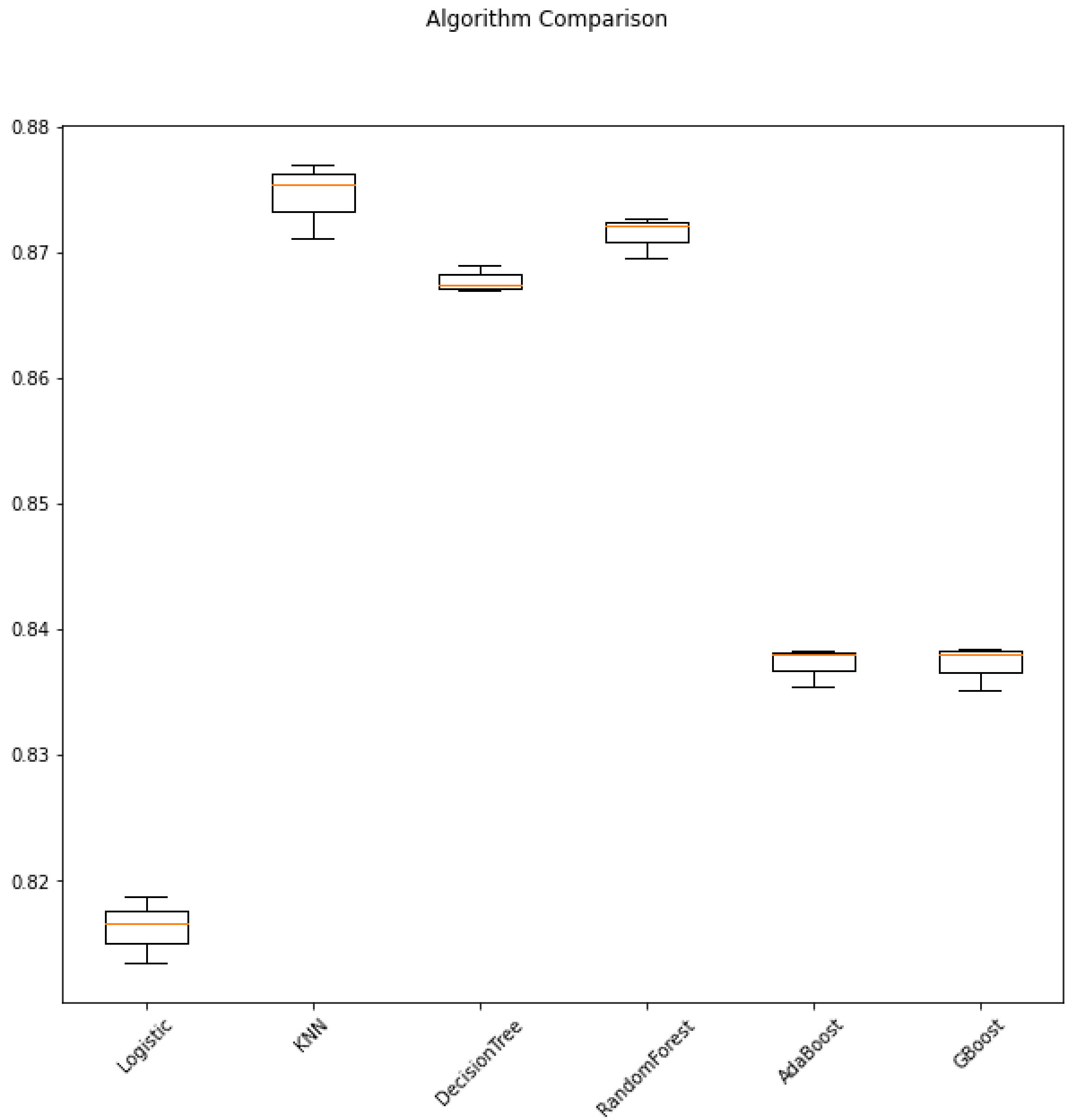
Comparing Models

In [50]:
```
LR=LogisticRegression()
NB=GaussianNB()
KNN=KNeighborsClassifier(n_neighbors=5,weights='distance')
DT=DecisionTreeClassifier(criterion='gini',max_depth=14,min_samples_leaf=19,random_state=0)
RF=RandomForestClassifier(criterion='entropy',n_estimators=7,random_state=0)
Bag=BaggingClassifier(n_estimators=3,random_state=0)
AB=AdaBoostClassifier(n_estimators=16,random_state=0)
GB=GradientBoostingClassifier(n_estimators=17)


models = []
models.append(('Logistic', LR))

models.append(('KNN',KNN))
models.append(('DecisionTree',DT))
models.append(('RandomForest',RF))

models.append(('AdaBoost',AB))
models.append(('GBoost',GB))
```

Results

In [51]:
```
results = []
names = []
for name, model in models:
        kfold = model_selection.KFold(shuffle=True,n_splits=3,random_state=0)
        cv_results = model_selection.cross_val_score(model, X_train, y_train,cv=kfold, scoring='f1_weighted
        results.append(cv_results)
        names.append(name)
        print("%s: %f (%f)" % (name, np.mean(cv_results),np.var(cv_results,ddof=1)))
   # boxplot algorithm comparison
fig = plt.figure(figsize=(10,9))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names,rotation=45)
plt.show()
```

```
Logistic: 0.816300 (0.000007)
KNN: 0.874537 (0.000009)
DecisionTree: 0.867824 (0.000001)
RandomForest: 0.871457 (0.000003)
AdaBoost: 0.837251 (0.000003)
GBoost: 0.837215 (0.000003)
```



In [52]:
```
KNN.fit(X_train,y_train)
```

Out[52]:

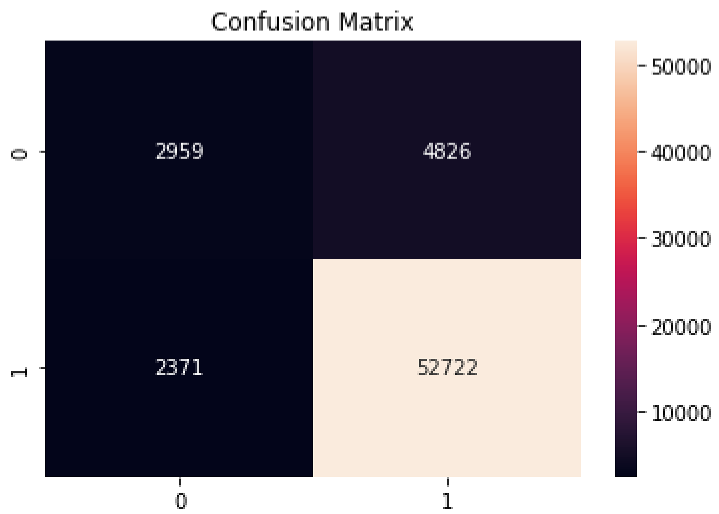
```
▾ KNeighborsClassifier
KNeighborsClassifier(weights='distance')
```

In [53]:

```
predictions = KNN.predict(X_test)
```

In [54]:

```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.show()
```



In [55]:

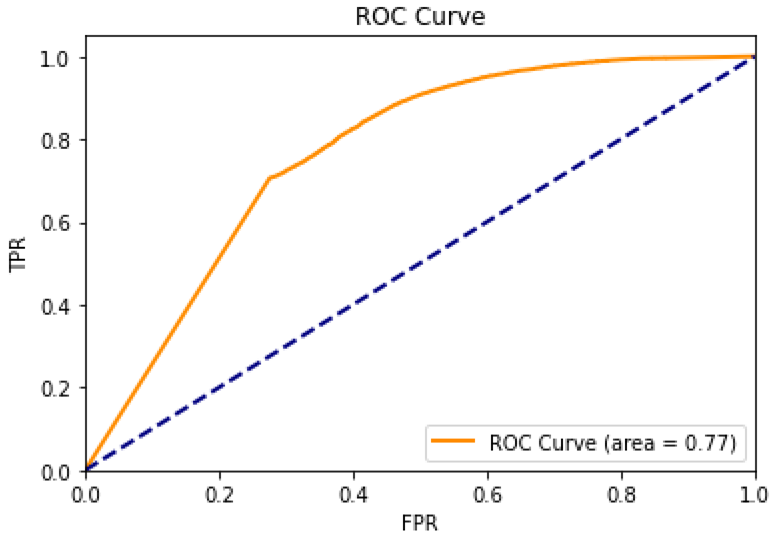
```
tn = cm[0,0]  #True Negative
tp = cm[1,1]  #True Positives
fp = cm[0,1]  #False Positives
fn = cm[1,0]  #False Negatives

accuracy = (tp+tn)/(tp+fn+fp+tn)
precision = tp / (tp+fp)
recall = tp / (tp+fn)
f1 = 2*precision*recall / (precision+recall)

print('Accuracy =',accuracy)
print('Precision =', precision)
```

```
print('Recall =', recall)
print('F1 Score =', f1)
```

```
Accuracy = 0.8855402525525621
Precision = 0.9161395704455411
Recall = 0.956963679596319
F1 Score = 0.936106746211415
```

In [56]:
```
from sklearn.metrics import roc_curve,roc_auc_score
ypred = KNN.predict_proba(X_test)
fpr,tpr,threshold = roc_curve(y_test,ypred[:,1])
roc_auc = roc_auc_score(y_test,ypred[:,1])

print('ROC AUC =', roc_auc)
plt.figure()
lw = 2
plt.plot(fpr,tpr,color='darkorange',lw=lw,label='ROC Curve (area = %0.2f)'%roc_auc)
plt.plot([0,1],[0,1],color='navy',lw=lw,linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
ROC AUC = 0.766416444589327
```



Based on the aforementioned outcomes, it is evident that the K Nearest Neighbor model outperforms the others. Through a comprehensive comparison of bias error and variance error across all algorithms, it is concluded that KNN is the most effective, and therefore, it will be employed for predicting loan defaulters.

```
In [ ]:
```