Churn Prediction Model

June 26, 2024

1 Importing Necessary Libraries

[19]:
```
import pandas as pd #Data Manipulation
import numpy as np
import seaborn as sns #Data Visualization
import matplotlib.pyplot as plt #Data Visualization
import plotly.express as px #Data Visualization
import plotly.graph_objects as go #Data Visualization
```

2 Load the Dataset

[20]:
```
df = pd.read_csv("Churn_ Data.csv")
```

3 Understanding the Dataset

[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Columns: 111 entries, s6.new.rev.p2.m2 to s3.rev.p1
dtypes: float64(80), int64(31)
memory usage: 21.2 MB
```

[22]:
```
df.head()
```

[22]:
```
   s6.new.rev.p2.m2  s1.new.rev.m1  s3.og.rev.4db.p5  s3.new.rev.4db.p5  \
0             -0.76        88.0482          3.106604           3.754955
1             -0.98        67.5039          3.094574           5.550865
2             -0.98        33.9248          2.324016           2.438114
3             -0.92        82.6780          2.630749           2.858961
4             -0.97        96.8379          2.674316           2.912397

   s4.usg.ins.p2  s4.og.unq.any.p2  s2.rch.val.p6  s1.og.rev.all.m1  \
0              4                14          39.29            57.320
1              1                 2          21.67            38.700
```

```
2              2                  3           30.00             15.320
3              2                  3           50.00             51.956
4              3                  2           22.50             66.886

   s8.new.rev.p6  s4.loc.ic.ins.p1  …  prop.og.mou.tot.mou.all.p6  \
0          -0.17                 1  …                    0.454642
1          -0.32                 3  …                    0.343190
2          -0.05                 3  …                    0.101838
3          -0.18                 4  …                    0.066602
4           0.01                 4  …                    0.219821

   prop.i2i.og.mou.p6  s4.loc.ic.ins.p2  s4.std.ic.ins.l14  \
0            0.497397                 4                  0
1            0.767617                 6                  0
2            0.619034                 6                  1
3            0.437088                 7                  2
4            0.585977                 6                  1

   s4.low.blnc.ins.p4  s3.og.rev.all.m2  s3.new.rev.m2  prop.og.mou.any.p6  \
0                   9              6.02           8.20           46.465636
1                  20              3.66           8.10           34.525456
2                  19              4.33           4.36           10.298451
3                  11              3.40           3.53            6.670783
4                  14              3.85           3.87           21.998905

   prop.loc.i2i.mou.og.mou.p3  s3.rev.p1
0                    0.609456       0.22
1                    1.000000       0.38
2                    0.699592       0.11
3                    0.086617       5.18
4                    0.683105       0.10

[5 rows x 111 columns]
```

[23]:
```
df.tail()
```

[23]:
```
       s6.new.rev.p2.m2  s1.new.rev.m1  s3.og.rev.4db.p5  s3.new.rev.4db.p5  \
24995              0.21       132.0365          2.652236           2.857739
24996              0.80        77.0154          3.763389           5.012503
24997              0.01       148.8337          3.823940           4.334250
24998              0.17      1012.4398         14.667580          14.579567
24999             -1.00       275.3530          5.134579           5.954062

       s4.usg.ins.p2  s4.og.unq.any.p2  s2.rch.val.p6  s1.og.rev.all.m1  \
24995              5                 8          26.67           123.396
24996              2                 8          27.88            62.140
24997              6                10          10.00            98.900
```

```
24998              7                 67          42.92            734.005
24999              1                  1          53.50            250.340

       s8.new.rev.p6  s4.loc.ic.ins.p1  …  prop.og.mou.tot.mou.all.p6  \
24995          -0.16                 4  …                    0.145831
24996           0.19                 4  …                    0.529829
24997          -0.03                 2  …                    0.327245
24998           0.70                 4  …                    0.824671
24999          -0.48                 4  …                    0.377281

       prop.i2i.og.mou.p6  s4.loc.ic.ins.p2  s4.std.ic.ins.l14  \
24995            0.200151                 7                  0
24996            0.169835                 7                  0
24997            0.407944                 3                  0
24998            0.889239                 7                  1
24999            0.609046                 7                  0

       s4.low.blnc.ins.p4  s3.og.rev.all.m2  s3.new.rev.m2  \
24995                  18              3.57           3.83
24996                  18              6.89           7.70
24997                  12              6.63           7.48
24998                   1             19.36          22.26
24999                  18              5.42           8.02

       prop.og.mou.any.p6  prop.loc.i2i.mou.og.mou.p3  s3.rev.p1
24995           14.896154                    0.328027       0.76
24996           55.156230                    0.288006      12.74
24997           33.222018                    0.235918       8.07
24998           82.549378                    0.952962      21.21
24999           38.590040                    1.000000       0.00

[5 rows x 111 columns]
```

[24]:
```python
print(df.shape)
```

```
(25000, 111)
```

[25]:
```python
df.describe().T
```

[25]:
```
                              count        mean         std       min  \
s6.new.rev.p2.m2            25000.0   -0.003730    2.727916 -1.000000
s1.new.rev.m1               25000.0  281.073083  276.075983  0.000000
s3.og.rev.4db.p5            25000.0    4.890003    4.212452  0.000000
s3.new.rev.4db.p5           25000.0    7.070194    6.318992  0.000833
s4.usg.ins.p2               25000.0    5.460080    2.184444  0.000000
…                                 …           …           …         …
s3.og.rev.all.m2            25000.0    8.008660    6.152429  0.000000
```

```
s3.new.rev.m2             25000.0   12.540182   11.540611  0.000000
prop.og.mou.any.p6        25000.0   53.594165   21.408486  0.000000
prop.loc.i2i.mou.og.mou.p3 25000.0   0.483975    0.292349  0.000000
s3.rev.p1                 25000.0    9.951366   17.648128  0.000000

                                 25%         50%         75%          max
s6.new.rev.p2.m2           -0.580000   -0.170000    0.280000   316.860000
s1.new.rev.m1             101.563800  204.859600  370.711650  5702.924300
s3.og.rev.4db.p5            2.367288    3.729944    5.993342   153.221695
s3.new.rev.4db.p5           3.318825    5.231268    8.395736   170.200441
s4.usg.ins.p2               5.000000    7.000000    7.000000     7.000000
…                                …           …           …            …
s3.og.rev.all.m2            4.207500    6.345000    9.830000   171.780000
s3.new.rev.m2               6.167500    9.350000   14.620000   386.480000
prop.og.mou.any.p6         39.378142   53.976203   68.312416   100.000000
prop.loc.i2i.mou.og.mou.p3  0.251304    0.477621    0.716538     1.000000
s3.rev.p1                   1.970000    5.380000   11.400000   585.500000

[111 rows x 8 columns]
```

4 Processing of the Data

4.0.1 Check for Misclassified Data Types

[26]:
```python
misclassified_columns = []
for col in df.columns:
    if df[col].dtype == 'object':
        try:
            df[col] = pd.to_numeric(df[col])
        except ValueError:
            misclassified_columns.append(col)
print("Misclassified columns:", misclassified_columns)
```

```
Misclassified columns: []
```

4.0.2 Check for NULL Values

[27]:
```python
df.isnull().sum()
df.isnull().sum() / df.shape[0] * 100
```

[27]:
```
s6.new.rev.p2.m2           0.0
s1.new.rev.m1              0.0
s3.og.rev.4db.p5           0.0
s3.new.rev.4db.p5          0.0
s4.usg.ins.p2              0.0
                          …
s3.og.rev.all.m2           0.0
```

```
s3.new.rev.m2               0.0
prop.og.mou.any.p6          0.0
prop.loc.i2i.mou.og.mou.p3  0.0
s3.rev.p1                   0.0
Length: 111, dtype: float64
```

4.0.3 Check for Duplicate Values

[28]:
```
duplicates = df.duplicated().sum()
df = df.drop_duplicates()
print(f"Removed {duplicates} duplicate rows")
print("New shape:", df.shape)
```

```
Removed 0 duplicate rows
New shape: (25000, 111)
```

4.0.4 Check for Unique values

[29]:
```
unique_value_columns = [col for col in df.columns if df[col].nunique() == df.
 ↪shape[0]]
df = df.drop(columns=unique_value_columns)
print("Removed unique value columns:", unique_value_columns)
print("New shape:", df.shape)
```

```
Removed unique value columns: []
New shape: (25000, 111)
```

4.0.5 Check for Zero Variance variables and Removal

[30]:
```
zero_variance_columns = [col for col in df.columns if df[col].std() == 0]
df = df.drop(columns=zero_variance_columns)
print("Removed zero variance columns:", zero_variance_columns)
print("New shape:", df.shape)
```

```
Removed zero variance columns: []
New shape: (25000, 111)
```

4.0.6 Outliers treatment (using IQR method and +/- Sigma Approach)

[31]:
```
# Step 4.6: Outliers treatment (using IQR method and +/- 3 Sigma Approach)
def treat_outliers(df):
  for column in df.select_dtypes(include=[np.number]).columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    # IQR method
```

```
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # +/- 3 Sigma approach (assuming normal distribution)
    mean = df[column].mean()
    std = df[column].std()
    sigma_lower_bound = mean - 3 * std
    sigma_upper_bound = mean + 3 * std

    # Combining IQR and sigma methods (consider the more extreme bound)
    final_lower_bound = max(lower_bound, sigma_lower_bound)
    final_upper_bound = min(upper_bound, sigma_upper_bound)

    df[column] = np.clip(df[column], final_lower_bound, final_upper_bound)
  return df

df = treat_outliers(df)
print(df.describe())
print("New shape:", df.shape)
```

```
       s6.new.rev.p2.m2  s1.new.rev.m1  s3.og.rev.4db.p5  s3.new.rev.4db.p5  \
count      25000.000000   25000.000000      25000.000000       25000.000000
mean          -0.083814     263.414113          4.559075           6.492778
std            0.668743     211.173061          2.897105           4.219776
min           -1.000000       0.000000          0.000000           0.000833
25%           -0.580000     101.563800          2.367288           3.318825
50%           -0.170000     204.859600          3.729944           5.231268
75%            0.280000     370.711650          5.993342           8.395736
max            1.570000     774.433425         11.432423          16.011101

       s4.usg.ins.p2  s4.og.unq.any.p2  s2.rch.val.p6  s1.og.rev.all.m1  \
count   25000.000000      25000.000000   25000.000000      25000.000000
mean        5.624880         27.137520      67.150729        201.146059
std         1.818213         23.163585      46.538585        167.480617
min         2.000000          0.000000       0.000000          0.000000
25%         5.000000          9.000000      33.000000         74.420000
50%         7.000000         21.000000      52.260000        151.168500
75%         7.000000         39.000000      89.852500        284.265000
max         7.000000         84.000000     175.131250        599.032500

       s8.new.rev.p6  s4.loc.ic.ins.p1  …  prop.og.mou.tot.mou.all.p6  \
count   25000.000000      25000.000000  …                25000.000000
mean       -0.026442          3.374020  …                    0.538407
std         0.253097          0.915725  …                    0.209203
min        -0.565000          1.500000  …                    0.000000
25%        -0.160000          3.000000  …                    0.394227
50%        -0.020000          4.000000  …                    0.539354
```

```
75%          0.110000          4.000000  …                    0.682695
max          0.515000          4.000000  …                    1.000000

       prop.i2i.og.mou.p6  s4.loc.ic.ins.p2  s4.std.ic.ins.l14  \
count        25000.000000      25000.000000       25000.000000
mean             0.485523          5.835480           1.322600
std              0.271146          1.667562           1.822707
min              0.000000          2.000000           0.000000
25%              0.274034          5.000000           0.000000
50%              0.476759          7.000000           0.000000
75%              0.694104          7.000000           2.000000
max              1.000000          7.000000           5.000000

       s4.low.blnc.ins.p4  s3.og.rev.all.m2  s3.new.rev.m2  \
count        25000.000000      25000.000000   25000.000000
mean             8.382160          7.578758      11.333164
std              8.961016          4.477694       6.941543
min              0.000000          0.000000       0.000000
25%              1.000000          4.207500       6.167500
50%              5.000000          6.345000       9.350000
75%             14.000000          9.830000      14.620000
max             30.000000         18.263750      27.298750

       prop.og.mou.any.p6  prop.loc.i2i.mou.og.mou.p3     s3.rev.p1
count        25000.000000                25000.000000  25000.000000
mean            53.594165                    0.483975      7.817712
std             21.408486                    0.292349      7.555598
min              0.000000                    0.000000      0.000000
25%             39.378142                    0.251304      1.970000
50%             53.976203                    0.477621      5.380000
75%             68.312416                    0.716538     11.400000
max            100.000000                    1.000000     25.545000

[8 rows x 111 columns]
New shape: (25000, 111)
```

4.0.7 Handling Missing Values

[32]:
```
def treat_missing_values(df):
    missing_percentage = df.isnull().mean() * 100
    columns_to_drop = missing_percentage[missing_percentage > 50].index
    df = df.drop(columns=columns_to_drop)
    print(f"Dropped columns with >50% missing values: {list(columns_to_drop)}")

    total_missing_percentage = df.isnull().sum().sum() / (df.shape[0] * df.
↪shape[1]) * 100
    if total_missing_percentage < 5:
```

```
        df = df.dropna()
        print("Removed records with missing values (less than 5% of total)")
    else:
        for column in df.columns:
            if df[column].dtype in ['int64', 'float64']:
                df[column].fillna(df[column].median(), inplace=True)
            elif df[column].dtype == 'object':
                df[column].fillna(df[column].mode()[0], inplace=True)
    return df

df = treat_missing_values(df)
print("New shape after handling missing values:", df.shape)
```

```
Dropped columns with >50% missing values: []
Removed records with missing values (less than 5% of total)
New shape after handling missing values: (25000, 111)
```

4.0.8 Removing Highly Correlated Variables

[33]:
```
def remove_highly_correlated(df, threshold=0.9):
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).
 ↪astype(bool))
    to_drop = [column for column in upper.columns if any(upper[column] >␣
 ↪threshold)]
    df = df.drop(columns=to_drop)
    print(f"Removed highly correlated columns: {to_drop}")
    return df

df = remove_highly_correlated(df)
print("New shape after removing highly correlated variables:", df.shape)
```

```
Removed highly correlated columns: ['s1.og.rev.all.m1', 's2.rch.val.l67',
's7.new.rev.p2.p6', 's7.rtd.mou.p2.p6', 's1.new.rev.p1', 's1.rtd.mou.p1',
's1.og.rev.all.p1', 's1.og.mou.all.p1', 'snd.dec.p2', 's1.og.mou.all.p2',
's8.og.rev.p6', 's1.og.hom.mou.p2', 's5.og.rev.all.p1', 's1.og.rev.all.p2',
's1.rtd.mou.p2', 's5.rtd.mou.p1', 's1.og.mou.any.p2', 's1.hom.rmg.rev.p2',
's5.og.mou.all.p1', 's5.og.hom.mou.p1', 's4.usg.ins.p1', 's2.s4.day.no.mou.p2',
's5.s4.day.no.mou.p2', 'tot.s4.day.no.mou.p2', 's1.rev.p1', 's4.og.any.p2',
's1.loc.og.mou.p2', 's5.new.rev.p2', 's5.new.rev.p1', 's4.low.blnc.ins.l14',
's3.og.hom.mou.p1', 's3.new.rev.p2', 'tot.s4.day.no.mou.p3', 's5.og.mou.all.p2',
's4.usg.ins.l14', 's4.loc.og.ins.p2', 's3.rtd.mou.p1', 's7.s5.s4.day.nomou.p2',
's5.og.hom.mou.p2', 'prop.og.mou.tot.mou.all.p2', 's7.s5.s4.day.nomou.p3',
's3.og.rev.3db.p5', 's8.rtd.mou.p6', 's4.low.blnc.ins.p2', 's4.low.blnc.ins.m2',
's4.dec.ins.p2', 's1.rev.p2', 'prop.i2i.og.mou.p6', 's4.loc.ic.ins.p2',
's4.std.ic.ins.l14', 's4.low.blnc.ins.p4', 's3.og.rev.all.m2', 's3.new.rev.m2',
'prop.og.mou.any.p6', 's3.rev.p1']
```

```
New shape after removing highly correlated variables: (25000, 56)
```

4.0.9 Multicollinearity (VIF > 5)

[34]:
```python
import warnings
warnings.filterwarnings("ignore")
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(df):
    vif_data = pd.DataFrame()
    vif_data["feature"] = df.columns
    vif_data["VIF"] = [variance_inflation_factor(df.values, i) for i in␣
 ↪range(len(df.columns))]
    return vif_data

def remove_high_vif_features(df, threshold=5):
    while True:
        vif_data = calculate_vif(df)
        if vif_data['VIF'].max() <= threshold:
            break
        feature_to_remove = vif_data.loc[vif_data['VIF'].idxmax(), 'feature']
        df = df.drop(columns=[feature_to_remove])
        print(f"Removed feature with high VIF: {feature_to_remove}")
    return df

df = remove_high_vif_features(df)
print("Final shape after removing features with high VIF:", df.shape)

# Step 5: Final preprocessed dataset
print(df.describe())
print("Final dataset shape:", df.shape)
```

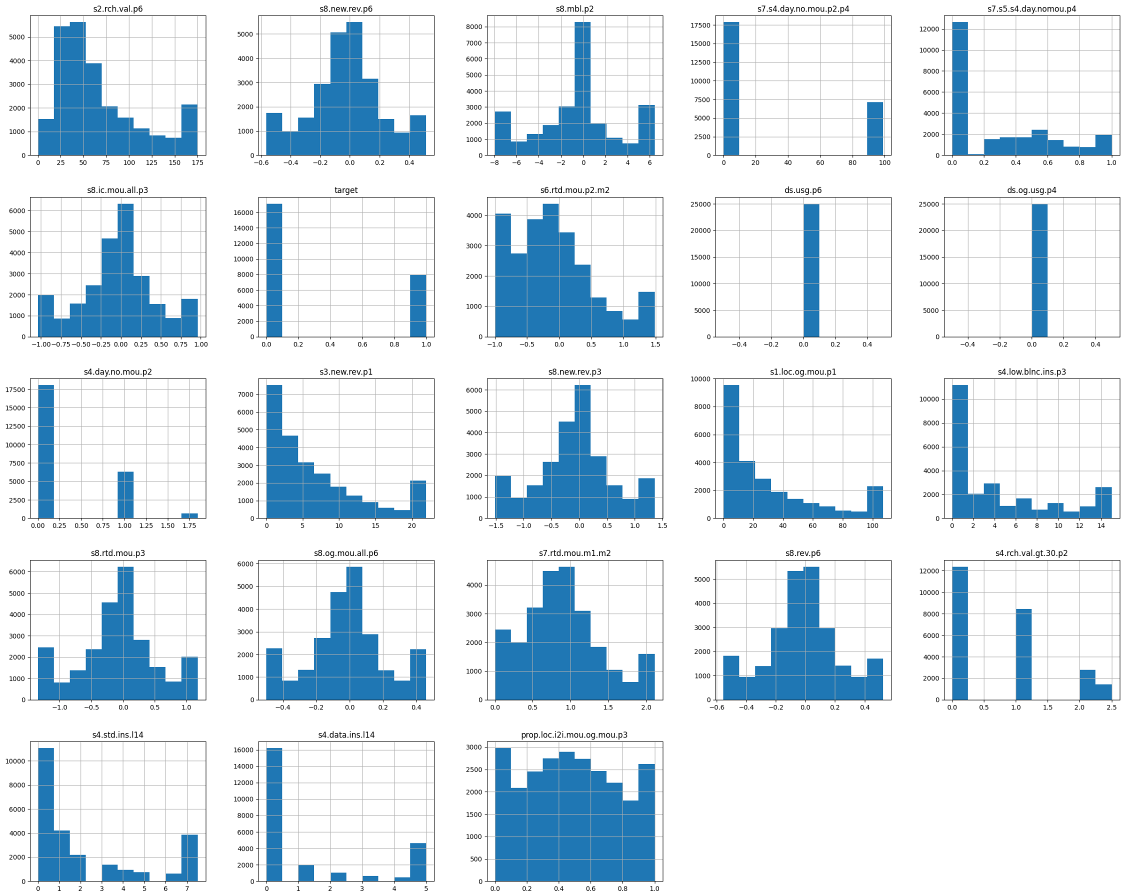
```
Removed feature with high VIF: s4.loc.ins.l14
Removed feature with high VIF: s7.rtd.mou.l21.p6
Removed feature with high VIF: s4.loc.ic.ins.l14
Removed feature with high VIF: s4.usg.ins.p2
Removed feature with high VIF: s7.new.rev.p3.p6
Removed feature with high VIF: s4.loc.og.ins.l14
Removed feature with high VIF: s1.new.rev.p2
Removed feature with high VIF: s7.new.rev.l21.p6
Removed feature with high VIF: s1.new.rev.m2
Removed feature with high VIF: s3.og.mou.all.p1
Removed feature with high VIF: s7.rtd.mou.p3.p6
Removed feature with high VIF: s4.loc.ic.ins.p1
Removed feature with high VIF: s1.og.hom.mou.p1
Removed feature with high VIF: prop.og.mou.tot.mou.all.p6
Removed feature with high VIF: s3.new.rev.4db.p5
Removed feature with high VIF: s7.rev.p2.p6
```

```
Removed feature with high VIF: prop.loc.i2i.mou.og.mou.p6
Removed feature with high VIF: s4.loc.og.ins.p1
Removed feature with high VIF: s1.og.hom.rev.p2
Removed feature with high VIF: s5.rev.p1
Removed feature with high VIF: s4.low.blnc.ins.p6
Removed feature with high VIF: s3.new.rev.p3
Removed feature with high VIF: s3.og.rev.all.p1
Removed feature with high VIF: s1.new.rev.m1
Removed feature with high VIF: s4.dec.ins.l14
Removed feature with high VIF: s5.rev.p2
Removed feature with high VIF: s3.og.mou.all.p2
Removed feature with high VIF: s4.og.unq.any.p2
Removed feature with high VIF: s3.og.rev.4db.p5
Removed feature with high VIF: s8.og.rev.p3
Removed feature with high VIF: s6.new.rev.p2.m2
Removed feature with high VIF: s2.s4.day.no.mou.p3
Removed feature with high VIF: prop.og.mou.any.p2
Final shape after removing features with high VIF: (25000, 23)
       s2.rch.val.p6  s8.new.rev.p6     s8.mbl.p2  s7.s4.day.no.mou.p2.p4  \
count   25000.000000   25000.000000  25000.000000            25000.000000
mean       67.150729      -0.026442     -0.547296               28.481545
std        46.538585       0.253097      3.921211               44.601940
min         0.000000      -0.565000     -7.960000                0.000000
25%        33.000000      -0.160000     -2.560000                0.100000
50%        52.260000      -0.020000     -0.080000                0.363636
75%        89.852500       0.110000      1.040000               99.000000
max       175.131250       0.515000      6.440000               99.000000

       s7.s5.s4.day.nomou.p4  s8.ic.mou.all.p3        target  \
count           25000.000000      25000.000000  25000.000000
mean                0.284181         -0.038190      0.316680
std                 0.333501          0.480536      0.465191
min                 0.000000         -1.040000      0.000000
25%                 0.000000         -0.290000      0.000000
50%                 0.000000         -0.020000      0.000000
75%                 0.500000          0.210000      1.000000
max                 1.000000          0.960000      1.000000

       s6.rtd.mou.p2.m2  ds.usg.p6  ds.og.usg.p4  …  s1.loc.og.mou.p1  \
count      25000.000000    25000.0       25000.0  …      25000.000000
mean          -0.080950        0.0           0.0  …         30.365185
std            0.646247        0.0           0.0  …         32.871037
min           -1.000000        0.0           0.0  …          0.000000
25%           -0.540000        0.0           0.0  …          4.483200
50%           -0.150000        0.0           0.0  …         17.999900
75%            0.270000        0.0           0.0  …         45.499325
max            1.485000        0.0           0.0  …        107.023513
```

```
       s4.low.blnc.ins.p3  s8.rtd.mou.p3  s8.og.mou.all.p6  s7.rtd.mou.m1.m2  \
count        25000.000000   25000.000000      25000.000000      25000.000000
mean             4.335200      -0.087113         -0.019145          0.897349
std              5.027382       0.633841          0.241952          0.520586
min              0.000000      -1.345000         -0.500000          0.000000
25%              0.000000      -0.400000         -0.140000          0.552297
50%              2.000000      -0.050000         -0.020000          0.860550
75%              7.000000       0.230000          0.100000          1.176028
max             15.000000       1.175000          0.460000          2.111626

          s8.rev.p6  s4.rch.val.gt.30.p2  s4.std.ins.l14  s4.data.ins.l14  \
count  25000.000000         25000.000000    25000.000000     25000.000000
mean      -0.017048             0.701160        2.103420         1.243200
std        0.254273             0.802347        2.712373         1.965291
min       -0.555000             0.000000        0.000000         0.000000
25%       -0.150000             0.000000        0.000000         0.000000
50%       -0.010000             1.000000        1.000000         0.000000
75%        0.120000             1.000000        3.000000         2.000000
max        0.525000             2.500000        7.500000         5.000000

       prop.loc.i2i.mou.og.mou.p3
count                25000.000000
mean                     0.483975
std                      0.292349
min                      0.000000
25%                      0.251304
50%                      0.477621
75%                      0.716538
max                      1.000000

[8 rows x 23 columns]
Final dataset shape: (25000, 23)
```

4.1 Visualize the Distribution

[35]:
```
df.hist(figsize=(30,24))
plt.show()
```



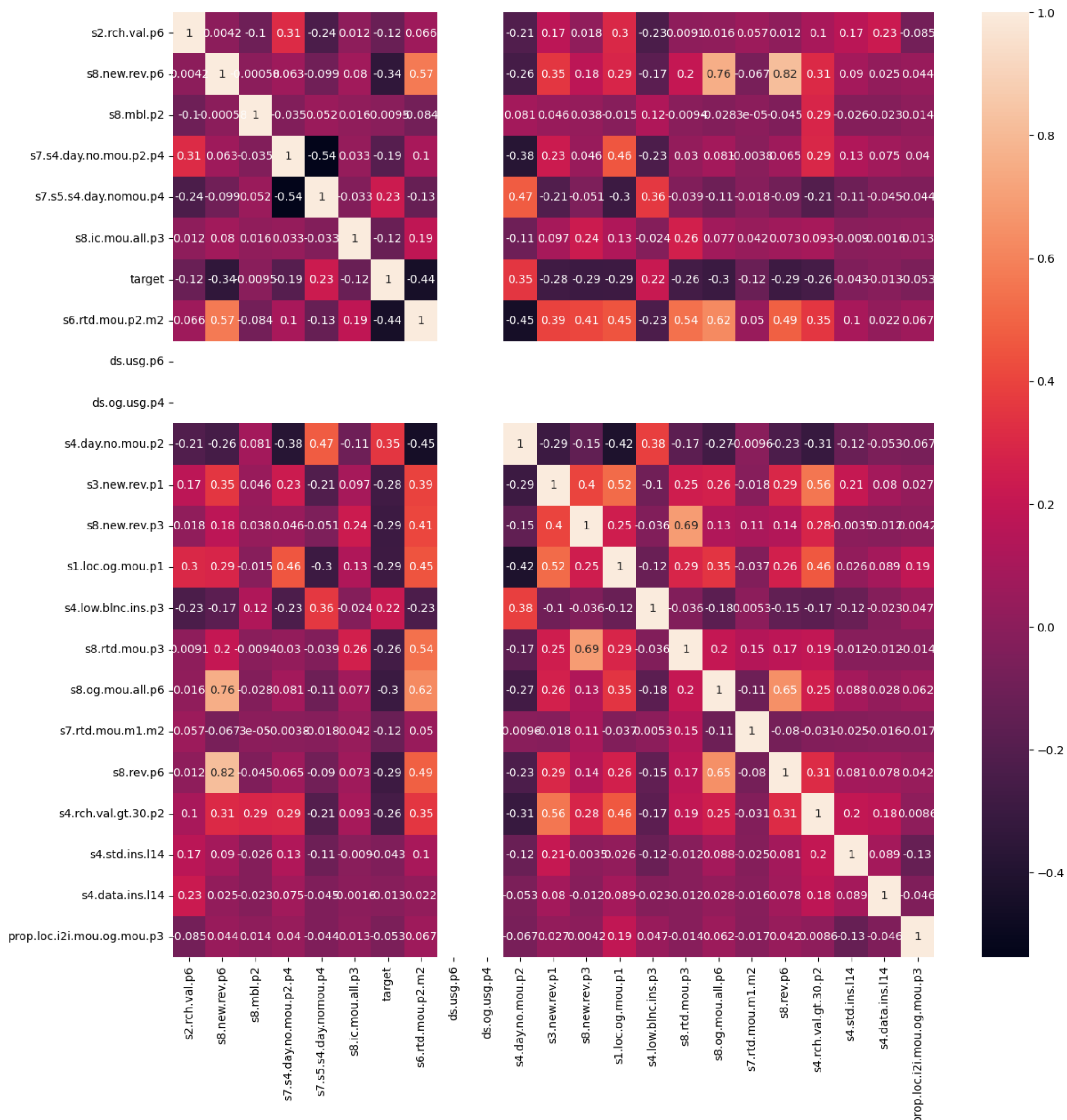
[36]:
```python
df['target'].value_counts()
```

[36]:
```
target
0    17083
1     7917
Name: count, dtype: int64
```

[37]:
```python
import plotly.graph_objects as go
fig=go.Figure(data=[go.Pie(labels =['Retained (0)', 'Exited (1)'],values=df['target'].value_counts())])
fig.update_layout(width=500, height=400)
fig.show()
```

[38]:
```python
s=df.select_dtypes(include=["integer","float"]).corr()
plt.figure(figsize=(15,15))
sns.heatmap(s,annot=True)
```

[38]:
```
<Axes: >
```



4.2 Droping the Irrevant Columns

[39]:
```
df = df.drop(['ds.og.usg.p4','ds.usg.p6'], axis=1)
#df = df.drop(['ds.og.usg.p4', 's4.day.no.mou.p2','ds.usg.p6'], axis=1)
#df = df.drop(['ds.usg.p6'], axis=1)
```

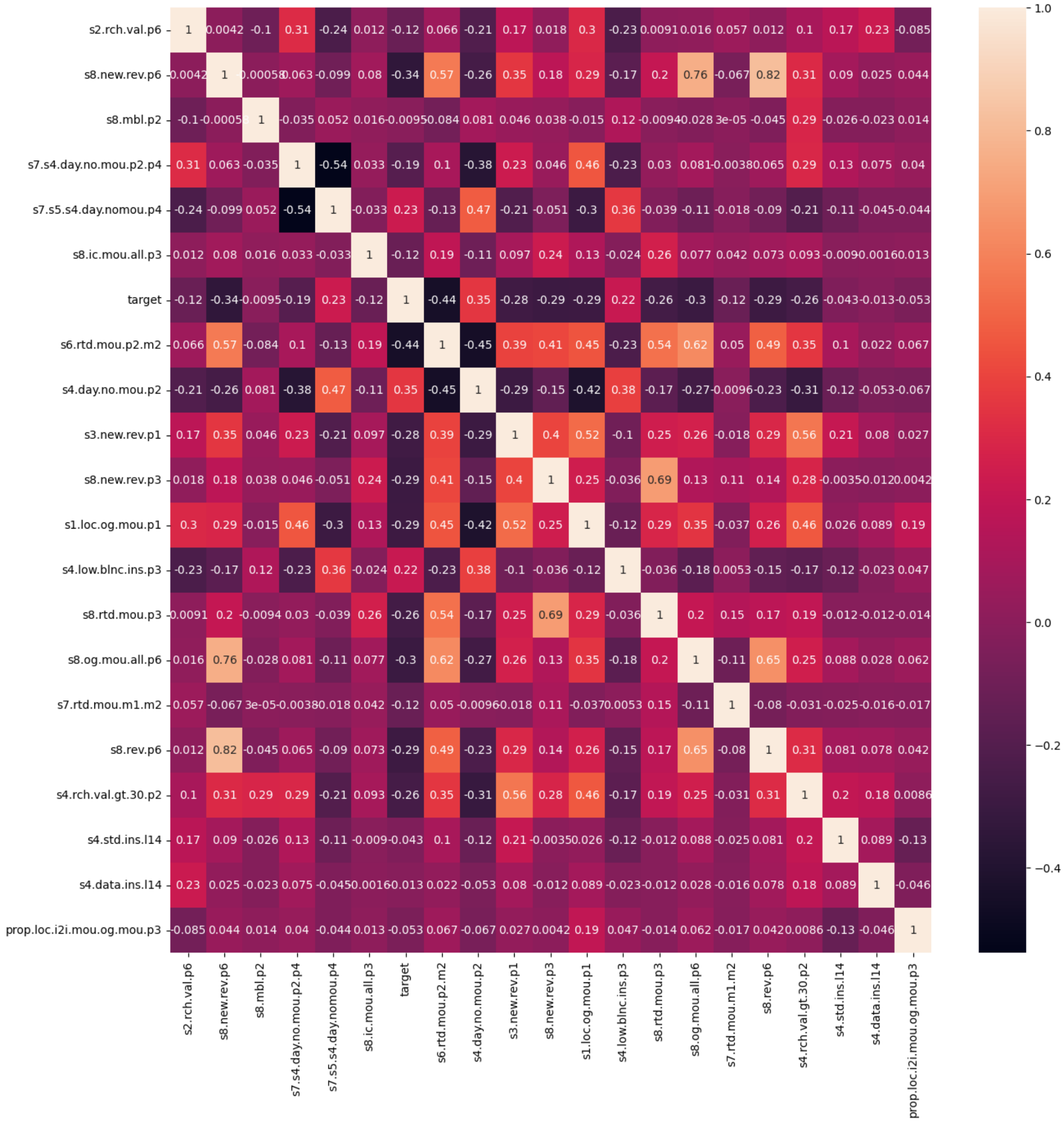
[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 21 columns):
```

```
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   s2.rch.val.p6              25000 non-null  float64
 1   s8.new.rev.p6              25000 non-null  float64
 2   s8.mbl.p2                  25000 non-null  float64
 3   s7.s4.day.no.mou.p2.p4     25000 non-null  float64
 4   s7.s5.s4.day.nomou.p4      25000 non-null  float64
 5   s8.ic.mou.all.p3           25000 non-null  float64
 6   target                     25000 non-null  int64
 7   s6.rtd.mou.p2.m2           25000 non-null  float64
 8   s4.day.no.mou.p2           25000 non-null  float64
 9   s3.new.rev.p1              25000 non-null  float64
 10  s8.new.rev.p3              25000 non-null  float64
 11  s1.loc.og.mou.p1           25000 non-null  float64
 12  s4.low.blnc.ins.p3         25000 non-null  int64
 13  s8.rtd.mou.p3              25000 non-null  float64
 14  s8.og.mou.all.p6           25000 non-null  float64
 15  s7.rtd.mou.m1.m2           25000 non-null  float64
 16  s8.rev.p6                  25000 non-null  float64
 17  s4.rch.val.gt.30.p2        25000 non-null  float64
 18  s4.std.ins.l14             25000 non-null  float64
 19  s4.data.ins.l14            25000 non-null  int64
 20  prop.loc.i2i.mou.og.mou.p3  25000 non-null  float64
dtypes: float64(18), int64(3)
memory usage: 4.0 MB
```

[41]:
```
s=df.select_dtypes(include=["integer","float"]).corr()
plt.figure(figsize=(15,15))
sns.heatmap(s,annot=True)
```

[41]: <Axes: >



5 Feature Selection and Prepare the Data

[42]:
```python
X = df.drop('target', axis='columns')
y = df['target']
```

[43]:
```python
X
```

[43]:
```
       s2.rch.val.p6  s8.new.rev.p6  s8.mbl.p2  s7.s4.day.no.mou.p2.p4  \
0              39.29         -0.170      -0.72                1.000000
1              21.67         -0.320      -0.08                0.500000
```

2	30.00	-0.050	-0.09	0.384615
3	50.00	-0.180	1.83	0.416667
4	22.50	0.010	-0.04	0.222222
…	…	…	…	…
24995	26.67	-0.160	0.76	0.250000
24996	27.88	0.190	0.37	0.454545
24997	10.00	-0.030	-0.79	0.083333
24998	42.92	0.515	-1.09	99.000000
24999	53.50	-0.480	0.00	0.400000

	s7.s5.s4.day.nomou.p4	s8.ic.mou.all.p3	s6.rtd.mou.p2.m2 \
0	0.666667	-0.73	-0.71
1	0.583333	0.00	-0.96
2	0.384615	-1.03	-0.98
3	0.250000	-0.43	-0.92
4	0.777778	-1.04	-0.98
…	…	…	…
24995	0.375000	0.68	-0.58
24996	0.636364	0.04	-0.07
24997	0.333333	0.27	-0.12
24998	0.000000	-0.42	0.25
24999	0.333333	0.54	-1.00

	s4.day.no.mou.p2	s3.new.rev.p1	s8.new.rev.p3	s1.loc.og.mou.p1 \
0	1.000000	0.22	-0.90	2.383200
1	1.000000	0.38	-0.14	0.650000
2	1.844649	0.11	-0.45	0.183300
3	1.844649	0.49	-0.02	0.816500
4	1.000000	0.10	-0.67	0.166600
…	…	…	…	…
24995	1.000000	0.76	0.26	3.499900
24996	1.000000	14.26	0.82	17.116500
24997	0.000000	8.39	0.38	3.299800
24998	0.000000	15.16	-1.52	107.023513
24999	1.000000	0.00	0.00	0.000000

	s4.low.blnc.ins.p3	s8.rtd.mou.p3	s8.og.mou.all.p6	s7.rtd.mou.m1.m2 \
0	7	-0.500	-0.11	0.240533
1	13	-0.110	-0.13	0.459725
2	10	-0.390	-0.12	0.111785
3	11	-0.020	-0.14	1.920826
4	0	-0.630	-0.02	1.728186
…	…	…	…	…
24995	10	-0.130	-0.16	1.423358
24996	10	0.220	0.08	0.688912
24997	3	0.250	-0.03	1.223699
24998	0	-1.345	0.46	0.579099

```
24999                  15          0.000             -0.50          1.423424

       s8.rev.p6  s4.rch.val.gt.30.p2  s4.std.ins.l14  s4.data.ins.l14  \
0         -0.120                  0.0             0.0                0
1         -0.220                  0.0             0.0                0
2         -0.070                  0.0             1.0                0
3         -0.210                  0.0             2.0                0
4          0.010                  0.0             1.0                2
…           …                     …              …                 …
24995     -0.140                  0.0             0.0                0
24996      0.050                  0.0             0.0                0
24997     -0.030                  0.0             1.0                1
24998      0.525                  2.5             1.0                3
24999     -0.460                  0.0             0.0                0

       prop.loc.i2i.mou.og.mou.p3
0                        0.609456
1                        1.000000
2                        0.699592
3                        0.086617
4                        0.683105
…                           …
24995                    0.328027
24996                    0.288006
24997                    0.235918
24998                    0.952962
24999                    1.000000

[25000 rows x 20 columns]
```

[44]:
```
y
```

[44]:
```
0        1
1        1
2        1
3        0
4        0
        ..
24995    1
24996    0
24997    0
24998    0
24999    1
Name: target, Length: 25000, dtype: int64
```

[45]:
```
X.shape
```

[45]: (25000, 20)

[46]:
```
y.shape
```

[46]: (25000,)

5.1 Splitting the dataset

[77]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

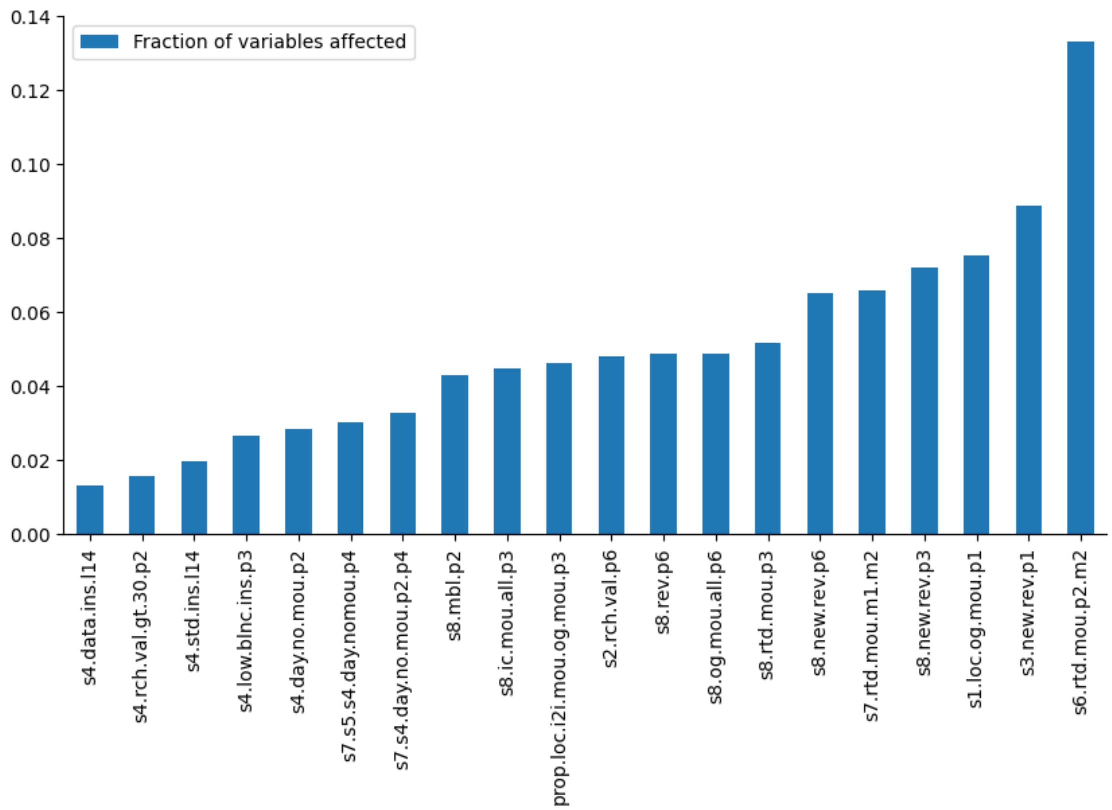
```
X_train shape: (20000, 20)
X_test shape: (5000, 20)
y_train shape: (20000,)
y_test shape: (5000,)
```

5.2 Check importance of Features in the Dataset

[78]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train.values.ravel())
```

[78]: RandomForestClassifier()

[79]:
```
feat_scores = pd.DataFrame({"Fraction of variables affected" : rf.
↪feature_importances_},index=X.columns)
feat_scores = feat_scores.sort_values(by = "Fraction of variables affected")
feat_scores.plot(kind ="bar", figsize = (10,5))
sns.despine()
```



6 Train the Models

6.1 Random Forest Classifier

[80]:
```
#from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier()
model_rf.fit(X_train,y_train)
```

[80]:
```
RandomForestClassifier()
```

[81]:
```
y_predict = model_rf.predict(X_test)
```

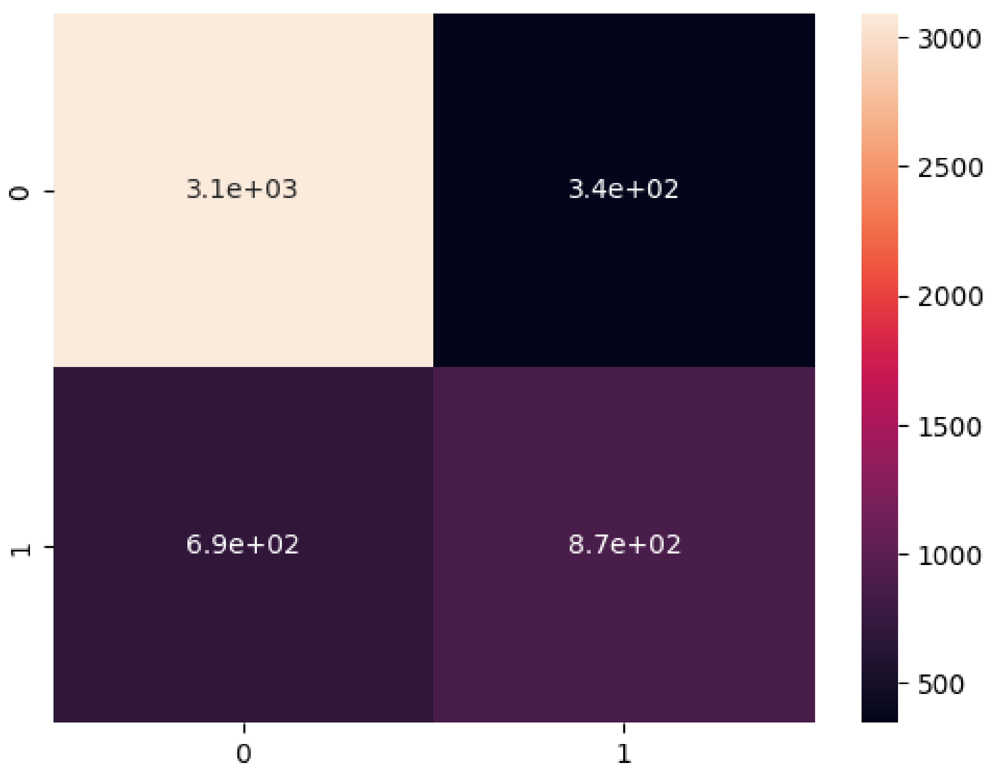
[82]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      3437
           1       0.72      0.56      0.63      1563

    accuracy                           0.79      5000
```

```
   macro avg       0.77      0.73      0.74      5000
weighted avg       0.79      0.79      0.79      5000
```

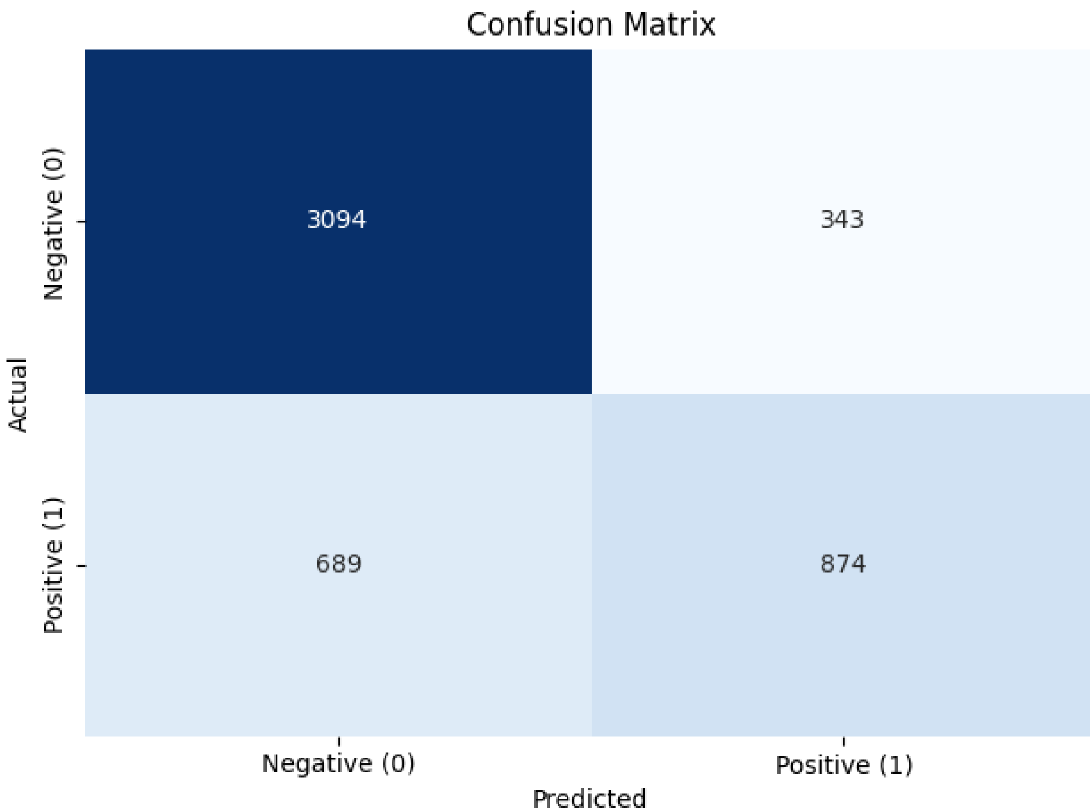
[83]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm,annot=True)
```

[83]: <Axes: >



[84]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Add class labels
class_names = ['Negative (0)', 'Positive (1)']
tick_marks = np.arange(len(class_names))
```

```
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)

# Show the plot
plt.tight_layout()
plt.show()
```



6.2 Linear Regression Model

[85]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

model_lr = LogisticRegression()
model_lr.fit(X_train,y_train)
```

[85]: LogisticRegression()

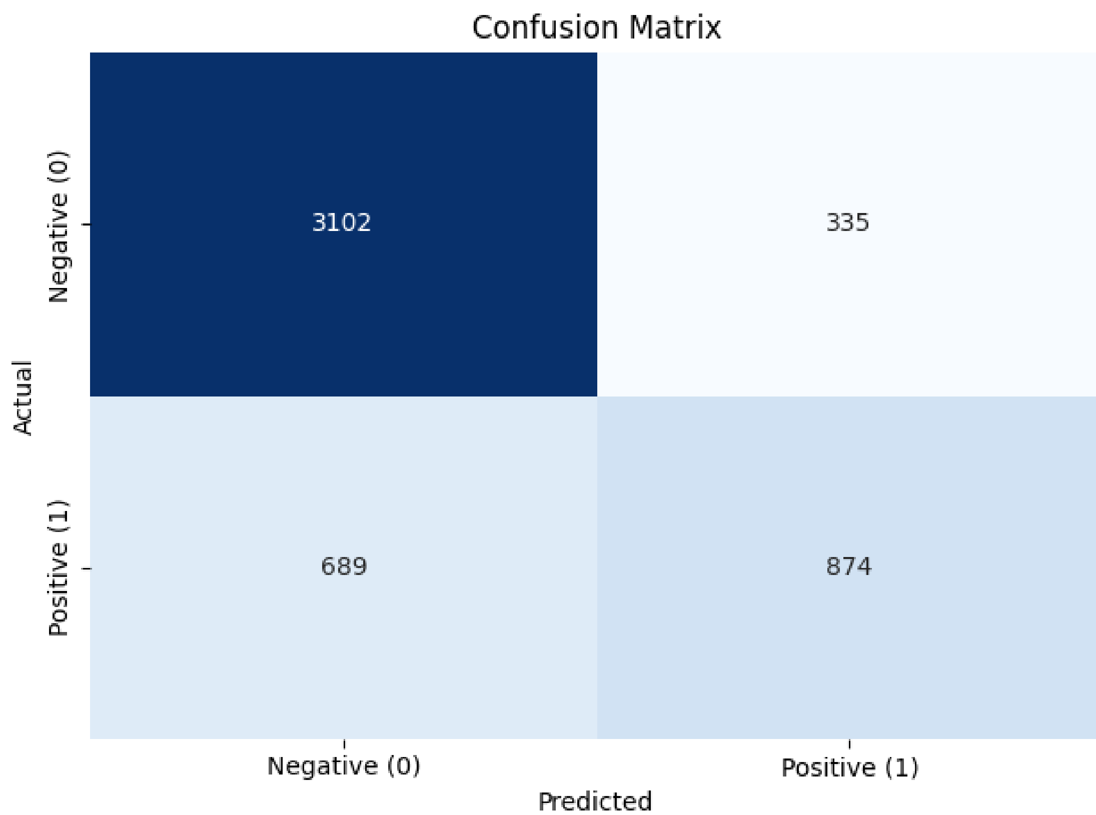
[86]:
```
y_predict = model_lr.predict(X_test)
```

[87]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      3437
           1       0.72      0.56      0.63      1563

    accuracy                           0.80      5000
   macro avg       0.77      0.73      0.74      5000
weighted avg       0.79      0.80      0.79      5000
```

[88]:

```python
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Add class labels
class_names = ['Negative (0)', 'Positive (1)']
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)

# Show the plot
plt.tight_layout()
plt.show()
```



6.3 Support Vector Machine

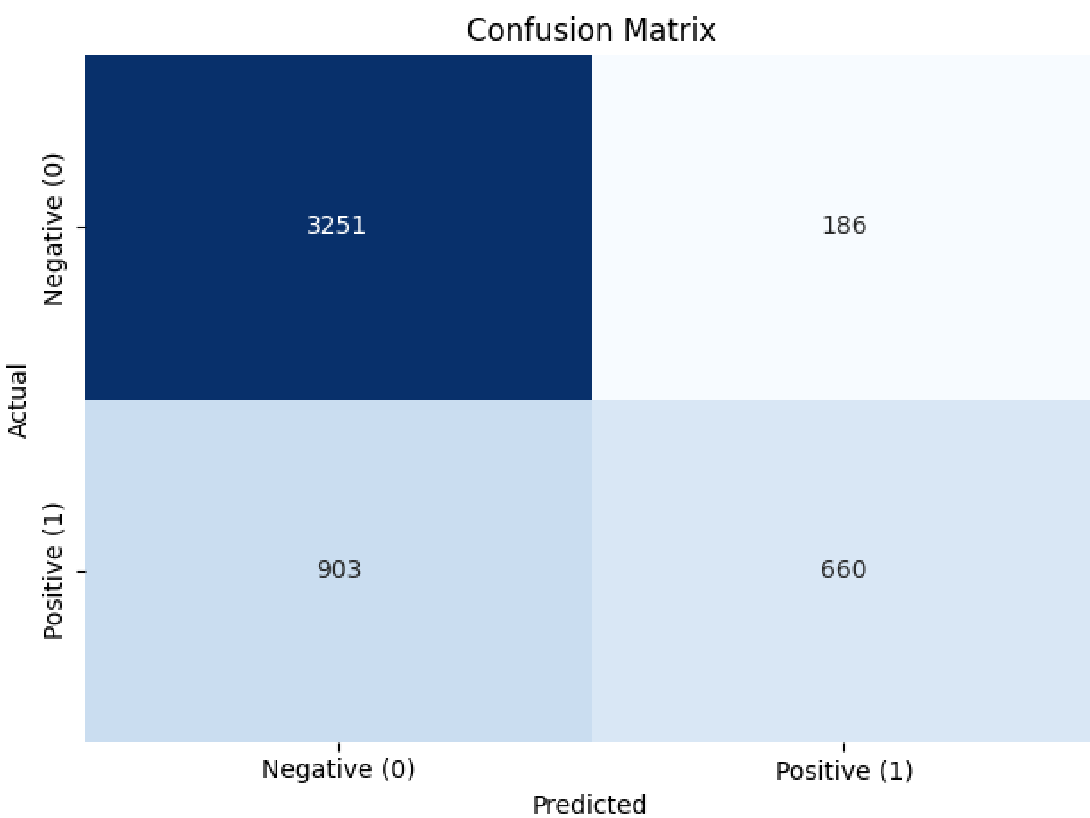
[89]:
```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import LinearSVC

model_svc = LinearSVC()
model_svc.fit(X_train,y_train)
```

[89]: LinearSVC()

[90]:
```
y_predict = model_svc.predict(X_test)
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.78      0.95      0.86      3437
           1       0.78      0.42      0.55      1563

    accuracy                           0.78      5000
   macro avg       0.78      0.68      0.70      5000
weighted avg       0.78      0.78      0.76      5000
```

[91]:
```
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Add class labels
class_names = ['Negative (0)', 'Positive (1)']
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)

# Show the plot
plt.tight_layout()
plt.show()
```



6.4 K- Nearest Neighbours

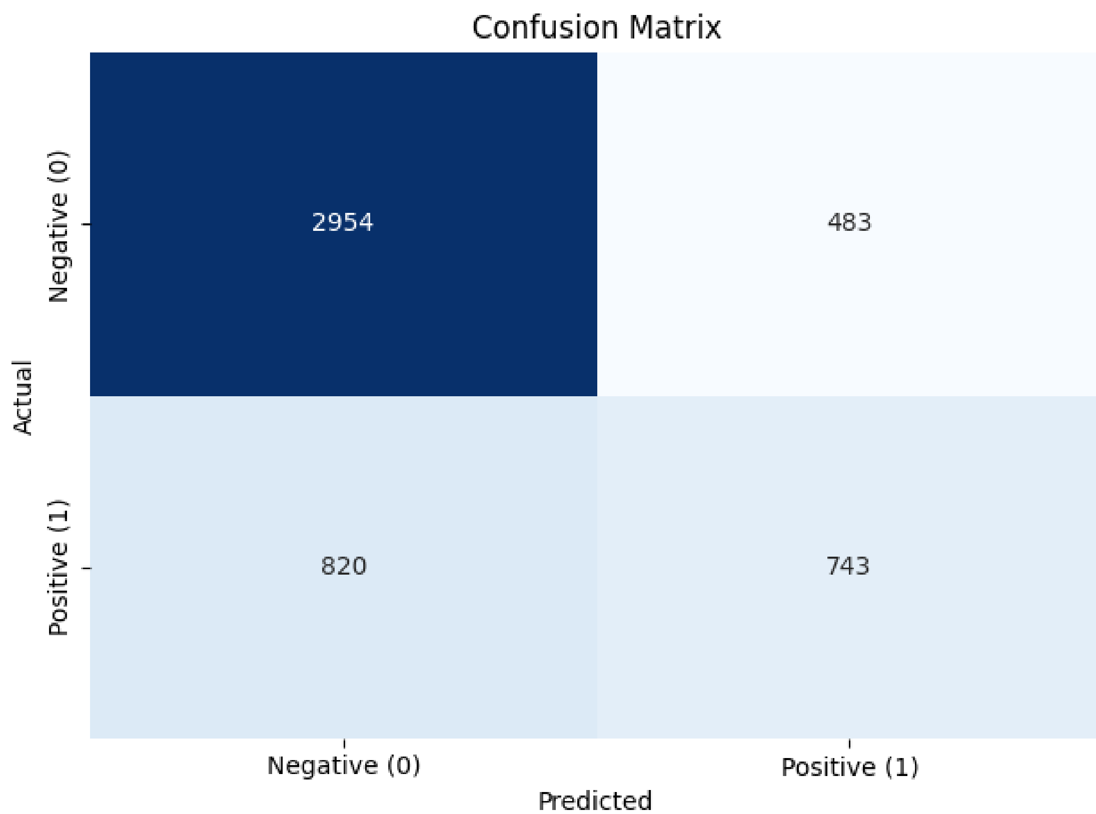
[92]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
model_knn = KNeighborsClassifier()
model_knn.fit(X_train,y_train)
```

[92]:
```
KNeighborsClassifier()
```

[93]:
```
y_predict = model_knn.predict(X_test)
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82      3437
           1       0.61      0.48      0.53      1563

    accuracy                           0.74      5000
   macro avg       0.69      0.67      0.68      5000
weighted avg       0.73      0.74      0.73      5000
```

[94]:
```
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Add class labels
class_names = ['Negative (0)', 'Positive (1)']
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)

# Show the plot
plt.tight_layout()
plt.show()
```



6.5 Naive Bayer Model

[95]:
```
from sklearn.naive_bayes import GaussianNB
model_nb = GaussianNB()
model_nb.fit(X_train,y_train)
```

[95]:
```
GaussianNB()
```

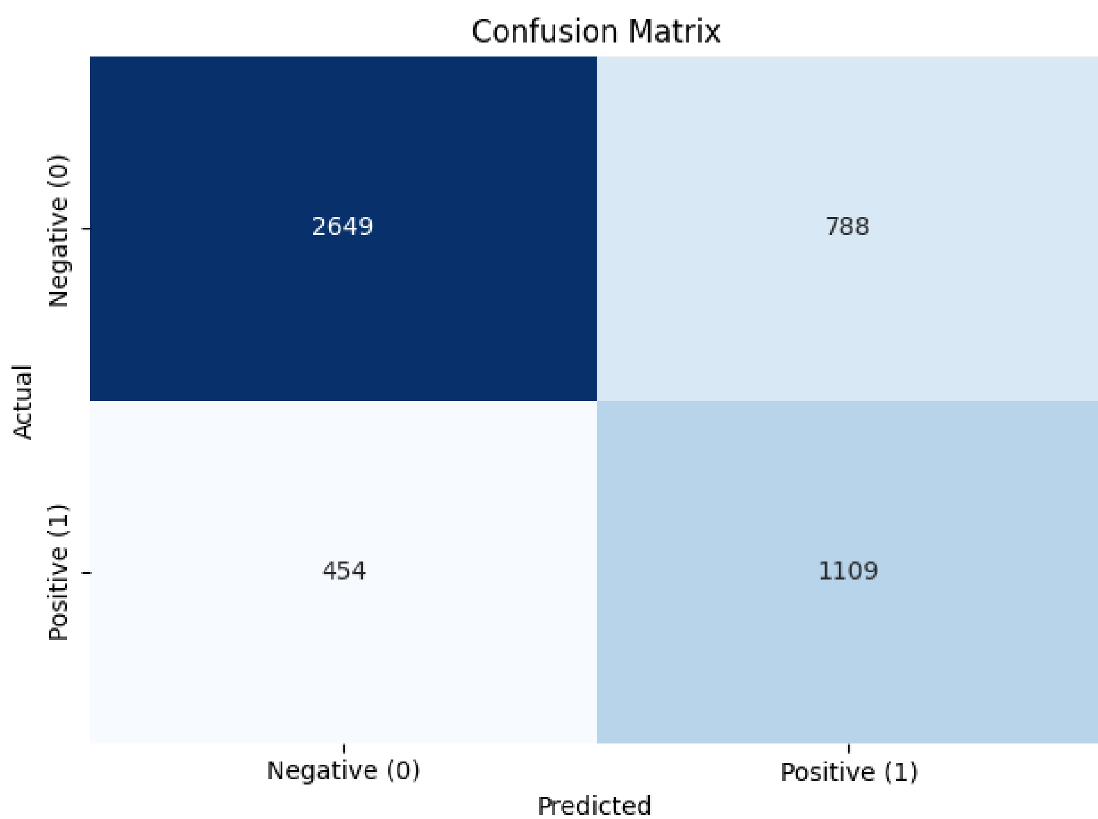
[96]:
```
y_predict = model_nb.predict(X_test)
```

[97]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.77      0.81      3437
           1       0.58      0.71      0.64      1563

    accuracy                           0.75      5000
   macro avg       0.72      0.74      0.73      5000
weighted avg       0.77      0.75      0.76      5000
```

[98]:

```python
cm = confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Add class labels
class_names = ['Negative (0)', 'Positive (1)']
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)

# Show the plot
plt.tight_layout()
plt.show()
```



7 ROC Curve

[99]:
```
model_lr.predict_proba(X_test)
```

[99]:
```
array([[0.99576888, 0.00423112],
       [0.51117157, 0.48882843],
       [0.80720218, 0.19279782],
       …,
       [0.75609067, 0.24390933],
       [0.83541985, 0.16458015],
       [0.57106124, 0.42893876]])
```

[100]:
```
model_lr.predict_proba(X_test)[:,1]
```

[100]:
```
array([0.00423112, 0.48882843, 0.19279782, …, 0.24390933, 0.16458015,
       0.42893876])
```

[101]:
```
y_test
```

[101]:
```
6868     0
24016    0
9668     0
13640    0
14018    0
        ..
8670     0
11839    0
4013     0
21147    0
695      0
Name: target, Length: 5000, dtype: int64
```

[111]:
```
from sklearn.metrics import roc_curve
fpr1,tpr1,thresh1 = roc_curve(y_test, model_rf.predict_proba(X_test)[:
 ↪,1],pos_label=1)
fpr2,tpr2,thresh2 = roc_curve(y_test, model_lr.predict_proba(X_test)[:
 ↪,1],pos_label=1)
fpr3,tpr3,thresh3 = roc_curve(y_test, model_svc.
 ↪decision_function(X_test),pos_label=1)
fpr4,tpr4,thresh4 = roc_curve(y_test, model_knn.predict_proba(X_test)[:
 ↪,1],pos_label=1)
fpr5,tpr5,thresh5 = roc_curve(y_test, model_nb.predict_proba(X_test)[:
 ↪,1],pos_label=1)
```

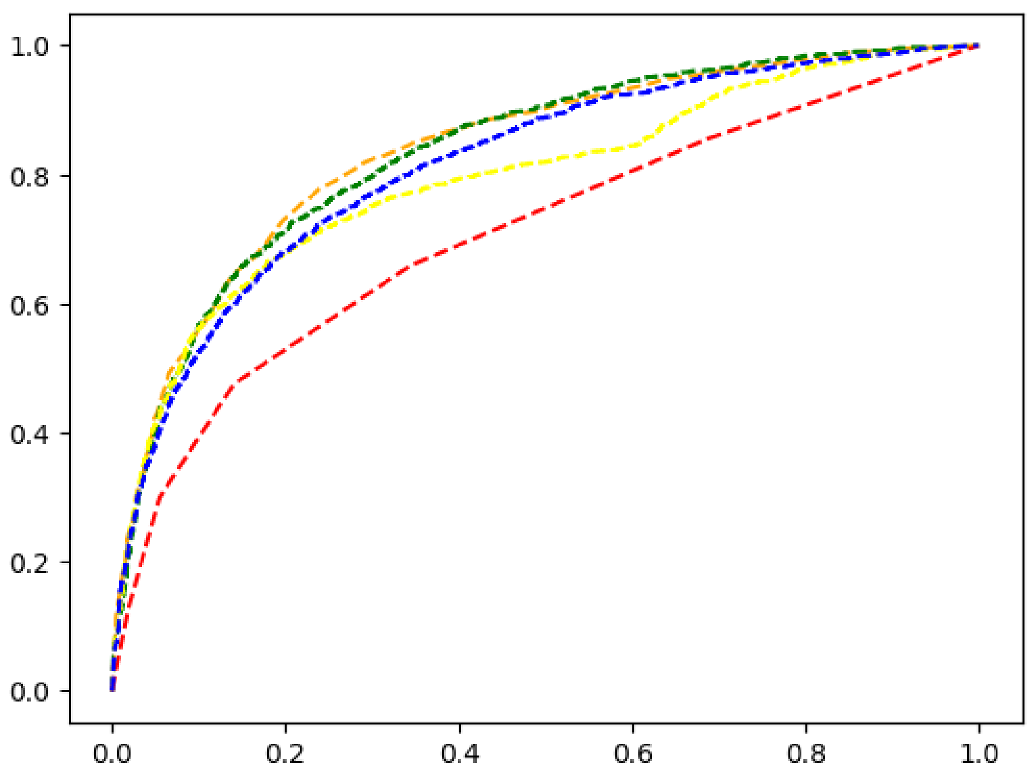
[112]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score1 = roc_auc_score(y_test, model_rf.predict_proba(X_test)[:,1])
roc_auc_score2 = roc_auc_score(y_test, model_lr.predict_proba(X_test)[:,1])
```

```
roc_auc_score3 = roc_auc_score(y_test, model_svc.decision_function(X_test))
roc_auc_score4 = roc_auc_score(y_test, model_knn.predict_proba(X_test)[:,1])
roc_auc_score5 = roc_auc_score(y_test, model_nb.predict_proba(X_test)[:,1])

print("Random Forest:", roc_auc_score1)
print("Logistic Regression", roc_auc_score2)
print("Support Vector Machine", roc_auc_score3)
print("K-Nearest Neighbours", roc_auc_score4)
print("Naive Bayes:", roc_auc_score5)
```

```
Random Forest: 0.8373767761206145
Logistic Regression 0.8340912403521127
Support Vector Machine 0.7929215226047653
K-Nearest Neighbours 0.7074472392285152
Naive Bayes: 0.8148504727541593
```

[113]:
```
plt.plot(fpr1,tpr1,linestyle='--', color='orange', label='Random Forest')
plt.plot(fpr2,tpr2,linestyle='--', color='green', label='Linear Regression')
plt.plot(fpr3,tpr3,linestyle='--', color='yellow', label='Support Vector␣
↪Machine')
plt.plot(fpr4,tpr4,linestyle='--', color='red', label='K-Nearest Neighbours')
plt.plot(fpr5,tpr5,linestyle='--', color='blue', label='Naive Bayes')
```

[113]:
```
[<matplotlib.lines.Line2D at 0x7963d4b13ee0>]
```



8 Conclusion

As per the Graph and ROC AUC Score, Random Forest Model performed Well.