Syed Faizan

Magazine Subscription Analysis using two Group Classification Models

We implement logistic regression and Support Vector Machine Models to classify and predict the magazine subscription behavious.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Set pandas options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)




file_path = r'C:\Users\sfaiz\OneDrive\Desktop\ALY 6020 Module 3
Project Syed Faizan\marketing_campaign.xlsx'
data = pd.read_excel(file_path)

data.head()

     ID  Year_Birth   Education Marital_Status   Income  Kidhome
Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0
0
1  2174        1954  Graduation         Single  46344.0        1
1
2  4141        1965  Graduation       Together  71613.0        0
0
3  6182        1984  Graduation       Together  26646.0        1
0
4  5324        1981         PhD        Married  58293.0        1
0

  Dt_Customer  Recency  MntWines  MntFruits  MntMeatProducts
MntFishProducts  \
0  2012-09-04       58       635         88              546
172
1  2014-03-08       38        11          1                6
2
2  2013-08-21       26       426         49              127
111
```

```
3  2014-02-10       26        11          4               20
10
4  2014-01-19       94       173         43              118
46

   MntSweetProducts  MntGoldProds  NumDealsPurchases  NumWebPurchases
\
0                88            88                  3                8

1                 1             6                  2                1

2                21            42                  1                8

3                 3             5                  2                2

4                27            15                  5                5


   NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth
AcceptedCmp3  \
0                   10                  4                  7
0
1                    1                  2                  5
0
2                    2                 10                  4
0
3                    0                  4                  6
0
4                    3                  6                  5
0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response
0              3         11         1
1              3         11         0
2              3         11         0
3              3         11         0
4              3         11         0
```

```
data.shape
```

```
(2240, 29)
```

```
data.isnull().sum()
```

```
ID                         0
Year_Birth                 0
Education                  0
Marital_Status             0
Income                    24
Kidhome                    0
Teenhome                   0
Dt_Customer                0
Recency                    0
MntWines                   0
MntFruits                  0
MntMeatProducts            0
MntFishProducts            0
MntSweetProducts           0
MntGoldProds               0
NumDealsPurchases          0
NumWebPurchases            0
NumCatalogPurchases        0
NumStorePurchases          0
NumWebVisitsMonth          0
AcceptedCmp3               0
AcceptedCmp4               0
AcceptedCmp5               0
AcceptedCmp1               0
AcceptedCmp2               0
Complain                   0
Z_CostContact              0
Z_Revenue                  0
Response                   0
dtype: int64
```

```
data = data[data['Income'].notnull()]
```

```
data.shape
```

```
(2216, 29)
```

```
data.isnull().sum()
```

```
ID                        0
Year_Birth                0
Education                 0
Marital_Status            0
Income                    0
Kidhome                   0
Teenhome                  0
Dt_Customer               0
Recency                   0
MntWines                  0
MntFruits                 0
```

```
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
NumStorePurchases      0
NumWebVisitsMonth      0
AcceptedCmp3           0
AcceptedCmp4           0
AcceptedCmp5           0
AcceptedCmp1           0
AcceptedCmp2           0
Complain               0
Z_CostContact          0
Z_Revenue              0
Response               0
dtype: int64
```

```
data.dtypes
```

```
ID                       int64
Year_Birth               int64
Education               object
Marital_Status          object
Income                 float64
Kidhome                  int64
Teenhome                 int64
Dt_Customer             object
Recency                  int64
MntWines                 int64
MntFruits                int64
MntMeatProducts          int64
MntFishProducts          int64
MntSweetProducts         int64
MntGoldProds             int64
NumDealsPurchases        int64
NumWebPurchases          int64
NumCatalogPurchases      int64
NumStorePurchases        int64
NumWebVisitsMonth        int64
AcceptedCmp3             int64
AcceptedCmp4             int64
AcceptedCmp5             int64
AcceptedCmp1             int64
AcceptedCmp2             int64
Complain                 int64
Z_CostContact            int64
Z_Revenue                int64
```

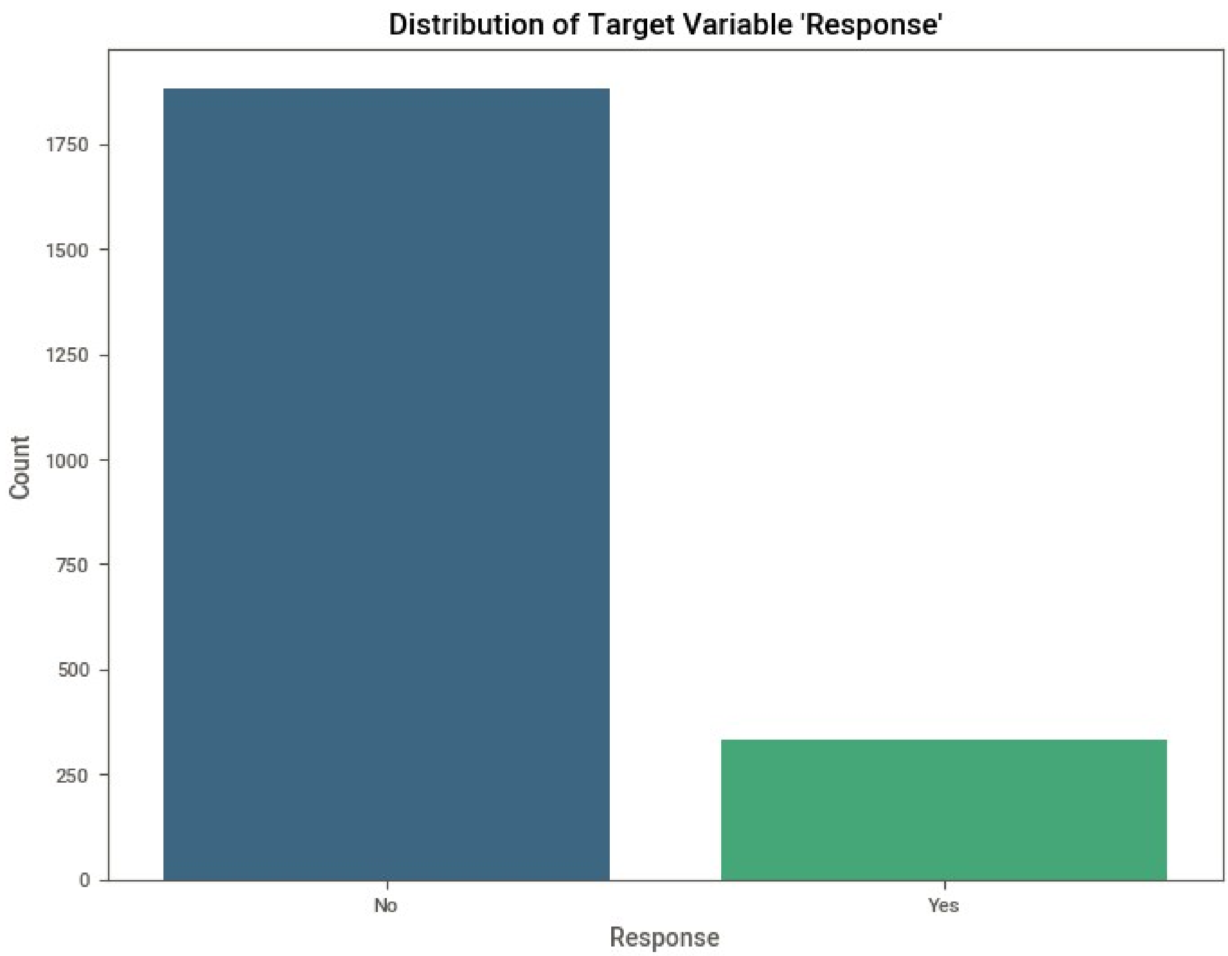
```
Response                        int64
dtype: object
```

Ignore warnings as some Hyperparameter tuning may produce extensive warnings

```python
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
```

Examining the target variable

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Count the occurrences of each class in the target variable
response_counts = data['Response'].value_counts()

# Plot the imbalance
plt.figure(figsize=(8, 6))
sns.barplot(x=response_counts.index, y=response_counts.values,
palette="viridis")

# Add labels and title
plt.title("Distribution of Target Variable 'Response'")
plt.xlabel("Response")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=["No", "Yes"], rotation=0)  #
Customize if labels are binary
plt.show()
```



Three issues to be dealt with by Data Cleansing :

1. The Dt_Customer column needs to be converted to an integer value that represents the time since subscription.
2. The 'object' columns need to be converted to numerical columns and encoded using ordinal encoding as one hot encoding may cause hyper-dimentionality.
3. The heavy imbalance in the target variable ought to be addressed and remedial measures ought to be taken.

Dt_Customer column converted to an integer value as 'Customer Tenure'.

```
# Dt_Customer column needs to be converted to an integer value
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'],
```

```
errors='coerce')
data = data.dropna(subset=['Dt_Customer'])
data['Customer_Tenure'] = (pd.Timestamp.now() -
data['Dt_Customer']).dt.days

data.head()

     ID  Year_Birth   Education Marital_Status   Income  Kidhome
Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0
0
1  2174        1954  Graduation         Single  46344.0        1
1
2  4141        1965  Graduation       Together  71613.0        0
0
3  6182        1984  Graduation       Together  26646.0        1
0
4  5324        1981         PhD        Married  58293.0        1
0

  Dt_Customer  Recency  MntWines  MntFruits  MntMeatProducts
MntFishProducts  \
0  2012-09-04       58       635         88              546
172
1  2014-03-08       38        11          1                6
2
2  2013-08-21       26       426         49              127
111
3  2014-02-10       26        11          4               20
10
4  2014-01-19       94       173         43              118
46

   MntSweetProducts  MntGoldProds  NumDealsPurchases  NumWebPurchases
\
0                88            88                  3                8

1                 1             6                  2                1

2                21            42                  1                8

3                 3             5                  2                2

4                27            15                  5                5


   NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth
AcceptedCmp3  \
0                   10                  4                  7
0
```

```
1                    1                  2                  5
0
2                    2                 10                  4
0
3                    0                  4                  6
0
4                    3                  6                  5
0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response  Customer_Tenure
0              3         11         1             4461
1              3         11         0             3911
2              3         11         0             4110
3              3         11         0             3937
4              3         11         0             3959
```

Removing dt_customer as it's data is found in customer tenure

```
# removing dt_customer as it's data is found in customer tenure
data_sansdt = data.drop(columns=['Dt_Customer'])
```

Encode categorical variables

```
# Encode categorical variables
categorical_columns = ['Education', 'Marital_Status']
for col in categorical_columns:
    le = LabelEncoder()
    data_sansdt[col] = le.fit_transform(data_sansdt[col])

data_sansdt.head()

     ID  Year_Birth  Education  Marital_Status   Income  Kidhome
Teenhome  \
0  5524        1957          2               4  58138.0        0
0
1  2174        1954          2               4  46344.0        1
1
```

```
2  4141        1965          2                5  71613.0        0
0
3  6182        1984          2                5  26646.0        1
0
4  5324        1981          4                3  58293.0        1
0

   Recency  MntWines  MntFruits  MntMeatProducts  MntFishProducts  \
0       58       635         88              546              172
1       38        11          1                6                2
2       26       426         49              127              111
3       26        11          4               20               10
4       94       173         43              118               46

   MntSweetProducts  MntGoldProds  NumDealsPurchases  NumWebPurchases
\
0                88            88                  3                8

1                 1             6                  2                1

2                21            42                  1                8

3                 3             5                  2                2

4                27            15                  5                5


   NumCatalogPurchases  NumStorePurchases  NumWebVisitsMonth
AcceptedCmp3  \
0                   10                  4                  7
0
1                    1                  2                  5
0
2                    2                 10                  4
0
3                    0                  4                  6
0
4                    3                  6                  5
0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response  Customer_Tenure
0              3         11         1             4461
1              3         11         0             3911
```

```
2                 3          11          0              4110
3                 3          11          0              3937
4                 3          11          0              3959
```

Combine children columns into one

```
# Combine children columns into one
data_sansdt['Total_Children'] = data_sansdt['Kidhome'] +
data_sansdt['Teenhome']
data_sansdt.drop(['Kidhome', 'Teenhome', 'ID'], axis=1, inplace=True)
```

Scale numerical features

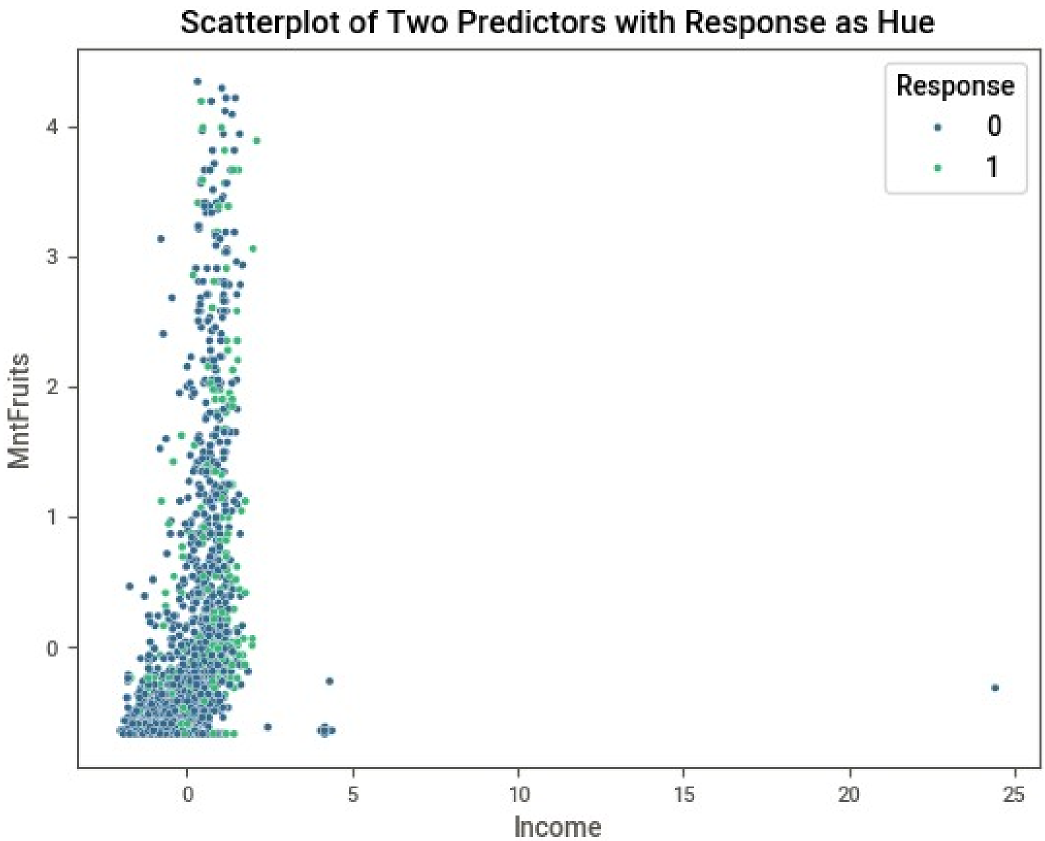
```
# Scale numerical features
numerical_columns = [
    'Income', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
    'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
    'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
    'NumStorePurchases', 'NumWebVisitsMonth', 'Customer_Tenure'
]
scaler = StandardScaler()
data_sansdt[numerical_columns] =
scaler.fit_transform(data_sansdt[numerical_columns])

count = data[data['Year_Birth'] < 1940].shape[0]
print(count)

3
```

Limiting dataset to those born only after 1940 by dropping three data points

```
data_sansdt = data_sansdt[data_sansdt['Year_Birth'] >= 1940]

from collections import Counter

Counter(data_sansdt['Response'])

Counter({0: 1880, 1: 333})

import seaborn as sns
```

Visually checking how target variable imbalance impacts the dataset by examining a scatterplot between two predictors

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

# Define the predictors (X) and target (y)
X = data_sansdt.drop(columns=['Response'])  # Replace 'Response' with
the actual column name if different
y = data_sansdt['Response']

# Scatterplot for the first two predictors
sns.scatterplot(
    x=X.iloc[:, 3],  # First predictor
    y=X.iloc[:, 6],  # Second predictor
    hue=y,           # Target variable
    palette="viridis"
)

# Customize the plot
plt.title("Scatterplot of Two Predictors with Response as Hue")
plt.xlabel(X.columns[3])  # Label for the first predictor
plt.ylabel(X.columns[6])  # Label for the second predictor
plt.legend(title='Response')
plt.show()
```




```
data1 = data_sansdt

data1.head()

   Year_Birth  Education  Marital_Status    Income   Recency  MntWines  \
0        1957          2               4  0.234063  0.310532  0.978226
1        1954          2               4 -0.234559 -0.380509 -0.872024
2        1965          2               5  0.769478 -0.795134  0.358511
3        1984          2               5 -1.017239 -0.795134 -0.872024
4        1981          4               3  0.240221  1.554407 -0.391671

   MntFruits  MntMeatProducts  MntFishProducts  MntSweetProducts  \
0   1.549429         1.690227         2.454568          1.484827
1  -0.637328        -0.717986        -0.651038         -0.633880
2   0.569159        -0.178368         1.340203         -0.146821
3  -0.561922        -0.655551        -0.504892         -0.585174
4   0.418348        -0.218505         0.152766         -0.000703
```

```
   MntGoldProds  NumDealsPurchases  NumWebPurchases  
NumCatalogPurchases  \
0      0.850031           0.351713         1.428553             
2.504712  
1     -0.732867          -0.168231        -1.125881            -
0.571082  
2     -0.037937          -0.688176         1.428553            -
0.229327  
3     -0.752171          -0.168231        -0.760962            -
0.912837  
4     -0.559135           1.391603         0.333796             
0.112428  

   NumStorePurchases  NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4  \
0          -0.554143           0.693232             0             0   
1          -1.169518          -0.131574             0             0   
2           1.291982          -0.543978             0             0   
3          -0.554143           0.280829             0             0   
4           0.061232          -0.131574             0             0   

   AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  
Z_CostContact  \
0             0             0             0         0              3   

1             0             0             0         0              3   

2             0             0             0         0              3   

3             0             0             0         0              3   

4             0             0             0         0              3   


   Z_Revenue  Response  Customer_Tenure  Total_Children  
0         11         1         1.529129               0  
1         11         0        -1.188411               2  
2         11         0        -0.205155               0  
3         11         0        -1.059945               1  
4         11         0        -0.951244               1  
```

Transforming year birth into the numerical 'Age' column for better manipulation

```
data_sansdt['Age'] = (pd.Timestamp.now() - 
pd.to_datetime(data_sansdt['Year_Birth'], format='%Y')).dt.days // 365

data_sansdt = data_sansdt.drop(columns = 'Year_Birth')
```

```
data_sansdt.head()
```

```
   Education  Marital_Status    Income   Recency  MntWines  MntFruits
\
0          2               4  0.234063  0.310532  0.978226   1.549429

1          2               4 -0.234559 -0.380509 -0.872024  -0.637328

2          2               5  0.769478 -0.795134  0.358511   0.569159

3          2               5 -1.017239 -0.795134 -0.872024  -0.561922

4          4               3  0.240221  1.554407 -0.391671   0.418348


   MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds  \
0         1.690227         2.454568          1.484827      0.850031
1        -0.717986        -0.651038         -0.633880     -0.732867
2        -0.178368         1.340203         -0.146821     -0.037937
3        -0.655551        -0.504892         -0.585174     -0.752171
4        -0.218505         0.152766         -0.000703     -0.559135

   NumDealsPurchases  NumWebPurchases  NumCatalogPurchases
NumStorePurchases  \
0           0.351713         1.428553             2.504712            -
0.554143
1          -0.168231        -1.125881            -0.571082            -
1.169518
2          -0.688176         1.428553            -0.229327
1.291982
3          -0.168231        -0.760962            -0.912837            -
0.554143
4           1.391603         0.333796             0.112428
0.061232

   NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4  AcceptedCmp5
AcceptedCmp1  \
0           0.693232             0             0             0
0
1          -0.131574             0             0             0
0
2          -0.543978             0             0             0
0
3           0.280829             0             0             0
0
4          -0.131574             0             0             0
0

   AcceptedCmp2  Complain  Z_CostContact  Z_Revenue  Response  \
0             0         0              3         11         1
```

```
1              0         0              3         11         0
2              0         0              3         11         0
3              0         0              3         11         0
4              0         0              3         11         0

   Customer_Tenure  Total_Children  Age
0         1.529129               0   67
1        -1.188411               2   70
2        -0.205155               0   59
3        -1.059945               1   40
4        -0.951244               1   43
```

Scaling the Age column

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_sansdt['Age'] = scaler.fit_transform(data_sansdt[['Age']])

data_sansdt.head()
```

```
   Education  Marital_Status    Income   Recency  MntWines  MntFruits
\
0          2               4  0.234063  0.310532  0.978226   1.549429

1          2               4 -0.234559 -0.380509 -0.872024  -0.637328

2          2               5  0.769478 -0.795134  0.358511   0.569159

3          2               5 -1.017239 -0.795134 -0.872024  -0.561922

4          4               3  0.240221  1.554407 -0.391671   0.418348


   MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds  \
0         1.690227         2.454568          1.484827      0.850031
1        -0.717986        -0.651038         -0.633880     -0.732867
2        -0.178368         1.340203         -0.146821     -0.037937
3        -0.655551        -0.504892         -0.585174     -0.752171
4        -0.218505         0.152766         -0.000703     -0.559135

   NumDealsPurchases  NumWebPurchases  NumCatalogPurchases
NumStorePurchases  \
0           0.351713         1.428553             2.504712           -
0.554143
1          -0.168231        -1.125881            -0.571082           -
1.169518
2          -0.688176         1.428553            -0.229327
1.291982
```

```
3          -0.168231        -0.760962            -0.912837          -
0.554143
4           1.391603         0.333796             0.112428
0.061232

   NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4  AcceptedCmp5
AcceptedCmp1  \
0           0.693232             0             0             0
0
1          -0.131574             0             0             0
0
2          -0.543978             0             0             0
0
3           0.280829             0             0             0
0
4          -0.131574             0             0             0
0

   AcceptedCmp2  Complain  Z_CostContact  Z_Revenue  Response  \
0             0         0              3         11         1
1             0         0              3         11         0
2             0         0              3         11         0
3             0         0              3         11         0
4             0         0              3         11         0

   Customer_Tenure  Total_Children       Age
0         1.529129               0  1.018785
1        -1.188411               2  1.275248
2        -0.205155               0  0.334882
3        -1.059945               1 -1.289387
4        -0.951244               1 -1.032923
```

Scaling the newly encoded columns

```
# Scale emncoded features
encoded_columns = [
    'Education', 'Marital_Status', 'Z_CostContact',
'Z_Revenue','Total_Children'
]
scaler = StandardScaler()
data_sansdt[encoded_columns] =
scaler.fit_transform(data_sansdt[encoded_columns])

data_sansdt.head()
```

```
   Education  Marital_Status    Income   Recency  MntWines  MntFruits
\
0  -0.352454        0.254202  0.234063  0.310532  0.978226   1.549429
```

```
1  -0.352454        0.254202 -0.234559 -0.380509 -0.872024  -0.637328

2  -0.352454        1.182503  0.769478 -0.795134  0.358511   0.569159

3  -0.352454        1.182503 -1.017239 -0.795134 -0.872024  -0.561922

4   1.430358       -0.674098  0.240221  1.554407 -0.391671   0.418348


   MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds  \
0         1.690227         2.454568          1.484827      0.850031
1        -0.717986        -0.651038         -0.633880     -0.732867
2        -0.178368         1.340203         -0.146821     -0.037937
3        -0.655551        -0.504892         -0.585174     -0.752171
4        -0.218505         0.152766         -0.000703     -0.559135

   NumDealsPurchases  NumWebPurchases  NumCatalogPurchases
NumStorePurchases  \
0           0.351713         1.428553             2.504712           -
0.554143
1          -0.168231        -1.125881            -0.571082           -
1.169518
2          -0.688176         1.428553            -0.229327
1.291982
3          -0.168231        -0.760962            -0.912837           -
0.554143
4           1.391603         0.333796             0.112428
0.061232

   NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4  AcceptedCmp5
AcceptedCmp1  \
0           0.693232             0             0             0
0
1          -0.131574             0             0             0
0
2          -0.543978             0             0             0
0
3           0.280829             0             0             0
0
4          -0.131574             0             0             0
0

   AcceptedCmp2  Complain  Z_CostContact  Z_Revenue  Response  \
0             0         0            0.0        0.0         1
1             0         0            0.0        0.0         0
2             0         0            0.0        0.0         0
3             0         0            0.0        0.0         0
4             0         0            0.0        0.0         0
```

```
   Customer_Tenure  Total_Children       Age
0         1.529129       -1.264914  1.018785
1        -1.188411        1.404857  1.275248
2        -0.205155       -1.264914  0.334882
3        -1.059945        0.069971 -1.289387
4        -0.951244        0.069971 -1.032923
```

```
dfSummary(data_sansdt)
```

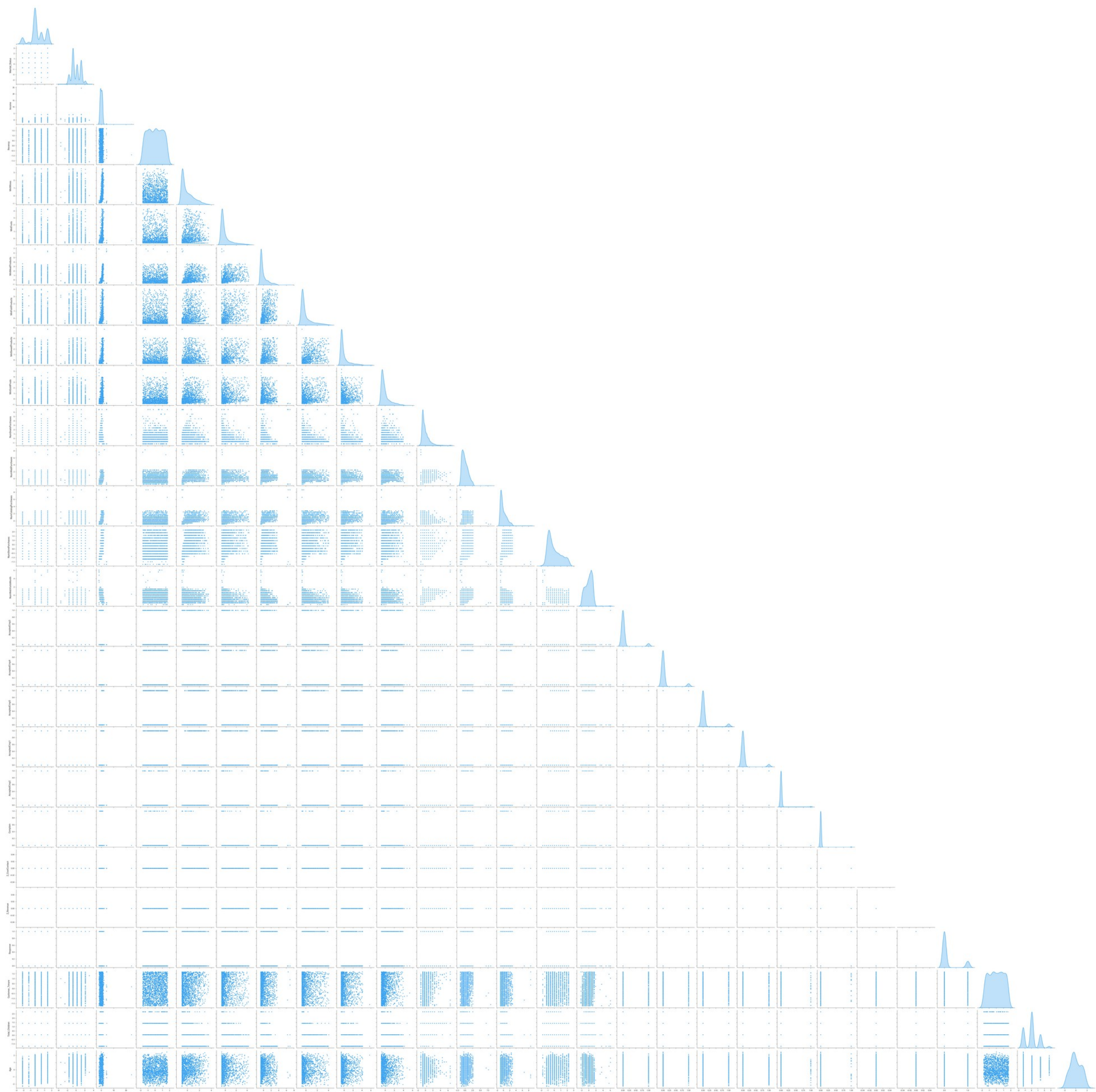
```
<pandas.io.formats.style.Styler at 0x1cf209a55e0>
```

```
dfSummary(data_sansdt)
```

```
<pandas.io.formats.style.Styler at 0x1cf209a55e0>
```

Checking for the assumptions of Regression - Linearity

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a pair plot
sns.pairplot(data_sansdt, diag_kind="kde", corner=True)

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



Dropping the Z Cost contact and Z Revenue variables due to there zero standard deviation and minimal information

```
columns_to_drop = ['Z_CostContact', 'Z_Revenue']
existing_columns = [col for col in columns_to_drop if col in
data_sansdt.columns]

if existing_columns:
    data_sansdt.drop(columns=existing_columns, inplace=True)
    print(f"Dropped columns: {existing_columns}")
else:
    print("No matching columns to drop.")
```

```
No matching columns to drop.
```

Examining the skewed features and log transformation application

```python
import numpy as np

# List of skewed predictors
skewed_predictors = [
    'Income', 'MntWines', 'MntFruits', 'MntMeatProducts',
    'MntFishProducts', 'MntSweetProducts', 'MntGoldProds'
]

# Handle negative values and apply log transformation
for col in skewed_predictors:
    # Replace negative values with NaN
    data_sansdt[col] = data_sansdt[col].apply(lambda x: np.nan if x < 
0 else x)
    # Apply log transformation
    data_sansdt[col] = np.log1p(data_sansdt[col])

for col in skewed_predictors:
    data_sansdt[col].fillna(data_sansdt[col].median(), inplace=True)


# Check if transformation was successful
print("Log transformation applied to skewed predictors:")
print(data_sansdt[skewed_predictors].head())
```

```
Log transformation applied to skewed predictors:
     Income  MntWines  MntFruits  MntMeatProducts  MntFishProducts  \
0  0.210312  0.682200   0.935870         0.989626         1.239697   
1  0.511969  0.620372   0.646287         0.683673         0.671193   
2  0.570684  0.306389   0.450540         0.683673         0.850238   
3  0.511969  0.620372   0.646287         0.683673         0.671193   
4  0.215290  0.620372   0.349493         0.683673         0.142164   

   MntSweetProducts  MntGoldProds  
0          0.910203      0.615202  
1          0.654775      0.625582  
2          0.654775      0.625582  
3          0.654775      0.625582  
4          0.654775      0.625582  
```

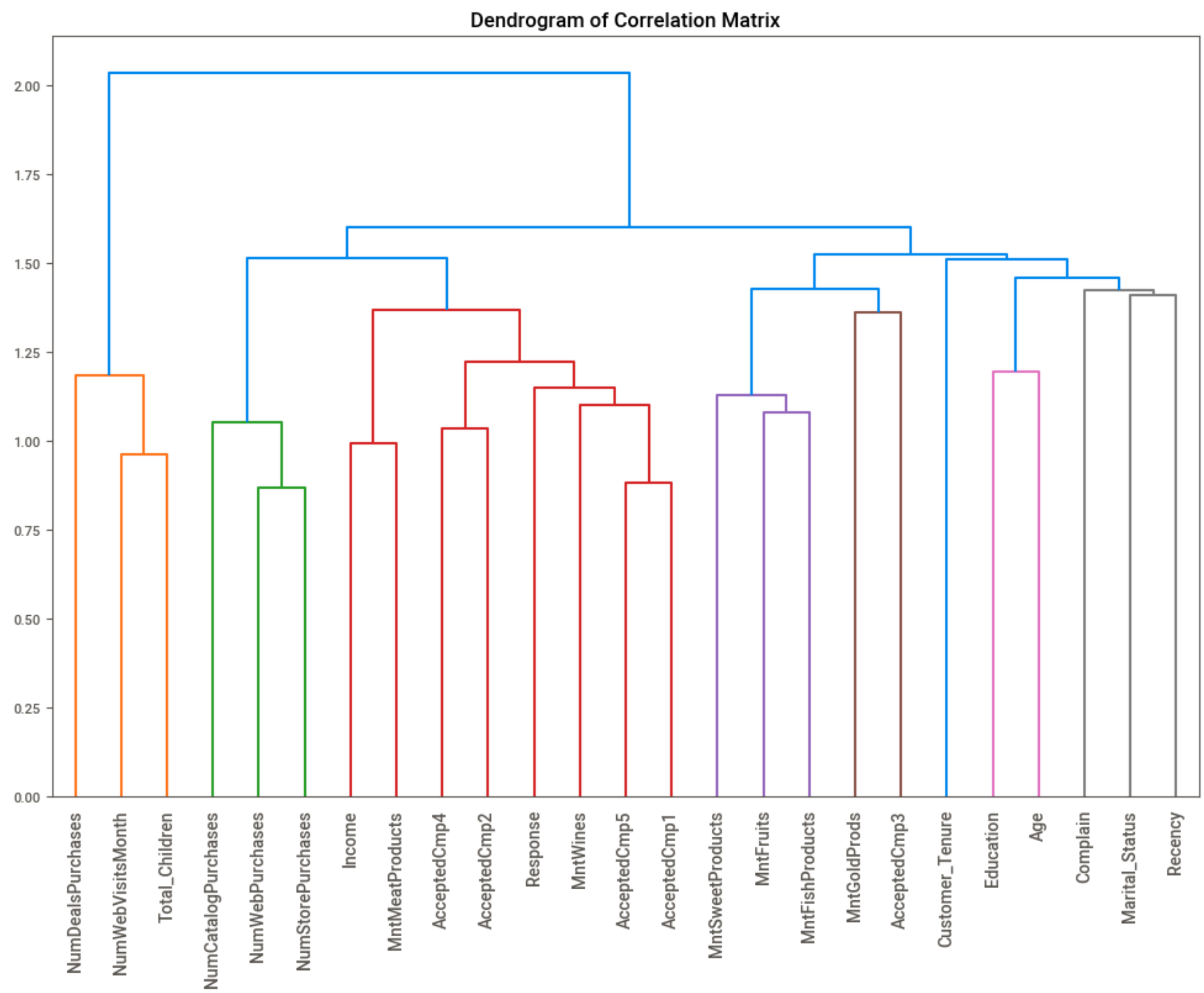
Checking skew after transformation

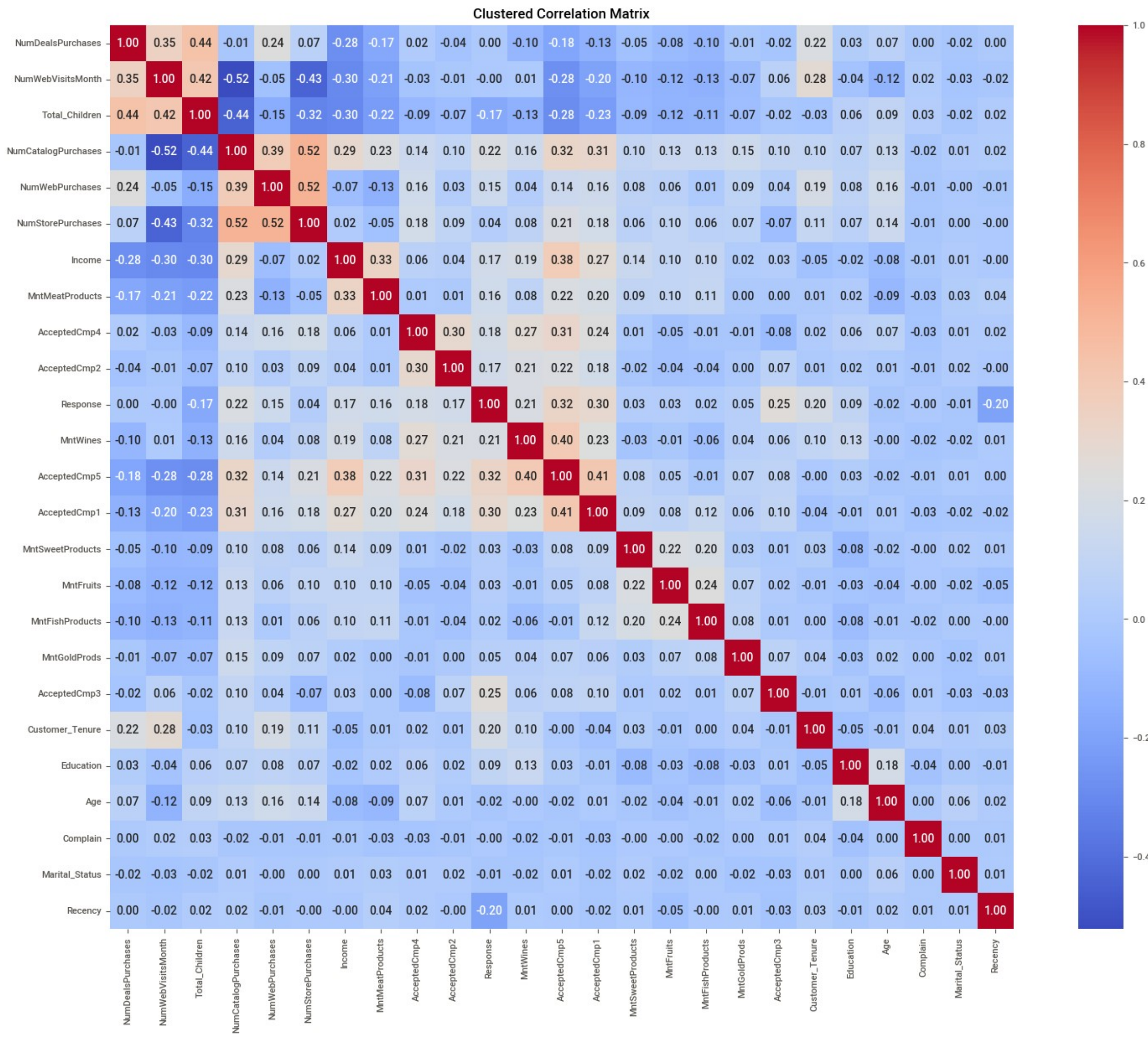
```
from scipy.stats import skew

for col in skewed_predictors:
    print(f"Skewness of {col}: {skew(data_sansdt[col]):.2f}")
```

```
Skewness of Income: 1.57
Skewness of MntWines: 0.58
Skewness of MntFruits: 1.04
Skewness of MntMeatProducts: 0.58
Skewness of MntFishProducts: 0.70
Skewness of MntSweetProducts: 0.95
Skewness of MntGoldProds: 0.89
```

Examining the second assumption of regression modelling - Limited Multicollinearity

```
from scipy.cluster.hierarchy import linkage, dendrogram
import numpy as np

# Compute linkage for clustering
linkage_matrix = linkage(correlation_matrix, method='average')

# Plot dendrogram
plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, labels=correlation_matrix.columns,
leaf_rotation=90)
plt.title("Dendrogram of Correlation Matrix")
plt.show()

# Reorder correlation matrix based on clustering
ordered_indices = dendrogram(linkage_matrix, no_plot=True)['leaves']
reordered_corr = correlation_matrix.iloc[ordered_indices,
ordered_indices]

# Plot the reordered heatmap
plt.figure(figsize=(20, 14))
sns.heatmap(reordered_corr, annot=True, fmt=".2f", cmap="coolwarm",
cbar=True, square=True)
plt.title("Clustered Correlation Matrix")
plt.show()
```


Dendrogram of Correlation Matrix
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
NumDealsPurchases
NumWebVisitsMonth
Total_Children
NumCatalogPurchases
NumWebPurchases
NumStorePurchases
Income
MntMeatProducts
AcceptedCmp4
AcceptedCmp2
Response
MntWines
AcceptedCmp5
AcceptedCmp1
MntSweetProducts
MntFruits
MntFishProducts
MntGoldProds
AcceptedCmp3
Customer_Tenure
Education
Age
Complain
Marital_Status
Recency



```python
from statsmodels.stats.outliers_influence import
variance_inflation_factor
X = data_sansdt.drop(columns=['Response'])



# Replace missing values
X.fillna(X.median(), inplace=True)

# Drop constant columns
constant_cols = [col for col in X.columns if X[col].std() == 0]
X.drop(columns=constant_cols, inplace=True)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
```

```
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]

print(vif_data)
```

```
                Feature       VIF
0             Education  1.090374
1        Marital_Status  1.009655
2                Income  9.953536
3               Recency  1.010645
4              MntWines  8.823181
5             MntFruits  8.648300
6       MntMeatProducts  9.890597
7       MntFishProducts  9.174705
8      MntSweetProducts  8.442964
9          MntGoldProds  6.763200
10    NumDealsPurchases  1.656132
11      NumWebPurchases  1.692045
12  NumCatalogPurchases  2.194465
13    NumStorePurchases  2.041905
14    NumWebVisitsMonth  2.349884
15         AcceptedCmp3  1.179194
16         AcceptedCmp4  1.392726
17         AcceptedCmp5  1.717832
18         AcceptedCmp1  1.420706
19         AcceptedCmp2  1.171125
20             Complain  1.015093
21      Customer_Tenure  1.268243
22       Total_Children  1.790751
23                  Age  1.131812
```

Data Splicing

```
# Split data into train-test sets
X = data_sansdt.drop('Response', axis=1)
y = data_sansdt['Response']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, stratify=y, random_state=42)

X_train.head()
```

```
      Education  Marital_Status    Income   Recency  MntWines
MntFruits  \
2211  -0.352454        1.182503  0.889335 -0.242301  0.491196
0.752421
925    0.538952        1.182503  0.511969  1.692615  0.620372
0.040490
160    0.538952        0.254202  0.511969 -1.693488  0.620372
0.646287
2014  -0.352454        0.254202  0.492917 -0.622374  1.113723
```

```
1.148251
725   -0.352454       -0.674098  0.511969  1.727167  0.620372
0.646287

      MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds
\
2211         1.408681         0.698818          0.654775      0.160760

925          0.201308         0.671193          0.654775      0.863408

160          0.683673         0.671193          0.654775      0.625582

2014         0.749022         0.076677          0.412579      0.625582

725          0.683673         0.671193          0.654775      0.625582


      NumDealsPurchases  NumWebPurchases  NumCatalogPurchases  \
2211          -0.688176         0.698715            -0.229327
925            0.351713         0.698715            -0.571082
160           -0.688176        -1.125881            -0.912837
2014          -0.688176        -0.760962             2.162957
725           -0.688176        -0.760962            -0.912837

      NumStorePurchases  NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4
\
2211           0.984294          -1.368784             0             0

925            0.061232           1.105635             0             0

160           -1.169518           0.693232             0             0

2014           2.215044          -0.131574             0             0

725           -0.861830           0.280829             0             0


      AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain
Customer_Tenure  \
2211             0             0             0         0         -
1.035240
925              0             0             0         0
1.548893
160              0             0             0         0
0.046835
2014             0             0             0         0
1.667476
725              0             0             0         0         -
1.642981

      Total_Children       Age
```

```
2211       -1.264914 -1.032923
925        -1.264914 -0.861947
160         2.739742  0.933297
2014       -1.264914  0.933297
725         0.069971  0.249394
```

```
X_test.head()
```

```
      Education  Marital_Status    Income   Recency  MntWines  
MntFruits  \
1359   0.538952        0.254202  0.511969 -0.518718  0.620372   
0.646287   
1779  -2.135266       -0.674098  0.419643 -1.658936  0.620372   
0.646287   
1839   1.430358       -1.602398  0.184051  0.206876  1.059719   
0.646287   
1151  -0.352454        1.182503  0.489537  1.623511  1.013451   
1.254066   
561   -0.352454        0.254202  0.775604  0.897917  0.620372   
1.431080   

      MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds
\
1359         0.683673         0.671193          0.654775      0.625582

1779         0.683673         1.041494          0.919956      0.625582

1839         0.705934         0.531107          0.562035      1.266595

1151         0.121641         0.680486          1.121449      0.625582

561          1.492304         0.316259          1.282339      0.625582


      NumDealsPurchases  NumWebPurchases  NumCatalogPurchases  \
1359           0.351713        -0.760962            -0.571082   
1779          -0.688176        -0.760962             0.112428   
1839           4.511271         1.428553             0.795937   
1151           0.351713         2.158392             0.454182   
561           -1.208121         0.698715             1.137692   

      NumStorePurchases  NumWebVisitsMonth  AcceptedCmp3  AcceptedCmp4
\
1359          -0.554143           0.280829             0             0

1779           2.215044          -1.781187             0             0

1839           0.984294           0.280829             0             0

1151           0.368919           0.280829             0             0
```

```
561              0.368919         -1.368784             0              0


      AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain
Customer_Tenure  \
1359             0             0             0         0         -
0.422559
1779             0             0             0         0         -
0.106336
1839             0             0             0         0
1.440191
1151             0             0             0         0
1.558774
561              0             0             0         0
0.120949

      Total_Children       Age
1359        1.404857 -0.178045
1779       -1.264914 -1.289387
1839        0.069971  1.275248
1151        0.069971  0.420370
561        -1.264914 -0.947435
```

Logistic Regression Model

```python
import statsmodels.api as sm
import pandas as pd

# Add a constant to the predictors (for intercept)
X_train_sm = sm.add_constant(X_train)

# Fit the logistic regression model
logit_model = sm.Logit(y_train, X_train_sm)
logit_result = logit_model.fit()

# Print the summary
print(logit_result.summary())



Optimization terminated successfully.
         Current function value: 0.272434
         Iterations 8
                           Logit Regression Results

======================================================================
========
Dep. Variable:               Response   No. Observations:
1549
Model:                          Logit   Df Residuals:
```

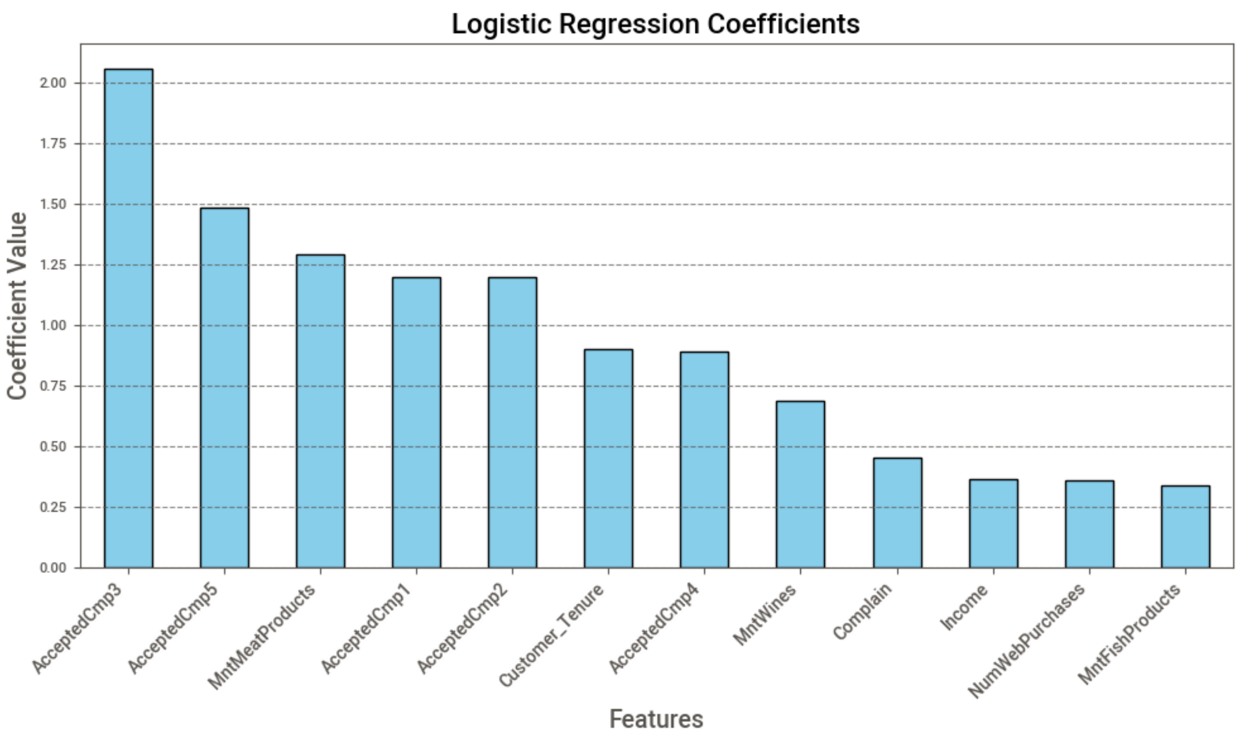
```
1524
Method:                           MLE   Df Model:
24
Date:                Thu, 21 Nov 2024   Pseudo R-squ.:
0.3566
Time:                        21:32:07   Log-Likelihood:
-422.00
converged:                       True   LL-Null:
-655.90
Covariance Type:            nonrobust   LLR p-value:
7.882e-84
======================================================================
=================
                          coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
-----------------
const                  -4.6890      0.603     -7.777      0.000      -
5.871      -3.507
Education               0.2652      0.097      2.736      0.006
0.075       0.455
Marital_Status         -0.0709      0.090     -0.788      0.431      -
0.247       0.105
Income                  0.3629      0.479      0.758      0.449      -
0.576       1.301
Recency                -0.7930      0.098     -8.067      0.000      -
0.986      -0.600
MntWines                0.6878      0.420      1.639      0.101      -
0.135       1.510
MntFruits               0.2425      0.367      0.661      0.509      -
0.476       0.961
MntMeatProducts         1.2905      0.414      3.120      0.002
0.480       2.101
MntFishProducts         0.3377      0.366      0.922      0.357      -
0.380       1.056
MntSweetProducts       -0.0958      0.360     -0.266      0.790      -
0.801       0.609
MntGoldProds           -0.2865      0.330     -0.868      0.385      -
0.934       0.361
NumDealsPurchases       0.2498      0.116      2.159      0.031
0.023       0.477
NumWebPurchases         0.3603      0.104      3.458      0.001
0.156       0.565
NumCatalogPurchases     0.1623      0.137      1.188      0.235      -
0.105       0.430
NumStorePurchases      -0.7931      0.140     -5.684      0.000      -
1.067      -0.520
NumWebVisitsMonth       0.0852      0.132      0.643      0.520      -
0.174       0.345
```

```
AcceptedCmp3             2.0571      0.263      7.816      0.000
1.541       2.573
AcceptedCmp4             0.8895      0.332      2.675      0.007
0.238       1.541
AcceptedCmp5             1.4826      0.347      4.278      0.000
0.803       2.162
AcceptedCmp1             1.1984      0.355      3.374      0.001
0.502       1.895
AcceptedCmp2             1.1976      0.637      1.881      0.060      -
0.050       2.445
Complain                 0.4513      0.884      0.511      0.610      -
1.281       2.183
Customer_Tenure          0.8997      0.110      8.197      0.000
0.685       1.115
Total_Children          -0.3735      0.133     -2.804      0.005      -
0.635      -0.112
Age                      0.0105      0.094      0.111      0.912      -
0.175       0.196
==============================================================================
=================
```

```python
logit_result.params.sort_values(ascending = False)
```

```
AcceptedCmp3           2.057070
AcceptedCmp5           1.482601
MntMeatProducts        1.290520
AcceptedCmp1           1.198401
AcceptedCmp2           1.197637
Customer_Tenure        0.899668
AcceptedCmp4           0.889458
MntWines               0.687785
Complain               0.451253
Income                 0.362861
NumWebPurchases        0.360328
MntFishProducts        0.337670
Education              0.265183
NumDealsPurchases      0.249827
MntFruits              0.242478
NumCatalogPurchases    0.162336
NumWebVisitsMonth      0.085186
Age                    0.010462
Marital_Status        -0.070877
MntSweetProducts      -0.095783
MntGoldProds          -0.286521
Total_Children        -0.373522
Recency               -0.793045
NumStorePurchases     -0.793099
const                 -4.688960
dtype: float64
```

```python
import matplotlib.pyplot as plt

# Assuming logit_result.params exists, create sorted coefficients
sorted_coefficients = logit_result.params.sort_values(ascending=False)

# Plot the coefficients as a colorful bar plot
plt.figure(figsize=(10, 6))
sorted_coefficients[0:12].plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Logistic Regression Coefficients', fontsize=16)
plt.xlabel('Features', fontsize=14)
plt.ylabel('Coefficient Value', fontsize=14)
plt.xticks(rotation=45, fontsize=10, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```




```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.linear_model import LogisticRegression
logistic=LogisticRegression()
logistic.fit(X_train,y_train)
y_pred=logistic.predict(X_test)
```

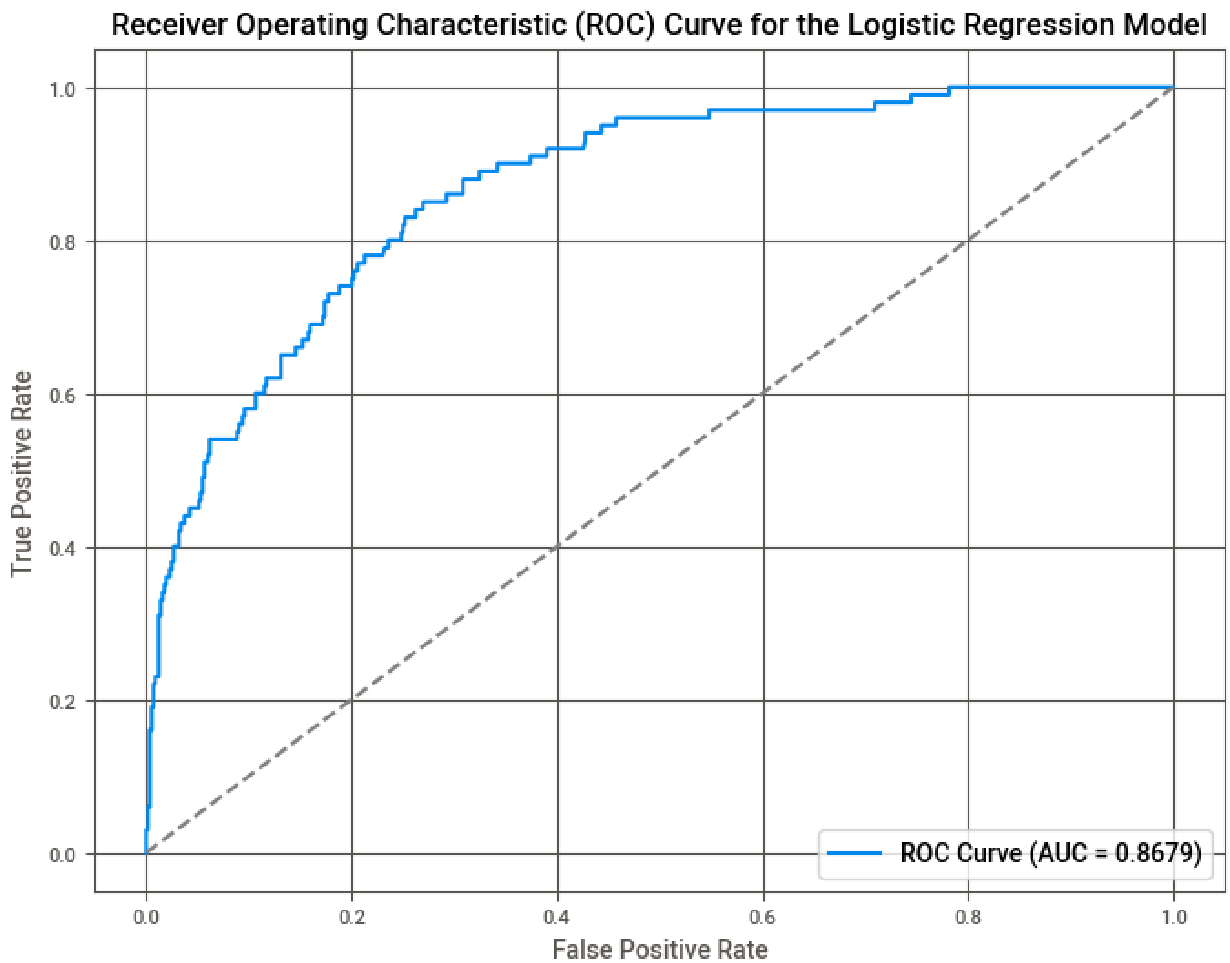
Logistic Regression Metrics

```
score=accuracy_score(y_pred,y_test)
print(f" The Logisitc Regression Model Metrics: \n The accuracy is : 
{score}")
print(f"The Classification Report is : \n 
{classification_report(y_pred,y_test)}")
print(f"The Confusion Matrix is : \n 
{confusion_matrix(y_pred,y_test)}")
```

```
 The Logisitc Regression Model Metrics: 
 The accuracy is : 0.8855421686746988
The Classification Report is : 
               precision    recall  f1-score   support

           0       0.97      0.90      0.94       608
           1       0.40      0.71      0.51        56

    accuracy                           0.89       664
   macro avg       0.69      0.81      0.72       664
weighted avg       0.92      0.89      0.90       664

The Confusion Matrix is : 
 [[548  60]
 [ 16  40]]
```

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities for the positive class
y_pred_proba = logistic.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal 
line
plt.title("Receiver Operating Characteristic (ROC) Curve for the 
Logistic Regression Model")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
AUC Score: 0.8679
```



Hyperparameter Tuning and rectifying the Target variable imbalance by assigning weights

```
## Hyperparamter tuning
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
penalty=['l1', 'l2', 'elasticnet']
c_values=[100,10,1.0,0.1,0.01]
solver=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
class_weight=[{0:w,1:y} for w in [1,10,50,100] for y in [1,10,50,100]]

params=dict(penalty=penalty,C=c_values,solver=solver,class_weight=clas
s_weight)

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
cv=StratifiedKFold()
```

```python
grid=GridSearchCV(estimator=model,param_grid=params,scoring='accuracy'
,cv=cv)

grid.fit(X_train,y_train)
```

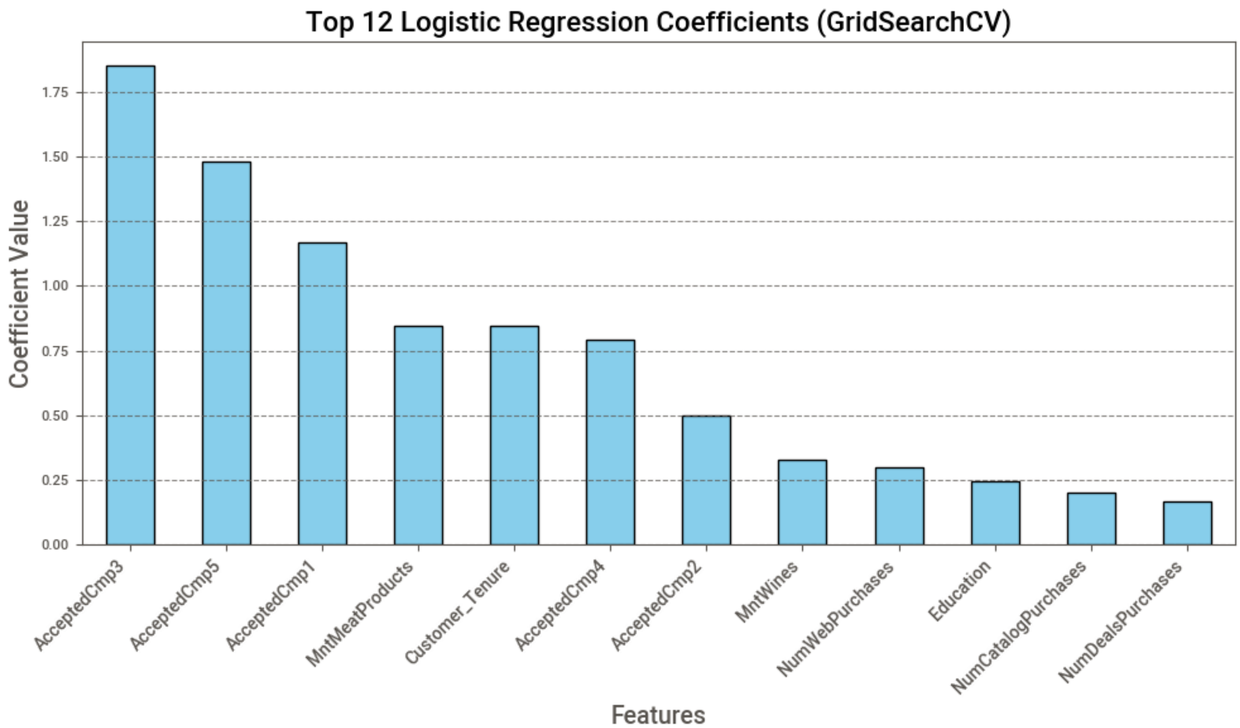
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None,
shuffle=False),
             estimator=LogisticRegression(),
             param_grid={'C': [100, 10, 1.0, 0.1, 0.01],
                         'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 10},
                                          {0: 1, 1: 50}, {0: 1, 1:
100},
                                          {0: 10, 1: 1}, {0: 10, 1:
10},
                                          {0: 10, 1: 50}, {0: 10, 1:
100},
                                          {0: 50, 1: 1}, {0: 50, 1:
10},
                                          {0: 50, 1: 50}, {0: 50, 1:
100},
                                          {0: 100, 1: 1}, {0: 100, 1:
10},
                                          {0: 100, 1: 50}, {0: 100, 1:
100}],
                         'penalty': ['l1', 'l2', 'elasticnet'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear',
'sag',
                                    'saga']},
             scoring='accuracy')
```

```python
grid.best_params_
```

```
{'C': 1.0, 'class_weight': {0: 1, 1: 1}, 'penalty': 'l1', 'solver':
'saga'}
```

```python
best_model = grid.best_estimator_

# Assuming coefficients are from a grid search fitted logistic
regression model
grid_coefficients = pd.Series(best_model.coef_.flatten(),
index=X_train.columns).sort_values(ascending=False)

# Plot the top 12 coefficients as a colorful bar plot
plt.figure(figsize=(10, 6))
grid_coefficients[:12].plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Top 12 Logistic Regression Coefficients (GridSearchCV)',
fontsize=16)
plt.xlabel('Features', fontsize=14)
plt.ylabel('Coefficient Value', fontsize=14)
plt.xticks(rotation=45, fontsize=10, ha='right')
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



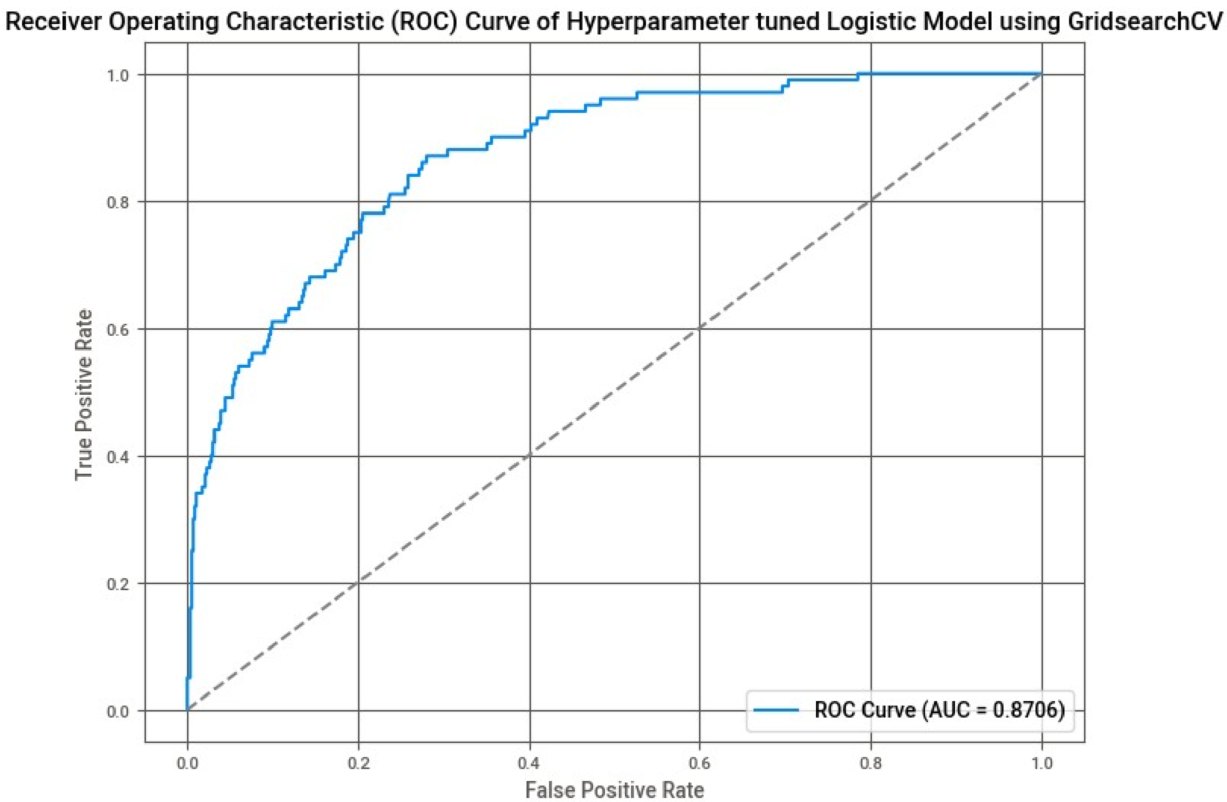

```
y_pred1=grid.predict(X_test)

score=accuracy_score(y_pred1,y_test)
print(f" The Hyperparameter Tuned Logisitc Regression Model Metrics: \
n The accuracy is : {score}")
print(f"The Classification Report is : \n
{classification_report(y_pred1,y_test)}")
print(f"The Confusion Matrix is : \n
{confusion_matrix(y_pred1,y_test)}")
```

```
 The Hyperparameter Tuned Logisitc Regression Model Metrics:
 The accuracy is : 0.8900602409638554
The Classification Report is :
               precision    recall  f1-score   support

           0       0.97      0.91      0.94       605
           1       0.43      0.73      0.54        59

    accuracy                           0.89       664
   macro avg       0.70      0.82      0.74       664
weighted avg       0.92      0.89      0.90       664
```

```
The Confusion Matrix is :
 [[548  57]
 [ 16  43]]
```

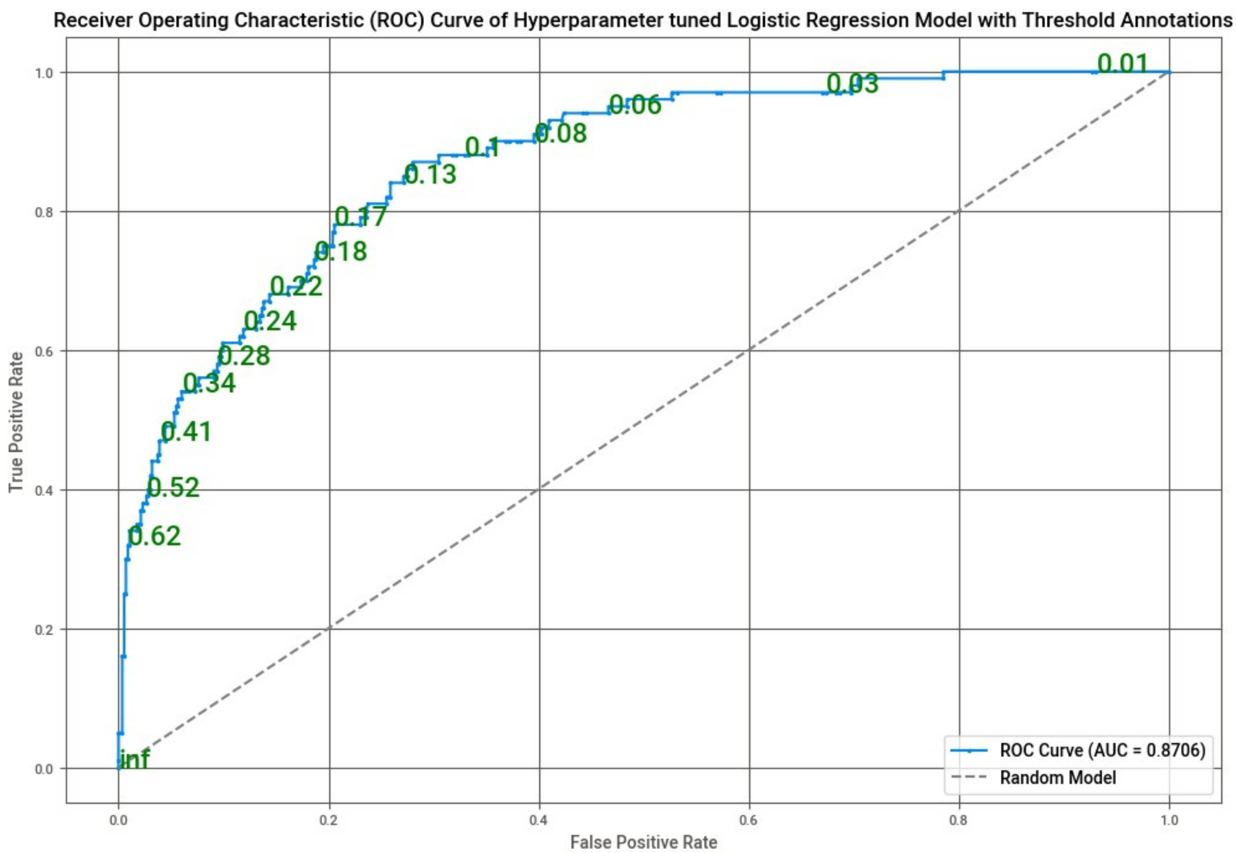
```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities for the positive class
y_pred_proba1 = grid.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba1)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba1)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal
line
plt.title("Receiver Operating Characteristic (ROC) Curve of
Hyperparameter tuned Logistic Model using GridsearchCV")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
AUC Score: 0.8706
```




```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import numpy as np

# Predict probabilities for the positive class
y_pred_proba1 = grid.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba1)

# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba1)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve with limited annotations
fig = plt.figure(figsize=(12, 8))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})",
marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')

# Add limited annotations for thresholds (e.g., every 10th point)

for i, xy in enumerate(zip(fpr, tpr, thresholds)):
    if i % 10 == 0:  # Annotate every 10th point
```

```
        plt.annotate(f'{np.round(xy[2], 2)}', xy=(xy[0], xy[1]),
fontsize=16, color='green')

# Axis labels and title
plt.title("Receiver Operating Characteristic (ROC) Curve of
Hyperparameter tuned Logistic Regression Model with Threshold
Annotations")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

# Show legend
plt.legend(loc="lower right")

# Show grid
plt.grid()

# Show the plot
plt.show()

AUC Score: 0.8706
```



Hyperparameter Tuning using randomized search CV and rectifying the Target variable imbalance by assigning weights

```
from sklearn.model_selection import RandomizedSearchCV

model=LogisticRegression()
randomcv=RandomizedSearchCV(estimator=model,param_distributions=params
,cv=5,scoring='accuracy')

randomcv.fit(X_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(),
                   param_distributions={'C': [100, 10, 1.0, 0.1,
0.01],
                                        'class_weight': [{0: 1, 1: 1},
                                                         {0: 1, 1:
10},
                                                         {0: 1, 1:
50},
                                                         {0: 1, 1:
100},
                                                         {0: 10, 1:
1},
                                                         {0: 10, 1:
10},
                                                         {0: 10, 1:
50},
                                                         {0: 10, 1:
100},
                                                         {0: 50, 1:
1},
                                                         {0: 50, 1:
10},
                                                         {0: 50, 1:
50},
                                                         {0: 50, 1:
100},
                                                         {0: 100, 1:
1},
                                                         {0: 100, 1:
10},
                                                         {0: 100, 1:
50},
                                                         {0: 100, 1:
100}],
                                        'penalty': ['l1', 'l2',
'elasticnet'],
                                        'solver': ['newton-cg',
```

```
'lbfgs',
                                                                   'liblinear', 'sag',
                                                                   'saga']},
                   scoring='accuracy')
```

```
randomcv.best_score_
```

```
0.8760622194383547
```

```
randomcv.best_params_
```

```
{'solver': 'liblinear',
 'penalty': 'l2',
 'class_weight': {0: 50, 1: 50},
 'C': 10}
```

```
y_pred2=randomcv.predict(X_test)
```

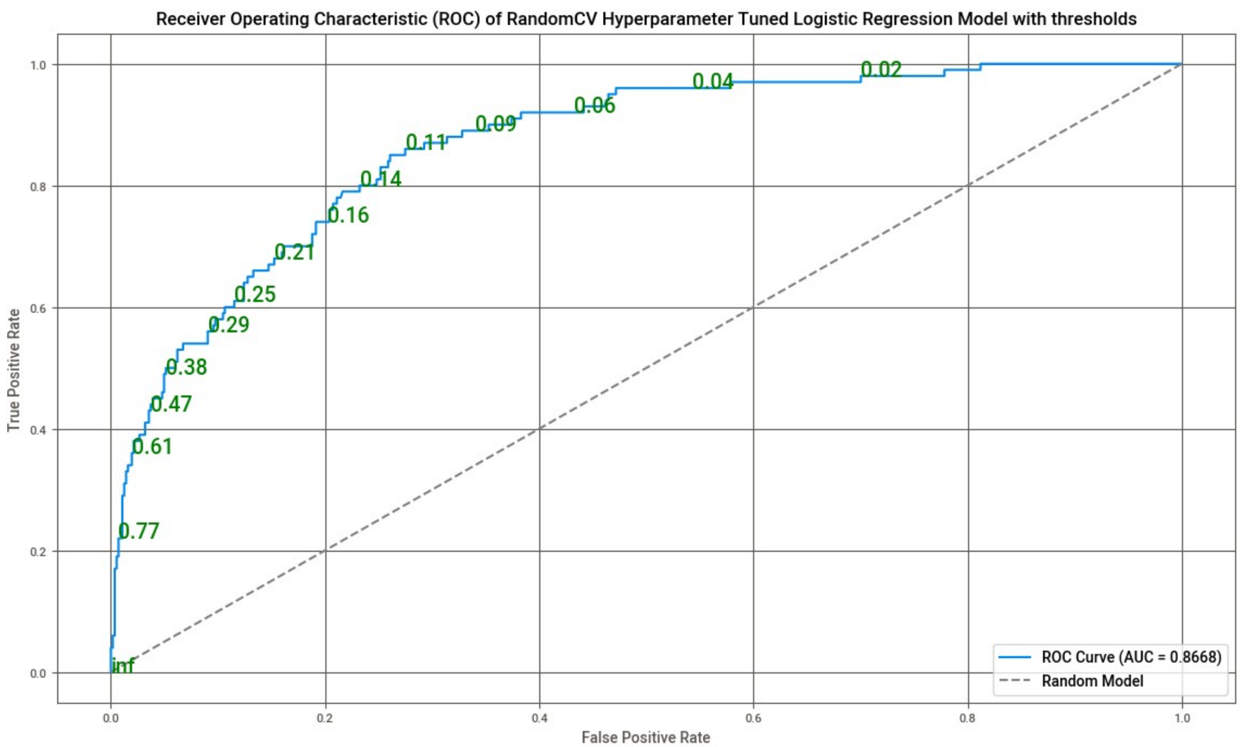
```
score=accuracy_score(y_pred2,y_test)
print(f" The Random Search CV Hyperparameter Tuned Logisitc Regression
Model Metrics: \n The accuracy is : {score}")
print(f"The Classification Report is : \n
{classification_report(y_pred2,y_test)}")
print(f"The Confusion Matrix is : \n
{confusion_matrix(y_pred2,y_test)}")
```

```
 The Random Search CV Hyperparameter Tuned Logisitc Regression Model
Metrics:
 The accuracy is : 0.8855421686746988
The Classification Report is :
               precision    recall  f1-score   support

           0       0.97      0.90      0.94       606
           1       0.41      0.71      0.52        58

    accuracy                           0.89       664
   macro avg       0.69      0.80      0.73       664
weighted avg       0.92      0.89      0.90       664

The Confusion Matrix is :
 [[547  59]
 [ 17  41]]
```

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import numpy as np

# Predict probabilities for the positive class
y_pred_proba2 = randomcv.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba2)
```

```
# Calculate the AUC
auc_score = roc_auc_score(y_test, y_pred_proba2)
print(f"AUC Score: {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(14, 8))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label="Random
Model")  # Diagonal line

# Add limited annotations for thresholds (e.g., every 50th point)
for i in range(0, len(thresholds), 10):  # Change step size to adjust
annotation density
    plt.annotate(f'{np.round(thresholds[i], 2)}', (fpr[i], tpr[i]),
fontsize=14, color='green')

# Axis labels and title
plt.title("Receiver Operating Characteristic (ROC) of RandomCV
Hyperparameter Tuned Logistic Regression Model with thresholds")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

# Add legend
plt.legend(loc="lower right")

# Add grid
plt.grid()

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

```
AUC Score: 0.8668
```



Support Vector Machine

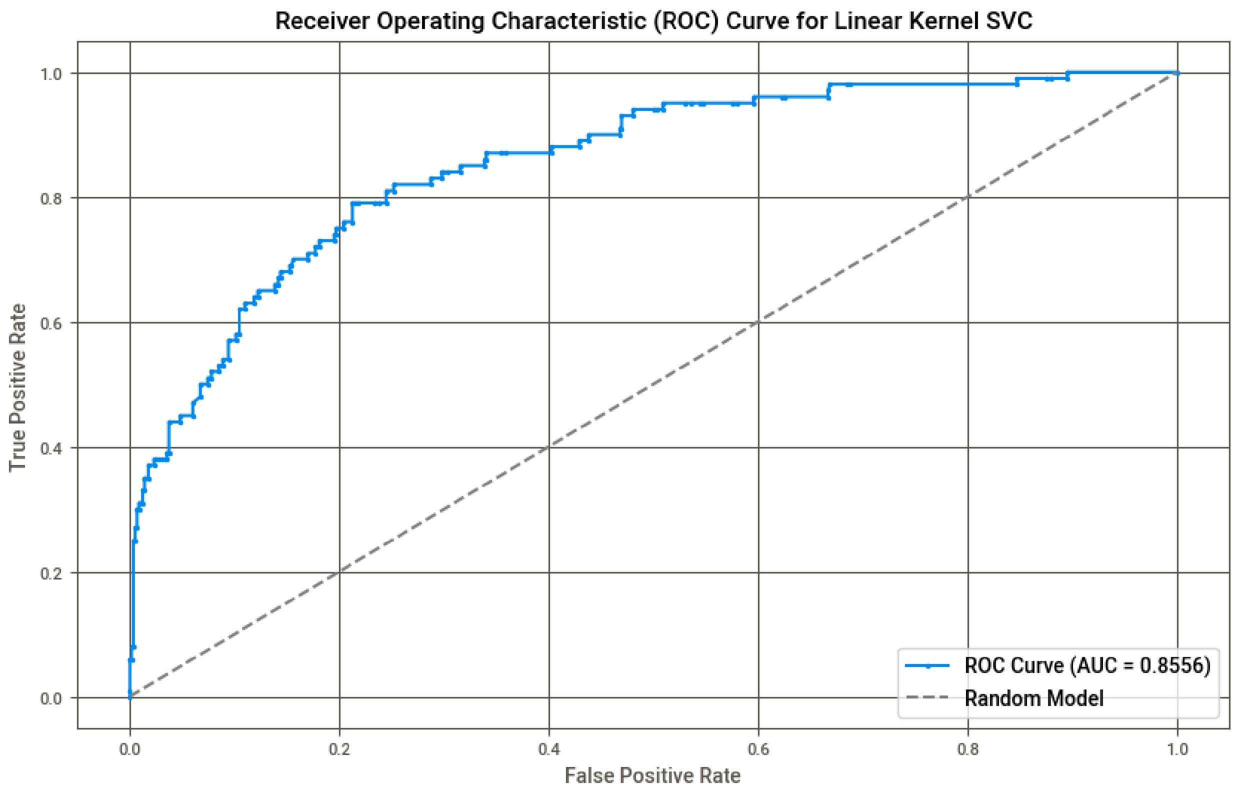
```
from sklearn.svm import SVC

svcl=SVC(kernel='linear', probability=True)

svcl.fit(X_train,y_train)

SVC(kernel='linear', probability=True)

# Get feature names and coefficients
coefficients = svcl.coef_[0]  # Coefficients for the first class
features = X_train.columns   # Feature names (if X_train is a
DataFrame)

# Create a DataFrame for better visualization
coef_df = pd.DataFrame({'Feature': features, 'Coefficient':
coefficients})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)

print("Coefficients for Predictors:")
print(coef_df)

Coefficients for Predictors:
                Feature  Coefficient
15          AcceptedCmp3     1.268511
17          AcceptedCmp5     0.961150
6       MntMeatProducts     0.928353
```

18	AcceptedCmp1	0.905359
4	MntWines	0.596882
16	AcceptedCmp4	0.513408
21	Customer_Tenure	0.441372
7	MntFishProducts	0.392129
2	Income	0.381166
19	AcceptedCmp2	0.268985
5	MntFruits	0.259621
20	Complain	0.211284
11	NumWebPurchases	0.192302
10	NumDealsPurchases	0.137807
0	Education	0.080210
12	NumCatalogPurchases	0.046844
23	Age	0.044956
14	NumWebVisitsMonth	-0.003640
9	MntGoldProds	-0.039200
1	Marital_Status	-0.049933
22	Total_Children	-0.188926
3	Recency	-0.355306
8	MntSweetProducts	-0.381656
13	NumStorePurchases	-0.417440

```python
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predictions
y_predl = svcl.predict(X_test)

# Accuracy
print("Accuracy of the SVM Linear Kernel is:", accuracy_score(y_test,
y_predl))

# Confusion Matrix
print("\nConfusion Matrix of the SVM Linear Kernel is:")
print(confusion_matrix(y_test, y_predl))

# Classification Report
print("\nClassification Report of the SVM Linear Kernel is:")
print(classification_report(y_test, y_predl))

# Predict probabilities for ROC curve
y_pred_proba = svcl.predict_proba(X_test)[:, 1]

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc_score = roc_auc_score(y_test, y_pred_proba)

# Print AUC
print(f"\nAUC Score: {auc_score:.4f}")
```

```python
# Plot ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})",
marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')
plt.title("Receiver Operating Characteristic (ROC) Curve for Linear
Kernel SVC")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
Accuracy of the SVM Linear Kernel is: 0.8689759036144579

Confusion Matrix of the SVM Linear Kernel is:
[[530  34]
 [ 53  47]]

Classification Report of the SVM Linear Kernel is:
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       564
           1       0.58      0.47      0.52       100

    accuracy                           0.87       664
   macro avg       0.74      0.70      0.72       664
weighted avg       0.86      0.87      0.86       664


AUC Score: 0.8556
```



Radial Basis Function Kernel Support Vector Classifier

```
# Train SVC with RBF kernel and probability=True
rbf = SVC(kernel='rbf', probability=True)
rbf.fit(X_train, y_train)
```

```
SVC(probability=True)
```

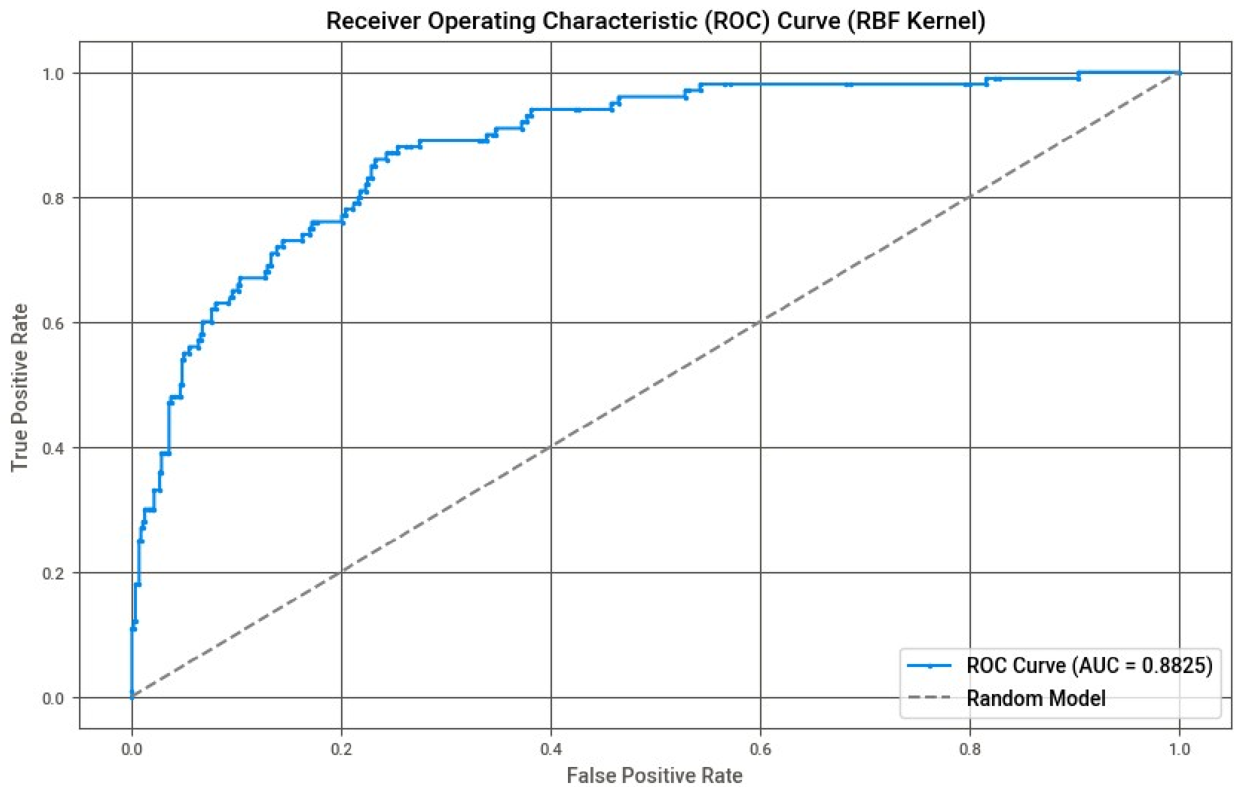
```
# Predictions
y_pred_rbf = rbf.predict(X_test)

# Accuracy
print("Accuracy of the SVM RBF Kernel is:", accuracy_score(y_test,
y_pred_rbf))

# Confusion Matrix
print("\nConfusion Matrix of the SVM RBF Kernel is:")
print(confusion_matrix(y_test, y_pred_rbf))

# Classification Report
print("\nClassification Report of the SVM RBF Kernel is:")
print(classification_report(y_test, y_pred_rbf))
```

```
Accuracy of the SVM RBF Kernel is: 0.8795180722891566

Confusion Matrix of the SVM RBF Kernel is:
```

```
[[553  11]
 [ 69  31]]

Classification Report of the SVM RBF Kernel is:
              precision    recall  f1-score   support

           0       0.89      0.98      0.93       564
           1       0.74      0.31      0.44       100

    accuracy                           0.88       664
   macro avg       0.81      0.65      0.68       664
weighted avg       0.87      0.88      0.86       664
```

```
# Predict probabilities for ROC curve
y_pred_proba_rbf = rbf.predict_proba(X_test)[:, 1]

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_rbf)
auc_score = roc_auc_score(y_test, y_pred_proba_rbf)

# Print AUC
print(f"\nAUC Score: {auc_score:.4f}")

# Plot ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})",
marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')
plt.title("Receiver Operating Characteristic (ROC) Curve (RBF
Kernel)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
AUC Score: 0.8825
```



Polynomial Kernel Support Vector Classifier

```
# Initialize SVC with Polynomial kernel and probability=True
poly = SVC(kernel='poly', probability=True)

# Fit the model to the training data
poly.fit(X_train, y_train)

SVC(kernel='poly', probability=True)

# Predict class labels on the test set
y_pred_poly = poly.predict(X_test)

# Predict probabilities for ROC curve
y_pred_proba_poly = poly.predict_proba(X_test)[:, 1]

# Calculate accuracy
accuracy_poly = accuracy_score(y_test, y_pred_poly)
print(f"Accuracy (Poly Kernel): {accuracy_poly:.4f}")

# Generate confusion matrix
conf_matrix_poly = confusion_matrix(y_test, y_pred_poly)
print("\nConfusion Matrix (Poly Kernel):")
print(conf_matrix_poly)

# Generate classification report
```

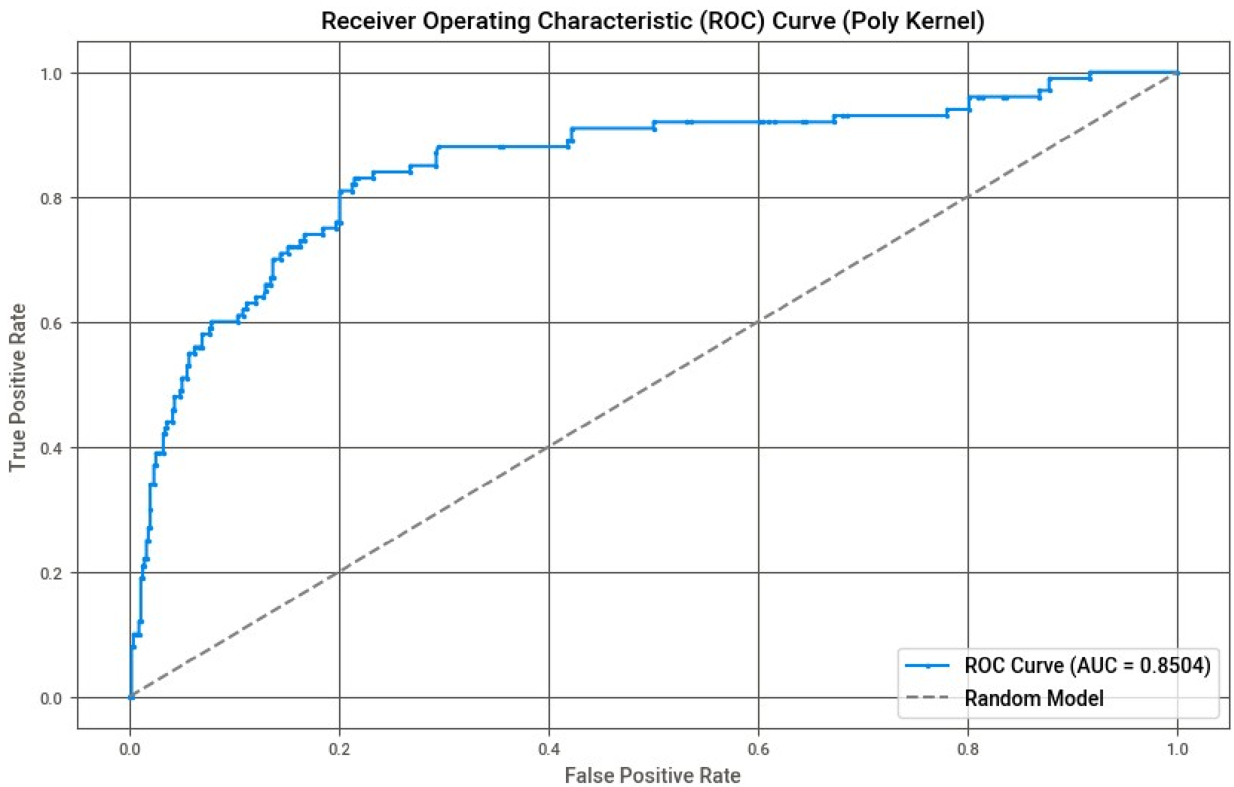
```
class_report_poly = classification_report(y_test, y_pred_poly)
print("\nClassification Report (Poly Kernel):")
print(class_report_poly)
```

```
Accuracy (Poly Kernel): 0.8795

Confusion Matrix (Poly Kernel):
[[549  15]
 [ 65  35]]

Classification Report (Poly Kernel):
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       564
           1       0.70      0.35      0.47       100

    accuracy                           0.88       664
   macro avg       0.80      0.66      0.70       664
weighted avg       0.86      0.88      0.86       664
```

```
# Compute ROC curve
fpr_poly, tpr_poly, thresholds_poly = roc_curve(y_test,
y_pred_proba_poly)

# Compute AUC score
auc_score_poly = roc_auc_score(y_test, y_pred_proba_poly)
print(f"\nAUC Score (Poly Kernel): {auc_score_poly:.4f}")
```

```
AUC Score (Poly Kernel): 0.8504
```

```
# Plot ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr_poly, tpr_poly, label=f"ROC Curve (AUC =
{auc_score_poly:.4f})", marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')
plt.title("Receiver Operating Characteristic (ROC) Curve (Poly
Kernel)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Sigmoid Kernel Support Vector Classifier

```
# Initialize SVC with Sigmoid kernel and probability=True
sigmoid = SVC(kernel='sigmoid', probability=True)

# Fit the model to the training data
sigmoid.fit(X_train, y_train)
```

```
SVC(kernel='sigmoid', probability=True)
```

```
# Predict class labels on the test set
y_pred_sigmoid = sigmoid.predict(X_test)

# Predict probabilities for ROC curve
y_pred_proba_sigmoid = sigmoid.predict_proba(X_test)[:, 1]

# Calculate accuracy
accuracy_sigmoid = accuracy_score(y_test, y_pred_sigmoid)
print(f"Accuracy (Sigmoid Kernel): {accuracy_sigmoid:.4f}")

# Generate confusion matrix
conf_matrix_sigmoid = confusion_matrix(y_test, y_pred_sigmoid)
print("\nConfusion Matrix (Sigmoid Kernel):")
print(conf_matrix_sigmoid)

# Generate classification report
```

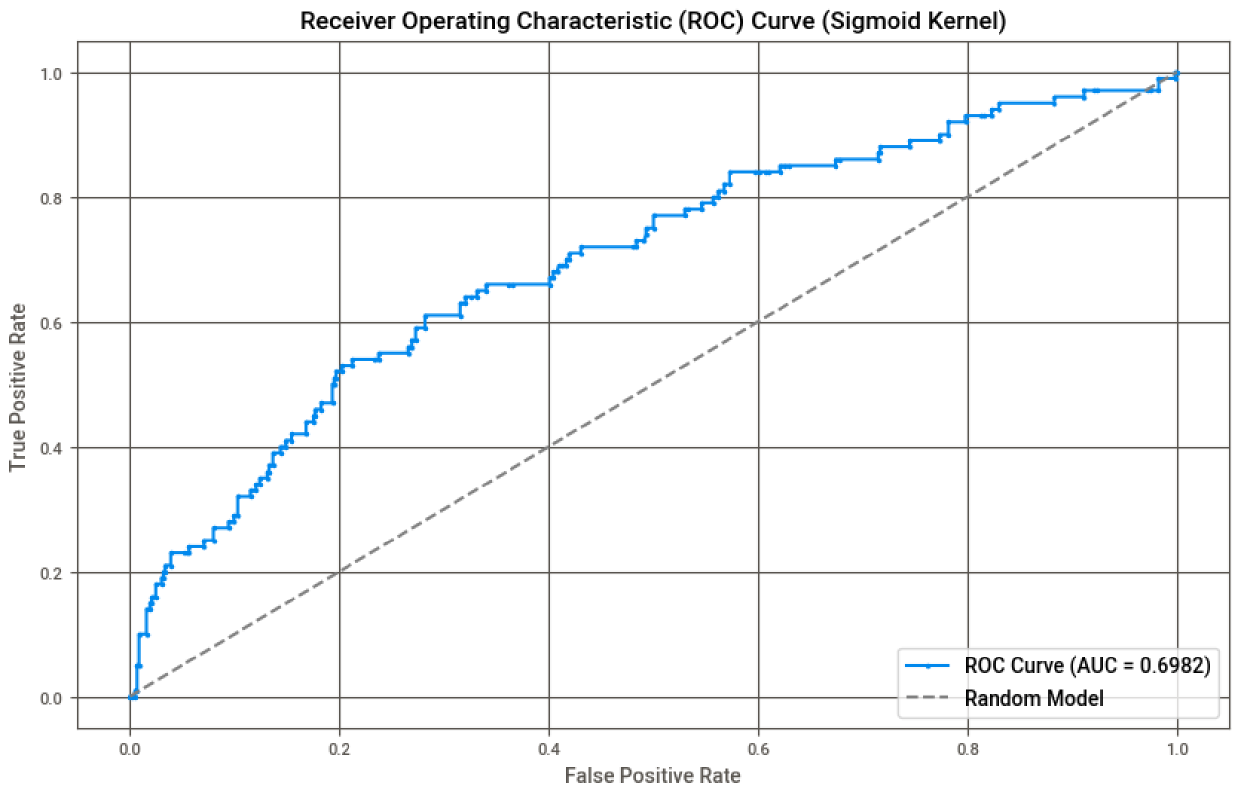
```
class_report_sigmoid = classification_report(y_test, y_pred_sigmoid)
print("\nClassification Report (Sigmoid Kernel):")
print(class_report_sigmoid)
```

```
Accuracy (Sigmoid Kernel): 0.8223

Confusion Matrix (Sigmoid Kernel):
[[515  49]
 [ 69  31]]

Classification Report (Sigmoid Kernel):
              precision    recall  f1-score   support

           0       0.88      0.91      0.90       564
           1       0.39      0.31      0.34       100

    accuracy                           0.82       664
   macro avg       0.63      0.61      0.62       664
weighted avg       0.81      0.82      0.81       664
```

```
# Compute ROC curve
fpr_sigmoid, tpr_sigmoid, thresholds_sigmoid = roc_curve(y_test,
y_pred_proba_sigmoid)

# Compute AUC score
auc_score_sigmoid = roc_auc_score(y_test, y_pred_proba_sigmoid)
print(f"\nAUC Score (Sigmoid Kernel): {auc_score_sigmoid:.4f}")
```

```
AUC Score (Sigmoid Kernel): 0.6982
```

```
# Plot ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr_sigmoid, tpr_sigmoid, label=f"ROC Curve (AUC =
{auc_score_sigmoid:.4f})", marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')
plt.title("Receiver Operating Characteristic (ROC) Curve (Sigmoid
Kernel)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Hyperparameter Tuning With SVC

```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

gridsvc=GridSearchCV(SVC(probability=True),param_grid=param_grid,refit
=True,cv=5,verbose=3)

gridsvc.fit(X_train,y_train)

Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.852 total
time=   0.9s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.848 total
time=   0.9s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.848 total
time=   0.9s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.848 total
time=   0.8s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.851 total
time=   0.8s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.852 total
```

```
time=   0.3s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.851 total
time=   0.3s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END .........C=1, gamma=1, kernel=rbf;, score=0.865 total
time=   0.8s
[CV 2/5] END .........C=1, gamma=1, kernel=rbf;, score=0.861 total
time=   0.8s
[CV 3/5] END .........C=1, gamma=1, kernel=rbf;, score=0.858 total
time=   0.9s
[CV 4/5] END .........C=1, gamma=1, kernel=rbf;, score=0.861 total
time=   0.9s
[CV 5/5] END .........C=1, gamma=1, kernel=rbf;, score=0.861 total
time=   0.8s
```

```
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.868 total
time=   0.3s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.881 total
time=   0.3s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.855 total
time=   0.3s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.874 total
time=   0.3s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.864 total
time=   0.3s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.852 total
time=   0.3s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.865 total
time=   0.9s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.855 total
time=   0.9s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.858 total
time=   0.9s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.858 total
time=   0.9s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.861 total
```

```
time=   0.8s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.839 total
time=   0.4s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.881 total
time=   0.4s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.865 total
time=   0.4s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.845 total
time=   0.4s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.858 total
time=   0.4s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.868 total
time=   0.2s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.897 total
time=   0.3s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.874 total
time=   0.2s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.858 total
time=   0.2s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.887 total
time=   0.2s
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.851 total
time=   0.2s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.865 total
time=   0.8s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.855 total
time=   0.8s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.858 total
time=   0.8s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.858 total
time=   0.9s
```

```
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.861 total
time=   0.9s
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.832 total
time=   0.4s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.868 total
time=   0.4s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.855 total
time=   0.4s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.823 total
time=   0.4s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.845 total
time=   0.4s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.868 total
time=   0.4s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.884 total
time=   0.4s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.871 total
time=   0.4s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.880 total
time=   0.4s
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.874 total
time=   0.3s
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.897 total
time=   0.3s
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.861 total
time=   0.3s
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.861 total
time=   0.3s
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.890 total
time=   0.3s
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.852 total
time=   0.3s
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.3s
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.848 total
time=   0.2s
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.865 total
time=   0.8s
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.855 total
time=   0.9s
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.858 total
time=   0.8s
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.858 total
```

```
time=   0.9s
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.861 total
time=   0.9s
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.835 total
time=   0.4s
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.839 total
time=   0.4s
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.842 total
time=   0.4s
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.816 total
time=   0.4s
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.854 total
time=   0.4s
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.848 total
time=   1.2s
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.887 total
time=   1.2s
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.868 total
time=   1.3s
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.842 total
time=   1.1s
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.871 total
time=   1.3s
[CV 1/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.861 total
time=   0.4s
[CV 2/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.897 total
time=   0.4s
[CV 3/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.865 total
time=   0.4s
[CV 4/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.861 total
time=   0.4s
[CV 5/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.896 total
time=   0.4s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.874 total
time=   0.3s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.900 total
time=   0.3s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.858 total
time=   0.3s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.868 total
time=   0.2s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.893 total
time=   0.3s

GridSearchCV(cv=5, estimator=SVC(probability=True),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']},
             verbose=3)
```

```python
# Print the best parameters
print("Best Parameters:", gridsvc.best_params_)
```

```
Best Parameters: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```python
# Access the best model
best_svc = gridsvc.best_estimator_
best_svc
```

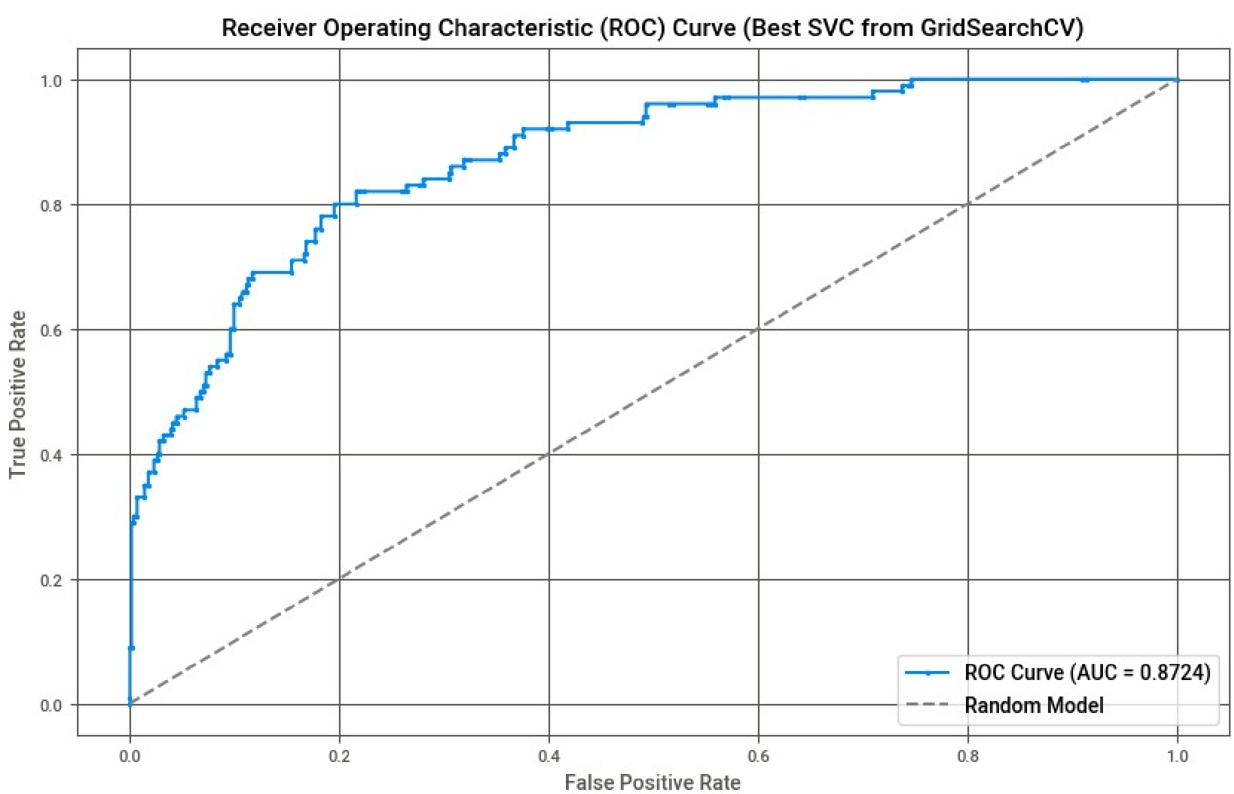
```
SVC(C=1000, gamma=0.0001, probability=True)
```

```python
## Prediction with hyperparameter tuned SVC model
y_pred4=gridsvc.predict(X_test)
# Accuracy
print("Accuracy of the hyperparameter tuned SVC is:",
accuracy_score(y_test, y_pred4))

# Confusion Matrix
print("\nConfusion Matrix of the hyperparameter tuned SVC is:")
print(confusion_matrix(y_test, y_pred4))

# Classification Report
print("\nClassification Report of the hyperparameter tuned SVC is:")
print(classification_report(y_test, y_pred4))
```

```
Accuracy of the hyperparameter tuned SVC is: 0.8870481927710844

Confusion Matrix of the hyperparameter tuned SVC is:
[[552  12]
 [ 63  37]]

Classification Report of the hyperparameter tuned SVC is:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       564
           1       0.76      0.37      0.50       100

    accuracy                           0.89       664
   macro avg       0.83      0.67      0.72       664
weighted avg       0.88      0.89      0.87       664
```

```python
# Use the best model to predict probabilities
y_pred_probasvcg = best_svc.predict_proba(X_test)[:, 1]

from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Compute ROC curve and AUC score for the probabilities
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probasvcg)
auc_score = roc_auc_score(y_test, y_pred_probasvcg)
```

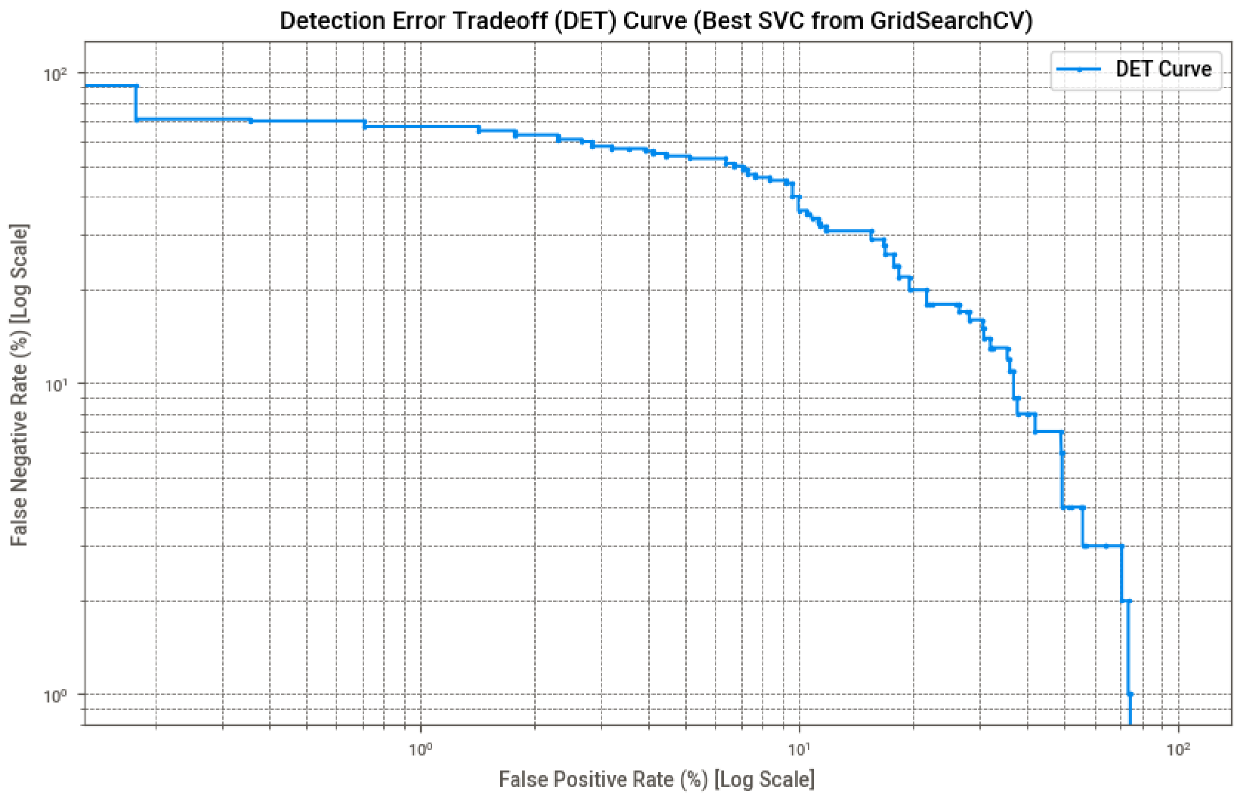
```
# Print AUC score
print(f"AUC Score (Best SVC): {auc_score:.4f}")

# Plot the ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})",
marker='.')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Model')
plt.title("Receiver Operating Characteristic (ROC) Curve (Best SVC
from GridSearchCV)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.show()

AUC Score (Best SVC): 0.8724
```




```
import numpy as np
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probasvcg)
```

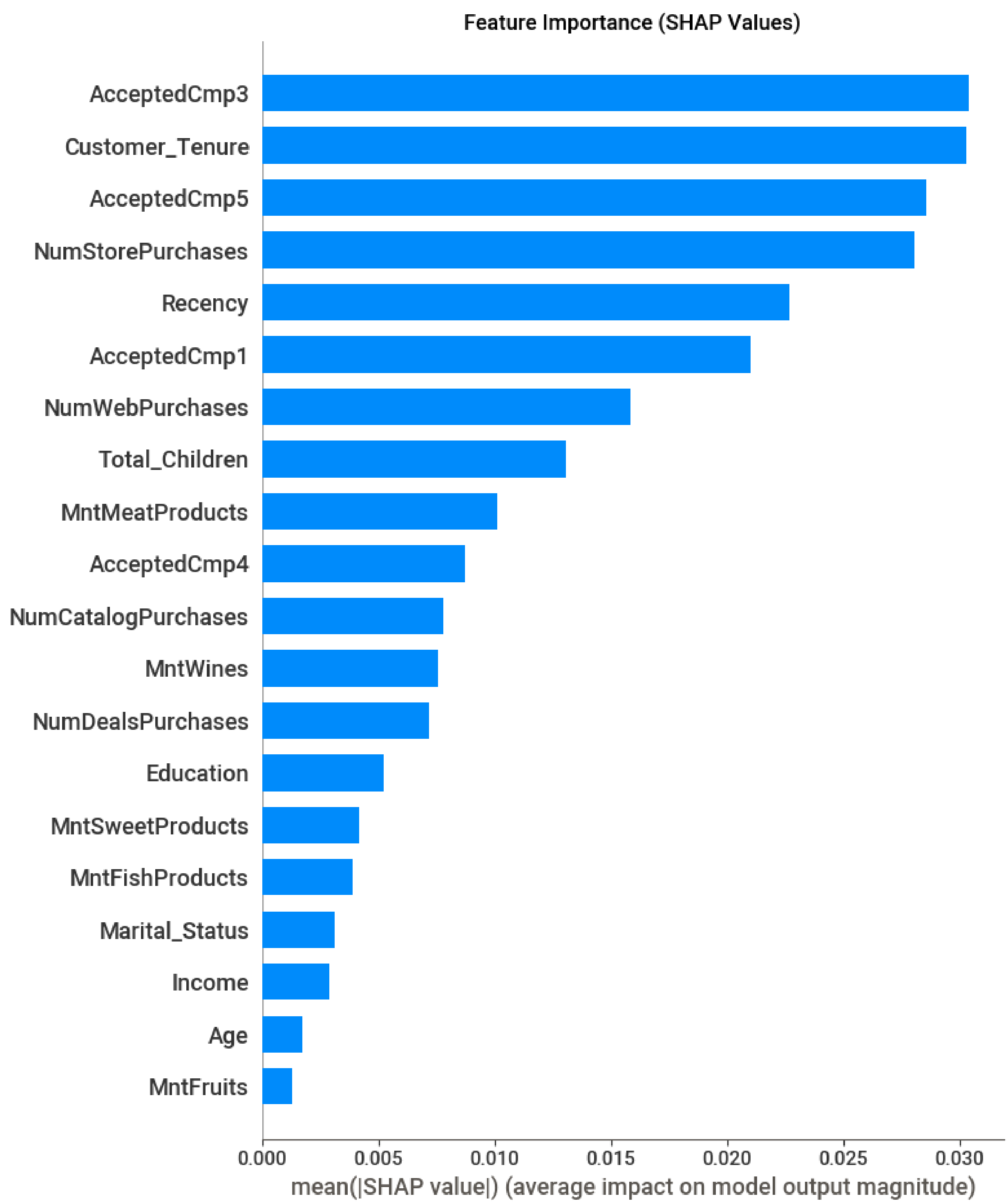
```
auc_score = roc_auc_score(y_test, y_pred_probasvcg)

# Compute False Negative Rate (FNR)
fnr = 1 - tpr

# Convert FPR and FNR to percentages
fpr_percentage = fpr * 100
fnr_percentage = fnr * 100

# Print AUC score
print(f"AUC Score (Best SVC): {auc_score:.4f}")

# Plot the DET curve
plt.figure(figsize=(10, 6))
plt.plot(fpr_percentage, fnr_percentage, label="DET Curve",
marker='.')

# Set logarithmic scale for both axes
plt.yscale("log")
plt.xscale("log")

# Add title and labels with percentages
plt.title("Detection Error Tradeoff (DET) Curve (Best SVC from
GridSearchCV)")
plt.xlabel("False Positive Rate (%) [Log Scale]")
plt.ylabel("False Negative Rate (%) [Log Scale]")

# Add legend and grid
plt.legend(loc="upper right")
plt.grid(which='both', linestyle='--', linewidth=0.5)

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

```
AUC Score (Best SVC): 0.8724
```



Feature Importance using SHAPly values

```
import shap
import matplotlib.pyplot as plt

# Train the best SVC model from GridSearchCV
best_svc = gridsvc.best_estimator_

# Explain the model predictions using SHAP
explainer = shap.Explainer(best_svc.predict, X_train)  # Use the
prediction function
shap_values = explainer(X_train)

# Feature importance plot
plt.title("Feature Importance (SHAP Values)")
shap.summary_plot(shap_values, X_train, plot_type="bar")

PermutationExplainer explainer: 1550it [1:17:29,  3.01s/it]
```


Feature Importance (SHAP Values)
AcceptedCmp3
Customer_Tenure
AcceptedCmp5
NumStorePurchases
Recency
AcceptedCmp1
NumWebPurchases
Total_Children
MntMeatProducts
AcceptedCmp4
NumCatalogPurchases
MntWines
NumDealsPurchases
Education
MntSweetProducts
MntFishProducts
Marital_Status
Income
Age
MntFruits
0.000
0.005
0.010
0.015
0.020
0.025
0.030
mean(|SHAP value|) (average impact on model output magnitude)