```
 What Does This Code Do?
✔ Loads the dataset
✔ Filters only numerical columns
✔ Creates 20 visualizations, including:

Histograms (Data Distribution)
Box Plots (Outlier Detection)
Pair Plots (Feature Relationships)
Heatmap (Feature Correlation)
Violin Plots (Density & Spread)
KDE Plots (Smooth Distributions)
Scatter Plots (Feature-Target Relationship)
 Data Preprocessing
✔ Drops non-numeric columns
✔ Splits data into training (80%) & testing (20%)
✔ Scales numerical features

 Model Training & Evaluation
✔ Trains 10 Classification Models
✔ Computes Accuracy & ROC-AUC for each model
✔ Displays Classification Report

 Visualization
✔ Model Performance Comparison (Bar Plot)
✔ ROC-AUC Curve for all models
✔ Feature Importance Plot (Random Forest)

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score, roc_curve
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

```
df =
pd.read_csv("/kaggle/input/credit-card-application/Credit_Card_Applica
tions.csv")

df.head()
```

```
   CustomerID  A1     A2     A3  A4  A5  A6     A7  A8  A9  A10  A11
A12  \
0    15776156   1  22.08  11.46   2   4   4  1.585   0   0    0    1
2
1    15739548   0  22.67   7.00   2   8   4  0.165   0   0    0    0
2
2    15662854   0  29.58   1.75   1   4   4  1.250   0   0    0    1
2
3    15687688   0  21.67  11.50   1   5   3  0.000   1   1   11    1
2
4    15715750   1  20.17   8.17   2   6   4  1.960   1   1   14    0
2

   A13   A14  Class
0  100  1213      0
1  160     1      0
2  280     1      0
3    0     1      1
4   60   159      1
```

```
df.isnull().sum()
```

```
CustomerID    0
A1            0
A2            0
A3            0
A4            0
A5            0
A6            0
A7            0
A8            0
A9            0
A10           0
A11           0
A12           0
A13           0
A14           0
Class         0
dtype: int64
```

```
df. duplicated().sum()
```

```
0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   CustomerID  690 non-null    int64  
 1   A1          690 non-null    int64  
 2   A2          690 non-null    float64
 3   A3          690 non-null    float64
 4   A4          690 non-null    int64  
 5   A5          690 non-null    int64  
 6   A6          690 non-null    int64  
 7   A7          690 non-null    float64
 8   A8          690 non-null    int64  
 9   A9          690 non-null    int64  
 10  A10         690 non-null    int64  
 11  A11         690 non-null    int64  
 12  A12         690 non-null    int64  
 13  A13         690 non-null    int64  
 14  A14         690 non-null    int64  
 15  Class       690 non-null    int64  
dtypes: float64(3), int64(13)
memory usage: 86.4 KB
```

```
df.describe().sum()
```

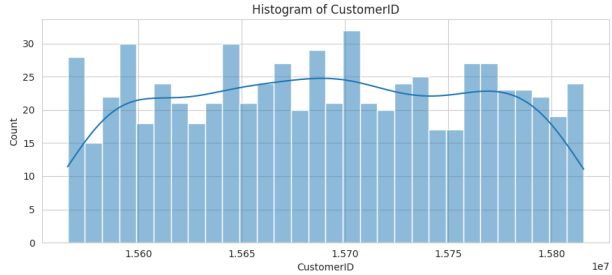
```
CustomerID    9.421757e+07
A1            6.941457e+02
A2            9.164240e+02
A3            7.386944e+02
A4            7.021967e+02
A5            7.380557e+02
A6            7.196851e+02
A7            7.278599e+02
A8            6.940230e+02
A9            6.929226e+02
A10           7.672629e+02
A11           6.929566e+02
A12           7.022278e+02
A13           3.558174e+03
A14           1.073240e+05
Class         6.929422e+02
dtype: float64
```

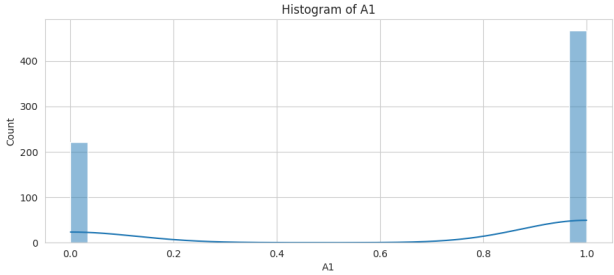
```
df.shape
```

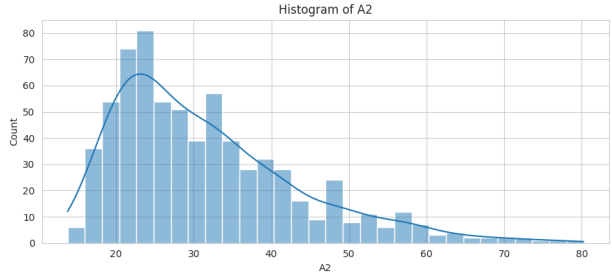
```
(690, 16)
```

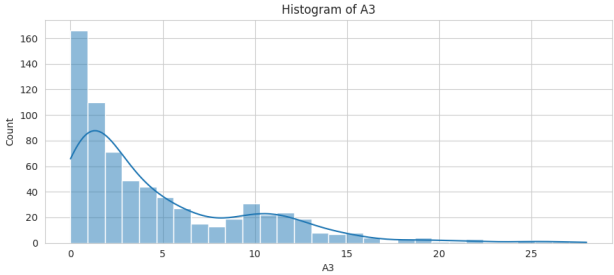
```
# Set style for seaborn
sns.set_style("whitegrid")
```

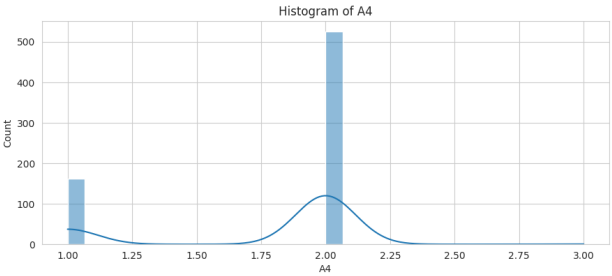
```
# Create subplots for visualization
fig, axes = plt.subplots(10, 2, figsize=(18, 40))
axes = axes.flatten()

# 1. Histogram for all numerical features
for i, col in enumerate(df.columns):
    sns.histplot(df[col], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(f'Histogram of {col}', fontsize=12)

plt.tight_layout()
plt.show()
```

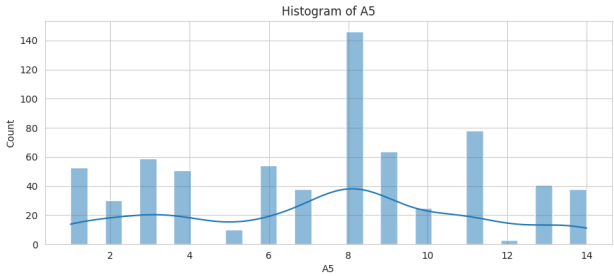

Histogram of CustomerID
Count
CustomerID
1e7

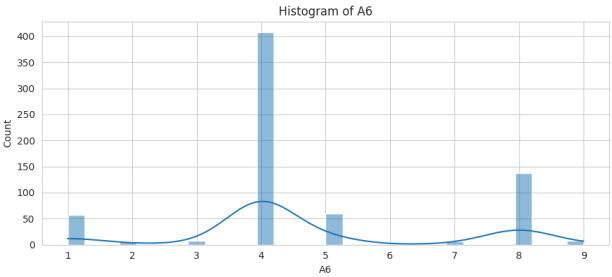

Histogram of A1
Count
A1

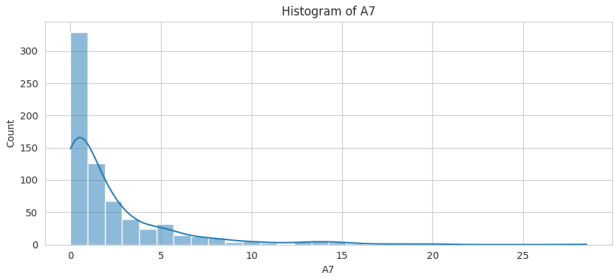

Histogram of A2
Count
A2

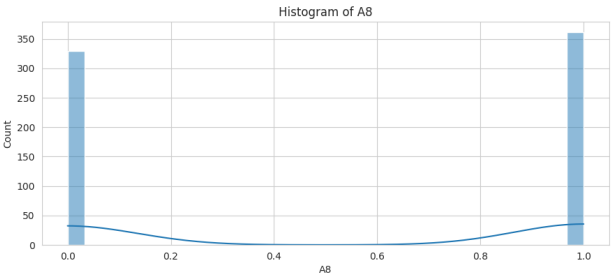

Histogram of A3
Count
A3

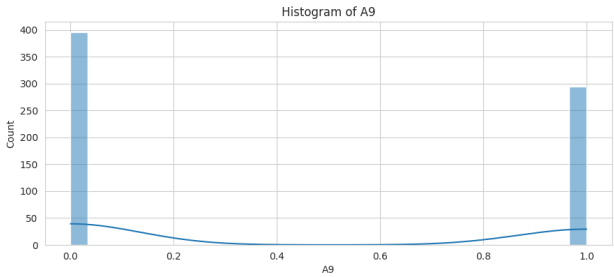

Histogram of A4
Count
A4

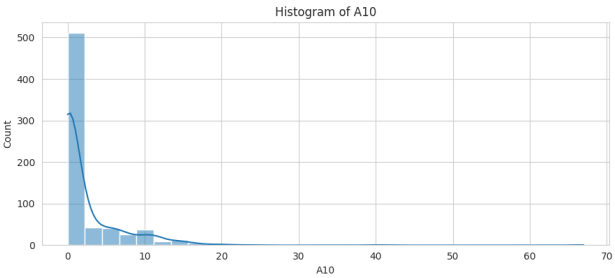

Histogram of A5
Count
A5

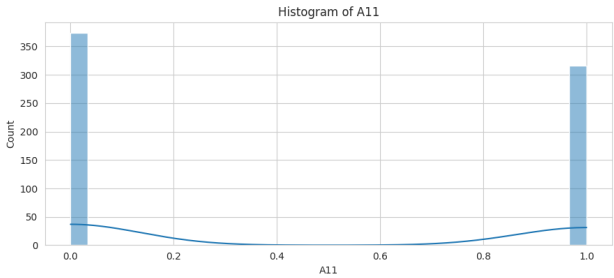

Histogram of A6
Count
A6

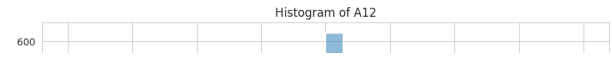

Histogram of A7
Count
A7


Histogram of A8
Count
A8


Histogram of A9
Count
A9

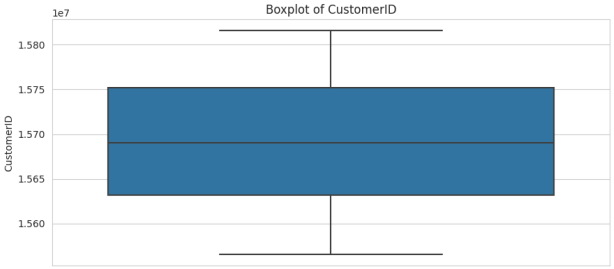

Histogram of A10
Count
A10

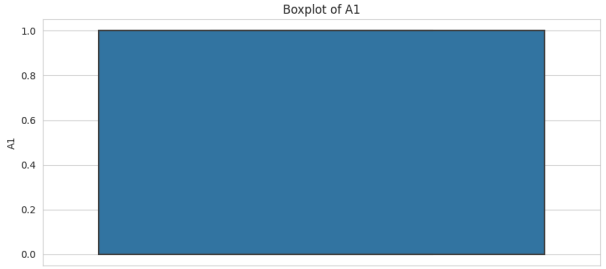

Histogram of A11
Count
A11

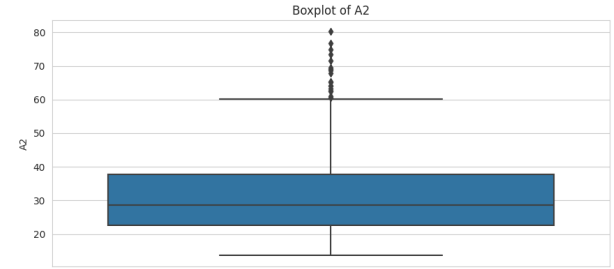

Histogram of A12

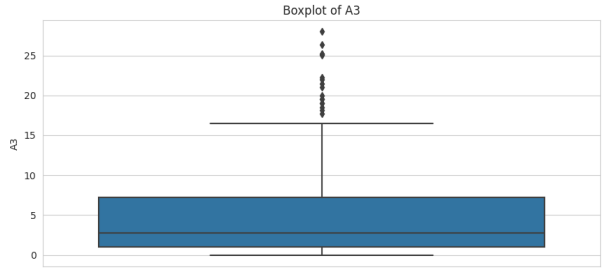

Histogram of A13

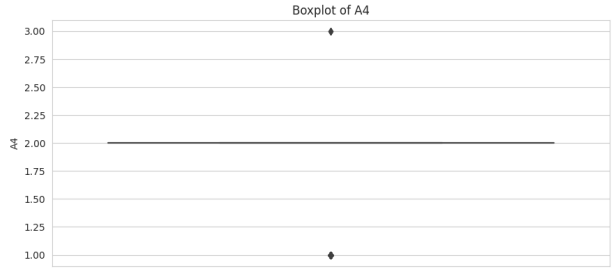
```
# 2. Box plots for all numerical features
fig, axes = plt.subplots(10, 2, figsize=(18, 40))
axes = axes.flatten()
for i, col in enumerate(df.columns):
    sns.boxplot(y=df[col], ax=axes[i])
    axes[i].set_title(f'Boxplot of {col}', fontsize=12)

plt.tight_layout()
plt.show()
```

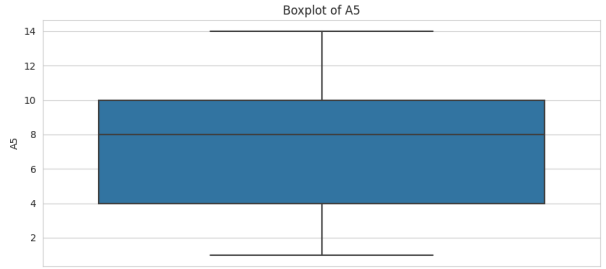

Boxplot of CustomerID
1e7
1.580
1.575
1.570
1.565
1.560
CustomerID

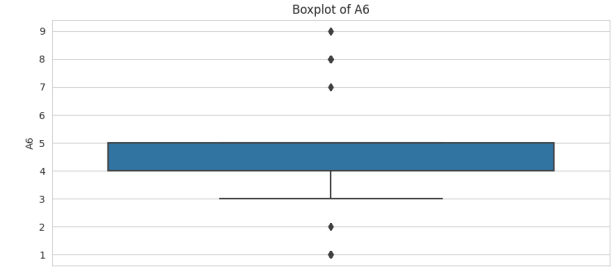

Boxplot of A1
1.0
0.8
0.6
0.4
0.2
0.0
A1

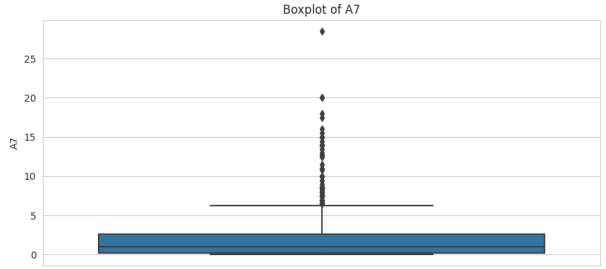

Boxplot of A2
80
70
60
50
40
30
20
A2

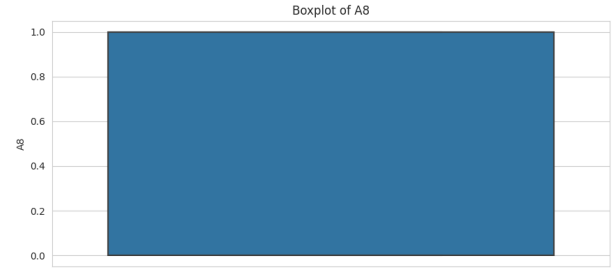

Boxplot of A3
25
20
15
10
5
0
A3

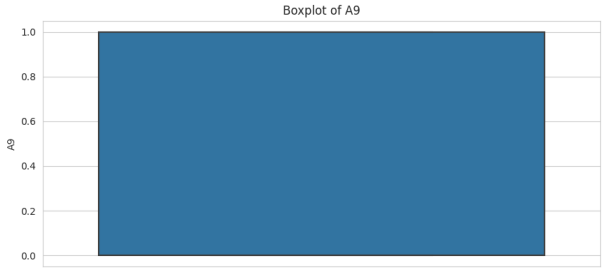

Boxplot of A4
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
A4

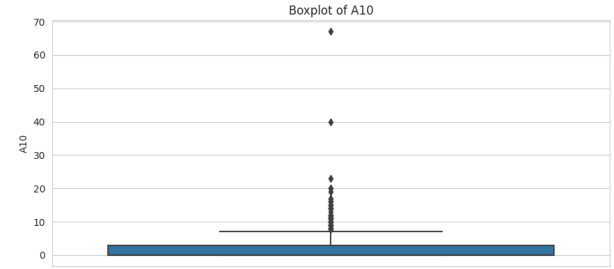

Boxplot of A5
14
12
10
8
6
4
2
A5

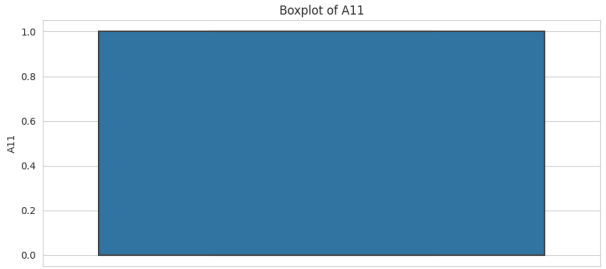

Boxplot of A6
9
8
7
6
5
4
3
2
1
A6


Boxplot of A7
25
20
15
10
5
0
A7


Boxplot of A8
1.0
0.8
0.6
0.4
0.2
0.0
A8


Boxplot of A9
1.0
0.8
0.6
0.4
0.2
0.0
A9

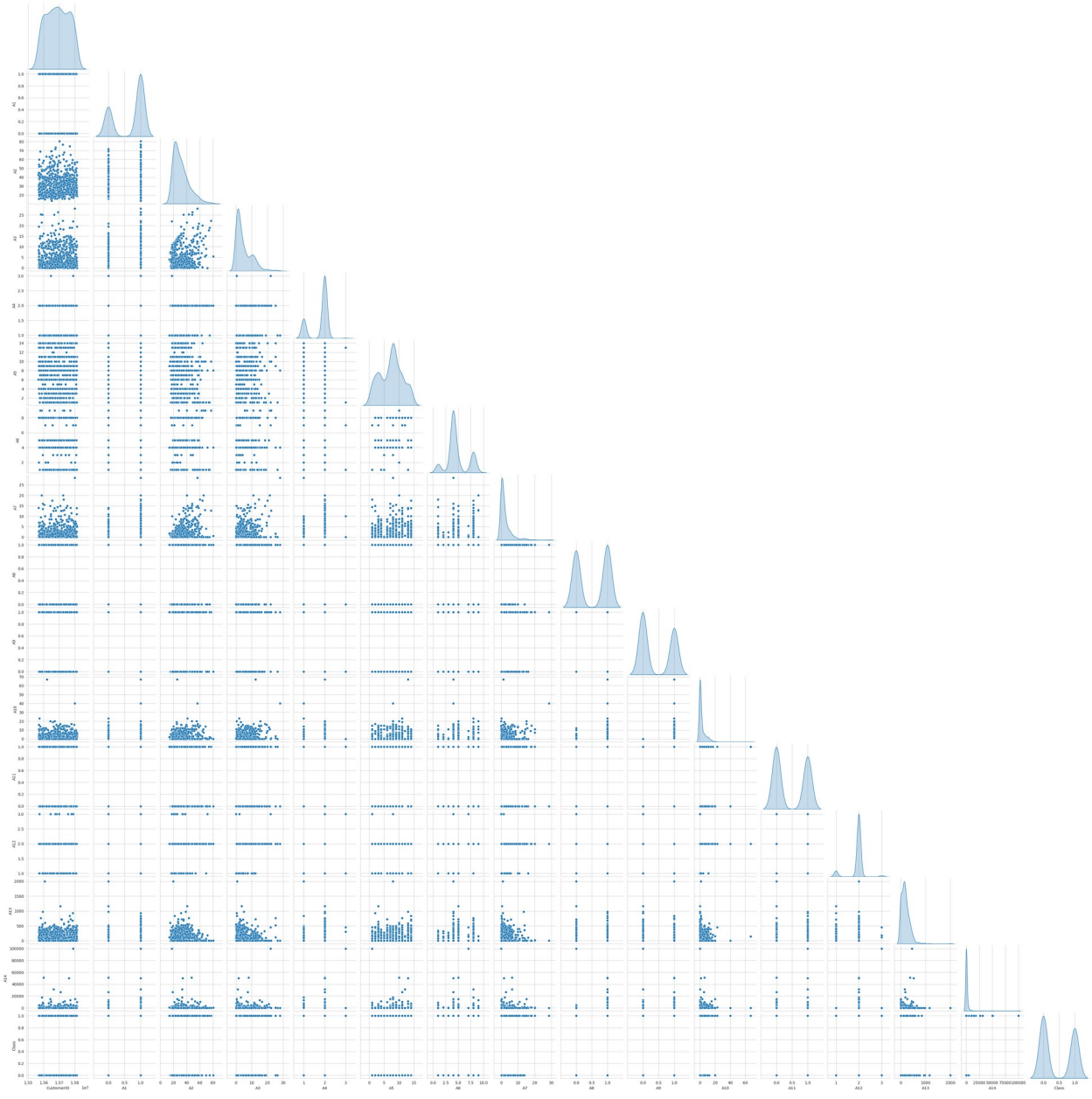

Boxplot of A10
70
60
50
40
30
20
10
0
A10


Boxplot of A11
1.0
0.8
0.6
0.4
0.2
0.0
A11

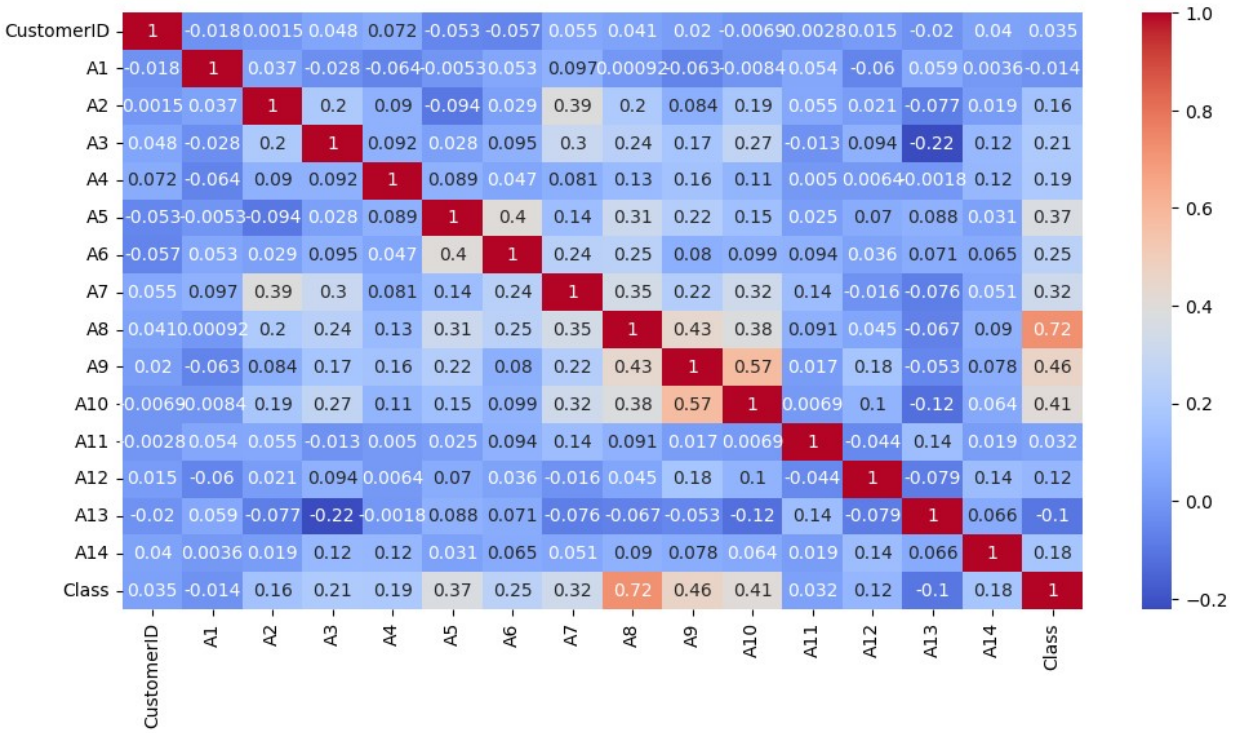

Boxplot of A12
3.00
2.75


Boxplot of A13
2000
1750

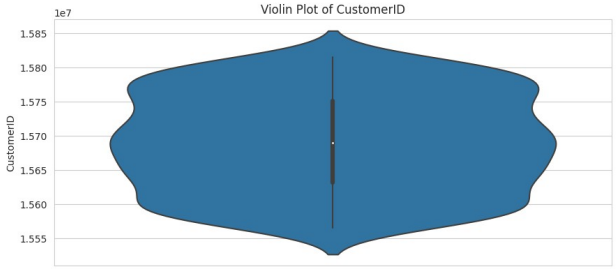
```
# 3. Pairplot of all numerical columns (useful for spotting correlations)
sns.pairplot(df, diag_kind='kde', corner=True)
plt.show()
```

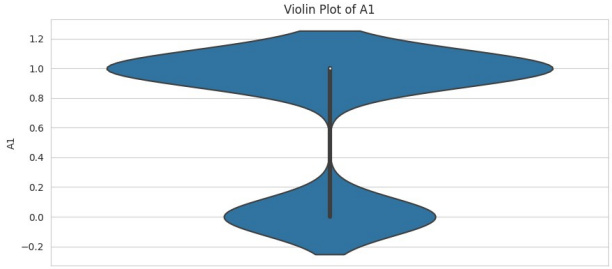


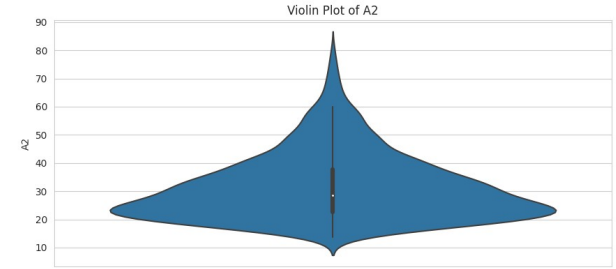
```
df

plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

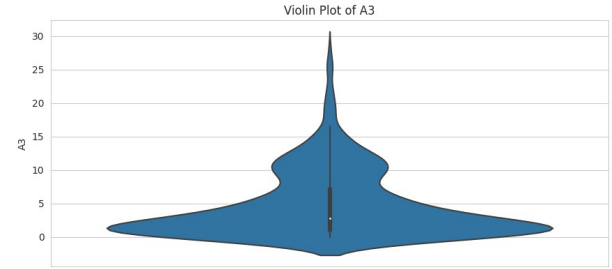


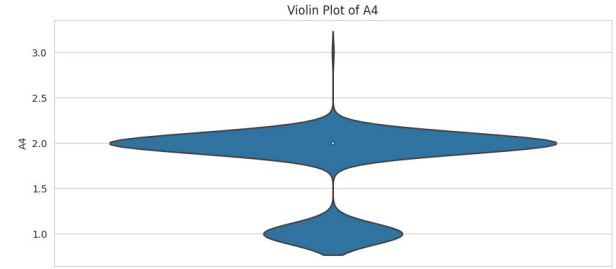

```python
# 5. Violin plots for all numerical features
fig, axes = plt.subplots(10, 2, figsize=(18, 40))
axes = axes.flatten()
for i, col in enumerate(df.columns):
    sns.violinplot(y=df[col], ax=axes[i])
    axes[i].set_title(f'Violin Plot of {col}', fontsize=12)

plt.tight_layout()
plt.show()
```

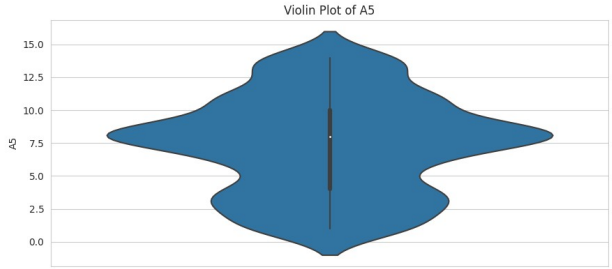

Violin Plot of CustomerID
1e7
1.585
1.580
1.575
1.570
1.565
1.560
1.555
CustomerID

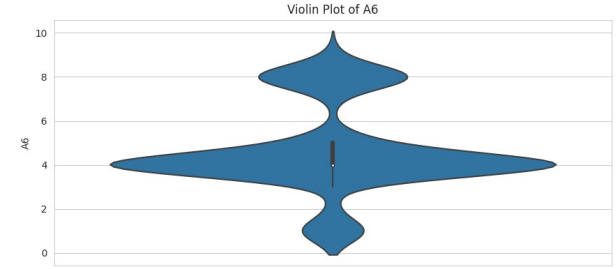

Violin Plot of A1
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
A1

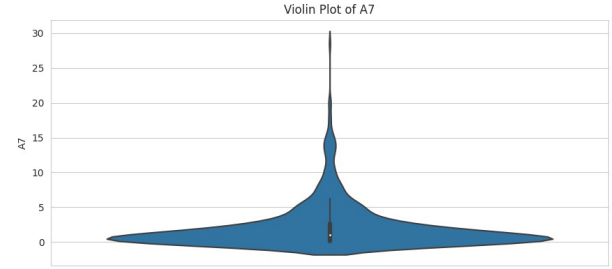

Violin Plot of A2
90
80
70
60
50
40
30
20
10
A2

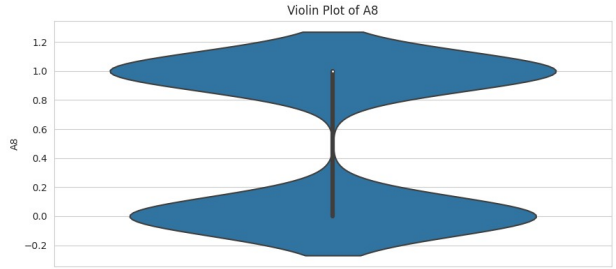

Violin Plot of A3
30
25
20
15
10
5
0
A3

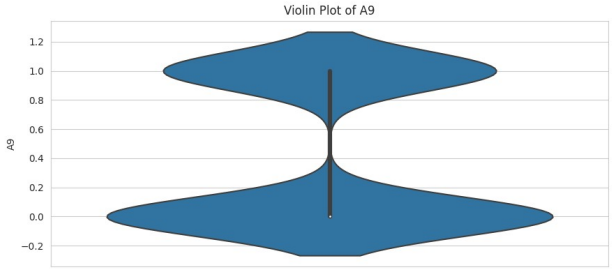

Violin Plot of A4
3.0
2.5
2.0
1.5
1.0
A4

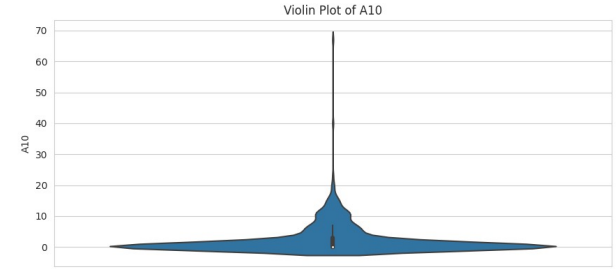

Violin Plot of A5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
A5

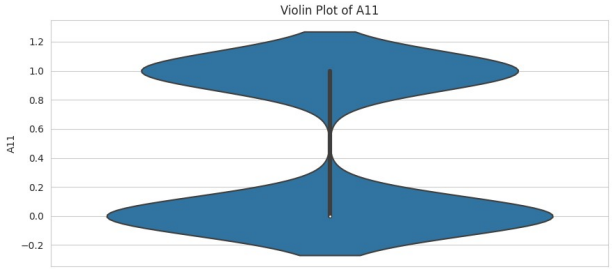

Violin Plot of A6
10
8
6
4
2
0
A6


Violin Plot of A7
30
25
20
15
10
5
0
A7


Violin Plot of A8
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
A8


Violin Plot of A9
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
A9

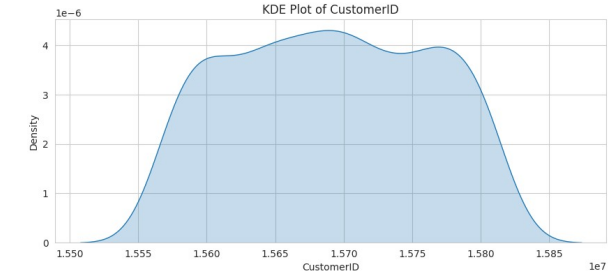

Violin Plot of A10
70
60
50
40
30
20
10
0
A10

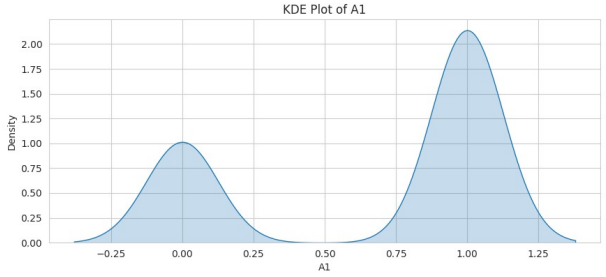

Violin Plot of A11
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
A11

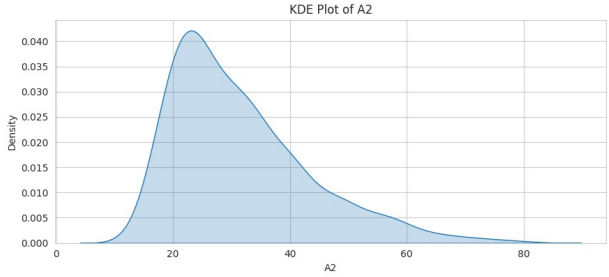

Violin Plot of A12
3.0

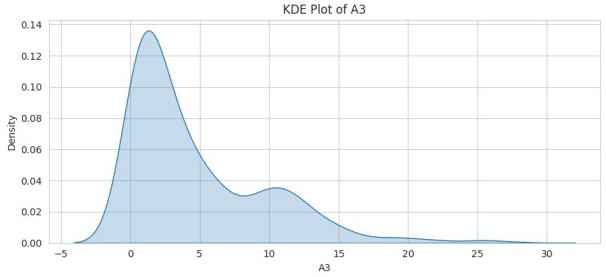

Violin Plot of A13
2000

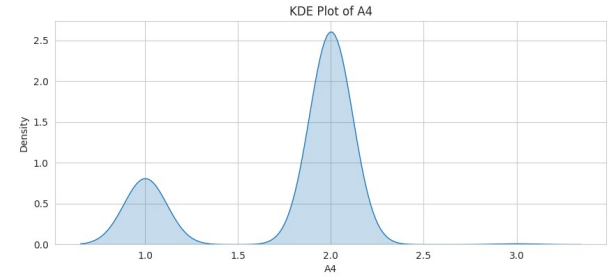
```
# 6. KDE plots for all numerical features
fig, axes = plt.subplots(10, 2, figsize=(18, 40))
axes = axes.flatten()
for i, col in enumerate(df.columns):
    sns.kdeplot(df[col], shade=True, ax=axes[i])
    axes[i].set_title(f'KDE Plot of {col}', fontsize=12)

plt.tight_layout()
plt.show()
```

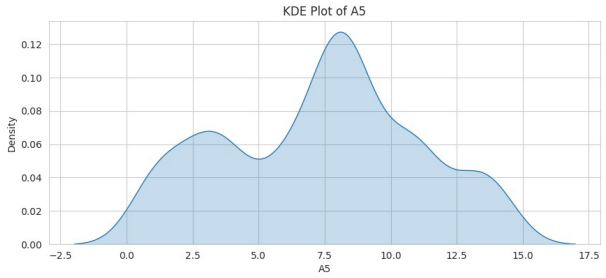

KDE Plot of CustomerID
1e−6
Density
CustomerID
1e7

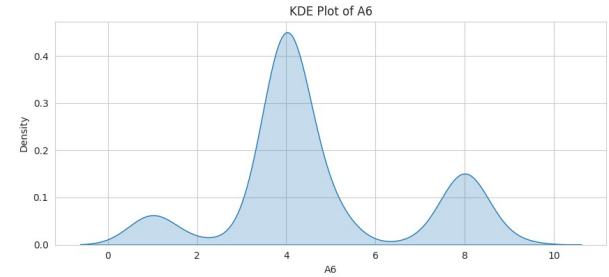

KDE Plot of A1
Density
A1

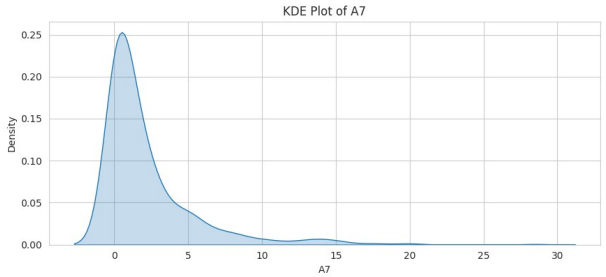

KDE Plot of A2
Density
A2

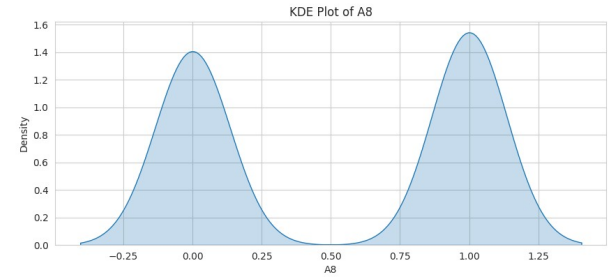

KDE Plot of A3
Density
A3

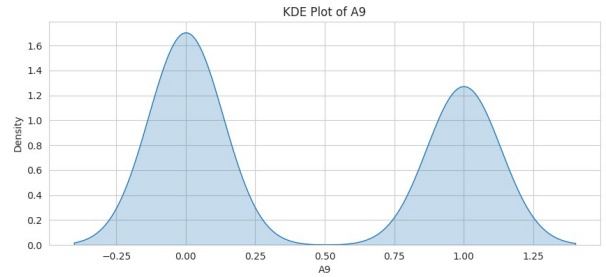

KDE Plot of A4
Density
A4

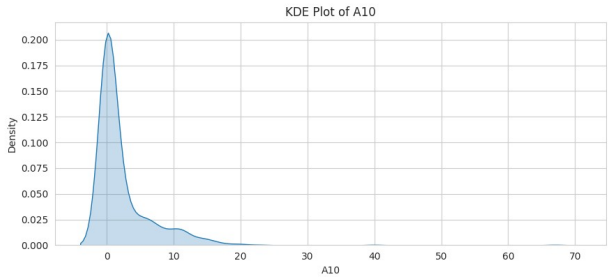

KDE Plot of A5
Density
A5

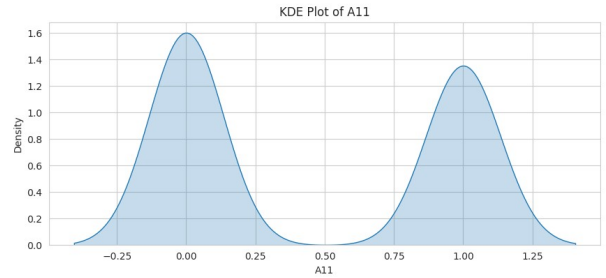

KDE Plot of A6
Density
A6


KDE Plot of A7
Density
A7


KDE Plot of A8
Density
A8


KDE Plot of A9
Density
A9

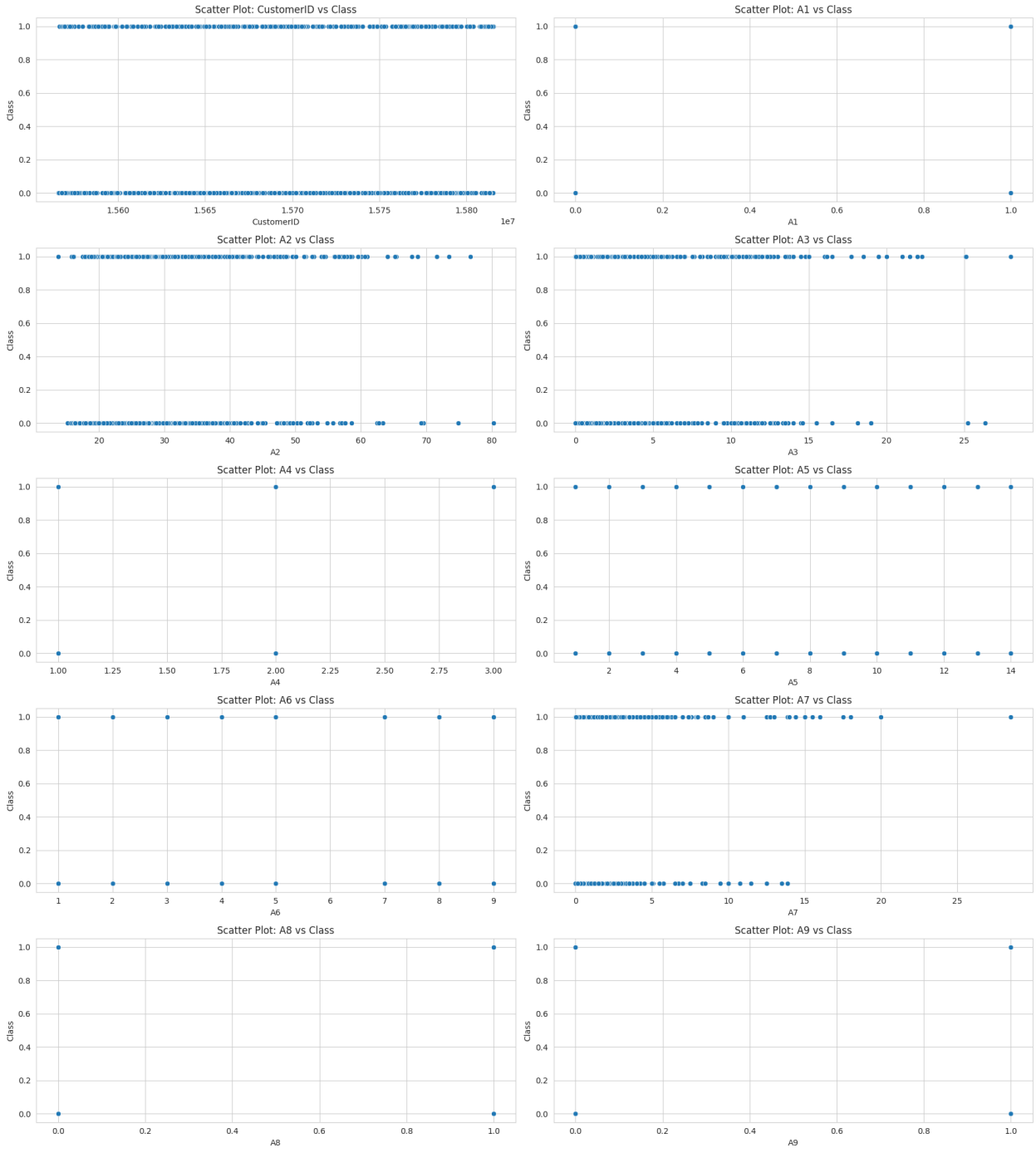

KDE Plot of A10
Density
A10


KDE Plot of A11
Density
A11


KDE Plot of A12


KDE Plot of A13

```
# 7. Scatter plots for some important features
fig, axes = plt.subplots(5, 2, figsize=(18, 20))
axes = axes.flatten()
for i, col in enumerate(df.columns[:10]):  # Selecting first 10
columns for scatter plots
    sns.scatterplot(x=df[col], y=df['Class'], ax=axes[i])
    axes[i].set_title(f'Scatter Plot: {col} vs Class', fontsize=12)

plt.tight_layout()
plt.show()
```




```python
# Function to remove outliers using IQR
def remove_outliers_iqr(df):
    Q1 = df.quantile(0.25)  # 25th percentile (Q1)
    Q3 = df.quantile(0.75)  # 75th percentile (Q3)
    IQR = Q3 - Q1  # Interquartile Range
    lower_bound = Q1 - 1.5 * IQR  # Lower Bound
    upper_bound = Q3 + 1.5 * IQR  # Upper Bound
```

```
    # Filtering values within bounds
    return df[~((df < lower_bound) | (df > upper_bound)).any(axis=1)]

# Print original shape
print("Original Dataset Shape:", df.shape)

# Remove outliers
dfd = remove_outliers_iqr(df)

# Print new shape after outlier removal
print("New Dataset Shape after Outlier Removal:", df.shape)

# Save cleaned dataset
df.to_csv("cleaned_dataset.csv", index=False)
print("Outlier-free dataset saved as 'cleaned_dataset.cs")
```

```
Original Dataset Shape: (690, 16)
New Dataset Shape after Outlier Removal: (690, 16)
Outlier-free dataset saved as 'cleaned_dataset.cs
```

```
# Drop non-numeric columns
df.drop(columns=['CustomerID'], inplace=True)

# Separate features and target
X = df.drop(columns=['Class'])  # Features
y = df['Class']  # Target variable


# Split dataset into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Scale numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define classification models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(probability=True),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "AdaBoost": AdaBoostClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False,
eval_metric="logloss"),
    "LightGBM": LGBMClassifier()
}
```

```python
# Store results
results = {}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)  # Train model
    y_pred = model.predict(X_test)  # Predictions
    y_prob = model.predict_proba(X_test)[:, 1]  # Probability for ROC

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob)

    # Store results
    results[name] = {"Accuracy": acc, "ROC_AUC": roc_auc, "Model":
model}

    # Print classification report
    print(f"\n{name} Classification Report:")
    print(classification_report(y_test, y_pred))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.75      0.83        77
           1       0.75      0.93      0.83        61

    accuracy                           0.83       138
   macro avg       0.84      0.84      0.83       138
weighted avg       0.85      0.83      0.83       138


Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.74      0.79        77
           1       0.72      0.84      0.77        61

    accuracy                           0.78       138
   macro avg       0.78      0.79      0.78       138
weighted avg       0.79      0.78      0.78       138


Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.78      0.83        77
           1       0.76      0.89      0.82        61

    accuracy                           0.83       138
```

```
   macro avg       0.83      0.83      0.83       138
weighted avg       0.84      0.83      0.83       138


SVM Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.71      0.81        77
           1       0.72      0.93      0.81        61

    accuracy                           0.81       138
   macro avg       0.83      0.82      0.81       138
weighted avg       0.84      0.81      0.81       138


K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.81      0.82        77
           1       0.77      0.80      0.78        61

    accuracy                           0.80       138
   macro avg       0.80      0.80      0.80       138
weighted avg       0.81      0.80      0.80       138


Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        77
           1       0.84      0.75      0.79        61

    accuracy                           0.83       138
   macro avg       0.83      0.82      0.82       138
weighted avg       0.83      0.83      0.82       138


Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.81      0.84        77
           1       0.78      0.85      0.81        61

    accuracy                           0.83       138
   macro avg       0.82      0.83      0.83       138
weighted avg       0.83      0.83      0.83       138


AdaBoost Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.91      0.82      0.86        77
           1       0.80      0.90      0.85        61

    accuracy                           0.86       138
   macro avg       0.86      0.86      0.85       138
weighted avg       0.86      0.86      0.86       138


XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.87      0.89        77
           1       0.85      0.90      0.87        61

    accuracy                           0.88       138
   macro avg       0.88      0.89      0.88       138
weighted avg       0.89      0.88      0.88       138

[LightGBM] [Info] Number of positive: 246, number of negative: 306
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.004654 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 544
[LightGBM] [Info] Number of data points in the train set: 552, number
of used features: 14
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.445652 ->
initscore=-0.218254
[LightGBM] [Info] Start training from score -0.218254
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
```

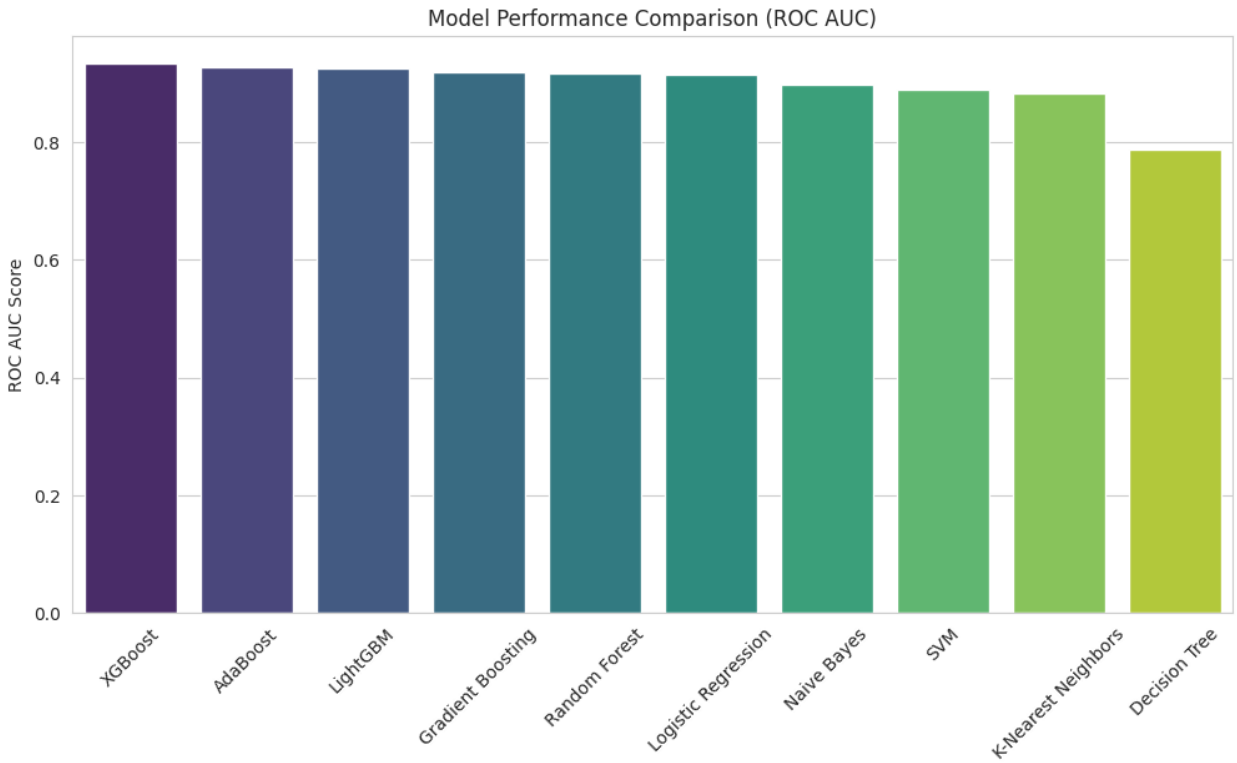
```
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf

LightGBM Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.82      0.85        77
           1       0.79      0.87      0.83        61

    accuracy                           0.84       138
   macro avg       0.84      0.84      0.84       138
weighted avg       0.84      0.84      0.84       138


# Convert results to DataFrame
results_df = pd.DataFrame(results).T.sort_values(by="ROC_AUC",
ascending=False)

# Display results
```

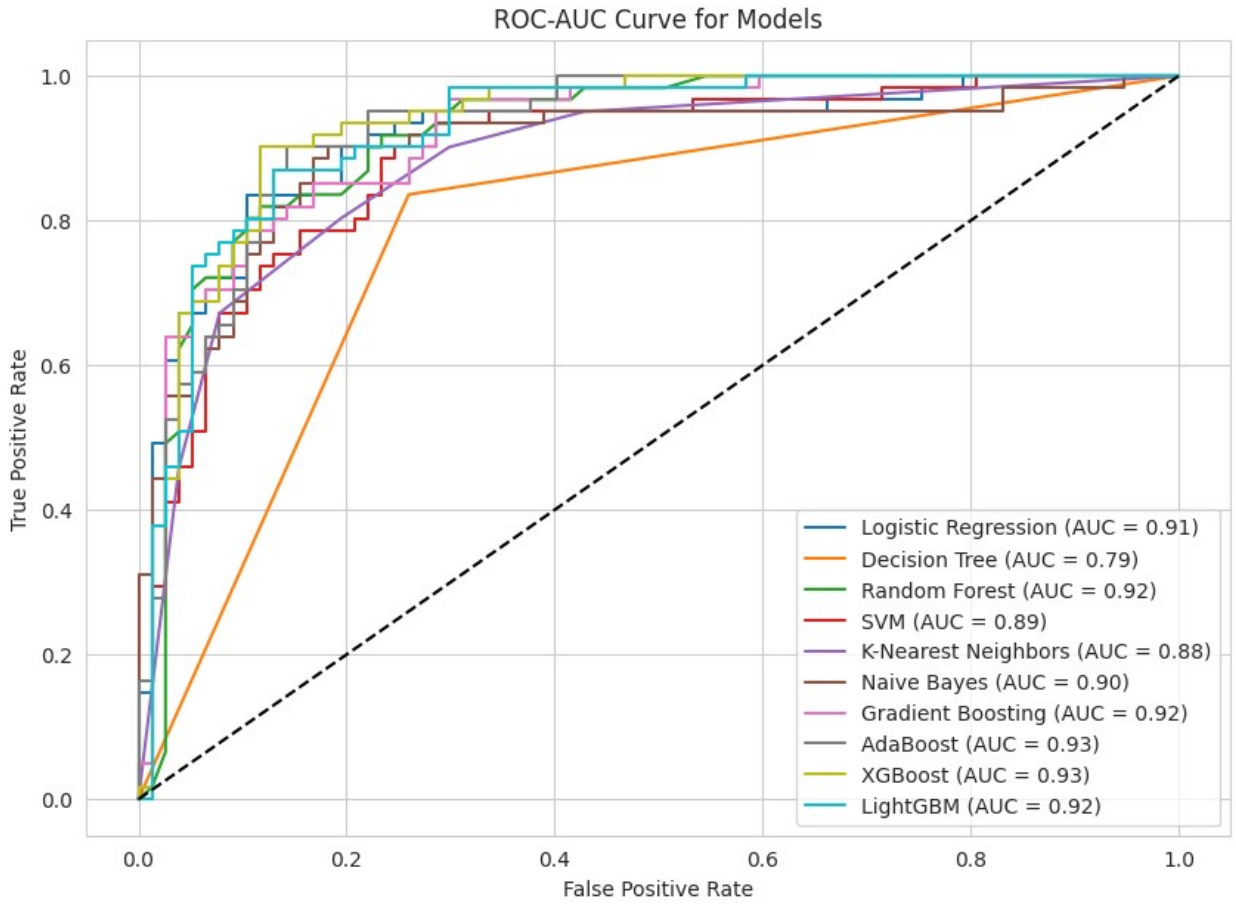
```
print("\nModel Performance Comparison:")
print(results_df[["Accuracy", "ROC_AUC"]])
```

```
Model Performance Comparison:
                     Accuracy   ROC_AUC
XGBoost              0.884058  0.933362
AdaBoost             0.855072  0.926762
LightGBM              0.84058  0.924846
Gradient Boosting    0.826087  0.918671
Random Forest        0.826087   0.91782
Logistic Regression  0.833333  0.913988
Naive Bayes          0.826087  0.898872
SVM                  0.811594  0.889078
K-Nearest Neighbors  0.804348  0.883436
Decision Tree        0.782609  0.788163
```

```
# Plot Model Comparison
plt.figure(figsize=(12, 6))
sns.barplot(x=results_df.index, y=results_df["ROC_AUC"],
palette="viridis")
plt.xticks(rotation=45)
plt.ylabel("ROC AUC Score")
plt.title("Model Performance Comparison (ROC AUC)")
plt.show()
```

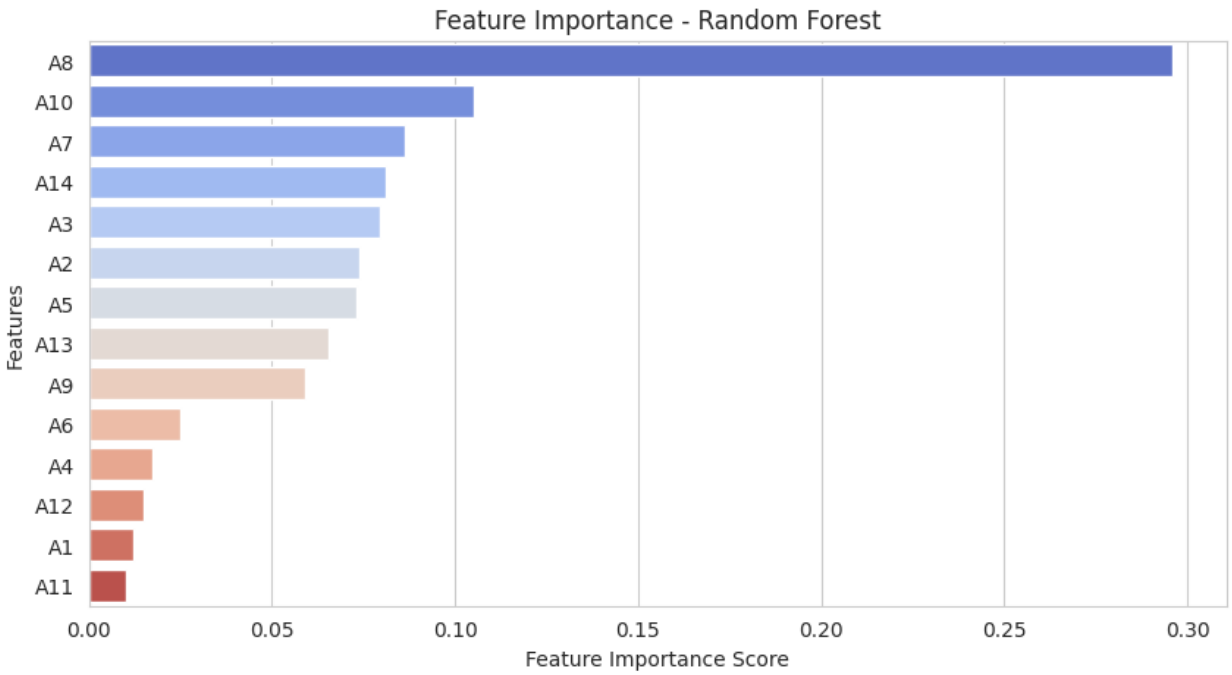


```
# ROC Curve Plot
plt.figure(figsize=(10, 7))
for name, data in results.items():
    model = data["Model"]
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {data['ROC_AUC']:.2f})")

plt.plot([0, 1], [0, 1], "k--")  # Diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC-AUC Curve for Models")
plt.legend()
plt.show()
```




```
# Feature Importance (Using RandomForest)
best_model = results["Random Forest"]["Model"]  # Using RF as an
example
feature_importances = pd.Series(best_model.feature_importances_,
index=X.columns).sort_values(ascending=False)
```

```
# Plot feature importance
plt.figure(figsize=(10, 5))
sns.barplot(x=feature_importances, y=feature_importances.index,
palette="coolwarm")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance - Random Forest")
plt.show()
```



About the Author

Name: Arif Mia

 Profession: Machine Learning Engineer & Data Scientist

Career Objective

 My goal is to contribute to groundbreaking advancements in artificial intelligence and data science, empowering companies and individuals with data-driven solutions. I strive to simplify complex challenges, craft innovative projects, and pave the way for a smarter and more connected future.

 As a **Machine Learning Engineer** and **Data Scientist**, I am passionate about using machine learning, deep learning, computer vision, and advanced analytics to solve real-world problems. My expertise lies in delivering impactful solutions by leveraging cutting-edge technologies.

Skills

- **Artificial Intelligence & Machine Learning**
- **Computer Vision & Predictive Analytics**
- **Deep Learning & Natural Language Processing (NLP)**
- **Python Programming & Automation**
- **Data Visualization & Analysis**
- **End-to-End Model Development & Deployment**

Featured Projects

Lung Cancer Prediction with Deep Learning
Achieved 99% accuracy in a computer vision project using 12,000 medical images across three classes. This project involved data preprocessing, visualization, and model training to detect cancer effectively.

Ghana Crop Disease Detection Challenge
Developed a model using annotated images to identify crop diseases with bounding boxes, addressing real-world agricultural challenges and disease mitigation.

Global Plastic Waste Analysis
Utilized GeoPandas, Matplotlib, and machine learning models like RandomForestClassifier and CatBoostClassifier to analyze trends in plastic waste management.

♪ **Twitter Emotion Classification**
Performed exploratory data analysis and built a hybrid machine learning model to classify Twitter sentiments, leveraging text data preprocessing and visualization techniques.

⚙ Technical Skills

- **Programming Languages:** Python , SQL , R
- **Data Visualization Tools:** Matplotlib , Seaborn , Tableau , Power BI
- **Machine Learning & Deep Learning:** Scikit-learn , TensorFlow , PyTorch
- **Big Data Technologies:** Hadoop , Spark ⚡
- **Model Deployment:** Flask ⚡FastAPI , Docker

Connect with Me

Email: arifmiahcse@gmail.com

LinkedIn: www.linkedin.com/in/arif-miah-8751bb217

GitHub: https://github.com/Arif-miad

Kaggle: https://www.kaggle.com/arifmia

Let’s turn ideas into reality! If you’re looking for innovative solutions or need collaboration on exciting projects, feel free to reach out.