

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import kagglehub
```

```
1 # Download latest version
2 path = kagglehub.dataset_download("mlg-ulb/creditcardfraud")
3
4 print("Path to dataset files:", path)
```

Using Colab cache for faster access to the 'creditcardfraud' dataset.  
Path to dataset files: /kaggle/input/creditcardfraud

```
1 df = pd.read_csv('/kaggle/input/creditcardfraud/creditcard.csv')
2 df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 31 columns

```
1 df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	284807.000000	2.848070e+05								
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+00
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-01
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

8 rows × 31 columns

1 Start coding or [generate](#) with AI.

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.feature_selection import SelectFromModel
3 from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
4 from sklearn.linear_model import LogisticRegression
5 from sklearn.ensemble import RandomForestClassifier
6 from sklearn.svm import LinearSVC
7 from sklearn.metrics import f1_score, matthews_corrcoef, roc_auc_score,
confusion_matrix
8 from xgboost import XGBClassifier
9 from imblearn.over_sampling import SMOTE
10 from imblearn.combine import SMOTEEENN
```

```
1 # Randomize the dataset
2 new_df = df.copy()
3 new_df = new_df.sample(frac=1, random_state=42)
```

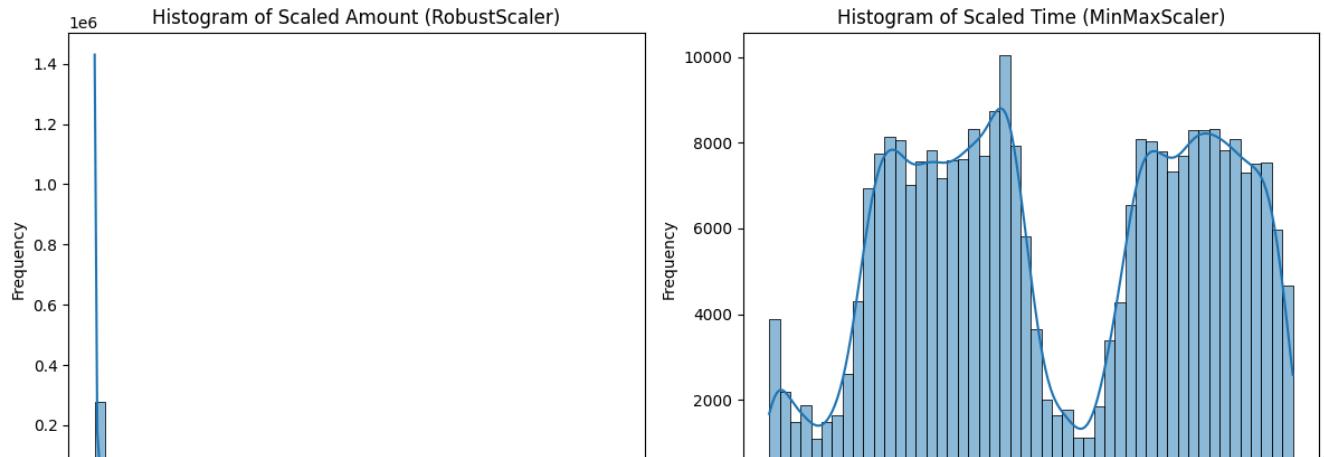
## Scaling Variables for Normalization

```
1 # @title Scaling Variables for Normalization
2
3 # Instantiate scikit-learn scalers
4 robust_scaler = RobustScaler()
5 minmax_scaler = MinMaxScaler()
6
7 # Apply RobustScaler to 'Amount' column
8 # Reshape the column to be 2D as fit_transform expects
```

```

9 scaled_amount = robust_scaler.fit_transform(new_df['Amount'].values.reshape(-1, 1))
10
11 # Apply MinMaxScaler to 'Time' column
12 # Reshape the column to be 2D
13 scaled_time = minmax_scaler.fit_transform(new_df['Time'].values.reshape(-1, 1))
14
15 # Plot histograms of the scaled columns
16 plt.figure(figsize=(12, 5))
17
18 plt.subplot(1, 2, 1)
19 sns.histplot(scaled_amount.flatten(), bins=50, kde=True) # flatten to get a 1D array
20 plt.title('Histogram of Scaled Amount (RobustScaler)')
21 plt.xlabel('Scaled Amount')
22 plt.ylabel('Frequency')
23
24 plt.subplot(1, 2, 2)
25 sns.histplot(scaled_time.flatten(), bins=50, kde=True) # flatten to get a 1D array
26 plt.title('Histogram of Scaled Time (MinMaxScaler)')
27 plt.xlabel('Scaled Time')
28 plt.ylabel('Frequency')
29
30 plt.tight_layout()
31 plt.show()
32

```



```

1 # Separate features (X) and target (y)
2 X = new_df.drop('Class', axis=1)
3 y = new_df['Class']
4
5 # Perform train-test split
6 # test_size=0.3 means 30% of data for testing, 70% for training
7 x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
8
9 print("Dataset split into training and testing sets.")
10 print(f"Training features shape: {x_train.shape}")
11 print(f"Testing features shape: {x_test.shape}")
12 print(f"Training target shape: {y_train.shape}")
13 print(f"Testing target shape: {y_test.shape}")
14
15 # Display the class distribution in train and test sets to verify stratification
16 print("\nClass distribution in training set:")
17 print(y_train.value_counts(normalize=True))
18 print("\nClass distribution in testing set:")
19 print(y_test.value_counts(normalize=True))

```

Dataset split into training and testing sets.

Training features shape: (199364, 30)

Testing features shape: (85443, 30)

Training target shape: (199364,)

Testing target shape: (85443,)

Class distribution in training set:

Class

0 0.998269

1 0.001731

Name: proportion, dtype: float64

Class distribution in testing set:

Class

0 0.99828

1 0.00172

Name: proportion, dtype: float64

## Batch Processing Models

### Logistic Regression Model

```
1 # @title Logistic Regression Model
2
3 # Instantiate the Logistic Regression model
4 model_lr = LogisticRegression(random_state=42, solver='liblinear')
5
6 # Train the model on the training data
7 model_lr.fit(x_train, y_train)
8 print("Training complete.")
9
10 # Make predictions on the test data
11 y_pred_lr = model_lr.predict(x_test)
12 y_pred_proba_lr = model_lr.predict_proba(x_test)[:, 1]
13
14 # Evaluate the model
15 f1_lr = f1_score(y_test, y_pred_lr)
16 mcc_lr = matthews_corrcoef(y_test, y_pred_lr)
17 roc_auc_lr = roc_auc_score(y_test, y_pred_proba_lr)
18
19 # Print the evaluation metrics
20 print("\nEvaluation Metrics on Test Data:")
21 print(f"F1 Score: {f1_lr:.4f}")
22 print(f"MCC Score: {mcc_lr:.4f}")
23 print(f"ROC AUC Score: {roc_auc_lr:.4f}")
```

Training complete.

Evaluation Metrics on Test Data:  
F1 Score: 0.6496  
MCC Score: 0.6508  
ROC AUC Score: 0.9045

### Scaled Logistic Regression Model

```
1 # @title Scaled Logistic Regression Model
2
3 # Instantiate the Logistic Regression model
4
5 model_lr_batch = LogisticRegression(random_state=42, solver='liblinear')
6
7 # Instantiate the StandardScaler
8 scaler_batch = StandardScaler()
9
10 # Scale the training and testing data
11 X_train_scaled = scaler_batch.fit_transform(x_train)
12 X_test_scaled = scaler_batch.transform(x_test)
13
14 model_lr_batch.fit(X_train_scaled, y_train)
15 print("Training complete.")
16
17 # Make predictions on the scaled test data
18 y_pred_lr_batch = model_lr_batch.predict(X_test_scaled)
19 y_pred_proba_lr_batch = model_lr_batch.predict_proba(X_test_scaled)[:, 1] # Get probability of the positive class (1)
20
21 # Evaluate the model
22 f1_lr_batch = f1_score(y_test, y_pred_lr_batch)
23 mcc_lr_batch = matthews_corrcoef(y_test, y_pred_lr_batch)
24 roc_auc_lr_batch = roc_auc_score(y_test, y_pred_proba_lr_batch)
25
26 # Print the evaluation metrics
27 print("\nEvaluation Metrics on Test Data:")
28 print(f"F1 Score: {f1_lr_batch:.4f}")
29 print(f"MCC Score: {mcc_lr_batch:.4f}")
30 print(f"ROC AUC Score: {roc_auc_lr_batch:.4f}")
```

Training complete.

Evaluation Metrics on Test Data:  
F1 Score: 0.7385  
MCC Score: 0.7445  
ROC AUC Score: 0.9772

### RandomForest Classifier Model

```
1 # @title RandomForest Classifier Model
2
3 # Basic Random Forest Classifier
```

```

4 model_rf_batch = RandomForestClassifier(max_depth=5, n_jobs=-1)
5 model_rf_batch.fit(x_train, y_train)
6
7 # Make predictions on the test data
8 y_pred_rf_batch = model_rf_batch.predict(x_test)
9 y_pred_proba_rf_batch = model_rf_batch.predict_proba(x_test)[:, 1]
10
11
12 # Evaluate the model
13 f1_rf_batch = f1_score(y_test, y_pred_rf_batch)
14 mcc_rf_batch = matthews_corrcoef(y_test, y_pred_rf_batch)
15 roc_auc_rf_batch = roc_auc_score(y_test, y_pred_proba_rf_batch)
16
17 # Print the evaluation metrics
18 print("\nEvaluation Metrics on Test Data:")
19 print(f"F1 Score: {f1_rf_batch:.4f}")
20 print(f"MCC Score: {mcc_rf_batch:.4f}")
21 print(f"ROC AUC Score: {roc_auc_rf_batch:.4f}")

```

Evaluation Metrics on Test Data:  
F1 Score: 0.8137  
MCC Score: 0.8191  
ROC AUC Score: 0.9685

## ↳ Scaled RandomForest Classifier Model

```

1 # @title Scaled RandomForest Classifier Model
2
3 # Basic Random Forest Classifier
4 model_rf_scaled_batch = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
5 model_rf_scaled_batch.fit(X_train_scaled, y_train)
6
7 # Make predictions on the scaled test data
8 y_pred_rf_scaled_batch = model_rf_scaled_batch.predict()
9 y_pred_proba_rf_scaled_batch = model_rf_scaled_batch.predict_proba(X_test_scaled)[:, 1]
10
11
12 # Evaluate the model
13 f1_rf_scaled_batch = f1_score(y_test, y_pred_rf_scaled_batch)
14 mcc_rf_scaled_batch = matthews_corrcoef(y_test, y_pred_rf_scaled_batch)
15 roc_auc_rf_scaled_batch = roc_auc_score(y_test, y_pred_proba_rf_scaled_batch)
16
17 # Print the evaluation metrics
18 print("\nEvaluation Metrics on Test Data (Scaled Random Forest Classifier):")
19 print(f"F1 Score: {f1_rf_scaled_batch:.4f}")
20 print(f"MCC Score: {mcc_rf_scaled_batch:.4f}")
21 print(f"ROC AUC Score: {roc_auc_rf_scaled_batch:.4f}")

```

Evaluation Metrics on Test Data (Scaled Random Forest Classifier):  
F1 Score: 0.8154  
MCC Score: 0.8222  
ROC AUC Score: 0.9679

## ↳ Linear Support Vector Class Model

```

1 # @title Linear Support Vector Class Model
2
3 # Instantiate the LinearSVC model
4 # Keeping dual=True for convergence
5 model_svc = LinearSVC(random_state=42, dual=True)
6
7 # Train the model on the training data
8 model_svc.fit(X_train_scaled, y_train)
9 print("Training complete.")
10
11 # Make predictions on the test data
12 y_pred_svc = model_svc.predict(X_test_scaled)
13
14 # Evaluate the model
15 f1_svc = f1_score(y_test, y_pred_svc)
16 mcc_svc = matthews_corrcoef(y_test, y_pred_svc)
17 roc_auc_svc = 0 #Not directly available for LinearSVC without calibration
18
19 # Print the evaluation metrics
20 print("\nEvaluation Metrics on Test Data (LinearSVC):")
21 print(f"F1 Score: {f1_svc:.4f}")
22 print(f"MCC Score: {mcc_svc:.4f}")
23 print("ROC AUC Score: Not directly available for LinearSVC without calibration")

```

Training complete.

```

Evaluation Metrics on Test Data (LinearSVC):
F1 Score: 0.8310
MCC Score: 0.8312
ROC AUC Score: Not directly available for LinearSVC without calibration
/usr/local/lib/python3.12/dist-packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to converge, increase the number of iter
warnings.warn(

```

## ▼ XGBoost Classifier Model

```

1 # @title XGBoost Classifier Model
2
3 # Instantiate the XGBoost classifier
4 model_xgb_batch = XGBClassifier(random_state=42,
5                                 use_label_encoder=False,
6                                 eval_metric='logloss'
7                                 )
8
9 # Train the model on the training data
10 model_xgb_batch.fit(X_train_scaled, y_train)
11 print("Training complete.")
12
13 # Make predictions on the test data
14 y_pred_xgb_batch = model_xgb_batch.predict(X_test_scaled)
15 y_pred_proba_xgb_batch = model_xgb_batch.predict_proba(X_test)[:, 1]
16
17 # Evaluate the model
18 f1_xgb_batch = f1_score(y_test, y_pred_xgb_batch)
19 mcc_xgb_batch = matthews_corrcoef(y_test, y_pred_xgb_batch)
20 roc_auc_xgb_batch = roc_auc_score(y_test, y_pred_proba_xgb_batch)
21
22 # Print the evaluation metrics
23 print("\nEvaluation Metrics on Test Data:")
24 print(f"F1 Score: {f1_xgb_batch:.4f}")
25 print(f"MCC Score: {mcc_xgb_batch:.4f}")
26 print(f"ROC AUC Score: {roc_auc_xgb_batch:.4f}")

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [17:30:35] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)
Training complete.

Evaluation Metrics on Test Data:
F1 Score: 0.8582
MCC Score: 0.8600
ROC AUC Score: 0.9599

```

1 Start coding or generate with AI.

1 Start coding or generate with AI.

## ▼ Testing Models with Balanced Dataset

### ▼ RandomForest Classifier with SMOTE Resampling

```

1 # @title RandomForest Classifier with SMOTE Resampling
2
3 # Apply SMOTE to the scaled training data
4 sm = SMOTE(random_state=42)
5 X_train_res, y_train_res = sm.fit_resample(X_train_scaled, y_train)
6
7 print(f"Shape of X_train after SMOTE: {X_train_res.shape}")
8 print(f"Shape of y_train after SMOTE: {y_train_res.shape}")
9 print("Class distribution after SMOTE:")
10 print(y_train_res.value_counts())
11
12 # Train a RandomForest Classifier on the SMOTE-resampled data
13 model_rf_smote = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
14 model_rf_smote.fit(X_train_res, y_train_res)
15 print("Training complete with SMOTE data.")
16
17 # Make predictions on the original scaled test data
18 y_pred_rf_smote = model_rf_smote.predict(X_test_scaled)
19 y_pred_proba_rf_smote = model_rf_smote.predict_proba(X_test_scaled)[:, 1]
20
21 # Evaluate the model
22 f1_rf_smote = f1_score(y_test, y_pred_rf_smote)
23 mcc_rf_smote = matthews_corrcoef(y_test, y_pred_rf_smote)
24 roc_auc_rf_smote = roc_auc_score(y_test, y_pred_proba_rf_smote)
25
26 # Print the evaluation metrics

```

```

27 print("\nEvaluation Metrics on Test Data (Random Forest with SMOTE):")
28 print(f"F1 Score: {f1_rf_smote:.4f}")
29 print(f"MCC Score: {mcc_rf_smote:.4f}")
30 print(f"ROC AUC Score: {roc_auc_rf_smote:.4f}")

Shape of X_train after SMOTE: (398038, 30)
Shape of y_train after SMOTE: (398038,)
Class distribution after SMOTE:
Class
0    199019
1    199019
Name: count, dtype: int64
Training complete with SMOTE data.

Evaluation Metrics on Test Data (Random Forest with SMOTE):
F1 Score: 0.3506
MCC Score: 0.4446
ROC AUC Score: 0.9829

```

## ▼ XGBoost Classifier with SMOTE Resampling

```

1 # @title XGBoost Classifier with SMOTE Resampling
2
3 # Apply SMOTE to the scaled training data for XGBoost
4 sm_xgb = SMOTE(random_state=42)
5 X_train_res_xgb, y_train_res_xgb = sm_xgb.fit_resample(X_train_scaled, y_train)
6
7 print(f"Shape of X_train after SMOTE for XGBoost: {X_train_res_xgb.shape}")
8 print(f"Shape of y_train after SMOTE for XGBoost: {y_train_res_xgb.shape}")
9 print("Class distribution after SMOTE for XGBoost:")
10 print(y_train_res_xgb.value_counts())
11
12 # Train an XGBoost Classifier on the SMOTE-resampled data
13 model_xgb_smote = XGBClassifier(random_state=42,
14                                  use_label_encoder=False,
15                                  eval_metric='logloss')
16 model_xgb_smote.fit(X_train_res_xgb, y_train_res_xgb)
17 print("Training complete with SMOTE data for XGBoost.")
18
19 # Make predictions on the original scaled test data
20 y_pred_xgb_smote = model_xgb_smote.predict(X_test_scaled)
21 y_pred_proba_xgb_smote = model_xgb_smote.predict_proba(X_test_scaled)[:, 1]
22
23 # Evaluate the model
24 f1_xgb_smote = f1_score(y_test, y_pred_xgb_smote)
25 mcc_xgb_smote = matthews_corrcoef(y_test, y_pred_xgb_smote)
26 roc_auc_xgb_smote = roc_auc_score(y_test, y_pred_proba_xgb_smote)
27
28 # Print the evaluation metrics
29 print("\nEvaluation Metrics on Test Data (XGBoost with SMOTE):")
30 print(f"F1 Score: {f1_xgb_smote:.4f}")
31 print(f"MCC Score: {mcc_xgb_smote:.4f}")
32 print(f"ROC AUC Score: {roc_auc_xgb_smote:.4f}")

Shape of X_train after SMOTE for XGBoost: (398038, 30)
Shape of y_train after SMOTE for XGBoost: (398038,)
Class distribution after SMOTE for XGBoost:
Class
0    199019
1    199019
Name: count, dtype: int64
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [18:04:01] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)
Training complete with SMOTE data for XGBoost.

Evaluation Metrics on Test Data (XGBoost with SMOTE):
F1 Score: 0.7914
MCC Score: 0.7949
ROC AUC Score: 0.9828

```

## ▼ Scaled RandomForest Classifier Model with Class Weight Balanced

```

1 # @title Scaled RandomForest Classifier Model with Class Weight Balanced
2
3 # Basic Random Forest Classifier with class_weight='balanced'
4 model_rf_scaled_balanced = RandomForestClassifier(max_depth=5,
5                                                 n_jobs=-1,
6                                                 random_state=42,
7                                                 class_weight='balanced')
8 model_rf_scaled_balanced.fit(X_train_scaled, y_train)
9 print("Training complete.")

10
11 # Make predictions on the scaled test data
12 y_pred_rf_scaled_balanced = model_rf_scaled_balanced.predict(X_test_scaled)

```

```

13 y_pred_proba_rf_scaled_balanced = model_rf_scaled_balanced.predict_proba(X_test_scaled)[:, 1]
14
15
16 # Evaluate the model
17 f1_rf_scaled_balanced = f1_score(y_test, y_pred_rf_scaled_balanced)
18 mcc_rf_scaled_balanced = matthews_corrcoef(y_test, y_pred_rf_scaled_balanced)
19 roc_auc_rf_scaled_balanced = roc_auc_score(y_test, y_pred_proba_rf_scaled_balanced)
20
21 # Print the evaluation metrics
22 print("\nEvaluation Metrics on Test Data (Scaled Random Forest Classifier with Balanced Class Weight):")
23 print(f"F1 Score: {f1_rf_scaled_balanced:.4f}")
24 print(f"MCC Score: {mcc_rf_scaled_balanced:.4f}")

```

Training complete.

Evaluation Metrics on Test Data (Scaled Random Forest Classifier with Balanced Class Weight):  
F1 Score: 0.5758  
MCC Score: 0.6172  
ROC AUC Score: 0.9848

## ↳ Scaled LinearSVC Classifier Model with Class Weight Balanced

```

1 # @title Scaled LinearSVC Classifier Model with Class Weight Balanced
2
3 # Instantiate the LinearSVC model
4 # Using 'balanced' class_weight to handle class imbalance
5 model_svc_balance = LinearSVC(random_state=42, class_weight='balanced',
dual=True)
6
7 # Train the model on the training data
8 model_svc_balance.fit(X_train_scaled, y_train)
9 print("Training complete.")
10
11 # Make predictions on the test data
12 y_pred_svc_balance = model_svc_balance.predict(X_test_scaled)
13
14 # Evaluate the model
15 f1_svc_balance = f1_score(y_test, y_pred_svc_balance)
16 mcc_svc_balance = matthews_corrcoef(y_test, y_pred_svc_balance)
17 roc_auc_svc_balance = 0 #Not directly available for LinearSVC without
calibration
18
19 # Print the evaluation metrics
20 print("\nEvaluation Metrics on Test Data (LinearSVC):")
21 print(f"F1 Score: {f1_svc_balance:.4f}")
22 print(f"MCC Score: {mcc_svc_balance:.4f}")
23 print("ROC AUC Score: Not directly available for LinearSVC without calibration")

```

Training complete.

Evaluation Metrics on Test Data (LinearSVC):  
F1 Score: 0.8025  
MCC Score: 0.8038  
ROC AUC Score: Not directly available for LinearSVC without calibration  
/usr/local/lib/python3.12/dist-packages/sklearn/svm/\_base.py:1249: ConvergenceWarning: Liblinear failed to converge, increase the number of iter
warnings.warn(

## ↳ Scaled XGBoost Classifier Model with Class Weight Balanced

```

1 # @title Scaled XGBoost Classifier Model with Class Weight Balanced
2
3 # Calculate scale_pos_weight for handling class imbalance
4 neg_count = np.sum(y_train == 0)
5 pos_count = np.sum(y_train == 1)
6 # Calculate scale_pos_weight
7 scale_pos_weight_value = neg_count / pos_count
8 print(f"Calculated scale_pos_weight: {scale_pos_weight_value:.2f}")
9
10
11 # Instantiate the XGBoost classifier
12 # Added scale_pos_weight parameter back for class balancing
13 model_xgb_balance = XGBClassifier(random_state=42,
use_label_encoder=False,
eval_metric='logloss',
scale_pos_weight=scale_pos_weight_value)
14
15
16
17
18 # Train the model on the training data
19 model_xgb_balance.fit(X_train_scaled, y_train)
20 print("Training complete.")
21
22 # Make predictions on the test data

```

```

23 y_pred_xgb_balance = model_xgb_balance.predict(X_test_scaled)
24 y_pred_proba_xgb_balance = model_xgb_balance.predict_proba(X_test_scaled)[:, 1] # Get probability of the positive class (1)
25
26 # Evaluate the model
27 f1_xgb_balance = f1_score(y_test, y_pred_xgb_balance)
28 mcc_xgb_balance = matthews_corrcoef(y_test, y_pred_xgb_balance)
29 roc_auc_xgb_balance = roc_auc_score(y_test, y_pred_proba_xgb_balance)
30
31 # Print the evaluation metrics
32 print("\nEvaluation Metrics on Test Data:")
33 print(f"F1 Score: {f1_xgb_balance:.4f}")
34 print(f"MCC Score: {mcc_xgb_balance:.4f}")
35 print(f"ROC AUC Score: {roc_auc_xgb_balance:.4f}")

```

Calculated scale\_pos\_weight: 576.87  
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [17:36:57] WARNING: /workspace/src/learner.cc:790:  
Parameters: { "use\_label\_encoder" } are not used.

```

bst.update(dtrain, iteration=i, fobj=obj)
Training complete.

```

Evaluation Metrics on Test Data:

```

F1 Score: 0.8889
MCC Score: 0.8889
ROC AUC Score: 0.9772

```

1 Start coding or generate with AI.

## Feature Selection on RandomForest Model

```

1 # @title Feature Selection on RandomForest Model
2
3 #Train an initial Random Forest model to get feature importances
4 initial_rf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced') # Use class_weight for imbalance
5 initial_rf.fit(X_train_scaled, y_train)
6 print("Initial Random Forest training complete.")
7
8 # Use SelectFromModel to select features based on a threshold or max_features
9 sfm = SelectFromModel(initial_rf, threshold='mean', prefit=True) # prefit=True since we already trained initial_rf
10
11 # Get the indices of the selected features
12 feature_indices = sfm.get_support(indices=True)
13 selected_features = X_train.columns[feature_indices] # Corrected from X_train_scaled.columns to X_train.columns
14
15 print(f"\nNumber of features before selection: {X_train_scaled.shape[1]}")
16 print(f"Number of features after selection: {len(selected_features)}")
17 print(f"Selected features: {list(selected_features)}")
18
19
20 # 3. Transform the training and testing data to include only selected features
21 X_train_selected = sfm.transform(X_train_scaled)
22 X_test_selected = sfm.transform(X_test_scaled)
23
24 # 4. Train a new Random Forest model on the selected features
25 print("\nTraining Random Forest model on selected features...")
26 # Using class_weight='balanced' again for the final model on imbalanced data
27 final_rf_selected = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
28 final_rf_selected.fit(X_train_selected, y_train)
29 print("Random Forest training on selected features complete.")
30
31
32 # 5. Make predictions on the test data with selected features
33 y_pred_rf_selected = final_rf_selected.predict(X_test_selected)
34 y_pred_proba_rf_selected = final_rf_selected.predict_proba(X_test_selected)[:, 1] # Get probability of the positive class (1)
35
36 # 6. Evaluate the model
37 f1_rf_selected = f1_score(y_test, y_pred_rf_selected)
38 mcc_rf_selected = matthews_corrcoef(y_test, y_pred_rf_selected)
39 roc_auc_rf_selected = roc_auc_score(y_test, y_pred_proba_rf_selected)
40
41 # Print the evaluation metrics
42 print("\nEvaluation Metrics on Test Data (Random Forest on Selected Features):")
43 print(f"F1 Score: {f1_rf_selected:.4f}")
44 print(f"MCC Score: {mcc_rf_selected:.4f}")
45 print(f"ROC AUC Score: {roc_auc_rf_selected:.4f}")

```

Initial Random Forest training complete.

Number of features before selection: 30  
Number of features after selection: 9

```
Selected features: ['V2', 'V3', 'V4', 'V10', 'V11', 'V12', 'V14', 'V16', 'V17']
```

```
Training Random Forest model on selected features...
Random Forest training on selected features complete.
```

```
Evaluation Metrics on Test Data (Random Forest on Selected Features):
```

```
F1 Score: 0.8832
MCC Score: 0.8854
ROC AUC Score: 0.9583
```

## ✓ Scaled LinearSVC Classifier Model with Class Weight Balanced and Feature Selection

```
1 # @title Scaled LinearSVC Classifier Model with Class Weight Balanced and Feature Selection
2
3 # Instantiate the LinearSVC model
4 # Using 'balanced' class_weight to handle class imbalance
5 model_svc_balance_selected = LinearSVC(random_state=42, class_weight='balanced', dual=True)
6
7 # Train the model on the selected and scaled training data
8 model_svc_balance_selected.fit(X_train_selected, y_train)
9 print("Training complete with selected features.")
10
11 # Make predictions on the selected and scaled test data
12 y_pred_svc_balance_selected = model_svc_balance_selected.predict(X_test_selected)
13
14 # Evaluate the model
15 f1_svc_balance_selected = f1_score(y_test, y_pred_svc_balance_selected)
16 mcc_svc_balance_selected = matthews_corrcoef(y_test, y_pred_svc_balance_selected)
17 roc_auc_svc_balance_selected = 0 #Not directly available for LinearSVC without calibration
18
19 # Print the evaluation metrics
20 print("\nEvaluation Metrics on Test Data (LinearSVC with Balanced Class Weight and Selected Features):")
21 print(f"F1 Score: {f1_svc_balance_selected:.4f}")
22 print(f"MCC Score: {mcc_svc_balance_selected:.4f}")
23 print("ROC AUC Score: Not directly available for LinearSVC without calibration")
```

```
Training complete with selected features.
```

```
Evaluation Metrics on Test Data (LinearSVC with Balanced Class Weight and Selected Features):
```

```
F1 Score: 0.7802
MCC Score: 0.7829
ROC AUC Score: Not directly available for LinearSVC without calibration
/usr/local/lib/python3.12/dist-packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to converge, increase the number of iter
warnings.warn(
```

```
1 # Collected final scores for each model from the metric variables
2 results = {
3     'Model': [
4         'Logistic Regression (No Scaling)',
5         'Scaled Logistic Regression',
6         'Adaptive Random Forest',
7         'Passive Aggressive Classifier',
8         'ARFClassifier (HardSampling)',
9         'AdaBoost (HardSampling)',
10        'AdaBoost (Hoeffding Tree)',
11        'AdaBoost (Scaled Hoeffding Tree)',
12        'ADWIN Boosting (Logistic Regression)'
13    ],
14    'F1 Score': [
15        ns_logreg.f1.get(),
16        logreg_f1.get(),
17        ARF_f1.get(),
18        f1_pa.get(),
19        f1_hs.get(),
20        f1_hs_ada.get(),
21        f1_ada.get(),
22        f1_ada_sc.get(),
23        f1_adwin_lr.get()
24    ],
25    'MCC Score': [
26        ns_logreg_mcc.get(),
27        logreg_mcc.get(),
28        ARF_mcc.get(),
29        mcc_pa.get(),
30        mcc_hs.get(),
31        mcc_hs_ada.get(),
32        mcc_ada.get(),
33        mcc_ada_sc.get(),
34        mcc_adwin_lr.get()
```

```

35     ],
36     'ROC AUC Score': [
37         ns_logreg_roc.get(),
38         logreg_roc.get(),
39         ARF_roc.get(),
40         roc_pa.get(),
41         roc_hs.get(),
42         roc_hs_ada.get(),
43         roc_ada.get(),
44         roc_ada_sc.get(),
45         roc_adwin_lr.get()
46     ]
47 }
48
49 df_results = pd.DataFrame(results)
50
51 # Set the index to model names for easier plotting
52 df_results = df_results.set_index('Model')
53
54 # Plotting the scores
55 fig, ax = plt.subplots(figsize=(10, 5))
56 fig.suptitle('Comparison of Model F1 Scores')
57
58 # F1 Score plot
59 df_results['F1 Score'].plot(kind='bar', ax=ax, color='pink')
60 ax.set_title('F1 Score')
61 ax.set_ylabel('Score')
62 ax.set_xlabel('Model')
63 ax.tick_params(axis='x')
64
65 plt.show()

```

1 Start coding or [generate](#) with AI.

```

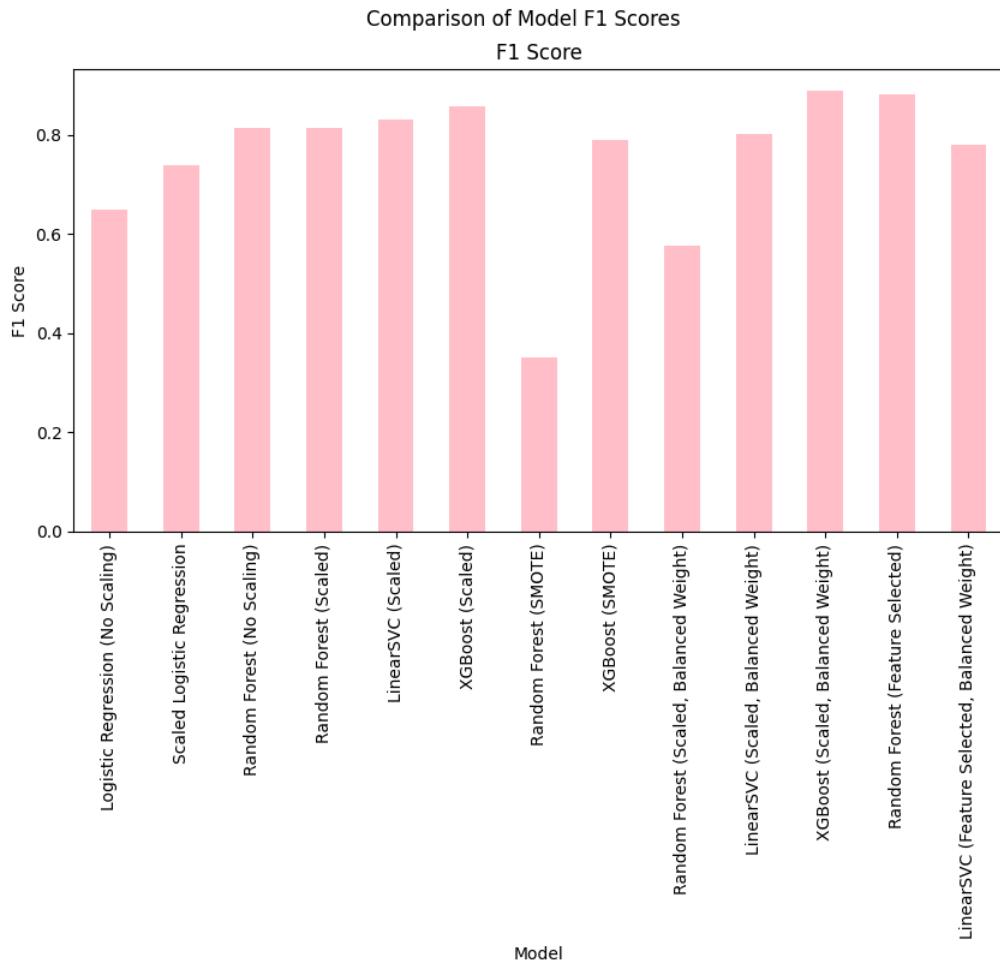
1   # Collected final scores for each model from the metric variables
2   results = {
3       'Model': [
4           'Logistic Regression (No Scaling)',
5           'Scaled Logistic Regression',
6           'Random Forest (No Scaling)',
7           'Random Forest (Scaled)',
8           'LinearSVC (Scaled)',
9           'XGBoost (Scaled)',
10          'Random Forest (SMOTE)',
11          'XGBoost (SMOTE)',
12          'Random Forest (Scaled, Balanced Weight)',
13          'LinearSVC (Scaled, Balanced Weight)',
14          'XGBoost (Scaled, Balanced Weight)',
15          'Random Forest (Feature Selected)',
16          'LinearSVC (Feature Selected, Balanced Weight)'
17      ],
18      'F1 Score': [
19          f1_lr,
20          f1_lr_batch,
21          f1_rf_batch,
22          f1_rf_scaled_batch,
23          f1_svc,
24          f1_xgb_batch,
25          f1_rf_smote,
26          f1_xgb_smote,
27          f1_rf_scaled_balanced,
28          f1_svc_balance,
29          f1_xgb_balance,
30          f1_rf_selected,
31          f1_svc_balance_selected
32      ],
33      'MCC Score': [
34          mcc_lr,
35          mcc_lr_batch,
36          mcc_rf_batch,
37          mcc_rf_scaled_batch,
38          mcc_svc,
39          mcc_xgb_batch,
40          mcc_rf_smote,
41          mcc_xgb_smote,
42          mcc_rf_scaled_balanced,
43          mcc_svc_balance,
44          mcc_xgb_balance,
45          mcc_rf_selected,
46          mcc_svc_balance_selected
47      ],
48      'ROC AUC Score': [
49          roc_auc_lr,
50          roc_auc_lr_batch,
51          roc_auc_rf_batch,

```

```

52     roc_auc_rf_scaled_batch,
53     roc_auc_svc,
54     roc_auc_xgb_batch,
55     roc_auc_rf_smote,
56     roc_auc_xgb_smote,
57     roc_auc_rf_scaled_balanced,
58     roc_auc_svc_balance,
59     roc_auc_xgb_balance,
60     roc_auc_rf_selected,
61     roc_auc_svc_balance_selected
62   ]
63 }
64
65 df_results = pd.DataFrame(results)
66
67 # Set the index to model names for easier plotting
68 df_results.set_index('Model')
69
70 fig, ax = plt.subplots(figsize=(10, 5))
71 fig.suptitle('Comparison of Model F1 Scores')
72
73 # F1 Score plot
74 df_results['F1 Score'].plot(kind='bar', ax=ax, color='pink')
75 ax.set_title('F1 Score')
76 ax.set_ylabel('F1 Score')
77 ax.tick_params(axis='x')
78
79 plt.show()

```

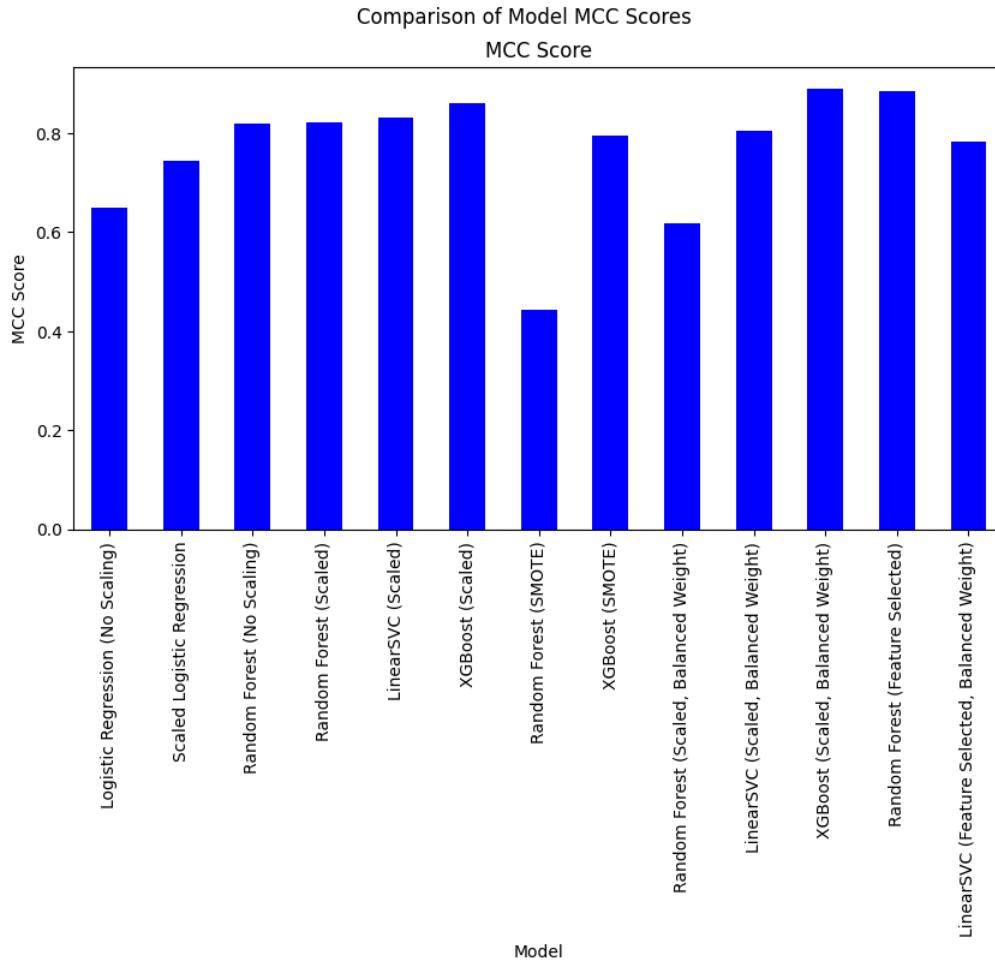


1 Start coding or [generate](#) with AI.

```

1 # Plotting the MCC scores
2 fig, ax = plt.subplots(figsize=(10, 5))
3 fig.suptitle('Comparison of Model MCC Scores')
4
5 # MCC Score plot
6 df_results['MCC Score'].plot(kind='bar', ax=ax, color='blue')
7 ax.set_title('MCC Score')
8 ax.set_ylabel('MCC Score')
9 ax.tick_params(axis='x')

```



```

1 # Plotting the ROC scores
2 fig, ax = plt.subplots(figsize=(10, 5))
3 fig.suptitle('Comparison of Model ROC AUC Scores')
4
5 # Filter out models with 0 ROC AUC Score (e.g., LinearSVC)
6 df_filtered_roc_results = df_results[df_results['ROC AUC Score'] != 0]
7
8 # MCC Score plot
9 df_filtered_roc_results['ROC AUC Score'].plot(kind='bar', ax=ax, color='green')
10 ax.set_title('ROC AUC Score')
11 ax.set_ylabel('ROC AUC Score')
12 ax.tick_params(axis='x')
13
14 plt.show()

```

Comparison of Model ROC AUC Scores

