```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import kagglehub
```

```python
# Download latest version
path = kagglehub.dataset_download("mlg-ulb/creditcardfraud")

print("Path to dataset files:", path)
```

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

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount
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	0.128539	-0.189115	0.133558	-0.021053	149.62
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	0.167170	0.125895	-0.008983	0.014724	2.69
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	-0.327642	-0.139097	-0.055353	-0.059752	378.66
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	0.647376	-0.221929	0.062723	0.061458	123.50
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	-0.206010	0.502292	0.219422	0.215153	69.99

5 rows × 31 columns

```python
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.84807
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	...	1.654067e-16	-3.568593e-16	2.57864
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	...	7.345240e-01	7.257016e-01	6.24460
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+01	...	-3.483038e+01	-1.093314e+01	-4.48077
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	...	-2.283949e-01	-5.423504e-01	-1.61846
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-02	...	-2.945017e-02	6.781943e-03	-1.11929
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	...	1.863772e-01	5.285536e-01	1.47642
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	...	2.720284e+01	1.050309e+01	2.25284

8 rows × 31 columns

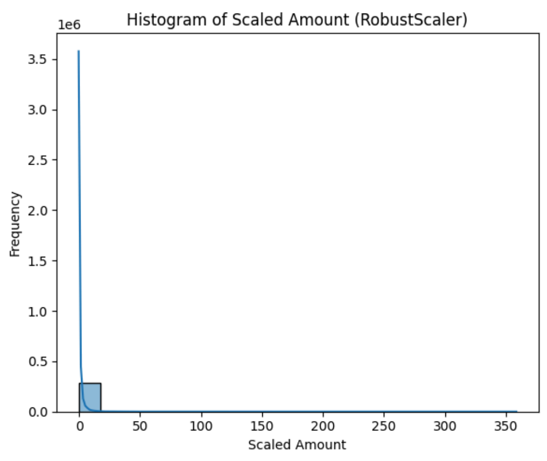
```
Start coding or generate with AI.
```

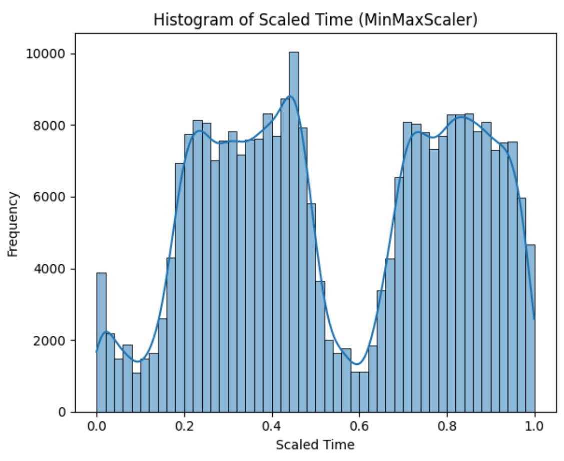
```python
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score, matthews_corrcoef, roc_auc_score, confusion_matrix
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
```

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

Scaling Variables for Normalization

```python
# @title Scaling Variables for Normalization

# Instantiate scikit-learn scalers
robust_scaler = RobustScaler()
minmax_scaler = MinMaxScaler()

# Apply RobustScaler to 'Amount' column
# Reshape the column to be 2D as fit_transform expects
scaled_amount = robust_scaler.fit_transform(new_df['Amount'].values.reshape(-1, 1))

# Apply MinMaxScaler to 'Time' column
# Reshape the column to be 2D
scaled_time = minmax_scaler.fit_transform(new_df['Time'].values.reshape(-1, 1))

# Plot histograms of the scaled columns
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(scaled_amount.flatten(), bins=20, kde=True) # flatten to get a 1D array
plt.title('Histogram of Scaled Amount (RobustScaler)')
plt.xlabel('Scaled Amount')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.histplot(scaled_time.flatten(), bins=50, kde=True) # flatten to get a 1D array
plt.title('Histogram of Scaled Time (MinMaxScaler)')
plt.xlabel('Scaled Time')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

```





```python
# Separate features (X) and target (y)
X = new_df.drop('Class', axis=1)
y = new_df['Class']

# Perform train-test split
# test_size=0.3 means 30% of data for testing, 70% for training
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print("Dataset split into training and testing sets.")
print(f"Training features shape: {x_train.shape}")
print(f"Testing features shape: {x_test.shape}")
print(f"Training target shape: {y_train.shape}")
print(f"Testing target shape: {y_test.shape}")

# Display the class distribution in train and test sets to verify stratification
print("\nClass distribution in training set:")
print(y_train.value_counts(normalize=True))
print("\nClass distribution in testing set:")
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

```python
# @title Logistic Regression Model

# Instantiate the Logistic Regression model
model_lr = LogisticRegression(random_state=42, solver='liblinear')

# Train the model on the training data
model_lr.fit(x_train, y_train)
print("Training complete.")

# Make predictions on the test data
y_pred_lr = model_lr.predict(x_test)
y_pred_proba_lr = model_lr.predict_proba(x_test)[:, 1]

# Evaluate the model
f1_lr = f1_score(y_test, y_pred_lr)
mcc_lr = matthews_corrcoef(y_test, y_pred_lr)
roc_auc_lr = roc_auc_score(y_test, y_pred_proba_lr)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data:")
print(f"F1 Score: {f1_lr:.4f}")
print(f"MCC Score: {mcc_lr:.4f}")
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
```

Scaling Preprocessing

```python
#@title Scaling Preprocessing

```

```
# Instantiate the StandardScaler
scaler_batch = StandardScaler()

# Scale the training and testing data
X_train_scaled = scaler_batch.fit_transform(x_train)
X_test_scaled = scaler_batch.transform(x_test)
```

Scaled Logistic Regression Model

```
# @title Scaled Logistic Regression Model

# Instantiate the Logistic Regression model

model_lr_batch = LogisticRegression(random_state=42, solver='liblinear')
model_lr_batch.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the scaled test data
y_pred_lr_batch = model_lr_batch.predict(X_test_scaled)
y_pred_proba_lr_batch = model_lr_batch.predict_proba(X_test_scaled)[:, 1] # Get probability of the positive class (1)

# Evaluate the model
f1_lr_batch = f1_score(y_test, y_pred_lr_batch)
mcc_lr_batch = matthews_corrcoef(y_test, y_pred_lr_batch)
roc_auc_lr_batch = roc_auc_score(y_test, y_pred_proba_lr_batch)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data:")
print(f"F1 Score: {f1_lr_batch:.4f}")
print(f"MCC Score: {mcc_lr_batch:.4f}")
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

```
  # @title RandomForest Classifier Model

  # Basic Random Forest Classifier
  model_rf_batch = RandomForestClassifier(max_depth=5, n_jobs=-1)
  model_rf_batch.fit(x_train, y_train)

  # Make predictions on the test data
  y_pred_rf_batch = model_rf_batch.predict(x_test)
  y_pred_proba_rf_batch = model_rf_batch.predict_proba(x_test)[:, 1]


  # Evaluate the model
  f1_rf_batch = f1_score(y_test, y_pred_rf_batch)
  mcc_rf_batch = matthews_corrcoef(y_test, y_pred_rf_batch)
  roc_auc_rf_batch = roc_auc_score(y_test, y_pred_proba_rf_batch)

  # Print the evaluation metrics
  print("\nEvaluation Metrics on Test Data:")
  print(f"F1 Score: {f1_rf_batch:.4f}")
  print(f"MCC Score: {mcc_rf_batch:.4f}")
  print(f"ROC AUC Score: {roc_auc_rf_batch:.4f}")
```

```

Evaluation Metrics on Test Data:
F1 Score: 0.8244
MCC Score: 0.8304
ROC AUC Score: 0.9742
```

Scaled RandomForest Classifier Model

```
# @title Scaled RandomForest Classifier Model

# Basic Random Forest Classifier
model_rf_scaled_batch = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
model_rf_scaled_batch.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
y_pred_rf_scaled_batch = model_rf_scaled_batch.predict(X_test_scaled)
y_pred_proba_rf_scaled_batch = model_rf_scaled_batch.predict_proba(X_test_scaled)[:, 1]


# Evaluate the model
f1_rf_scaled_batch = f1_score(y_test, y_pred_rf_scaled_batch)
mcc_rf_scaled_batch = matthews_corrcoef(y_test, y_pred_rf_scaled_batch)
roc_auc_rf_scaled_batch = roc_auc_score(y_test, y_pred_proba_rf_scaled_batch)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (Scaled Random Forest Classifier):")
print(f"F1 Score: {f1_rf_scaled_batch:.4f}")
print(f"MCC Score: {mcc_rf_scaled_batch:.4f}")
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

```
# @title Linear Support Vector Class Model

# Instantiate the LinearSVC model
```

```python
# Keeping dual=True for convergence
model_svc = LinearSVC(random_state=42, dual=True)

# Train the model on the training data
model_svc.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the test data
y_pred_svc = model_svc.predict(X_test_scaled)

# Evaluate the model
f1_svc = f1_score(y_test, y_pred_svc)
mcc_svc = matthews_corrcoef(y_test, y_pred_svc)
roc_auc_svc = 0 #Not directly available for LinearSVC without calibration

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (LinearSVC):")
print(f"F1 Score: {f1_svc:.4f}")
print(f"MCC Score: {mcc_svc:.4f}")
```

```
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 iterations.
  warnings.warn(
```

XGBoost Classifier Model

```python
# @title XGBoost Classifier Model

# Instantiate the XGBoost classifier
model_xgb_batch = XGBClassifier(random_state=42,
                                use_label_encoder=False,
                                eval_metric='logloss'
                                )

# Train the model on the training data
model_xgb_batch.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the test data
y_pred_xgb_batch = model_xgb_batch.predict(X_test_scaled)
y_pred_proba_xgb_batch = model_xgb_batch.predict_proba(x_test)[:, 1]

# Evaluate the model
f1_xgb_batch = f1_score(y_test, y_pred_xgb_batch)
mcc_xgb_batch = matthews_corrcoef(y_test, y_pred_xgb_batch)
roc_auc_xgb_batch = roc_auc_score(y_test, y_pred_proba_xgb_batch)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data:")
print(f"F1 Score: {f1_xgb_batch:.4f}")
print(f"MCC Score: {mcc_xgb_batch:.4f}")
print(f"ROC AUC Score: {roc_auc_xgb_batch:.4f}")
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [00:39:49] 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
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Testing Models with Balanced Dataset

RandomForest Classifier with SMOTE Resampling

```python
# @title RandomForest Classifier with SMOTE Resampling

# Apply SMOTE to the scaled training data
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train_scaled, y_train)

print(f"Shape of X_train after SMOTE: {X_train_res.shape}")
print(f"Shape of y_train after SMOTE: {y_train_res.shape}")
print("Class distribution after SMOTE:")
print(y_train_res.value_counts())

# Train a RandomForest Classifier on the SMOTE-resampled data
model_rf_smote = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
model_rf_smote.fit(X_train_res, y_train_res)
print("Training complete with SMOTE data.")

# Make predictions on the original scaled test data
y_pred_rf_smote = model_rf_smote.predict(X_test_scaled)
y_pred_proba_rf_smote = model_rf_smote.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
f1_rf_smote = f1_score(y_test, y_pred_rf_smote)
mcc_rf_smote = matthews_corrcoef(y_test, y_pred_rf_smote)
roc_auc_rf_smote = roc_auc_score(y_test, y_pred_proba_rf_smote)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (Random Forest with SMOTE):")
print(f"F1 Score: {f1_rf_smote:.4f}")
print(f"MCC Score: {mcc_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

```python

# @title XGBoost Classifier with SMOTE Resampling

# Apply SMOTE to the scaled training data for XGBoost
sm_xgb = SMOTE(random_state=42)
X_train_res_xgb, y_train_res_xgb = sm_xgb.fit_resample(X_train_scaled, y_train)

print(f"Shape of X_train after SMOTE for XGBoost: {X_train_res_xgb.shape}")
print(f"Shape of y_train after SMOTE for XGBoost: {y_train_res_xgb.shape}")
print("Class distribution after SMOTE for XGBoost:")
print(y_train_res_xgb.value_counts())

# Train an XGBoost Classifier on the SMOTE-resampled data
model_xgb_smote = XGBClassifier(random_state=42,
                                use_label_encoder=False,
                                eval_metric='logloss')
model_xgb_smote.fit(X_train_res_xgb, y_train_res_xgb)
print("Training complete with SMOTE data for XGBoost.")

# Make predictions on the original scaled test data
y_pred_xgb_smote = model_xgb_smote.predict(X_test_scaled)
y_pred_proba_xgb_smote = model_xgb_smote.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
f1_xgb_smote = f1_score(y_test, y_pred_xgb_smote)
mcc_xgb_smote = matthews_corrcoef(y_test, y_pred_xgb_smote)
roc_auc_xgb_smote = roc_auc_score(y_test, y_pred_proba_xgb_smote)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (XGBoost with SMOTE):")
print(f"F1 Score: {f1_xgb_smote:.4f}")
print(f"MCC Score: {mcc_xgb_smote:.4f}")
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: [00:45:54] 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

```python
# @title Scaled RandomForest Classifier Model with Class Weight Balanced

# Basic Random Forest Classifier with class_weight='balanced'
model_rf_scaled_balanced = RandomForestClassifier(max_depth=5,
                                                  n_jobs=-1,
                                                  random_state=42,
                                                  class_weight='balanced')
model_rf_scaled_balanced.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the scaled test data
y_pred_rf_scaled_balanced = model_rf_scaled_balanced.predict(X_test_scaled)
y_pred_proba_rf_scaled_balanced = model_rf_scaled_balanced.predict_proba(X_test_scaled)[:, 1]


# Evaluate the model
f1_rf_scaled_balanced = f1_score(y_test, y_pred_rf_scaled_balanced)
mcc_rf_scaled_balanced = matthews_corrcoef(y_test, y_pred_rf_scaled_balanced)
roc_auc_rf_scaled_balanced = roc_auc_score(y_test, y_pred_proba_rf_scaled_balanced)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (Scaled Random Forest Classifier with Balanced Class Weight):")
print(f"F1 Score: {f1_rf_scaled_balanced:.4f}")
print(f"MCC Score: {mcc_rf_scaled_balanced:.4f}")
print(f"ROC AUC Score: {roc_auc_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

```
# @title Scaled LinearSVC Classifier Model with Class Weight Balanced

# Instantiate the LinearSVC model
# Using 'balanced' class_weight to handle class imbalance
model_svc_balance = LinearSVC(random_state=42, class_weight='balanced', dual=True)

# Train the model on the training data
model_svc_balance.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the test data
y_pred_svc_balance = model_svc_balance.predict(X_test_scaled)

# Evaluate the model
f1_svc_balance = f1_score(y_test, y_pred_svc_balance)
mcc_svc_balance = matthews_corrcoef(y_test, y_pred_svc_balance)
roc_auc_svc_balance = 0 #Not directly available for LinearSVC without calibration

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (LinearSVC):")
print(f"F1 Score: {f1_svc_balance:.4f}")
print(f"MCC Score: {mcc_svc_balance:.4f}")
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 iterations.
  warnings.warn(
```

Scaled XGBoost Classifier Model with Class Weight Balanced

```
# @title Scaled XGBoost Classifier Model with Class Weight Balanced

# Calculate scale_pos_weight for handling class imbalance
neg_count = np.sum(y_train == 0)
pos_count = np.sum(y_train == 1)
# Calculate scale_pos_weight
scale_pos_weight_value = neg_count / pos_count
print(f"Calculated scale_pos_weight: {scale_pos_weight_value:.2f}")


# Instantiate the XGBoost classifier
# Added scale_pos_weight parameter back for class balancing
model_xgb_balance = XGBClassifier(random_state=42,
                                use_label_encoder=False,
                                eval_metric='logloss',
                                scale_pos_weight=scale_pos_weight_value)

# Train the model on the training data
model_xgb_balance.fit(X_train_scaled, y_train)
print("Training complete.")

# Make predictions on the test data
y_pred_xgb_balance = model_xgb_balance.predict(X_test_scaled)
y_pred_proba_xgb_balance = model_xgb_balance.predict_proba(X_test_scaled)[:, 1] # Get probability of the positive class (1)

# Evaluate the model
f1_xgb_balance = f1_score(y_test, y_pred_xgb_balance)
mcc_xgb_balance = matthews_corrcoef(y_test, y_pred_xgb_balance)
roc_auc_xgb_balance = roc_auc_score(y_test, y_pred_proba_xgb_balance)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data:")
print(f"F1 Score: {f1_xgb_balance:.4f}")
print(f"MCC Score: {mcc_xgb_balance:.4f}")
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: [00:48:53] 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
```

```
Start coding or generate with AI.
```

Feature Selection on RandomForest Model

```
# @title Feature Selection on RandomForest Model

#Train an initial Random Forest model to get feature importances
initial_rf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced') # Use class_weight for imbalance
initial_rf.fit(X_train_scaled, y_train)
print("Initial Random Forest training complete.")

# Use SelectFromModel to select features based on a threshold or max_features
sfm = SelectFromModel(initial_rf, threshold='mean', prefit=True) # prefit=True since we already trained initial_rf

# Get the indices of the selected features
feature_indices = sfm.get_support(indices=True)
selected_features = x_train.columns[feature_indices] # Corrected from X_train_scaled.columns to x_train.columns

print(f"\nNumber of features before selection: {X_train_scaled.shape[1]}")
print(f"Number of features after selection: {len(selected_features)}")
print(f"Selected features: {list(selected_features)}")


```

```python
# 3. Transform the training and testing data to include only selected features
X_train_selected = sfm.transform(X_train_scaled)
X_test_selected = sfm.transform(X_test_scaled)

# 4. Train a new Random Forest model on the selected features
print("\nTraining Random Forest model on selected features...")
# Using class_weight='balanced' again for the final model on imbalanced data
final_rf_selected = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
final_rf_selected.fit(X_train_selected, y_train)
print("Random Forest training on selected features complete.")


# 5. Make predictions on the test data with selected features
y_pred_rf_selected = final_rf_selected.predict(X_test_selected)
y_pred_proba_rf_selected = final_rf_selected.predict_proba(X_test_selected)[:, 1] # Get probability of the positive class (1)

# 6. Evaluate the model
f1_rf_selected = f1_score(y_test, y_pred_rf_selected)
mcc_rf_selected = matthews_corrcoef(y_test, y_pred_rf_selected)
roc_auc_rf_selected = roc_auc_score(y_test, y_pred_proba_rf_selected)

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (Random Forest on Selected Features):")
print(f"F1 Score: {f1_rf_selected:.4f}")
print(f"MCC Score: {mcc_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
```

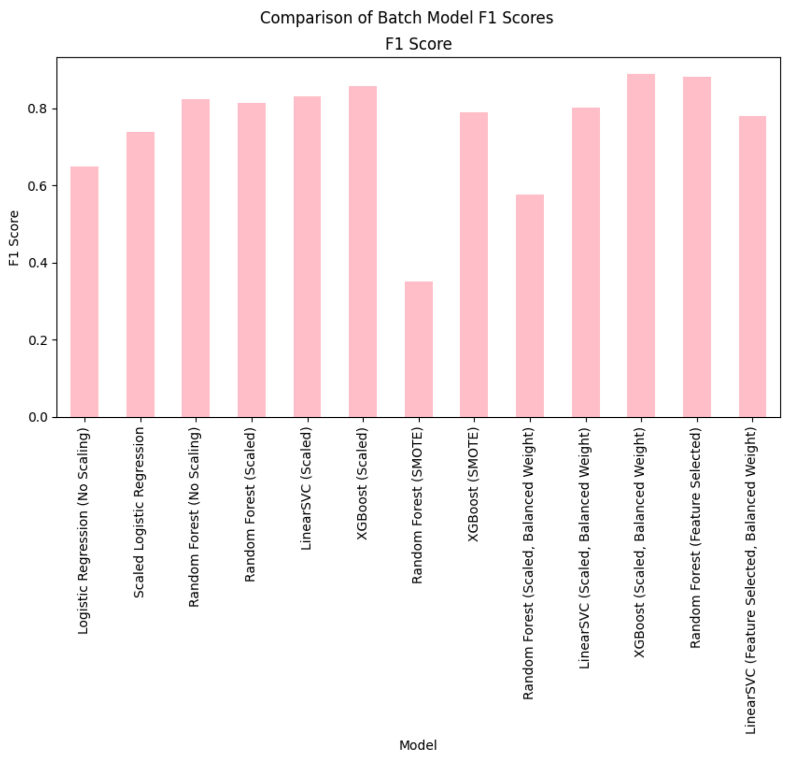
Scaled LinearSVC Classifier Model with Class Weight Balanced and Feature Selection

```python
# @title Scaled LinearSVC Classifier Model with Class Weight Balanced and Feature Selection

# Instantiate the LinearSVC model
# Using 'balanced' class_weight to handle class imbalance
model_svc_balance_selected = LinearSVC(random_state=42, class_weight='balanced', dual=True)

# Train the model on the selected and scaled training data
model_svc_balance_selected.fit(X_train_selected, y_train)
print("Training complete with selected features.")

# Make predictions on the selected and scaled test data
y_pred_svc_balance_selected = model_svc_balance_selected.predict(X_test_selected)

# Evaluate the model
f1_svc_balance_selected = f1_score(y_test, y_pred_svc_balance_selected)
mcc_svc_balance_selected = matthews_corrcoef(y_test, y_pred_svc_balance_selected)
roc_auc_svc_balance_selected = 0 #Not directly available for LinearSVC without calibration

# Print the evaluation metrics
print("\nEvaluation Metrics on Test Data (LinearSVC with Balanced Class Weight and Selected Features):")
print(f"F1 Score: {f1_svc_balance_selected:.4f}")
print(f"MCC Score: {mcc_svc_balance_selected:.4f}")
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 iterations.
  warnings.warn(
```

```
Start coding or generate with AI.
```

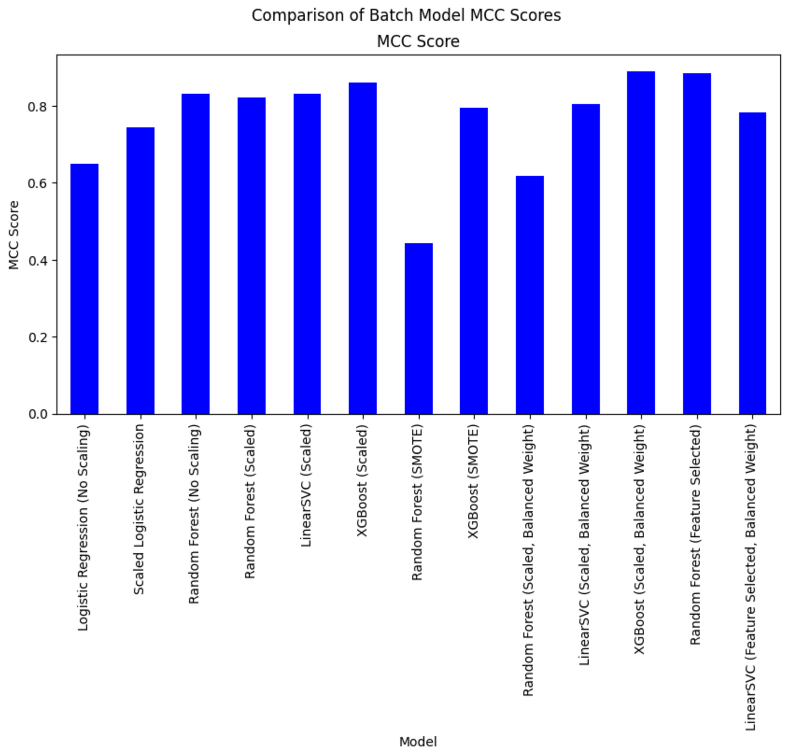
```python
# Collected final scores for each model from the metric variables
results = {
    'Model': [
        'Logistic Regression (No Scaling)',
        'Scaled Logistic Regression',
        'Random Forest (No Scaling)',
        'Random Forest (Scaled)',
        'LinearSVC (Scaled)',
        'XGBoost (Scaled)',
        'Random Forest (SMOTE)',
        'XGBoost (SMOTE)',
        'Random Forest (Scaled, Balanced Weight)',
        'LinearSVC (Scaled, Balanced Weight)',
        'XGBoost (Scaled, Balanced Weight)',
        'Random Forest (Feature Selected)',
        'LinearSVC (Feature Selected, Balanced Weight)'
    ],
    'F1 Score': [
        f1_lr,
        f1_lr_batch,
        f1_rf_batch,
        f1_rf_scaled_batch,
        f1_svc,
        f1_xgb_batch,
        f1_rf_smote,
        f1_xgb_smote,
        f1_rf_scaled_balanced,
        f1_svc_balance,
        f1_xgb_balance,
        f1_rf_selected,
        f1_svc_balance_selected
    ],
    'MCC Score': [
        mcc_lr,
        mcc_lr_batch,
        mcc_rf_batch,
        mcc_rf_scaled_batch,
        mcc_svc,
        mcc_xgb_batch,
        mcc_rf_smote,
        mcc_xgb_smote,
        mcc_rf_scaled_balanced,
        mcc_svc_balance,
        mcc_xgb_balance,
        mcc_rf_selected,
        mcc_svc_balance_selected
    ],
    'ROC AUC Score': [
        roc_auc_lr,
        roc_auc_lr_batch,
        roc_auc_rf_batch,
        roc_auc_rf_scaled_batch,
        roc_auc_svc,
        roc_auc_xgb_batch,
        roc_auc_rf_smote,
        roc_auc_xgb_smote,
        roc_auc_rf_scaled_balanced,
        roc_auc_svc_balance,
        roc_auc_xgb_balance,
        roc_auc_rf_selected,
        roc_auc_svc_balance_selected
    ]
}

df_results = pd.DataFrame(results)

# Set the index to model names for easier plotting
df_results = df_results.set_index('Model')

fig, ax = plt.subplots(figsize=(10, 5))
fig.suptitle('Comparison of Batch Model F1 Scores')

# F1 Score plot
df_results['F1 Score'].plot(kind='bar', ax=ax, color='pink')
ax.set_title('F1 Score')
ax.set_ylabel('F1 Score')
ax.tick_params(axis='x')

```



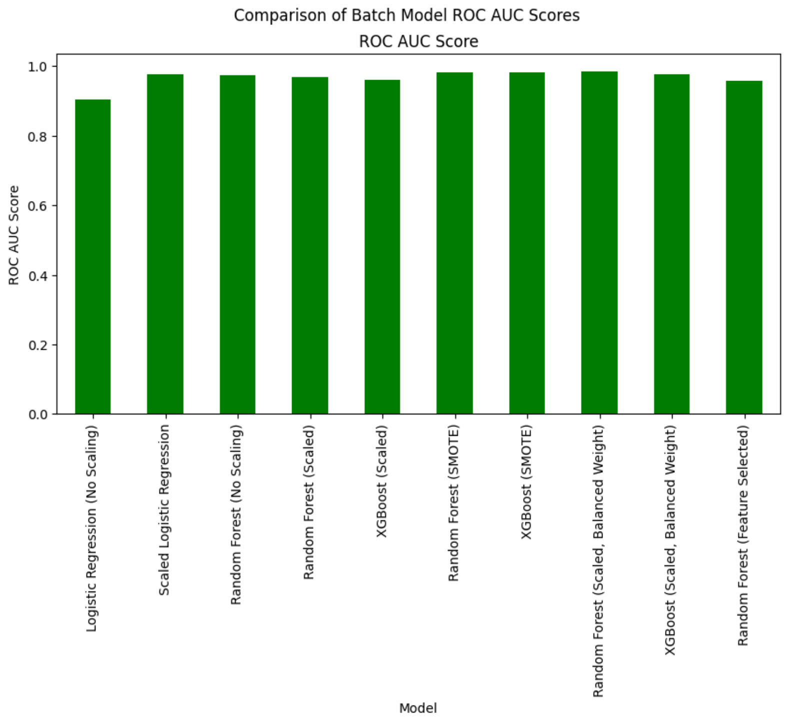
```
Start coding or generate with AI.
```

```
# Plotting the MCC scores
fig, ax = plt.subplots(figsize=(10, 5))
fig.suptitle('Comparison of Batch Model MCC Scores')

# MCC Score plot
```

```
df_results['MCC Score'].plot(kind='bar', ax=ax, color='blue')
ax.set_title('MCC Score')
ax.set_ylabel('MCC Score')
ax.tick_params(axis='x')

plt.show()
```



```
# Plotting the ROC scores
fig, ax = plt.subplots(figsize=(10, 5))
fig.suptitle('Comparison of Batch Model ROC AUC Scores')

# Filter out models with 0 ROC AUC Score (e.g., LinearSVC)
df_filtered_roc_results = df_results[df_results['ROC AUC Score'] != 0]

# MCC Score plot
df_filtered_roc_results['ROC AUC Score'].plot(kind='bar', ax=ax, color='green')
ax.set_title('ROC AUC Score')
ax.set_ylabel('ROC AUC Score')
ax.tick_params(axis='x')

plt.show()
```



```
Start coding or generate with AI.
```