```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score, confusion_matrix, classification_report
from sklearn.metrics import precision_recall_curve, roc_curve
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')

# Load the dataset (replace with your actual data source)
# For demonstration, we'll use a sample dataset structure
# In practice, you would load your actual customer data
def load_data():
    # Example: df = pd.read_csv('customer_churn_data.csv')
    # For this demo, we'll create synthetic data
    np.random.seed(42)
    n_samples = 1000

    data = {
        'customer_id': np.arange(n_samples),
        'age': np.random.randint(18, 70, size=n_samples),
        'gender': np.random.choice(['Male', 'Female'], size=n_samples),
        'tenure': np.random.randint(1, 72, size=n_samples),
        'usage_frequency': np.random.randint(1, 100, size=n_samples),
        'support_calls': np.random.randint(0, 10, size=n_samples),
        'payment_delay': np.random.randint(0, 30, size=n_samples),
        'subscription_type': np.random.choice(['Basic', 'Standard', 'Premium'], size=n_sa
        'monthly_charges': np.round(np.random.uniform(20, 100, size=n_samples), 2),
        'total_charges': np.round(np.random.u                                    2),
        'churn': np.random.choice([0, 1], size
    }

    df = pd.DataFrame(data)
    return df

# Data Exploration and Visualization
def explore_data(df):
    print("Dataset Overview:")
    print(df.head())
    print("\nDataset Info:")
    print(df.info())
    print("\nDescriptive Statistics:")
    print(df.describe())
    print("\nClass Distribution:")
```

```
    print(df['churn'].value_counts())

    # Visualizations
    plt.figure(figsize=(15, 10))

    # Churn distribution
    plt.subplot(2, 2, 1)
    sns.countplot(x='churn', data=df)
    plt.title('Churn Distribution')

    # Age distribution by churn
    plt.subplot(2, 2, 2)
    sns.boxplot(x='churn', y='age', data=df)
    plt.title('Age Distribution by Churn')

    # Tenure distribution by churn
    plt.subplot(2, 2, 3)
    sns.boxplot(x='churn', y='tenure', data=df)
    plt.title('Tenure Distribution by Churn')

    # Monthly charges by churn
    plt.subplot(2, 2, 4)
    sns.boxplot(x='churn', y='monthly_charges', data=df)
    plt.title('Monthly Charges by Churn')

    plt.tight_layout()
    plt.show()

    # Correlation matrix
    numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
    plt.figure(figsize=(10, 8))
    sns.heatmap(df[numeric_cols].corr(), annot=True, cmap='coolwarm', center=0)
    plt.title('Correlation Matrix')
    plt.show()

# Data Preprocessing
def preprocess_data(df):
    # Drop customer ID as it's not a feature
    df = df.drop('customer_id', axis=1)

    # Convert categorical variables to numeri
    categorical_cols = ['gender', 'subscripti
    label_encoders = {}

    for col in categorical_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

    # Handle missing values (if any)
    # In our synthetic data there are none, but in real data you might need:
    # df = df.fillna(df.mean())  # for numerical
    # df = df.fillna(df.mode().iloc[0])  # for categorical

    # Separate features and target
```

Run this cell to mount your Google Drive.
Learn more

```python
    X = df.drop('churn', axis=1)
    y = df['churn']

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y)

    # Scale numerical features
    scaler = StandardScaler()
    numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
    X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
    X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])

    # Handle class imbalance using SMOTE
    smote = SMOTE(random_state=42)
    X_train, y_train = smote.fit_resample(X_train, y_train)

    return X_train, X_test, y_train, y_test, scaler, label_encoders

# Model Training and Evaluation
def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    models = {
        'Logistic Regression': LogisticRegression(random_state=42),
        'Random Forest': RandomForestClassifier(random_state=42),
        'Gradient Boosting': GradientBoostingClassifier(random_state=42),
        'SVM': SVC(probability=True, random_state=42)
    }

    results = {}

    for name, model in models.items():
        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1]

        # Calculate metrics
        accuracy = accuracy_score(y_test, y_pr
        precision = precision_score(y_test, y_
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        roc_auc = roc_auc_score(y_test, y_prob

        # Store results
        results[name] = {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'roc_auc': roc_auc,
            'model': model
        }
```

Run this cell to mount your Google Drive.
Learn more

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

        # Plot confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(5, 5))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
        plt.title(f'{name} Confusion Matrix')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.show()

    return results

# Feature Importance Analysis
def analyze_feature_importance(model, feature_names):
    if hasattr(model, 'feature_importances_'):
        importances = model.feature_importances_
        indices = np.argsort(importances)[::-1]

        plt.figure(figsize=(10, 6))
        plt.title('Feature Importances')
        plt.bar(range(len(importances)), importances[indices], align='center')
        plt.xticks(range(len(importances)), [feature_names[i] for i in indices], rotation
        plt.tight_layout()
        plt.show()
    elif hasattr(model, 'coef_'):
        coefficients = model.coef_[0]
        indices = np.argsort(np.abs(coefficients))[::-1]

        plt.figure(figsize=(10, 6))
        plt.title('Feature Coefficients (Absolute Values)')
        plt.bar(range(len(coefficients)), np.abs(coefficients[indices]), align='center')
        plt.xticks(range(len(coefficients)), [feature_names[i] for i in indices], rotatic
        plt.tight_layout()
        plt.show()

# Hyperparameter Tuning
def tune_hyperparameters(X_train, y_train):
    # Example with Random Forest
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    rf = RandomForestClassifier(random_state=42)
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                               cv=5, n_jobs=-1, scoring='roc_auc')
    grid_search.fit(X_train, y_train)

    print("Best parameters found: ", grid_search.best_params_)
    print("Best ROC AUC score: ", grid_search.best_score_)
```

```
    return grid_search.best_estimator_

# Main execution
def main():
    # Step 1: Load data
    print("Loading data...")
    df = load_data()

    # Step 2: Explore data
    print("\nExploring data...")
    explore_data(df)

    # Step 3: Preprocess data
    print("\nPreprocessing data...")
    X_train, X_test, y_train, y_test, scaler, label_encoders = preprocess_data(df)

    # Step 4: Train and evaluate models
    print("\nTraining and evaluating models...")
    results = train_and_evaluate_models(X_train, X_test, y_train, y_test)

    # Step 5: Analyze feature importance (using the best model)
    best_model_name = max(results, key=lambda x: results[x]['roc_auc'])
    best_model = results[best_model_name]['model']
    print(f"\nBest model: {best_model_name} with ROC AUC: {results[best_model_name]['roc_

    feature_names = X_train.columns
    print("\nAnalyzing feature importance...")
    analyze_feature_importance(best_model, feature_names)

    # Step 6: Hyperparameter tuning (optional)
    print("\nPerforming hyperparameter tuning...")
    tuned_model = tune_hyperparameters(X_train, y_train)

    # Evaluate tuned model
    y_pred_tuned = tuned_model.predict(X_test)
    y_prob_tuned = tuned_model.predict_proba(X_test)[:, 1]
    roc_auc_tuned = roc_auc_score(y_test, y_prob_tuned)
    print(f"\nTuned model ROC AUC: {roc_auc_tuned:.4f}")

    # Plot ROC curves for all models
    plt.figure(figsize=(10, 8))
    for name, result in results.items():
        y_prob = result['model'].predict_prob
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {result["roc_auc"]:.2f})')

    # Add tuned model to ROC plot
    fpr_tuned, tpr_tuned, _ = roc_curve(y_test, y_prob_tuned)
    plt.plot(fpr_tuned, tpr_tuned, label=f'Tuned {best_model_name} (AUC = {roc_auc_tuned:

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve Comparison')
```

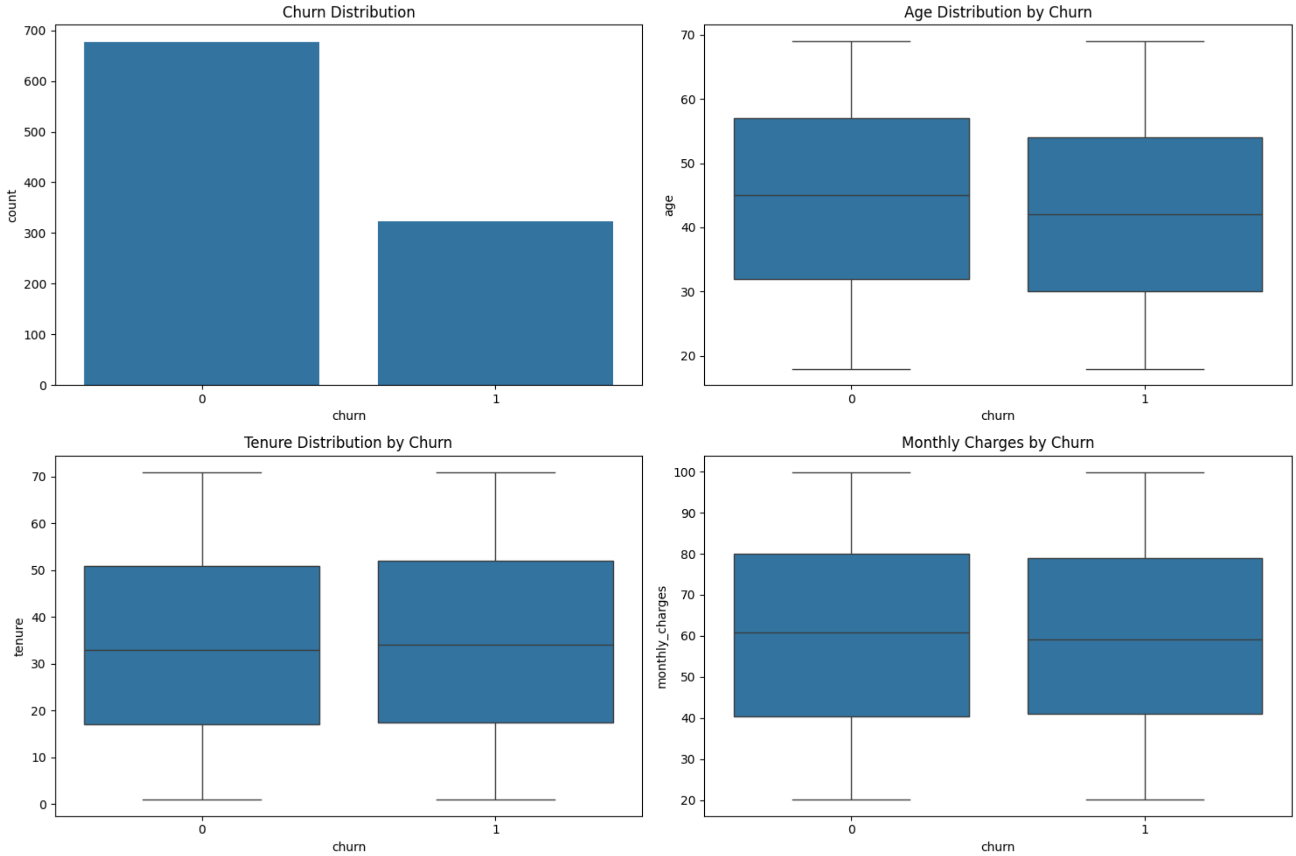
```
        plt.legend(loc='lower right')
        plt.show()

    if __name__ == "__main__":
        main()
```

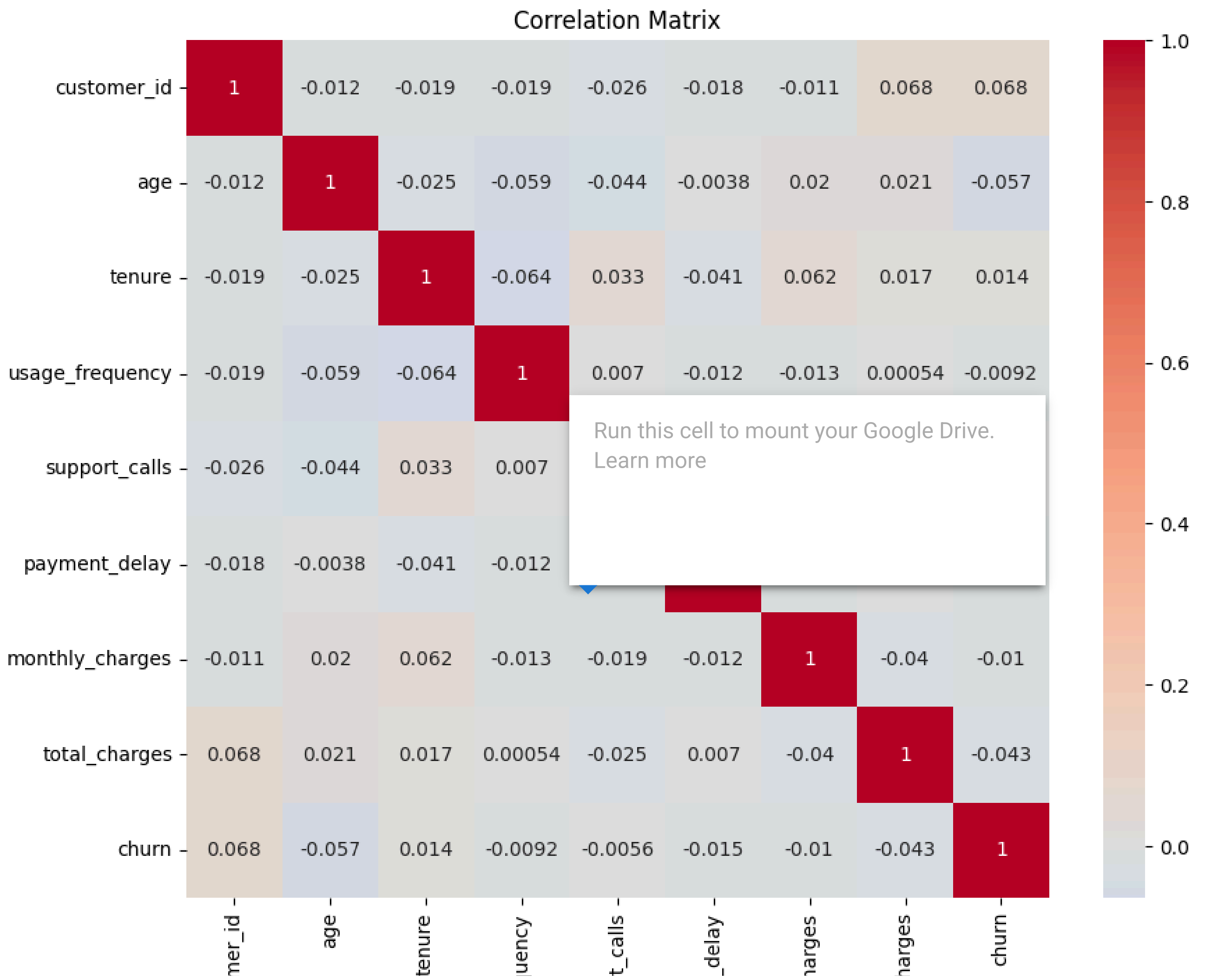
Run this cell to mount your Google Drive.
Learn more

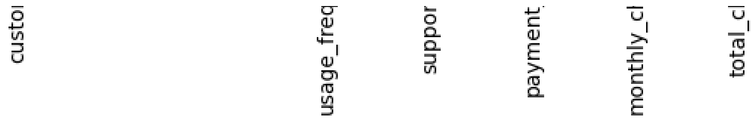
```
Loading data...

Exploring data...
Dataset Overview:
   customer_id  age  gender  tenure  usage_frequency  support_calls  \
0            0   56    Male      40               26              3
1            1   69    Male      45               14              1
2            2   46    Male      62               57              2
3            3   32  Female      58               37              5
4            4   60    Male      67               58              4

   payment_delay subscription_type  monthly_charges  total_charges  churn
0             21           Premium            70.37         287.99      1
1             19             Basic            89.08        2443.69      0
2             29           Premium            30.58         122.50      0
3              5           Premium            98.21        4609.66      0
4              2           Premium            66.61        4881.63      0

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        1000 non-null   int64
 1   age                1000 non-null   int64
 2   gender             1000 non-null   object
 3   tenure             1000 non-null   int64
 4   usage_frequency    1000 non-null   int64
 5   support_calls      1000 non-null   int64
 6   payment_delay      1000 non-null   int64
 7   subscription_type  1000 non-null   object
 8   monthly_charges    1000 non-null   float64
 9   total_charges      1000 non-null   float64
 10  churn              1000 non-null   int64
dtypes: float64(2), int64(7), object(2)
memory usage: 86.1+ KB
None

Descriptive Statistics:
       customer_id         age       tenure  usage_frequency  support_calls  \
count  1000.000000  1000.00000  1000.000000      1000.000000    1000.000000
mean    499.500000    43.81900    34.627[illegible]00
std     288.819436    14.99103    20.334[illegible]99
min       0.000000    18.00000     1.000[illegible]00
25%     249.750000    31.00000    17.000[illegible]00
50%     499.500000    44.00000    33.500[illegible]00
75%     749.250000    56.00000    52.000[illegible]00
max     999.000000    69.00000    71.000000        99.000000       9.000000

       payment_delay  monthly_charges  total_charges        churn
count    1000.000000      1000.000000    1000.000000  1000.000000
mean       15.039000        59.841710    2499.624500     0.323000
std         8.409768        22.818466    1440.588333     0.467857
min         0.000000        20.130000      53.110000     0.000000
25%         8.000000        40.445000    1228.335000     0.000000
50%        15.000000        60.120000    2522.160000     0.000000
75%        22.000000        79.607500    3678.687500     1.000000
max        29.000000        99.920000    4999.250000     1.000000
```

Run this cell to mount your Google Drive.
Learn more

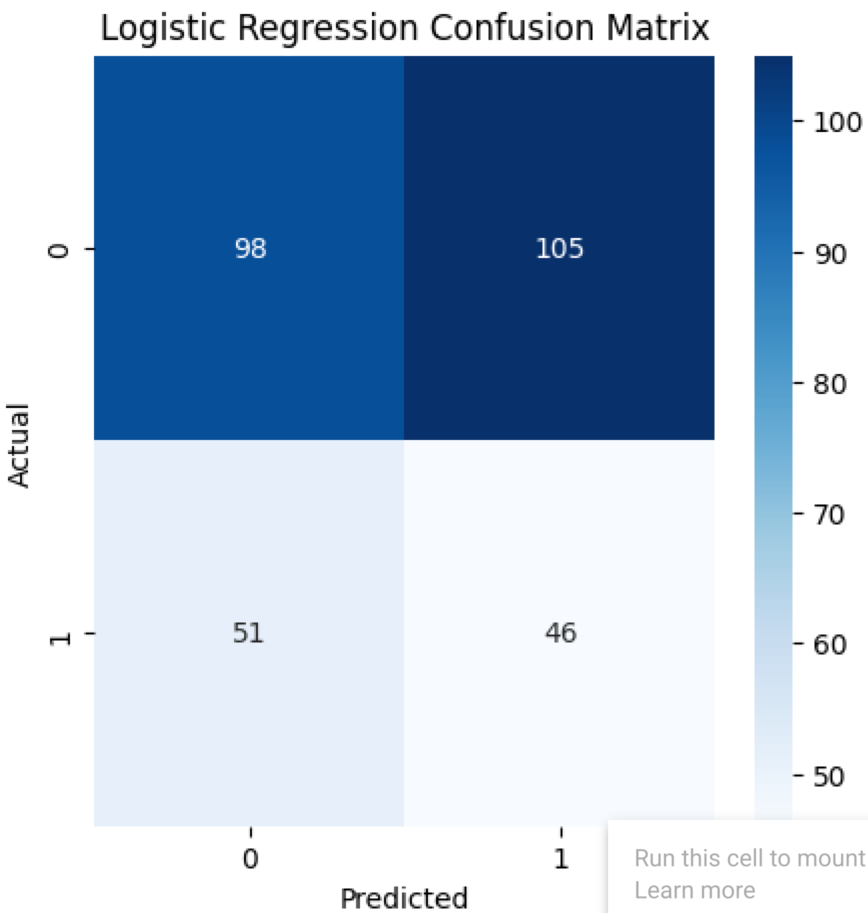
```
Class Distribution:
churn
0    677
1    323
Name: count, dtype: int64
```



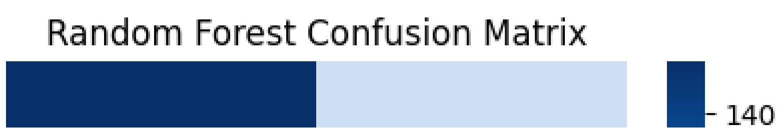


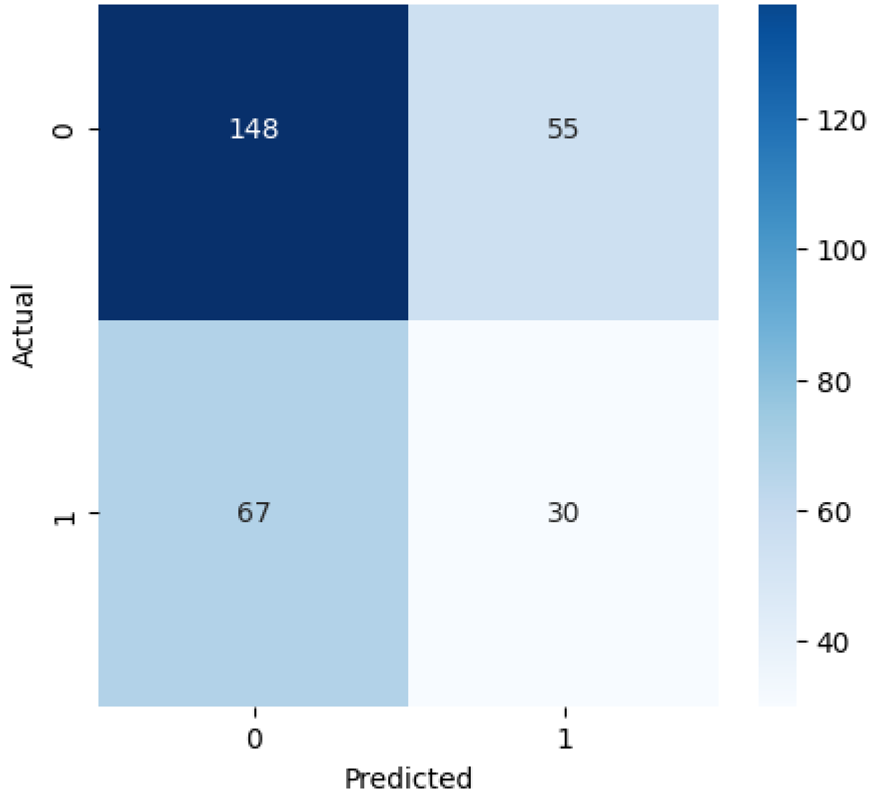



```
Preprocessing data...

Training and evaluating models...

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.48      0.56       203
           1       0.30      0.47      0.37        97

    accuracy                           0.48       300
   macro avg       0.48      0.48      0.46       300
weighted avg       0.54      0.48      0.50       300
```



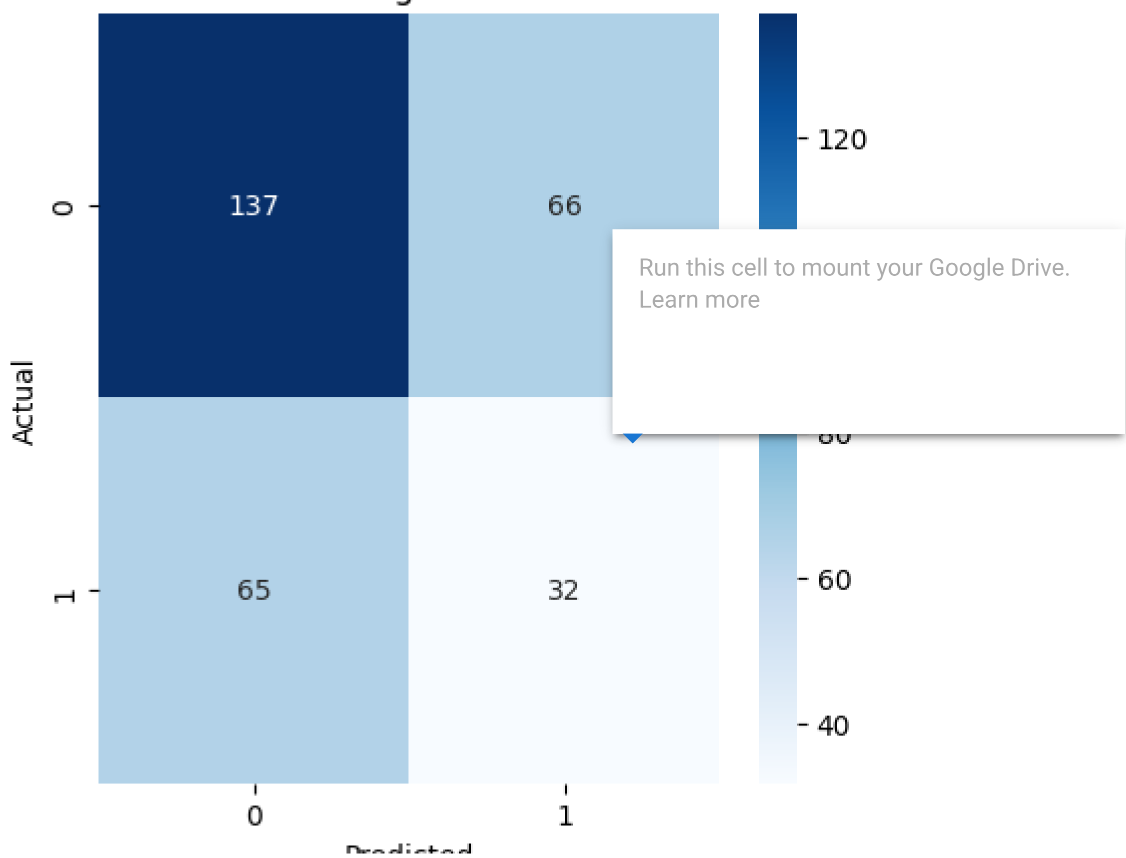

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.73      0.71       203
           1       0.35      0.31      0.33        97

    accuracy                           0.59       300
   macro avg       0.52      0.52      0.52       300
weighted avg       0.58      0.59      0.59       300
```



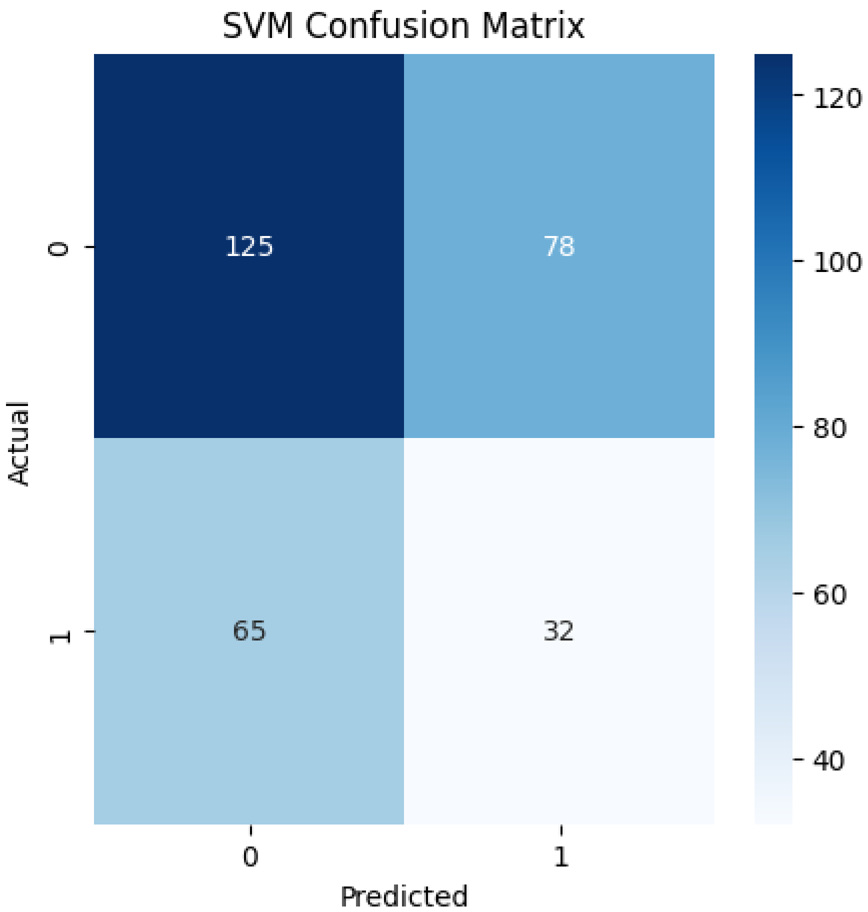



```
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.67      0.68       203
           1       0.33      0.33      0.33        97

    accuracy                           0.56       300
   macro avg       0.50      0.50      0.50       300
weighted avg       0.56      0.56      0.56       300
```

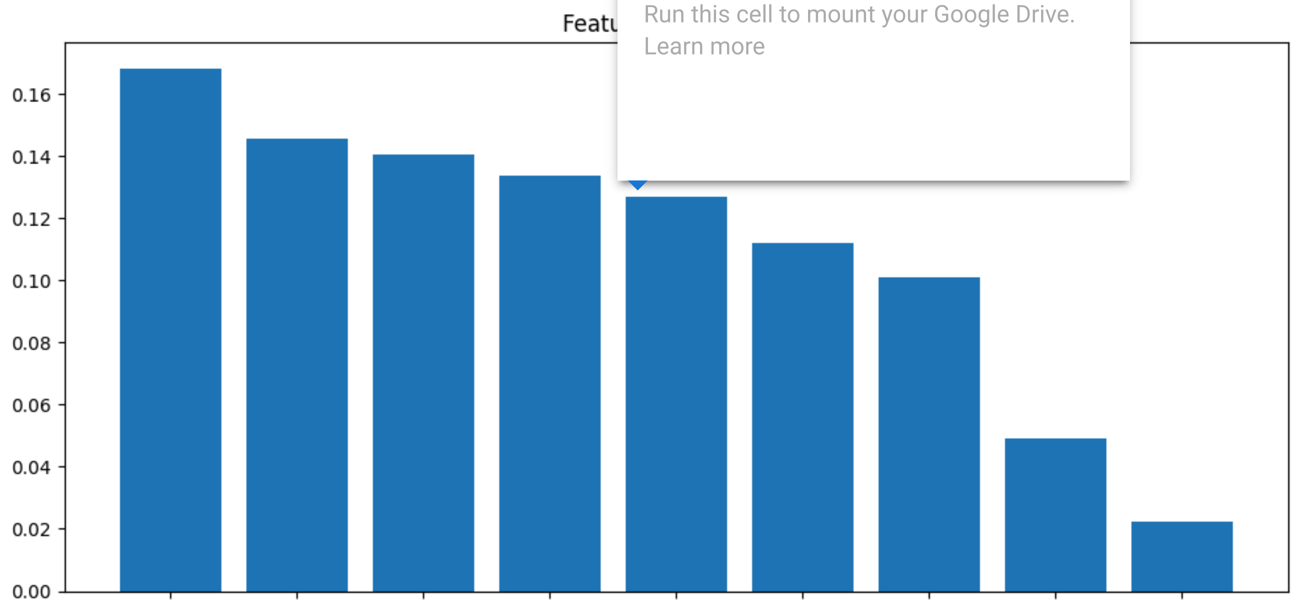
Gradient Boosting Confusion Matrix



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

           0       0.66      0.62      0.64       203
           1       0.29      0.33      0.31        97

    accuracy                           0.52       300
   macro avg       0.47      0.47      0.47       300
weighted avg       0.54      0.52      0.53       300
```

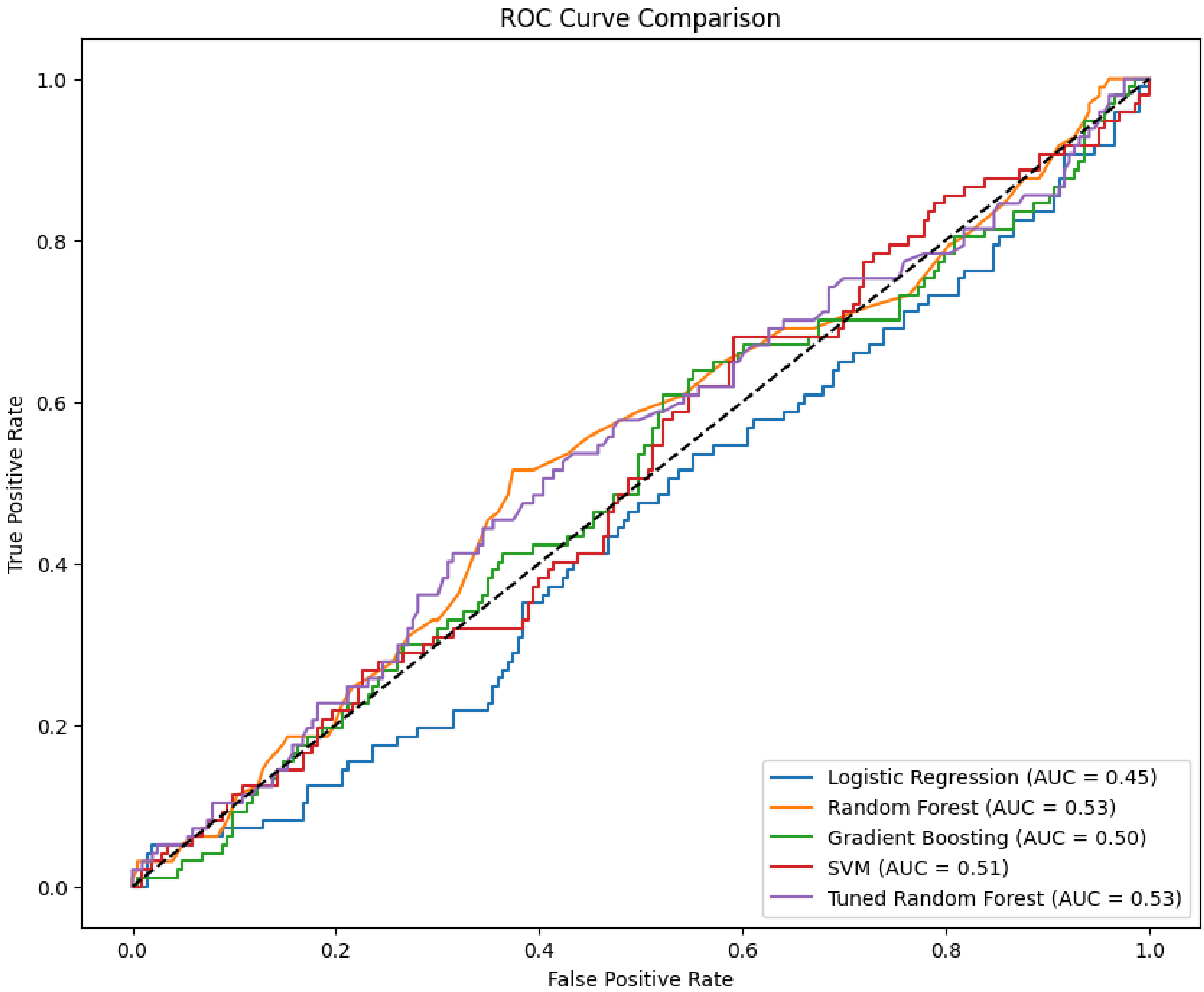



```
Best model: Random Forest with ROC AUC: 0.5332

Analyzing feature importance...
```



Run this cell to mount your Google Drive.
Learn more


```
Performing hyperparameter tuning...
Best parameters found:  {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split':
Best ROC AUC score:  0.8234952554959627

Tuned model ROC AUC: 0.5327
```



Run this cell to mount your Google Drive.
Learn more