Business Problem:

Develop a model to predict customer churn for a subscription based service or business. Use historical customer data, including features like usage behavior and customer demographics, and try algorithms like Logistic Regression, Random Forests, or Gradient Boosting to predict churn.

Importing Necessary Libraries

```
import pandas as pd
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.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, roc_auc_score, precision_score, recall_score, f1_score, roc_curve, auc, confusion_matrix
from imblearn.over_sampling import SMOTE
```

Load the dataset

```
# Load the dataset
data = pd.read_csv('Churn_Modelling.csv')
```

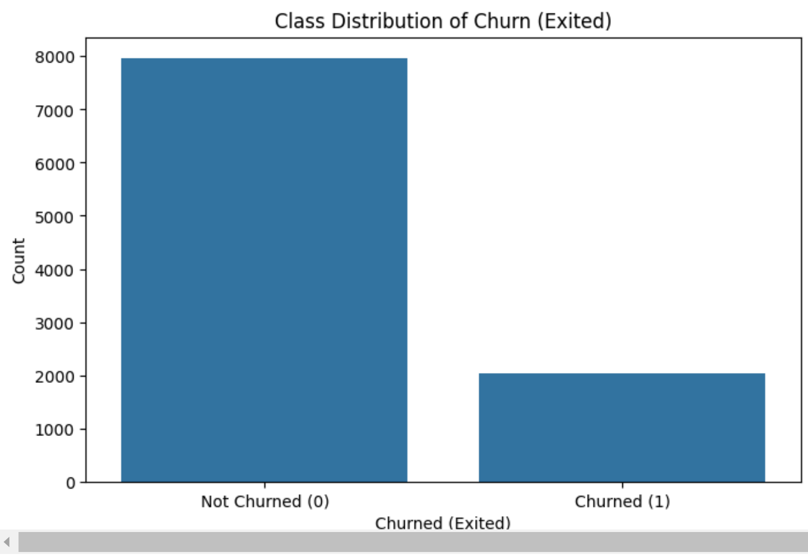
```
# Drop unnecessary columns
data = data.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
```

```
# Encode categorical variables
label_encoder = LabelEncoder()
data['Geography'] = label_encoder.fit_transform(data['Geography'])
data['Gender'] = label_encoder.fit_transform(data['Gender'])
```

```
# Check the distribution of the target variable
class_distribution = data['Exited'].value_counts(normalize=True)  # Using normalize=True for percentage
print("Class distribution before resampling:\n", class_distribution)
```

```
Class distribution before resampling:
 Exited
0    0.7963
1    0.2037
Name: proportion, dtype: float64
```

```
# Plotting the distribution for better visualization
plt.figure(figsize=(8, 5))
sns.countplot(x='Exited', data=data)
plt.title('Class Distribution of Churn (Exited)')
plt.xlabel('Churned (Exited)')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Not Churned (0)', 'Churned (1)'])
plt.show()
```



```
# Define features and target variable
# Separate features and target
X = data.drop(['Exited'], axis=1)  # Assuming all other columns are features
y = data['Exited']
```

```
# Create a SMOTE object
smote = SMOTE(random_state=42)
```

```
# Apply SMOTE
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
# Check the new class distribution
print("Class distribution after resampling:\n", pd.Series(y_resampled).value_counts(normalize=True))
```

```
Class distribution after resampling:
 Exited
1    0.5
0    0.5
Name: proportion, dtype: float64
```

```
# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

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

Model Training and Evaluation

Logistic Regression

```
# Initialize the model
lr = LogisticRegression()

# Train the model
lr.fit(X_train, y_train)

# Predict
y_pred_lr = lr.predict(X_test)
y_pred_proba_lr = lr.predict_proba(X_test)[:, 1]

# Evaluation
print('Logistic Regression Classification Report:')
print(classification_report(y_test, y_pred_lr))
print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_proba_lr)}\n")
```

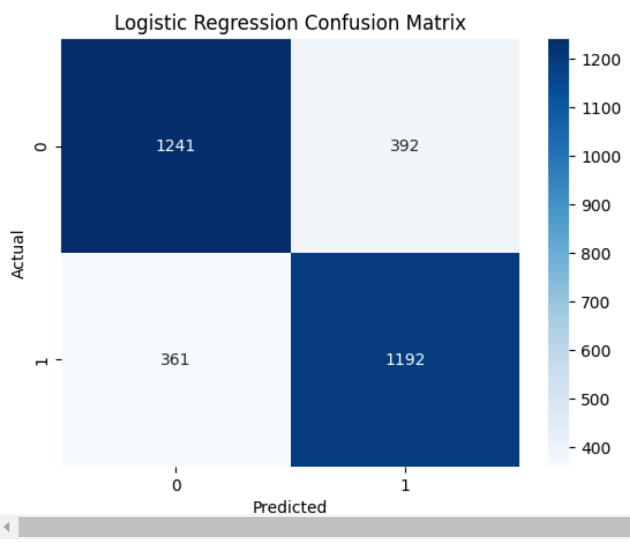
```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.76      0.77      1633
           1       0.75      0.77      0.76      1553

    accuracy                           0.76      3186
   macro avg       0.76      0.76      0.76      3186
weighted avg       0.76      0.76      0.76      3186

AUC-ROC: 0.8469737769262344
```

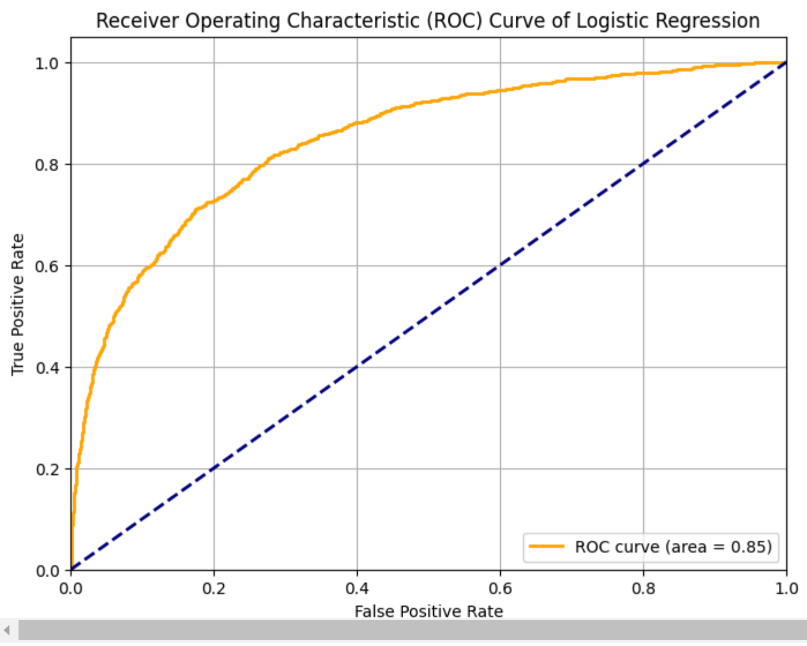
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



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

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Logistic Regression')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Random Forest Classifier

```
# Initialize the model
rf = RandomForestClassifier()

# Train the model
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)
y_pred_proba_rf = rf.predict_proba(X_test)[:, 1]

# Evaluation
print('Random Forest Classification Report:')
print(classification_report(y_test, y_pred_rf))
print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_proba_rf)}\n")
```

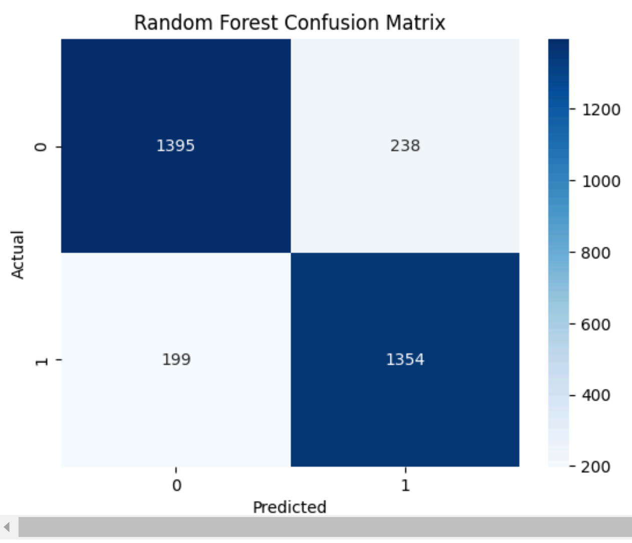
```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.85      0.86      1633
           1       0.85      0.87      0.86      1553

    accuracy                           0.86      3186
   macro avg       0.86      0.86      0.86      3186
weighted avg       0.86      0.86      0.86      3186

AUC-ROC: 0.9372305109246707
```

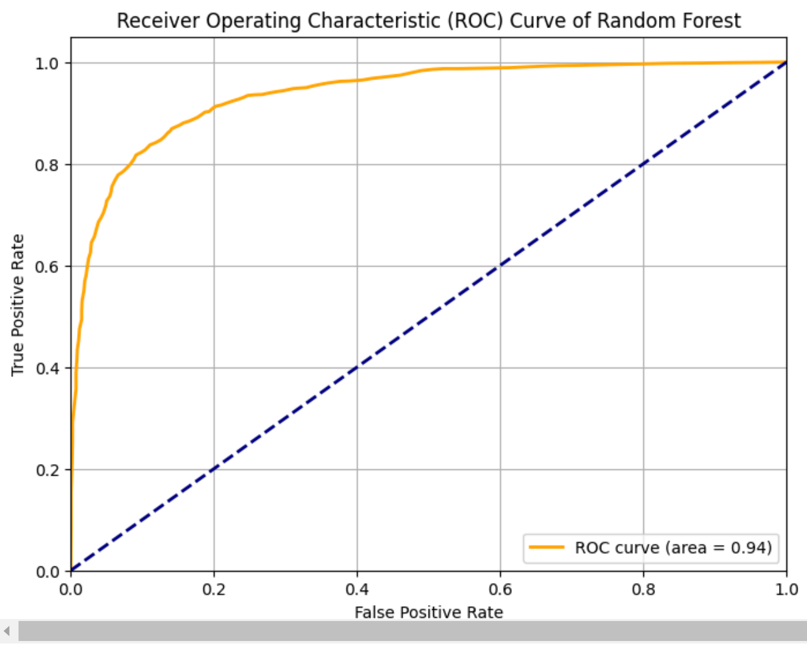
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



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

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Random Forest')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Gradient Boosting Classifier

```
gb=GradientBoostingClassifier()

# Train the model
gb.fit(X_train, y_train)

# Predict
y_pred_gb = gb.predict(X_test)
y_pred_proba_gb = gb.predict_proba(X_test)[:, 1]

# Evaluation
print('Gradient Boosting Classifier Report:')
print(classification_report(y_test, y_pred_gb))
print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_proba_gb)}\n")
```

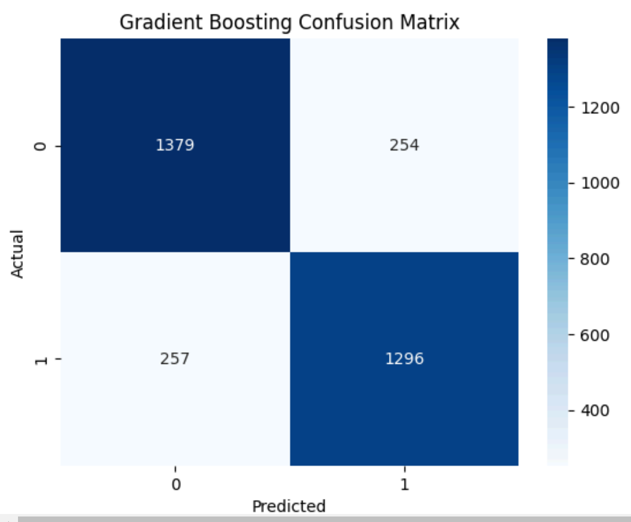
```
Gradient Boosting Classifier Report:
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      1633
           1       0.84      0.83      0.84      1553

    accuracy                           0.84      3186
   macro avg       0.84      0.84      0.84      3186
weighted avg       0.84      0.84      0.84      3186

AUC-ROC: 0.9202613987348036
```

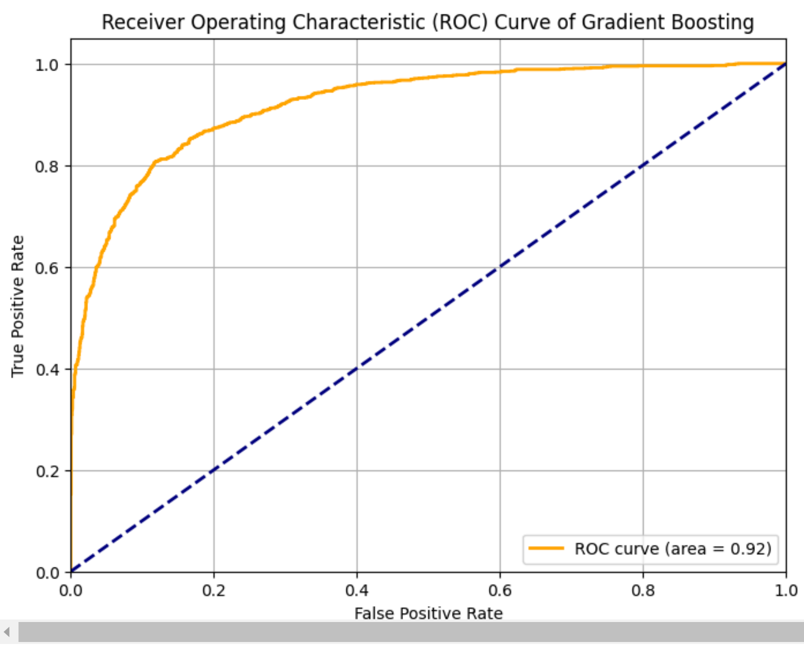
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_gb)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Gradient Boosting Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



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

# Calculate AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Gradient Boosting')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Comparing Models

```
# Create a dataframe to compare metrics
models = ['Logistic Regression', 'Random Forest', 'Gradient Boosting']
precision = [
    precision_score(y_test, y_pred_lr),
    precision_score(y_test, y_pred_rf),
    precision_score(y_test, y_pred_gb)
]
recall = [
    recall_score(y_test, y_pred_lr),
    recall_score(y_test, y_pred_rf),
    recall_score(y_test, y_pred_gb)
]
f1 = [
    f1_score(y_test, y_pred_lr),
    f1_score(y_test, y_pred_rf),
    f1_score(y_test, y_pred_gb)
]
roc_auc = [
    roc_auc_score(y_test, y_pred_proba_lr),
    roc_auc_score(y_test, y_pred_proba_rf),
    roc_auc_score(y_test, y_pred_proba_gb)
]

comparison = pd.DataFrame({
    'Model': models,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1,
    'ROC AUC': roc_auc
})

print(comparison)
```

```
                 Model  Precision    Recall  F1-Score   ROC AUC
0  Logistic Regression   0.752525  0.767547  0.759962  0.846974
1        Random Forest   0.850503  0.871861  0.861049  0.937231
2    Gradient Boosting   0.836129  0.834514  0.835321  0.920261
```

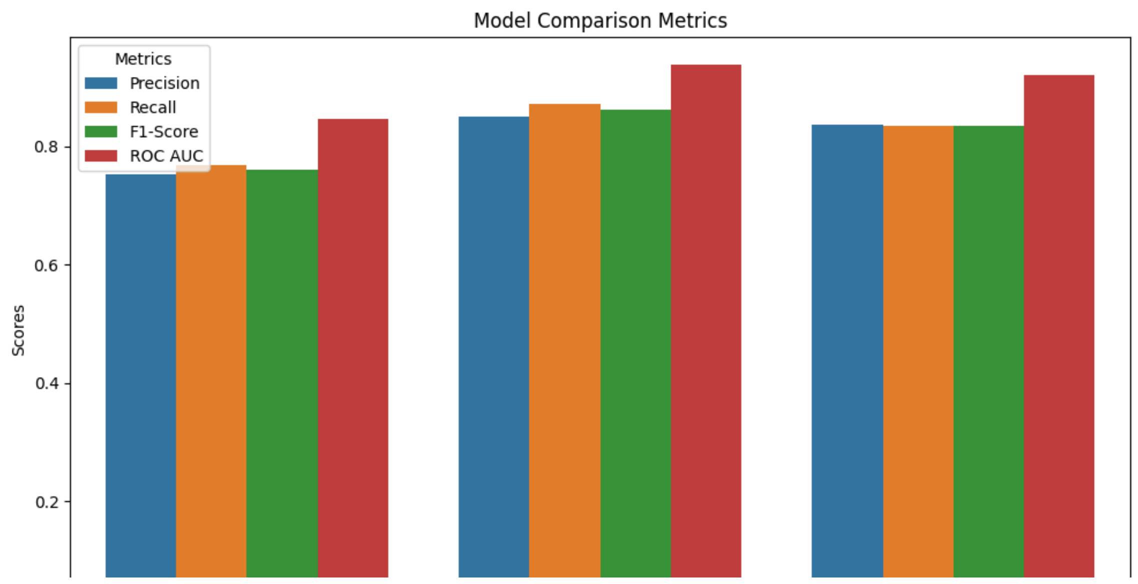
```
import seaborn as sns

# Reshape the DataFrame to long format
comparison_melted = comparison.melt(id_vars='Model', var_name='Metric', value_name='Score')

# Create the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=comparison_melted, x='Model', y='Score', hue='Metric')

# Add labels and title
plt.title('Model Comparison Metrics')
plt.ylabel('Scores')
plt.xlabel('Models')
plt.legend(title='Metrics')
plt.tight_layout()

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


Model Comparison Metrics
Metrics
Precision
Recall
F1-Score
ROC AUC
Scores
0.8
0.6
0.4
0.2