```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import VotingClassifier

# Load the training dataset
train_data = pd.read_csv('/content/Churn_TRAIN.csv')

# Display the first few rows of the training dataset
print(train_data.head())

# Load the test dataset
test_data = pd.read_csv('/content/Churn_TEST.csv')

# Display the first few rows of the test dataset
print(test_data.head())
```

```
   Call  Failure  Complains  Subscription  Length  Charge  Amount  \
0           10         0                  37              1
1            4         0                  36              0
2            2         0                   9              1
3           14         0                  22              5
4            6         0                  25              0

   Seconds of Use  Frequency of use  Frequency of SMS  \
0            6908                76               117
1           15295               182               175
2            4390                40               215
3            3238                53                48
4             395                19                 3

   Distinct Called Numbers  Age Group  Tariff Plan  Status  Age  \
0                       40          2            1       1   25
1                       32          2            1       1   25
2                       10          3            1       1   30
3                       25          3            2       1   30
4                        5          2            1       2   25

   Customer Value  Churn
0         840.780      0
1        1483.965      0
2        1037.200      0
3         323.640      0
4          32.130      0
   Call  Failure  Complains  Subscription  Length  Charge  Amount  \
0            4         0                  42              0
1            3         0                  10              2
2           13         0                  27              0
3            5         0                  42              0
4           19         0                  19              2

   Seconds of Use  Frequency of use  Frequency of SMS  \
0            2315                43               293
1            2593                35                16
2             945                28                12
3             888                17                24
4            6453               144                90

   Distinct Called Numbers  Age Group  Tariff Plan  Status  Age  \
0                       22          3            1       1   30
1                       13          3            1       1   30
2                        7          3            1       2   30
3                       10          4            1       2   45
4                       44          2            2       1   25

   Customer Value  Churn
0        1266.320      0
1         169.120      0
2          86.920      0
3          82.625      1
4         701.865      0
```

```
# Separate features and target variable in the training dataset
X_train = train_data.drop(columns=['Churn'])
y_train = train_data['Churn']

# Separate features and target variable in the test dataset
```

```
X_test = test_data.drop(columns=['Churn'])
y_test = test_data['Churn']

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



# Train a RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Train a LogisticRegression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)

# Train a SVC
svc = SVC(probability=True)
svc.fit(X_train, y_train)
```

```
▾ SVC
SVC(probability=True)
```

```
# Make predictions using the RandomForestClassifier
rf_predictions = rf.predict(X_test)

# Evaluate the RandomForestClassifier
print('Classification Report for RandomForestClassifier:')
print(classification_report(y_test, rf_predictions))
print('Confusion Matrix for RandomForestClassifier:')
print(confusion_matrix(y_test, rf_predictions))

# Make predictions using the LogisticRegression
lr_predictions = lr.predict(X_test)

# Evaluate the LogisticRegression
print('Classification Report for LogisticRegression:')
print(classification_report(y_test, lr_predictions))
print('Confusion Matrix for LogisticRegression:')
print(confusion_matrix(y_test, lr_predictions))

# Make predictions using the SVC
svc_predictions = svc.predict(X_test)

# Evaluate the SVC
print('Classification Report for SVC:')
print(classification_report(y_test, svc_predictions))
print('Confusion Matrix for SVC:')
print(confusion_matrix(y_test, svc_predictions))
```

```
Classification Report for RandomForestClassifier:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1324
           1       0.89      0.77      0.83       251

    accuracy                           0.95      1575
   macro avg       0.92      0.88      0.90      1575
weighted avg       0.95      0.95      0.95      1575

Confusion Matrix for RandomForestClassifier:
[[1299   25]
 [  57  194]]
Classification Report for LogisticRegression:
              precision    recall  f1-score   support

           0       0.90      0.97      0.94      1324
           1       0.76      0.43      0.55       251

    accuracy                           0.89      1575
   macro avg       0.83      0.70      0.74      1575
weighted avg       0.88      0.89      0.87      1575

Confusion Matrix for LogisticRegression:
[[1290   34]
 [ 143  108]]
Classification Report for SVC:
              precision    recall  f1-score   support
```

```
              0       0.92      0.99      0.95      1324
              1       0.94      0.54      0.68       251

       accuracy                           0.92      1575
      macro avg       0.93      0.77      0.82      1575
   weighted avg       0.92      0.92      0.91      1575

   Confusion Matrix for SVC:
   [[1315    9]
    [ 116  135]]
```

```
# Hyperparameter tuning for RandomForestClassifier
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='roc_auc')
grid_search.fit(X_train, y_train)
print('Best parameters for RandomForestClassifier:', grid_search.best_params_)
print('Best score for RandomForestClassifier:', grid_search.best_score_)

# Ensemble learning using VotingClassifier
voting_clf = VotingClassifier(estimators=[
    ('lr', LogisticRegression(max_iter=1000)),
    ('rf', RandomForestClassifier()),
    ('svc', SVC(probability=True))],
    voting='soft')
voting_clf.fit(X_train, y_train)
voting_predictions = voting_clf.predict(X_test)
print('Classification Report for VotingClassifier:')
print(classification_report(y_test, voting_predictions))
print('Confusion Matrix for VotingClassifier:')
print(confusion_matrix(y_test, voting_predictions))

# Handling imbalanced data using SMOTE
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

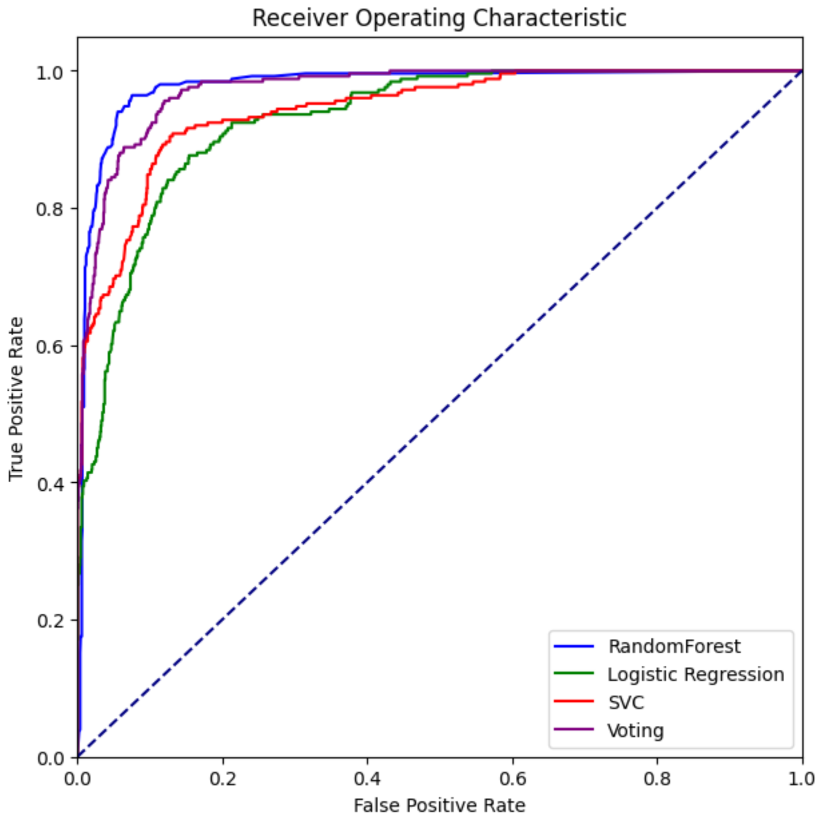
```
   Best parameters for RandomForestClassifier: {'max_depth': 30, 'min_samples_split': 5, 'n_estimators': 300}
   Best score for RandomForestClassifier: 0.980417347446427
   Classification Report for VotingClassifier:
                 precision    recall  f1-score   support

              0       0.93      0.99      0.96      1324
              1       0.90      0.62      0.73       251

       accuracy                           0.93      1575
      macro avg       0.91      0.80      0.84      1575
   weighted avg       0.93      0.93      0.92      1575

   Confusion Matrix for VotingClassifier:
   [[1306   18]
    [  96  155]]
```

```
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve, auc

# Calculate the ROC AUC scores
rf_roc_auc = roc_auc_score(y_test, rf.predict(X_test))
lr_roc_auc = roc_auc_score(y_test, lr.predict(X_test))
svc_roc_auc = roc_auc_score(y_test, svc.predict(X_test))
voting_roc_auc = roc_auc_score(y_test, voting_clf.predict(X_test))

print('ROC AUC score for RandomForestClassifier: ', rf_roc_auc)
print('ROC AUC score for LogisticRegression: ', lr_roc_auc)
print('ROC AUC score for SVC: ', svc_roc_auc)
print('ROC AUC score for VotingClassifier: ', voting_roc_auc)
```

```
   ROC AUC score for RandomForestClassifier:  0.8770130956536392
   ROC AUC score for LogisticRegression:  0.7022995630769973
   ROC AUC score for SVC:  0.7655255112480591
   ROC AUC score for VotingClassifier:  0.8019673571574727
```

```
# Plot the ROC curves
plt.figure(figsize=(7, 7))

for model, name, color in zip([rf, lr, svc, voting_clf], ['RandomForest', 'Logistic Regression', 'SVC', 'Voting'], ['blue', 'green', 'rec
    fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
    plt.plot(fpr, tpr, color=color, label=name)
```

```
plt.plot([0, 1], [0, 1], color='navy', 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')
plt.legend(loc="lower right")
plt.show()
```




```
# Cost analysis
# Please replace the values for cost_fp and cost_fn with your actual costs
cost_fp = 1  # Cost of a false positive
cost_fn = 1  # Cost of a false negative

cm_rf = confusion_matrix(y_test, rf_predictions)
cm_lr = confusion_matrix(y_test, lr_predictions)
cm_svc = confusion_matrix(y_test, svc_predictions)
cm_voting = confusion_matrix(y_test, voting_predictions)

cost_rf = cm_rf[0, 1] * cost_fp + cm_rf[1, 0] * cost_fn
cost_lr = cm_lr[0, 1] * cost_fp + cm_lr[1, 0] * cost_fn
cost_svc = cm_svc[0, 1] * cost_fp + cm_svc[1, 0] * cost_fn
cost_voting = cm_voting[0, 1] * cost_fp + cm_voting[1, 0] * cost_fn

print('Total cost for RandomForestClassifier: ', cost_rf)
print('Total cost for LogisticRegression: ', cost_lr)
print('Total cost for SVC: ', cost_svc)
print('Total cost for VotingClassifier: ', cost_voting)
```

```
Total cost for RandomForestClassifier:  82
Total cost for LogisticRegression:  177
Total cost for SVC:  125
Total cost for VotingClassifier:  114
```

```
# Import necessary libraries
from sklearn.metrics import precision_score, recall_score, f1_score, log_loss, matthews_corrcoef

# Calculate metrics for RandomForestClassifier
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)
rf_log_loss = log_loss(y_test, rf.predict_proba(X_test))
rf_matthews = matthews_corrcoef(y_test, rf_predictions)

print('RandomForestClassifier:')
```

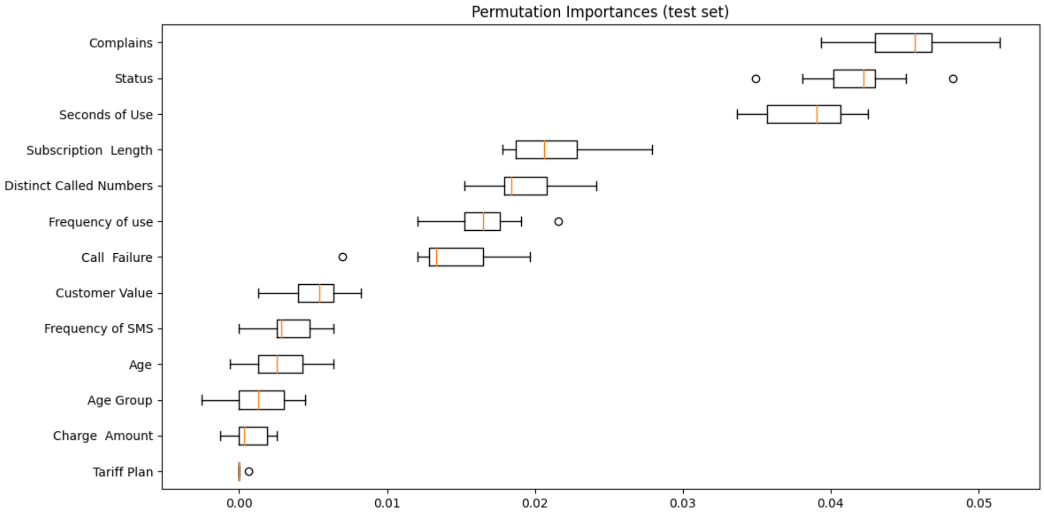
```
print('Precision: ', rf_precision)
print('Recall: ', rf_recall)
print('F1-score: ', rf_f1)
print('Log Loss: ', rf_log_loss)
print('Matthews Correlation Coefficient: ', rf_matthews)

# Calculate metrics for LogisticRegression
lr_precision = precision_score(y_test, lr_predictions)
lr_recall = recall_score(y_test, lr_predictions)
lr_f1 = f1_score(y_test, lr_predictions)
lr_log_loss = log_loss(y_test, lr.predict_proba(X_test))
lr_matthews = matthews_corrcoef(y_test, lr_predictions)

print('\nLogisticRegression:')
print('Precision: ', lr_precision)
print('Recall: ', lr_recall)
print('F1-score: ', lr_f1)
print('Log Loss: ', lr_log_loss)
print('Matthews Correlation Coefficient: ', lr_matthews)

# Calculate metrics for SVC
svc_precision = precision_score(y_test, svc_predictions)
svc_recall = recall_score(y_test, svc_predictions)
svc_f1 = f1_score(y_test, svc_predictions)
svc_log_loss = log_loss(y_test, svc.predict_proba(X_test))
svc_matthews = matthews_corrcoef(y_test, svc_predictions)

print('\nSVC:')
print('Precision: ', svc_precision)
print('Recall: ', svc_recall)
print('F1-score: ', svc_f1)
print('Log Loss: ', svc_log_loss)
print('Matthews Correlation Coefficient: ', svc_matthews)

# Calculate metrics for VotingClassifier
voting_precision = precision_score(y_test, voting_predictions)
voting_recall = recall_score(y_test, voting_predictions)
voting_f1 = f1_score(y_test, voting_predictions)
voting_log_loss = log_loss(y_test, voting_clf.predict_proba(X_test))
voting_matthews = matthews_corrcoef(y_test, voting_predictions)

print('\nVotingClassifier:')
print('Precision: ', voting_precision)
print('Recall: ', voting_recall)
print('F1-score: ', voting_f1)
print('Log Loss: ', voting_log_loss)
print('Matthews Correlation Coefficient: ', voting_matthews)
```

```
RandomForestClassifier:
Precision:  0.8858447488584474
Recall:  0.7729083665338645
F1-score:  0.825531914893617
Log Loss:  0.220508289503508
Matthews Correlation Coefficient:  0.7976555339516986

LogisticRegression:
Precision:  0.7605633802816901
Recall:  0.4302788844621514
F1-score:  0.549618320610687
Log Loss:  0.2300518912407888
Matthews Correlation Coefficient:  0.5170571241823865

SVC:
Precision:  0.9375
Recall:  0.5378486055776892
F1-score:  0.6835443037974683
Log Loss:  0.21601221478887933
Matthews Correlation Coefficient:  0.6743976367376933

VotingClassifier:
Precision:  0.8959537572254336
Recall:  0.6175298804780877
F1-score:  0.7311320754716981
Log Loss:  0.1665534415242046
Matthews Correlation Coefficient:  0.7069257293672487
```

```
# Convert numpy arrays back into dataframes
X_train = pd.DataFrame(X_train, columns=train_data.drop(columns=['Churn']).columns)
X_test = pd.DataFrame(X_test, columns=train_data.drop(columns=['Churn']).columns)
```

```
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# Calculate permutation importance for RandomForestClassifier
result = permutation_importance(rf, X_test, y_test, n_repeats=10, random_state=0, n_jobs=-1)

# Sort features by importance
sorted_idx = result.importances_mean.argsort()

# Plot
fig, ax = plt.subplots(figsize=(12, 6))
ax.boxplot(result.importances[sorted_idx].T, vert=False, labels=X_test.columns[sorted_idx])
ax.set_title("Permutation Importances (test set)")
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Ranc
  warnings.warn(
```




```
# Define cost values
cost_tp = -100  # Cost of true positive (retention cost)
cost_fp = -100  # Cost of false positive (unnecessary retention cost)
cost_tn = 0  # Cost of true negative (no cost)
cost_fn = -2300  # Cost of false negative (lost revenue + cost of acquiring a new customer)

# Calculate total cost for RandomForestClassifier
cost_rf = cm_rf[0, 1] * cost_fp + cm_rf[1, 0] * cost_fn + cm_rf[1, 1] * cost_tp + cm_rf[0, 0] * cost_tn

# Calculate total cost for LogisticRegression
cost_lr = cm_lr[0, 1] * cost_fp + cm_lr[1, 0] * cost_fn + cm_lr[1, 1] * cost_tp + cm_lr[0, 0] * cost_tn

# Calculate total cost for SVC
cost_svc = cm_svc[0, 1] * cost_fp + cm_svc[1, 0] * cost_fn + cm_svc[1, 1] * cost_tp + cm_svc[0, 0] * cost_tn

# Calculate total cost for VotingClassifier
cost_voting = cm_voting[0, 1] * cost_fp + cm_voting[1, 0] * cost_fn + cm_voting[1, 1] * cost_tp + cm_voting[0, 0] * cost_tn

print('Total cost for RandomForestClassifier: ', cost_rf)
print('Total cost for LogisticRegression: ', cost_lr)
print('Total cost for SVC: ', cost_svc)
print('Total cost for VotingClassifier: ', cost_voting)
```

```
Total cost for RandomForestClassifier:  -153000
Total cost for LogisticRegression:  -343100
Total cost for SVC:  -281200
Total cost for VotingClassifier:  -238100
```

```
# Define profit values
profit_tn = 1000  # Profit from true negative (customer correctly identified as not churning)
profit_tp = 1000  # Profit from true positive (customer correctly identified as churning and retained)

# Calculate total profit for RandomForestClassifier
profit_rf = cm_rf[0, 0] * profit_tn + cm_rf[1, 1] * profit_tp

# Calculate total profit for LogisticRegression
profit_lr = cm_lr[0, 0] * profit_tn + cm_lr[1, 1] * profit_tp

# Calculate total profit for SVC
profit_svc = cm_svc[0, 0] * profit_tn + cm_svc[1, 1] * profit_tp

# Calculate total profit for VotingClassifier
profit_voting = cm_voting[0, 0] * profit_tn + cm_voting[1, 1] * profit_tp

print('Total profit for RandomForestClassifier: ', profit_rf)
print('Total profit for LogisticRegression: ', profit_lr)
print('Total profit for SVC: ', profit_svc)
print('Total profit for VotingClassifier: ', profit_voting)
```

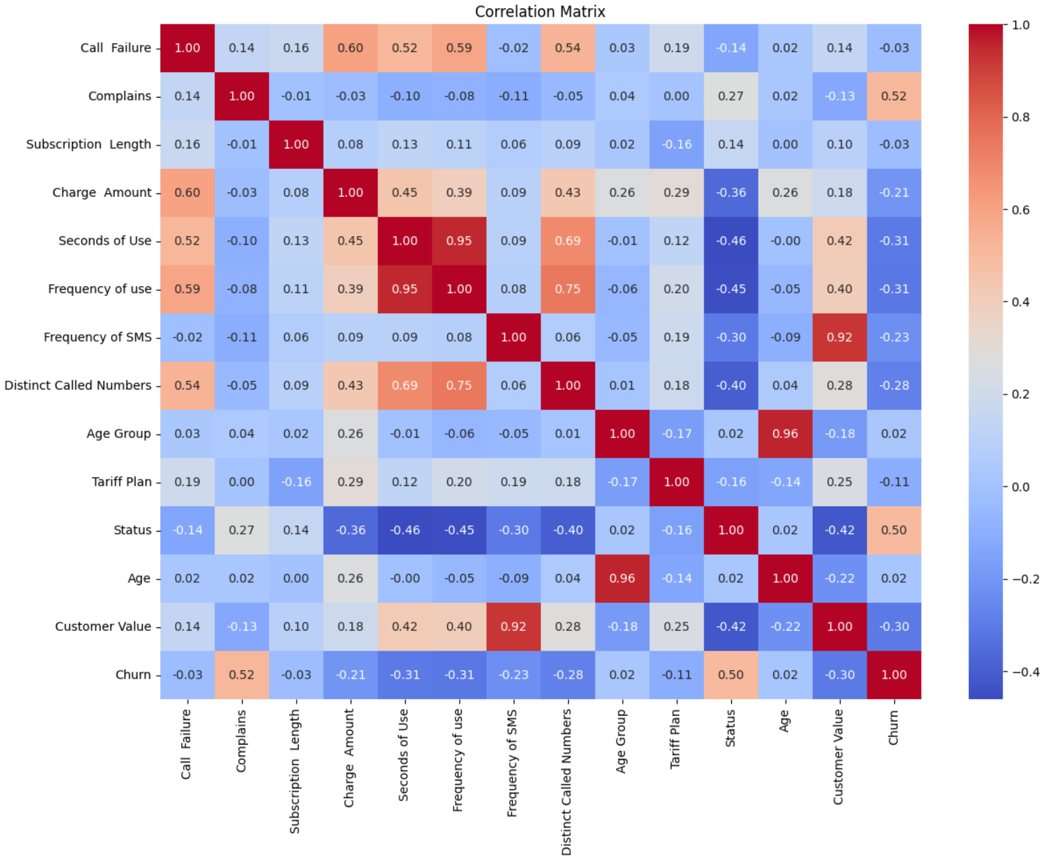
```
Total profit for RandomForestClassifier:  1493000
Total profit for LogisticRegression:  1398000
Total profit for SVC:  1450000
Total profit for VotingClassifier:  1461000
```

```
import numpy as np
```

```
from sklearn.model_selection import cross_val_score

# Perform cross-validation on RandomForestClassifier
rf_cv_scores = cross_val_score(rf, X_train, y_train, cv=5)
print('Cross-validation scores for RandomForestClassifier: ', rf_cv_scores)
print('Mean cross-validation score for RandomForestClassifier: ', np.mean(rf_cv_scores))

# Perform cross-validation on LogisticRegression
lr_cv_scores = cross_val_score(lr, X_train, y_train, cv=5)
print('\nCross-validation scores for LogisticRegression: ', lr_cv_scores)
print('Mean cross-validation score for LogisticRegression: ', np.mean(lr_cv_scores))

# Perform cross-validation on SVC
svc_cv_scores = cross_val_score(svc, X_train, y_train, cv=5)
print('\nCross-validation scores for SVC: ', svc_cv_scores)
print('Mean cross-validation score for SVC: ', np.mean(svc_cv_scores))

# Perform cross-validation on VotingClassifier
voting_cv_scores = cross_val_score(voting_clf, X_train, y_train, cv=5)
print('\nCross-validation scores for VotingClassifier: ', voting_cv_scores)
print('Mean cross-validation score for VotingClassifier: ', np.mean(voting_cv_scores))
```

```
Cross-validation scores for RandomForestClassifier:  [0.95555556 0.93650794 0.95238095 0.93333333 0.95555556]
Mean cross-validation score for RandomForestClassifier:  0.9466666666666667

Cross-validation scores for LogisticRegression:  [0.9047619  0.9015873  0.87619048 0.85714286 0.92063492]
Mean cross-validation score for LogisticRegression:  0.8920634920634921

Cross-validation scores for SVC:  [0.92063492 0.93015873 0.89206349 0.88253968 0.93333333]
Mean cross-validation score for SVC:  0.9117460317460317

Cross-validation scores for VotingClassifier:  [0.94920635 0.93015873 0.92698413 0.8984127  0.94285714]
Mean cross-validation score for VotingClassifier:  0.9295238095238094
```

```
import seaborn as sns

# Calculate correlation matrix
corr = train_data.corr()

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Correlation Matrix')
plt.show()
```



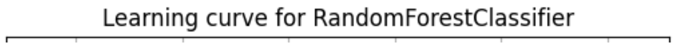

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Define the model with n_init specified
kmeans = KMeans(n_clusters=3, n_init=10, random_state=0)

# Define data for clustering (only numerical variables)
cluster_data = train_data.select_dtypes(include=[np.number])

# Scale the data
scaler = StandardScaler()
cluster_data_scaled = scaler.fit_transform(cluster_data)

# Define the model
kmeans = KMeans(n_clusters=3, random_state=0)  # we choose 3 clusters, but this could be any number

# Fit the model
kmeans.fit(cluster_data_scaled)

# Get cluster labels
train_data['cluster'] = kmeans.labels_

# Check the size of each cluster
print(train_data['cluster'].value_counts())

# Explore the mean values in each cluster
cluster_summary = train_data.groupby('cluster').mean()
print(cluster_summary)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fr
  warnings.warn(
0    878
2    363
1    334
```

```
Name: cluster, dtype: int64
         Call  Failure  Complains  Subscription  Length  Charge  Amount  \
cluster
0            5.446469   0.119590            32.082005        0.381549
1           15.209581   0.029940            33.979042        2.407186
2            6.093664   0.019284            31.016529        0.947658

         Seconds of Use  Frequency of use  Frequency of SMS  \
cluster
0           2089.438497         38.136674         13.514806
1          10603.736527        151.011976         50.538922
2           5003.107438         75.451791        241.440771

         Distinct Called Numbers  Age Group  Tariff Plan    Status        Age  \
cluster
0                      15.480638   2.769932     1.001139  1.439636  30.461276
1                      44.224551   3.245509     1.095808  1.000000  35.284431
2                      23.988981   2.490358     1.253444  1.000000  27.410468

         Customer Value     Churn
cluster
0            137.852215  0.275626
1            575.416317  0.000000
2           1222.589518  0.005510
```

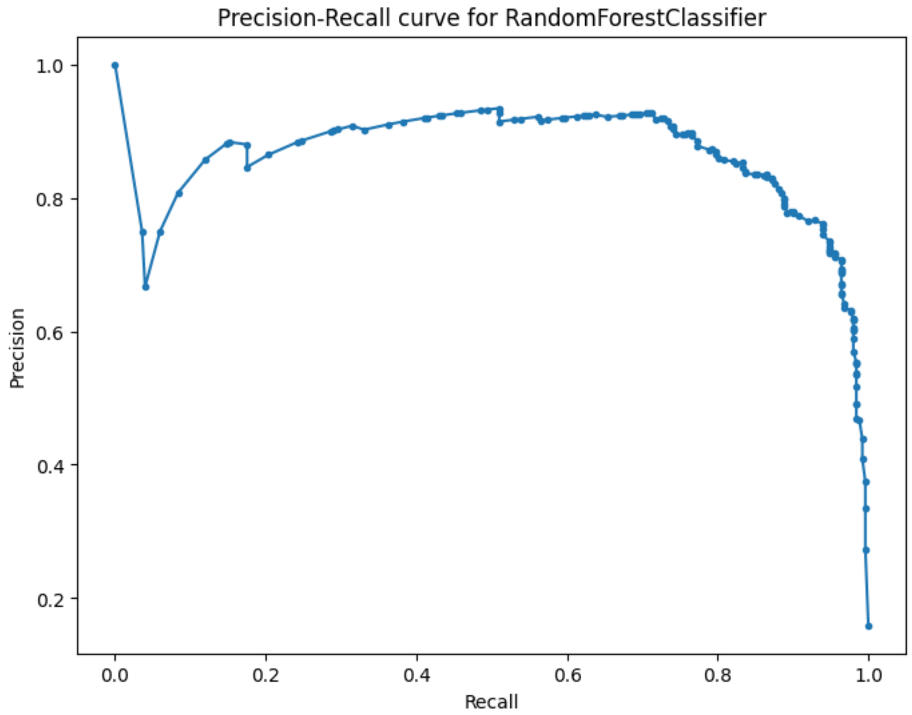
```
from sklearn.model_selection import learning_curve

# Define a function to plot the learning curves
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=None):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Generate learning curves
plot_learning_curve(rf, 'Learning curve for RandomForestClassifier', X_train, y_train, cv=5)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```




```
from sklearn.metrics import precision_recall_curve

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(y_test, rf.predict_proba(X_test)[:, 1])

# Plot the precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall curve for RandomForestClassifier')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Ranc
  warnings.warn(
```




```
# Binning the 'Age' column into three categories
train_data['Age_bin'] = pd.cut(train_data['Age'], bins=[0, 30, 60, 100], labels=['Young', 'Middle-aged', 'Senior'])
```

```
from imblearn.over_sampling import SMOTE

# Initialize SMOTE
smote = SMOTE()

# Fit SMOTE to the data
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
import xgboost as xgb

# Initialize XGBClassifier
xgb_clf = xgb.XGBClassifier()

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

# Make predictions
y_pred_xgb = xgb_clf.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1324
```

```
           1       0.88      0.78      0.83       251

    accuracy                           0.95      1575
   macro avg       0.92      0.88      0.90      1575
weighted avg       0.95      0.95      0.95      1575
```

```
# Make predictions using multiple models
y_pred_rf = rf.predict_proba(X_test)[:, 1]
y_pred_lr = lr.predict_proba(X_test)[:, 1]
y_pred_xgb = xgb_clf.predict_proba(X_test)[:, 1]

# Average the predictions
y_pred_ensemble = (y_pred_rf + y_pred_lr + y_pred_xgb) / 3

# Convert probabilities to class labels
y_pred_ensemble = [1 if prob > 0.5 else 0 for prob in y_pred_ensemble]

# Evaluate the ensemble
print(classification_report(y_test, y_pred_ensemble))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fitte
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted wi
  warnings.warn(
              precision    recall  f1-score   support

           0       0.95      0.98      0.97      1324
           1       0.88      0.75      0.81       251

    accuracy                           0.94      1575
   macro avg       0.92      0.87      0.89      1575
weighted avg       0.94      0.94      0.94      1575
```

```
import joblib
joblib.dump(rf, 'model.joblib')
```

```
['model.joblib']
```

```
!pip install shap
```

```
Collecting shap
  Downloading shap-0.42.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (54
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 547.1/547.1 kB 14.2 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.10.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.1)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from numba->shap) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2022.7.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap) (1.1
Installing collected packages: slicer, shap
Successfully installed shap-0.42.0 slicer-0.0.7
```

```
from sklearn.ensemble import RandomForestClassifier

# Define your RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to your training data
# (Substitute X_train and y_train with your actual training data)
rf.fit(Churn_TRAIN, Churn_TEST)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-5c1ce8ec3bbf> in <cell line: 8>()
      6 # Fit the classifier to your training data
      7 # (Substitute X_train and y_train with your actual training data)
----> 8 rf.fit(Churn_TRAIN, Churn_TEST)

NameError: name 'Churn_TRAIN' is not defined
```

SEARCH STACK OVERFLOW

0s completed at 01:03