```
import pandas as pd
import numpy as np
import os
import sys
import json
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = pd.read_csv("customer_shopping_data.csv")
```

Developing a predictive model, the goal could be to classify gender based on the features in your dataset.

```
data["invoice_date"] = pd.to_datetime(data["invoice_date"],
format="%d/%m/%Y")

# Extract year and month for grouping
data['year'] = data['invoice_date'].dt.year
data['month'] = data['invoice_date'].dt.month
data['total_spent'] = data['quantity'] * data['price']

from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(dtype=int)
data[['gender', 'category', 'payment_method', 'shopping_mall']] =
encoder.fit_transform(data[['gender', 'category', 'payment_method',
'shopping_mall']])

# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 2: Separate features (x) and target variable (y)
y = data['gender']  # Target variable
x =
data.drop(columns=['gender','invoice_no','customer_id','invoice_date']
)  # Features
```

```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.30, random_state=42)

# Verifying the split
print("Training data shape (features):", x_train.shape)
print("Testing data shape (features):", x_test.shape)
print("Training data shape (target):", y_train.shape)
print("Testing data shape (target):", y_test.shape)
```

```
Training data shape (features): (69619, 9)
Testing data shape (features): (29838, 9)
Training data shape (target): (69619,)
Testing data shape (target): (29838,)
```

```
def metric_score (clf, x_train,x_test,y_train, y_test, train=True):
    if train:
        y_pred = clf.predict(x_train)
        print("\n================Train Result====================")
        print (f"Accuracy Score: {accuracy_score (y_train, y_pred) *
100:.2f}%")
    elif train==False:
        pred = clf.predict(x_test)
        print("\n=========== Test Result=========================")
        print(f"Accuracy Score: {accuracy_score (y_test, pred) *
100:.2f}%")
        print ('\n \n Test Classification Report \n',
classification_report (y_test, pred, digits=2))
```

```
# 1. Logistic Regression
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred_logreg = logreg.predict(x_test)
```

```
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```python
from sklearn.metrics import classification_report, confusion_matrix

# Generate classification metrics
report = classification_report(y_test, y_pred_logreg)

metric_score (logreg,x_train,x_test,y_train,y_test, train=True)
metric_score (logreg,x_train,x_test,y_train,y_test,train=False)

# Confusion Matrix for reference
cm = confusion_matrix(y_test, y_pred_logreg)
print("\nConfusion Matrix:\n")
print(cm)
```

```
================Train Result====================
Accuracy Score: 60.02%

=========== Test Result=========================
Accuracy Score: 59.31%


 Test Classification Report
               precision    recall  f1-score   support

           0       0.59      1.00      0.74     17698
           1       0.00      0.00      0.00     12140

    accuracy                           0.59     29838
   macro avg       0.30      0.50      0.37     29838
weighted avg       0.35      0.59      0.44     29838


Confusion Matrix:

[[17698     0]
 [12140     0]]

C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
```

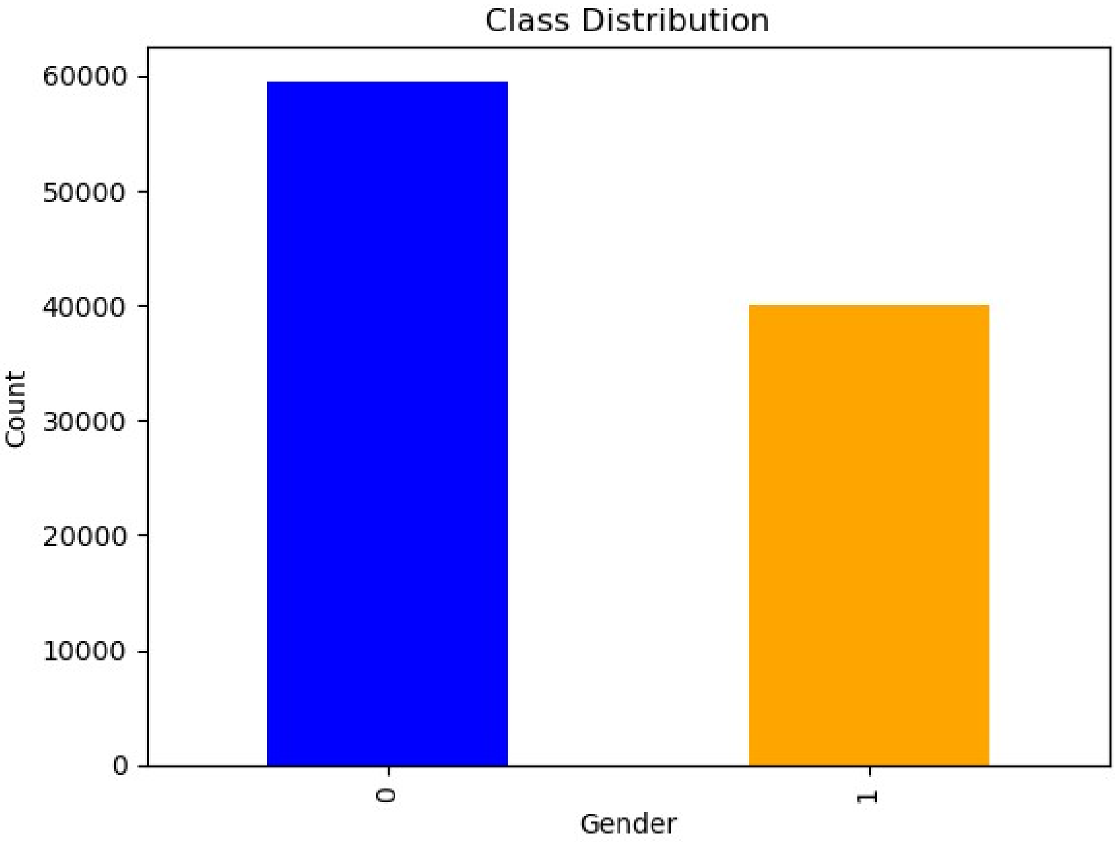
```
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\patil\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

- The classifier did not predict any positive cases, as there are no true positives or false positives. This suggests that the model is biased towards predicting the negative class.

- The absence of positive predictions could be due to imbalanced data, so Using SMOTE (Synthetic Minority Oversampling Technique) is a solid step to address class imbalance. It generates synthetic samples for the minority class to balance the dataset.

```python
import matplotlib.pyplot as plt

# Visualizing class distribution
class_counts = data['gender'].value_counts()
print("Class Distribution:")
print(class_counts)

# Plotting the distribution
class_counts.plot(kind='bar', color=['blue', 'orange'])
plt.title('Class Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

```
Class Distribution:
gender
0    59482
1    39975
Name: count, dtype: int64
```



```
from imblearn.over_sampling import SMOTE

# Apply SMOTE
smote = SMOTE(random_state=42)
x_resampled, y_resampled = smote.fit_resample(x, y)

# Verify new class distribution
print(pd.Series(y_resampled).value_counts())

gender
0    59482
1    59482
Name: count, dtype: int64

logreg1 = LogisticRegression(class_weight='balanced', max_iter=1000,
random_state=42)
logreg1.fit(x_train, y_train)
y_pred_logreg1 = logreg1.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix

# Generate classification metrics
report = classification_report(y_test, y_pred_logreg1)
```

```
metric_score (logreg1,x_train,x_test,y_train,y_test, train=True)
metric_score (logreg1,x_train,x_test,y_train,y_test,train=False)

# Confusion Matrix for reference
cm = confusion_matrix(y_test, y_pred_logreg1)
print("\nConfusion Matrix:\n")
print(cm)


================Train Result====================
Accuracy Score: 49.02%

=========== Test Result========================
Accuracy Score: 48.39%


 Test Classification Report
               precision    recall  f1-score   support

           0       0.59      0.43      0.50     17698
           1       0.40      0.56      0.47     12140

    accuracy                           0.48     29838
   macro avg       0.50      0.50      0.48     29838
weighted avg       0.51      0.48      0.49     29838


Confusion Matrix:

[[ 7597 10101]
 [ 5298  6842]]

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define the model
dtree = DecisionTreeClassifier()

# Define the parameter grid
#parameter tuning
grid_param={'criterion':['gini','entropy'],
           'max_depth':range(4,6),
           'max_leaf_nodes':range(10,12),
           'min_samples_leaf':range(6,8),
           'min_samples_split':range(21,26)}

# Use GridSearchCV to find the best combination of hyperparameters
grid_search_best = GridSearchCV(estimator=dtree,
param_grid=grid_param, cv=5)

# Fit the grid search to the training data
```

```python
grid_search_best.fit(x_train, y_train)

best=grid_search_best.best_params_

print(best)
```

```
{'criterion': 'entropy', 'max_depth': 5, 'max_leaf_nodes': 11,
'min_samples_leaf': 6, 'min_samples_split': 23}
```

```python
Dtree= DecisionTreeClassifier(criterion='entropy',
max_depth=5,max_leaf_nodes=11,min_samples_leaf=6,min_samples_split=23)
Dtree.fit(x_train, y_train)
y_pred_Dtree = Dtree.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix

# Generate classification metrics
report = classification_report(y_test, y_pred_Dtree)

metric_score (Dtree,x_train,x_test,y_train,y_test, train=True)
metric_score (Dtree,x_train,x_test,y_train,y_test,train=False)

# Confusion Matrix for reference
cm = confusion_matrix(y_test, y_pred_Dtree)
print("\nConfusion Matrix:\n")
print(cm)
```

```
================Train Result====================
Accuracy Score: 60.06%

=========== Test Result=========================
Accuracy Score: 59.31%


 Test Classification Report
               precision    recall  f1-score   support

           0       0.59      1.00      0.74     17698
           1       0.49      0.00      0.00     12140

    accuracy                           0.59     29838
   macro avg       0.54      0.50      0.37     29838
weighted avg       0.55      0.59      0.44     29838


Confusion Matrix:

[[17679    19]
 [12122    18]]
```

```python
# 3. Random Forest Classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(x_train, y_train)
y_pred_rf = rf.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix

# Generate classification metrics
report = classification_report(y_test, y_pred_rf)

metric_score (rf,x_train,x_test,y_train,y_test, train=True)
metric_score (rf,x_train,x_test,y_train,y_test,train=False)

# Confusion Matrix for reference
cm = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix:\n")
print(cm)
```

```
================Train Result====================
Accuracy Score: 97.84%

=========== Test Result=========================
Accuracy Score: 54.62%


 Test Classification Report
               precision    recall  f1-score   support

           0       0.60      0.71      0.65     17698
           1       0.42      0.30      0.35     12140

    accuracy                           0.55     29838
   macro avg       0.51      0.51      0.50     29838
weighted avg       0.53      0.55      0.53     29838


Confusion Matrix:

[[12629  5069]
 [ 8472  3668]]
```

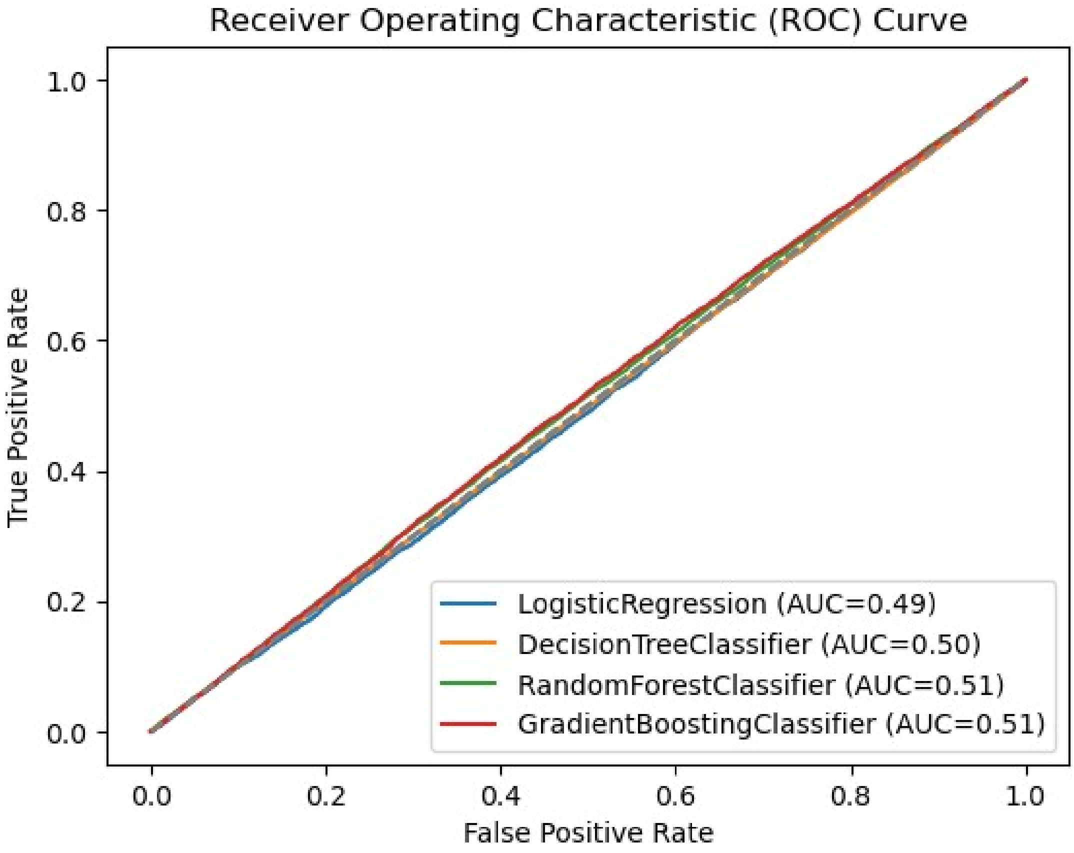
```python
# 4. Gradient Boosting Classifier
gb = GradientBoostingClassifier(random_state=42)
gb.fit(x_train, y_train)
y_pred_gb = gb.predict(x_test)

from sklearn.metrics import classification_report, confusion_matrix

# Generate classification metrics
report = classification_report(y_test, y_pred_gb)
```

```
metric_score (gb,x_train,x_test,y_train,y_test, train=True)
metric_score (gb,x_train,x_test,y_train,y_test,train=False)

# Confusion Matrix for reference
cm = confusion_matrix(y_test, y_pred_gb)
print("\nConfusion Matrix:\n")
print(cm)
```

```
================Train Result====================
Accuracy Score: 60.05%

=========== Test Result=========================
Accuracy Score: 59.30%


 Test Classification Report
              precision    recall  f1-score   support

           0       0.59      1.00      0.74     17698
           1       0.35      0.00      0.00     12140

    accuracy                           0.59     29838
   macro avg       0.47      0.50      0.37     29838
weighted avg       0.50      0.59      0.44     29838


Confusion Matrix:

[[17687    11]
 [12134     6]]
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from ipywidgets import interact
models = {
    'LogisticRegression': logreg1,
    'DecisionTreeClassifier': Dtree,
    'RandomForestClassifier': rf,
    'GradientBoostingClassifier': gb
}

for name, model in models.items():
    # Fit the model
    model.fit(x_train, y_train)

    # Get predicted probabilities for the positive class
    y_pred = model.predict_proba(x_test)[:, 1]
    # Calculate ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)

    # Print thresholds for each model
    print('Thresholds for', name, ':', thresholds)

    # Calculate AUC
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.plot(fpr, tpr, label='{} (AUC={:.2f})'.format(name, roc_auc))

    plt.plot([0, 1], [0, 1], linestyle='--', color='grey' )

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title('Receiver Operating Characteristic (ROC) Curve')

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

```
Thresholds for LogisticRegression : [       inf 0.50912682
0.50886621 ... 0.48743466 0.48743308 0.48692559]
Thresholds for DecisionTreeClassifier : [       inf 0.66666667 0.6375
0.4964539  0.48809524 0.42168675
 0.40194702 0.40151515 0.38672427 0.37027708 0.34011628 0.        ]
Thresholds for RandomForestClassifier : [       inf 0.98
0.97666667 ... 0.01375    0.01       0.        ]
Thresholds for GradientBoostingClassifier : [       inf 0.62492711
0.56438216 ... 0.294686   0.293735   0.27026518]
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
LogisticRegression (AUC=0.49)
DecisionTreeClassifier (AUC=0.50)
RandomForestClassifier (AUC=0.51)
GradientBoostingClassifier (AUC=0.51)