Business Problem

This is a dataset from one bank in the United States.

Besides usual services, this bank also provides car insurance services.

The bank organizes regular campaigns to attract new clients.

The bank has potential customers' data, and bank's employees call them for advertising available car insurance options.

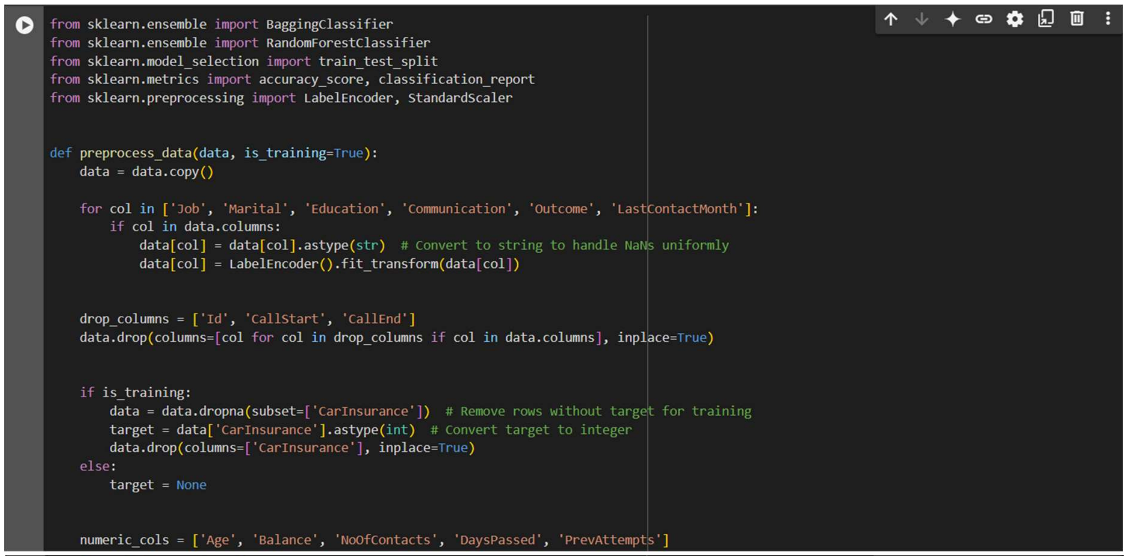
We are provided with general information about clients (age, job, etc.) as well as more specific information about the current insurance sell campaign (communication, last contact day) and previous campaigns (attributes like previous attempts, outcome).

You have data about 4000 customers who were contacted during the last campaign and for whom the results of campaign (did the customer buy insurance or not) are known.

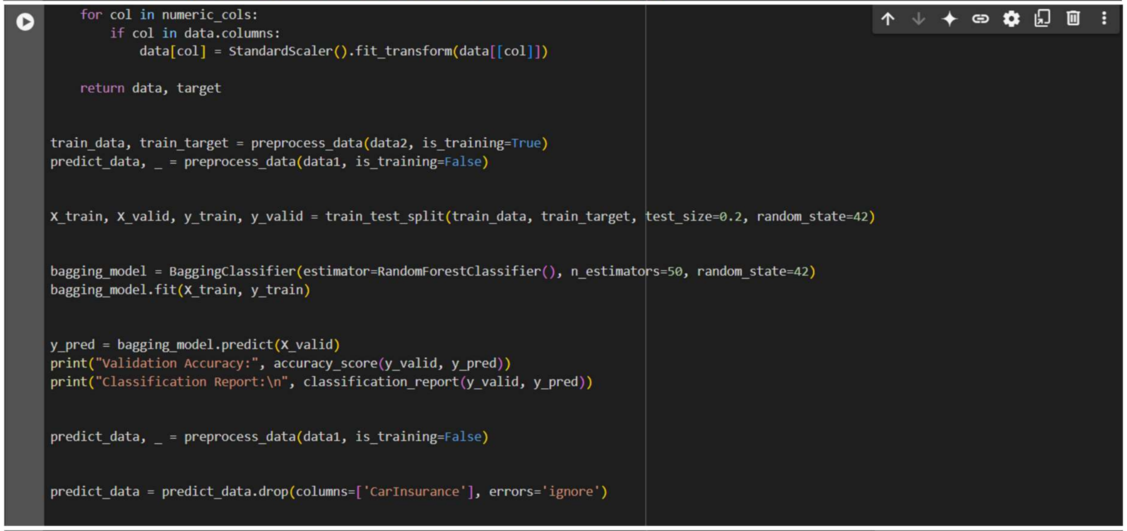
Task and Approach:

The task is to predict for 1000 customers who were contacted during the current campaign, whether they will buy car insurance or not.

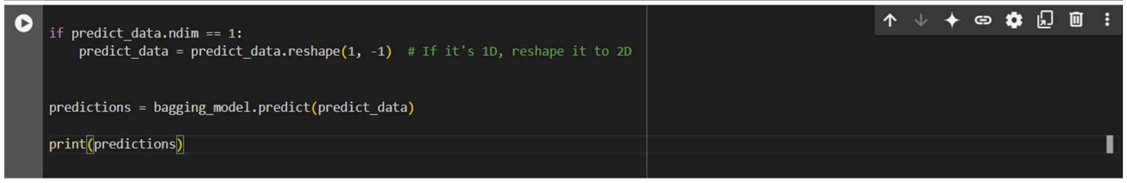
We will be using **Bagging techniques ** to predict it



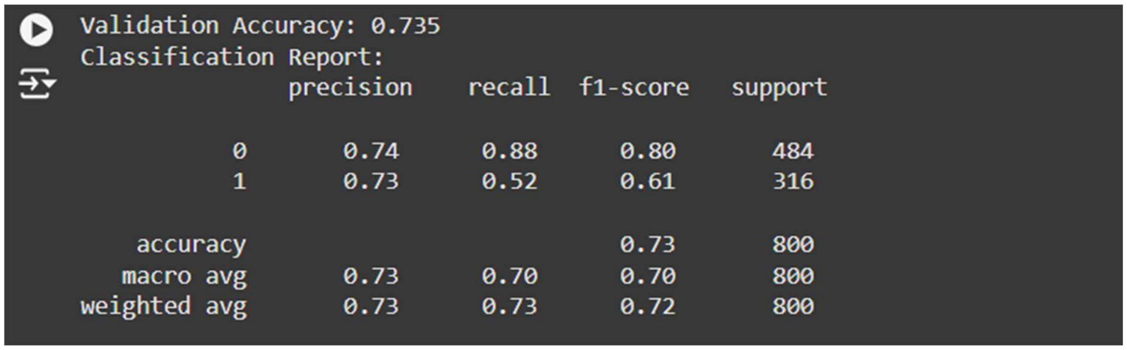
```python
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler


def preprocess_data(data, is_training=True):
    data = data.copy()

    for col in ['Job', 'Marital', 'Education', 'Communication', 'Outcome', 'LastContactMonth']:
        if col in data.columns:
            data[col] = data[col].astype(str)  # Convert to string to handle NaNs uniformly
            data[col] = LabelEncoder().fit_transform(data[col])


    drop_columns = ['Id', 'CallStart', 'CallEnd']
    data.drop(columns=[col for col in drop_columns if col in data.columns], inplace=True)


    if is_training:
        data = data.dropna(subset=['CarInsurance'])  # Remove rows without target for training
        target = data['CarInsurance'].astype(int)  # Convert target to integer
        data.drop(columns=['CarInsurance'], inplace=True)
    else:
        target = None


    numeric_cols = ['Age', 'Balance', 'NoOfContacts', 'DaysPassed', 'PrevAttempts']
```



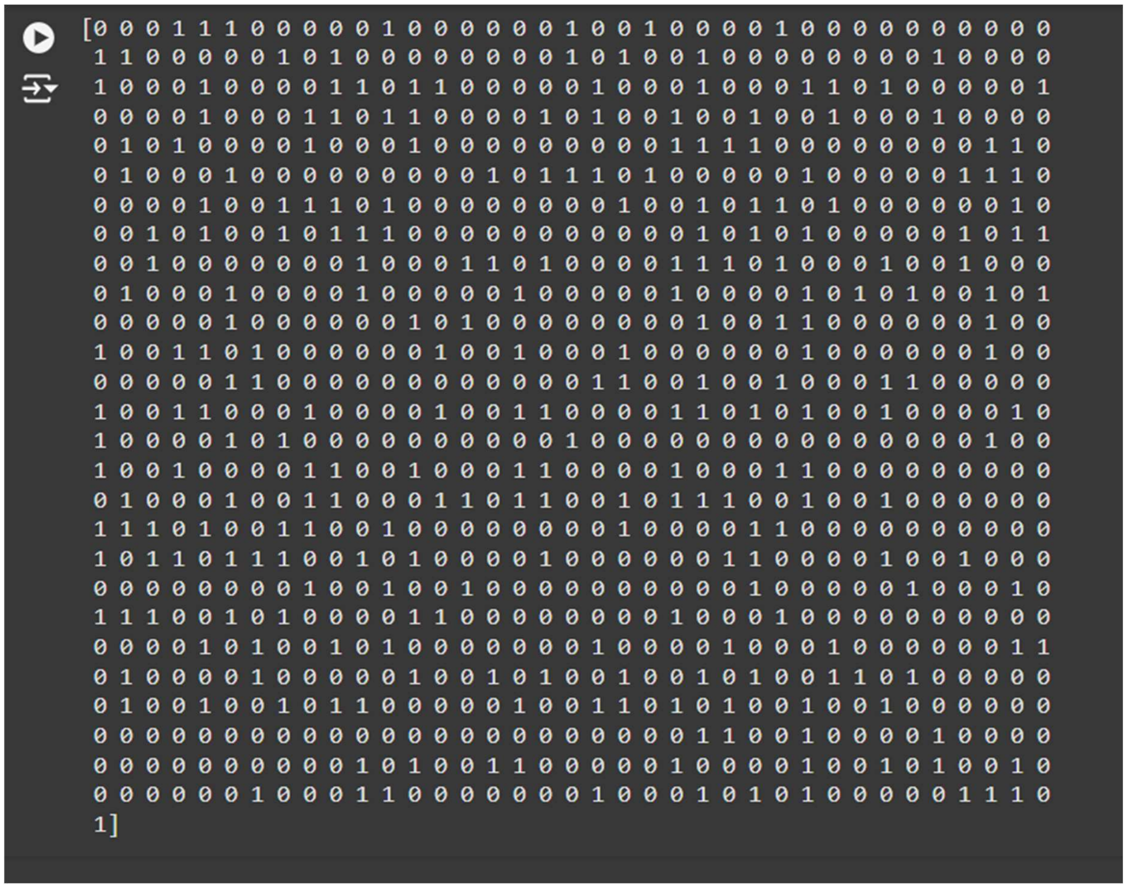
```python
    for col in numeric_cols:
        if col in data.columns:
            data[col] = StandardScaler().fit_transform(data[[col]])

    return data, target


train_data, train_target = preprocess_data(data2, is_training=True)
predict_data, _ = preprocess_data(data1, is_training=False)


X_train, X_valid, y_train, y_valid = train_test_split(train_data, train_target, test_size=0.2, random_state=42)


bagging_model = BaggingClassifier(estimator=RandomForestClassifier(), n_estimators=50, random_state=42)
bagging_model.fit(X_train, y_train)


y_pred = bagging_model.predict(X_valid)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))


predict_data, _ = preprocess_data(data1, is_training=False)


predict_data = predict_data.drop(columns=['CarInsurance'], errors='ignore')
```



```python
if predict_data.ndim == 1:
    predict_data = predict_data.reshape(1, -1)  # If it's 1D, reshape it to 2D


predictions = bagging_model.predict(predict_data)

print(predictions)
```



```
Validation Accuracy: 0.735
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.88      0.80       484
           1       0.73      0.52      0.61       316

    accuracy                           0.73       800
   macro avg       0.73      0.70      0.70       800
weighted avg       0.73      0.73      0.72       800
```



```
[0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1
 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0
 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0
 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0
 0 0 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1
 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0
 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1
 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0
 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0
 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0
 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0
 1 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1
 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 0
 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 0
 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 0
 1]
```