Quarto Document Template for R and Python

Roshani Gladson

September 29, 2024

Table of contents

Python **1**
step 1: missing values . . . 1
step 1 - Python . . . 2
step 2: necessary changes . . . 4
step 2 - Python . . . 4
step 3: selected features . . . 7
step 3 - Python . . . 7
step 4: categorical encoding . . . 7
step 4 - Python . . . 7
step 5: Split the dataset . . . 8
step 5 - Python . . . 8
step 6: first model choice . . . 8
step 6 - Python . . . 9
step 7: Second model choice . . . 12
step 7 - Python . . . 12
step 8: Models analysis . . . 16
step 9: Final Model Choice . . . 19
step 9 - Python . . . 20
Final Analysis . . . 20

Python

step 1: missing values

check for missing values in the dataset

step 1 - Python

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

# Load data
path = "C:/Users/Harri/Downloads/"
pred = pd.read_csv(path + "campaign_offer_rev-1.csv")

# Check for missing values and dataset info
print(pred.isnull().sum())
print(pred.shape)
print(pred.info())

# Impute missing values with mean
mean_imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
for col in ['Age', 'Income', 'Recency', 'AcceptedCmpOverall']:
    pred[col] = mean_imputer.fit_transform(pred[col].values.reshape(-1, 1))
```

```
CustID               0
Marital_Status       0
Education            0
Kidhome              0
Teenhome             0
AcceptedCmp3         0
AcceptedCmp4         0
AcceptedCmp5         0
AcceptedCmp1         0
AcceptedCmp2         0
Complain             0
Income             302
```

```
Age                   129
Recency               447
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumAppPurchases         0
NumStorePurchases       0
NumWebVisitsMonth       0
MntTotal                0
MntRegularProds         0
AcceptedCmpOverall    322
Response                0
dtype: int64
(2205, 29)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2205 entries, 0 to 2204
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CustID               2205 non-null   int64
 1   Marital_Status       2205 non-null   object
 2   Education            2205 non-null   object
 3   Kidhome              2205 non-null   object
 4   Teenhome             2205 non-null   object
 5   AcceptedCmp3         2205 non-null   int64
 6   AcceptedCmp4         2205 non-null   int64
 7   AcceptedCmp5         2205 non-null   int64
 8   AcceptedCmp1         2205 non-null   int64
 9   AcceptedCmp2         2205 non-null   int64
 10  Complain             2205 non-null   int64
 11  Income               1903 non-null   float64
 12  Age                  2076 non-null   float64
 13  Recency              1758 non-null   float64
 14  MntWines             2205 non-null   int64
 15  MntFruits            2205 non-null   int64
 16  MntMeatProducts      2205 non-null   int64
 17  MntFishProducts      2205 non-null   int64
 18  MntSweetProducts     2205 non-null   int64
```

```
 19  MntGoldProds         2205 non-null   int64
 20  NumDealsPurchases    2205 non-null   int64
 21  NumWebPurchases      2205 non-null   int64
 22  NumAppPurchases      2205 non-null   int64
 23  NumStorePurchases    2205 non-null   int64
 24  NumWebVisitsMonth    2205 non-null   int64
 25  MntTotal             2205 non-null   int64
 26  MntRegularProds      2205 non-null   int64
 27  AcceptedCmpOverall   1883 non-null   float64
 28  Response             2205 non-null   int64
dtypes: float64(4), int64(21), object(4)
memory usage: 499.7+ KB
None
```

step 2: necessary changes

perform any necessary coercions

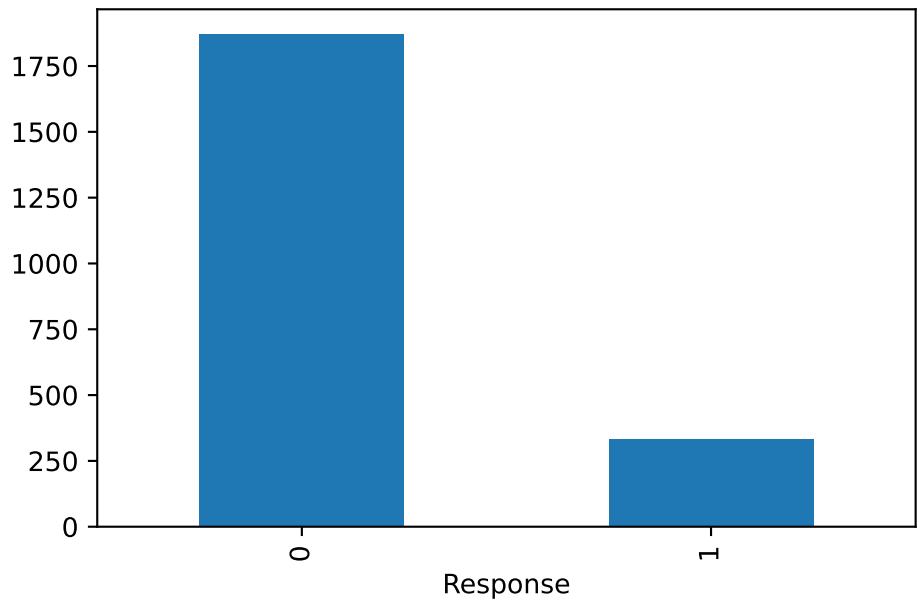
step 2 - Python

```
# Convert 'Kidhome' and 'Teenhome' to binary
pred['Kidhome'] = pd.to_numeric(pred['Kidhome'], errors='coerce').fillna(0).astype(int)
pred['Teenhome'] = pd.to_numeric(pred['Teenhome'], errors='coerce').fillna(0).astype(int)
pred['Kidhome'] = pred['Kidhome'].apply(lambda x: True if x >= 1 else False).astype('bool')
pred['Teenhome'] = pred['Teenhome'].apply(lambda x: True if x >= 1 else False).astype('bool')
print(pred.info())
print(pred.isnull().sum())
# Distribution of the target variable
pred['Response'] = pred['Response'].astype('category')
print(pred['Response'].value_counts())
pred['Response'].value_counts().plot.bar()
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2205 entries, 0 to 2204
Data columns (total 29 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   CustID              2205 non-null   int64
 1   Marital_Status      2205 non-null   object
```

```
 2   Education           2205 non-null   object
 3   Kidhome             2205 non-null   bool
 4   Teenhome            2205 non-null   bool
 5   AcceptedCmp3        2205 non-null   int64
 6   AcceptedCmp4        2205 non-null   int64
 7   AcceptedCmp5        2205 non-null   int64
 8   AcceptedCmp1        2205 non-null   int64
 9   AcceptedCmp2        2205 non-null   int64
 10  Complain            2205 non-null   int64
 11  Income              2205 non-null   float64
 12  Age                 2205 non-null   float64
 13  Recency             2205 non-null   float64
 14  MntWines            2205 non-null   int64
 15  MntFruits           2205 non-null   int64
 16  MntMeatProducts     2205 non-null   int64
 17  MntFishProducts     2205 non-null   int64
 18  MntSweetProducts    2205 non-null   int64
 19  MntGoldProds        2205 non-null   int64
 20  NumDealsPurchases   2205 non-null   int64
 21  NumWebPurchases     2205 non-null   int64
 22  NumAppPurchases     2205 non-null   int64
 23  NumStorePurchases   2205 non-null   int64
 24  NumWebVisitsMonth   2205 non-null   int64
 25  MntTotal            2205 non-null   int64
 26  MntRegularProds     2205 non-null   int64
 27  AcceptedCmpOverall  2205 non-null   float64
 28  Response            2205 non-null   int64
dtypes: bool(2), float64(4), int64(21), object(2)
memory usage: 469.6+ KB
None
CustID               0
Marital_Status       0
Education            0
Kidhome              0
Teenhome             0
AcceptedCmp3         0
AcceptedCmp4         0
AcceptedCmp5         0
AcceptedCmp1         0
AcceptedCmp2         0
Complain             0
Income               0
Age                  0
```

```
Recency               0
MntWines              0
MntFruits             0
MntMeatProducts       0
MntFishProducts       0
MntSweetProducts      0
MntGoldProds          0
NumDealsPurchases     0
NumWebPurchases       0
NumAppPurchases       0
NumStorePurchases     0
NumWebVisitsMonth     0
MntTotal              0
MntRegularProds       0
AcceptedCmpOverall    0
Response              0
dtype: int64
Response
0    1872
1     333
Name: count, dtype: int64
```



step 3: selected features

selected features: excluded features:complaint and CustID Complain: This column has little to no correlation with other columns, this will not be useful in predicting the target column CustID: This is just an identifier and does not contain predictive information.

choosing these features: I ran a correlation matrix and found AcceptedCmp1, AcceptedCmp2, AcceptedCmp3, AcceptedCmp5, AcceptedCmpOverall to have positive correlation with the target column and have strong correlations with AcceptedCmpOverall. In addition, I ran feature selection plot to see which features were useful in the overall accuracy for both decision trees and random forests. With all these considerations including my initution I included features that were useful but also made sure to have enough to give the model as much access as possible.

step 3 - Python

```
# Define features and target
features = [
    'AcceptedCmpOverall', 'MntTotal', 'MntRegularProds', 'MntWines', 'MntMeatProducts',
    'NumAppPurchases', 'Recency', 'NumWebVisitsMonth', 'Income', 'Age', 'Marital_Status',
    'Education'
]
X = pred[features]
y = pred['Response']
```

step 4: categorical encoding

I used categorical encoding, however I did not standardized or normalize since they aren't sensitive to feature scales since they split based on feature values directly

step 4 - Python

```
# Define pipelining steps
numeric_features = ['AcceptedCmpOverall', 'MntTotal', 'MntRegularProds', 'MntWines', 'MntMeat
                    'NumAppPurchases', 'Recency', 'NumWebVisitsMonth', 'Income', 'Age']
categorical_features = ['Marital_Status', 'Education']

# Create transformers
numeric_transformer = Pipeline(steps=[
```

```
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

step 5: Split the dataset

split the training and test set, test size is 20%

step 5 - Python

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, str

# Preprocess the data
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# Get feature names after preprocessing
encoded_feature_names = (
    numeric_features +
    list(preprocessor.named_transformers_['cat']['onehot'].get_feature_names_out(categorical_
)
```

step 6: first model choice

I chose Decision trees because our dataset structure is not linear in nature and this model handles non-linear patterns, can manage interactions between features (important given the

correlations), and doesn't require intensive preprocessing.

step 6 - Python

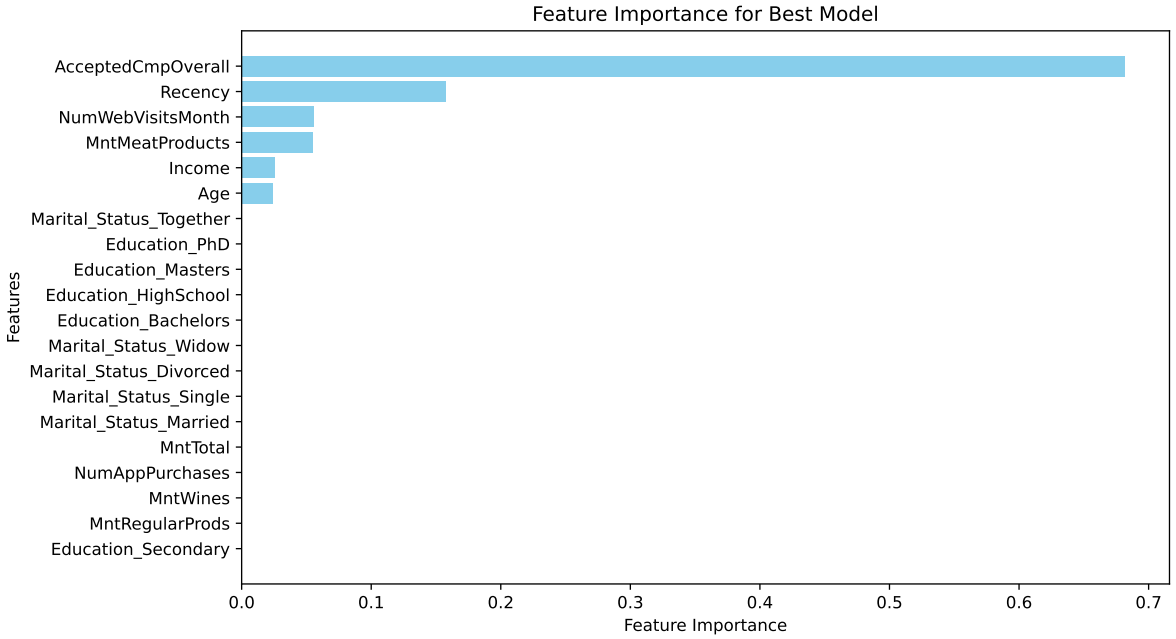
```
# Define the function to train the initial model
def train_model1(X_train, y_train, X_test, y_test):
     # Apply SMOTE to address class imbalance
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

    # Train the model
    model1 = DecisionTreeClassifier(random_state=42)
    model1.fit(X_train_smote, y_train_smote)
    # # Train the model
    # model1 = DecisionTreeClassifier(random_state=42)
    # model1.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model1.predict(X_test)
    print("Model1 Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    return model1

# Define the function to train the best model with hyperparameter tuning
def train_best_model1(X_train, y_train, X_test, y_test, feature_names):
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
    # Define parameter grid for GridSearchCV
    param_grid = {
        'max_depth': [3, 5, 7, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    # Initialize and fit the model with GridSearchCV
    grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5, sc
    grid_search.fit(X_train, y_train)

    # Train the best model
```

```
    best_model1 = grid_search.best_estimator_

    print("Best Parameters:", grid_search.best_params_)

    # Predict and evaluate
    y_pred = best_model1.predict(X_test)
    print("Best Model1 Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Extract and plot feature importance
    feature_importances = best_model1.feature_importances_
    feature_importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': feature_importances
    }).sort_values(by='Importance', ascending=False)

    print("\nFeature Importances:\n", feature_importance_df)

    # Plot feature importances
    plt.figure(figsize=(10, 6))
    plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='sk
    plt.xlabel("Feature Importance")
    plt.ylabel("Features")
    plt.title("Feature Importance for Best Model")
    plt.gca().invert_yaxis()  # Invert the y-axis for better readability
    plt.show()

    return best_model1

# Train and evaluate the initial model
model1 = train_model1(X_train_transformed, y_train, X_test_transformed, y_test)

# Train and evaluate the best model with the updated function
best_model1 = train_best_model1(X_train_transformed, y_train, X_test_transformed, y_test, enc
```

```
Model1 Accuracy: 0.8163265306122449
Confusion Matrix:
 [[329  45]
 [ 36  31]]
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.90      0.88      0.89       374
           1       0.41      0.46      0.43        67

    accuracy                           0.82       441
   macro avg       0.65      0.67      0.66       441
weighted avg       0.83      0.82      0.82       441

Best Parameters: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Model1 Accuracy: 0.8639455782312925
Confusion Matrix:
 [[367   7]
 [ 53  14]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       374
           1       0.67      0.21      0.32        67

    accuracy                           0.86       441
   macro avg       0.77      0.60      0.62       441
weighted avg       0.84      0.86      0.83       441


Feature Importances:
                     Feature  Importance
0        AcceptedCmpOverall    0.682011
6                   Recency    0.157784
7         NumWebVisitsMonth    0.055405
4           MntMeatProducts    0.055164
8                    Income    0.025504
9                       Age    0.024132
13  Marital_Status_Together    0.000000
18            Education_PhD    0.000000
17        Education_Masters    0.000000
16     Education_HighSchool    0.000000
15      Education_Bachelors    0.000000
14     Marital_Status_Widow    0.000000
10  Marital_Status_Divorced    0.000000
12    Marital_Status_Single    0.000000
11   Marital_Status_Married    0.000000
1                  MntTotal    0.000000
5           NumAppPurchases    0.000000
```

```
3                  MntWines    0.000000
2            MntRegularProds    0.000000
19       Education_Secondary    0.000000
```



step 7: Second model choice

I chose Random forest next because for its ability to handle non-linear relationships and interactions between features

step 7 - Python

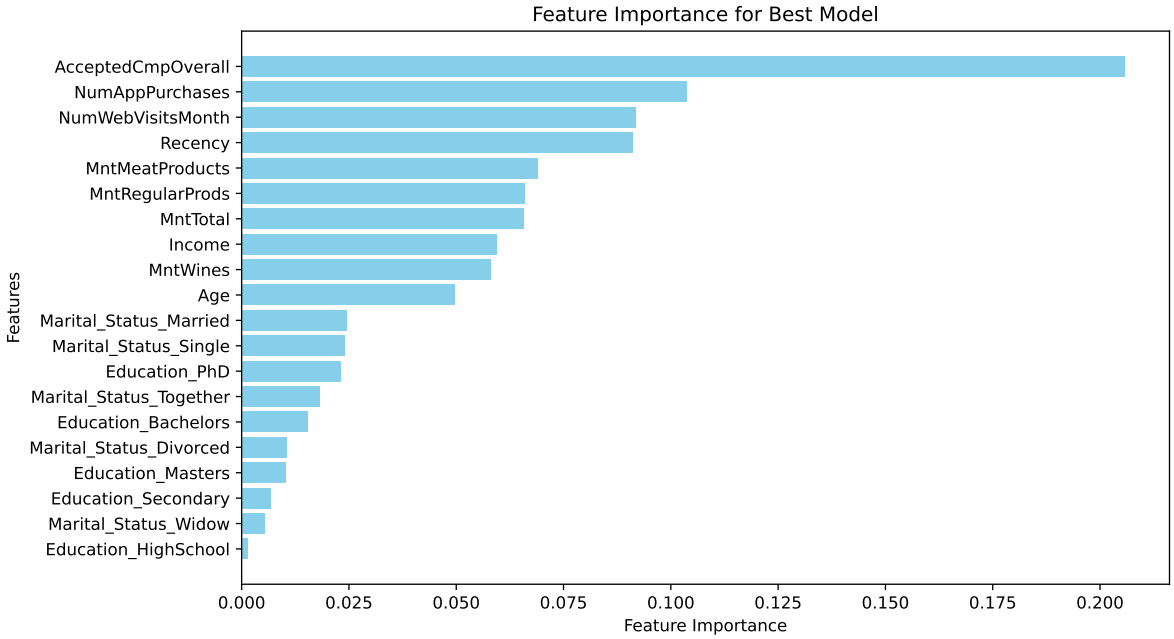
```python
# Define the function to train the initial Random Forest model with SMOTE
def train_model2(X_train, y_train, X_test, y_test):
    # Apply SMOTE to address class imbalance
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

    # Train the initial Random Forest model
    model2 = RandomForestClassifier(random_state=42)
    model2.fit(X_train_smote, y_train_smote)
```

```
    # Predict and evaluate
    y_pred = model2.predict(X_test)
    print("Model2 Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    return model2

# Define the function to train the best Random Forest model with hyperparameter tuning and SM
def train_best_model2(X_train, y_train, X_test, y_test, feature_names):
    # Apply SMOTE to address class imbalance
    smote = SMOTE(random_state=42)
    X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

    # Define the parameter grid for RandomizedSearchCV
    param_dist = {
        'n_estimators': [100, 200, 300, 400, 500],
        'max_depth': [10, 20, 30, 40, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'bootstrap': [True, False]
    }

    # Initialize RandomizedSearchCV for Random Forest
    randomized_search = RandomizedSearchCV(
        RandomForestClassifier(random_state=42),
        param_distributions=param_dist,
        n_iter=50,
        scoring='accuracy',
        cv=5,
        random_state=42,
        n_jobs=-1
    )

    # Fit the model with RandomizedSearchCV using SMOTE data
    randomized_search.fit(X_train_smote, y_train_smote)

    # Train the best model found by RandomizedSearchCV
    best_model2 = randomized_search.best_estimator_
    print("Best Parameters:", randomized_search.best_params_)

    # Predict and evaluate
```

```
    y_pred = best_model2.predict(X_test)
    print("Best Model2 Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Extract and plot feature importance
    feature_importances = best_model2.feature_importances_
    feature_importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': feature_importances
    }).sort_values(by='Importance', ascending=False)

    print("\nFeature Importances:\n", feature_importance_df)

    # Plot feature importances
    plt.figure(figsize=(10, 6))
    plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='sk
    plt.xlabel("Feature Importance")
    plt.ylabel("Features")
    plt.title("Feature Importance for Best Model")
    plt.gca().invert_yaxis()  # Invert the y-axis for better readability
    plt.show()

    return best_model2

# Train and evaluate the initial Random Forest model
model2 = train_model2(X_train_transformed, y_train, X_test_transformed, y_test)

# Train and evaluate the best Random Forest model with hyperparameter tuning
feature_names = encoded_feature_names  # Replace with the list of feature names after preproc
best_model2 = train_best_model2(X_train_transformed, y_train, X_test_transformed, y_test, fea
```

```
Model2 Accuracy: 0.8684807256235828
Confusion Matrix:
 [[355  19]
 [ 39  28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92       374
           1       0.60      0.42      0.49        67
```

```
    accuracy                           0.87       441
   macro avg       0.75      0.68      0.71       441
weighted avg       0.85      0.87      0.86       441

Best Parameters: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_de
Best Model2 Accuracy: 0.8639455782312925
Confusion Matrix:
 [[356  18]
 [ 42  25]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92       374
           1       0.58      0.37      0.45        67

    accuracy                           0.86       441
   macro avg       0.74      0.66      0.69       441
weighted avg       0.85      0.86      0.85       441


Feature Importances:
                     Feature  Importance
0        AcceptedCmpOverall    0.205890
5           NumAppPurchases    0.103629
7         NumWebVisitsMonth    0.091803
6                   Recency    0.091130
4           MntMeatProducts    0.069016
2           MntRegularProds    0.065996
1                  MntTotal    0.065676
8                    Income    0.059525
3                  MntWines    0.058078
9                       Age    0.049561
11   Marital_Status_Married    0.024584
12    Marital_Status_Single    0.024068
18            Education_PhD    0.023049
13  Marital_Status_Together    0.018170
15     Education_Bachelors    0.015412
10  Marital_Status_Divorced    0.010592
17       Education_Masters    0.010340
19     Education_Secondary    0.006743
14     Marital_Status_Widow    0.005355
16     Education_HighSchool    0.001382
```



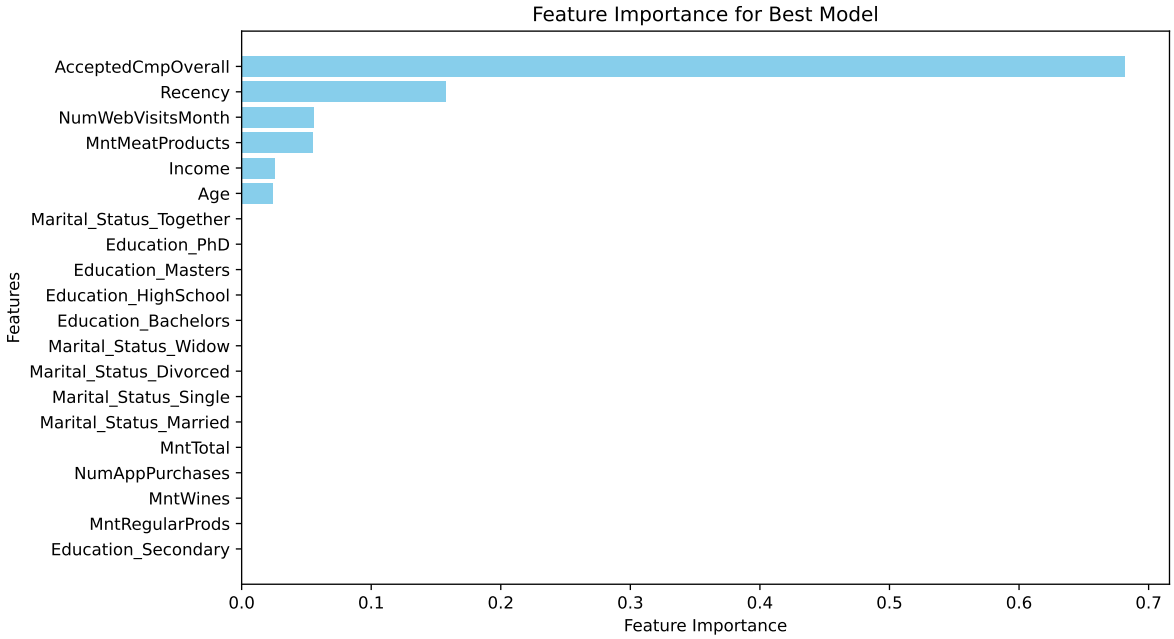
step 8: Models analysis

best model 2 is better compared to best model 1, even though they have the same over all accuracy but best model 2 is better because of higher f1score Based on the results, Model 2 (Random Forest) demonstrates superior performance compared to other models across most evaluation metrics and should be selected. With an accuracy of 87%, it correctly predicts a high proportion of instances overall. Its precision of 90% and recall of 95% for class 0 indicate strong performance in correctly identifying negatives. For class 1, while precision is 60% and recall is 42%, it performs better in capturing true positives compared to other models. The F1-score of 49% for class 1 highlights a better balance between precision and recall for positive predictions. Additionally, Model 2 shows fewer False Positives and False Negatives, leading to stronger weighted averages for precision, recall, and F1-score. These results make Random Forest the most effective and reliable choice, particularly for maximizing overall accuracy and reducing errors in the dominant class (class 0). I also ran model 2 with no SMOTE, the results were good as well: Accuracy: 88% Precision (Class 1): 76% Recall (Class 1): 33% F1-Score (Class 1): 46% False Positives (FP): 7 False Negatives (FN): 45 ### step 8 - Python

```
# Train and evaluate the best model with the updated function
best_model1 = train_best_model1(X_train_transformed, y_train, X_test_transformed, y_test, enc
best_model2 = train_best_model2(X_train_transformed, y_train, X_test_transformed, y_test, fea
```

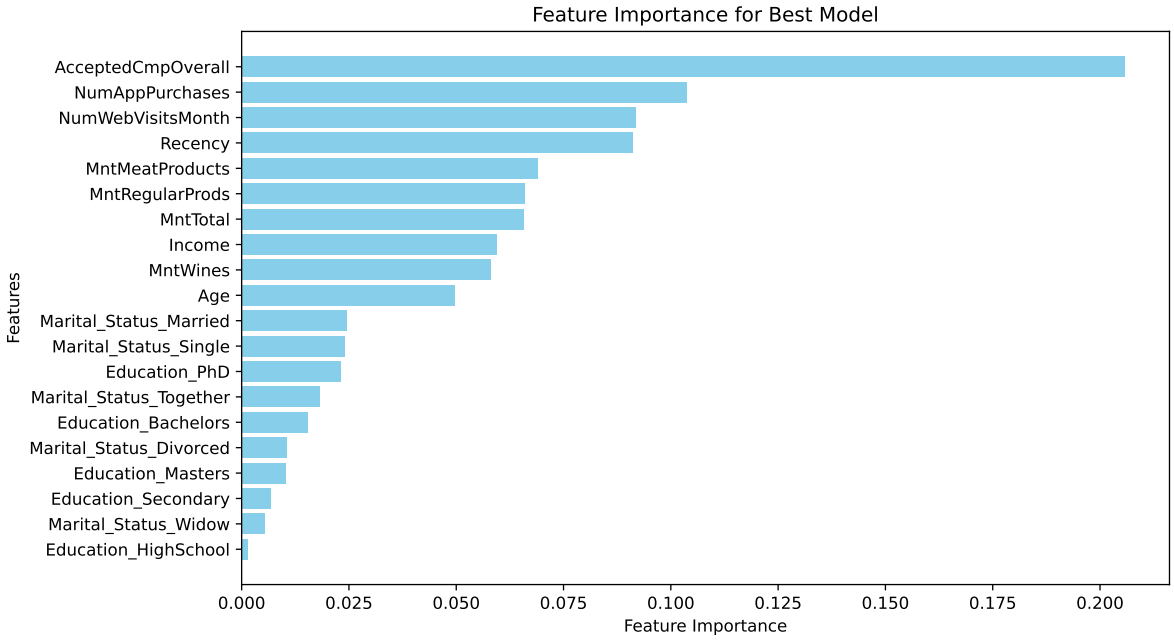
```
Best Parameters: {'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
Best Model1 Accuracy: 0.8639455782312925
Confusion Matrix:
 [[367   7]
 [ 53  14]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       374
           1       0.67      0.21      0.32        67

    accuracy                           0.86       441
   macro avg       0.77      0.60      0.62       441
weighted avg       0.84      0.86      0.83       441


Feature Importances:
                     Feature  Importance
0        AcceptedCmpOverall    0.682011
6                   Recency    0.157784
7         NumWebVisitsMonth    0.055405
4           MntMeatProducts    0.055164
8                    Income    0.025504
9                       Age    0.024132
13  Marital_Status_Together    0.000000
18            Education_PhD    0.000000
17        Education_Masters    0.000000
16     Education_HighSchool    0.000000
15      Education_Bachelors    0.000000
14     Marital_Status_Widow    0.000000
10  Marital_Status_Divorced    0.000000
12    Marital_Status_Single    0.000000
11   Marital_Status_Married    0.000000
1                  MntTotal    0.000000
5           NumAppPurchases    0.000000
3                  MntWines    0.000000
2           MntRegularProds    0.000000
19      Education_Secondary    0.000000
```




```
Best Parameters: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_de
Best Model2 Accuracy: 0.8639455782312925
Confusion Matrix:
 [[356  18]
 [ 42  25]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92       374
           1       0.58      0.37      0.45        67

    accuracy                           0.86       441
   macro avg       0.74      0.66      0.69       441
weighted avg       0.85      0.86      0.85       441


Feature Importances:
                    Feature  Importance
0        AcceptedCmpOverall    0.205890
5           NumAppPurchases    0.103629
7         NumWebVisitsMonth    0.091803
6                   Recency    0.091130
4           MntMeatProducts    0.069016
```

2	MntRegularProds	0.065996
1	MntTotal	0.065676
8	Income	0.059525
3	MntWines	0.058078
9	Age	0.049561
11	Marital_Status_Married	0.024584
12	Marital_Status_Single	0.024068
18	Education_PhD	0.023049
13	Marital_Status_Together	0.018170
15	Education_Bachelors	0.015412
10	Marital_Status_Divorced	0.010592
17	Education_Masters	0.010340
19	Education_Secondary	0.006743
14	Marital_Status_Widow	0.005355
16	Education_HighSchool	0.001382



step 9: Final Model Choice

Based on the results, Model 2 (Random Forest) demonstrates superior performance across most evaluation metrics and should be selected. With an accuracy of 87%, it correctly predicts a high proportion of instances overall. For class 0, the precision of 90% and recall of 95% indicate excellent performance in identifying negatives. For class 1, the precision is 60%, and recall

is 42%, demonstrating a better ability to capture true positives compared to other models. The F1-score of 49% for class 1 highlights a balanced approach between precision and recall, making Model 2 more reliable for positive predictions. Additionally, Model 2 has fewer false positives and false negatives, leading to stronger weighted averages across precision, recall, and F1-score. This makes it the most effective choice for overall accuracy and minimizing errors, especially in the dominant class (class 0).

When running Model 2 without SMOTE, the performance remains strong, with an accuracy of 88%. For class 1, the precision improves to 76%, although recall drops to 33%, resulting in an F1-score of 46%. The model exhibits 7 false positives and 45 false negatives, which is still an improvement compared to other models. These results confirm that Model 2 (Random Forest), both with and without SMOTE, is the most robust and balanced option for this dataset.

step 9 - Python

```
model2 = train_model2(X_train_transformed, y_train, X_test_transformed, y_test)
```

```
Model2 Accuracy: 0.8684807256235828
Confusion Matrix:
 [[355  19]
 [ 39  28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92       374
           1       0.60      0.42      0.49        67

    accuracy                           0.87       441
   macro avg       0.75      0.68      0.71       441
weighted avg       0.85      0.87      0.86       441
```

Final Analysis

Interpretability:

The model is moderately interpretable, especially when we consider elements like feature importance and the confusion matrix. With Random Forest, we can analyze feature importance scores to determine which variables have the most significant impact on predictions. For instance, features such as income, recency, or purchase history could have higher importance, highlighting their role in predicting customer responses.

To explain the results to someone unfamiliar with data science, I would focus on a few key points. Firstly, the model has an overall accuracy of 87%, meaning it successfully predicts customer responses in 87 out of 100 cases. Secondly, for customers who responded positively (class 1), the model achieves a precision of 60%, meaning that 60% of the customers it predicted as responders were accurate. Its recall for class 1 is 42%, which means the model identifies 42% of all actual responders. For non-responders (class 0), the model performs better, with a precision of 90% and a recall of 95%. The confusion matrix provides a clear breakdown of correct and incorrect predictions, showing where the model performs well (e.g., high accuracy for non-responders) and where it has limitations (e.g., missing some true responders). Lastly, using feature importance helps identify key factors driving predictions, making it easier to draw actionable insights. Visual aids like bar charts and confusion matrices would be helpful to simplify these concepts and make them understandable for a non-technical audience.

Decisions and Limitations:

Based on the model's performance, I would recommend focusing on targeted campaigns. By leveraging the model's ability to predict responders (class 1), decision-makers can prioritize features like customer recency, spending habits, and product preferences to design campaigns that target the most likely responders effectively. Additionally, using feature importance scores, we could segment the audience further to create personalized marketing strategies, increasing the likelihood of conversion. Furthermore, resources should be allocated strategically to high-value customers identified as potential responders, ensuring a better return on investment for marketing efforts.

However, it is essential to communicate the limitations of the model as well. One significant limitation is the class imbalance in the dataset, which makes it harder for the model to predict positive responders (class 1) accurately, resulting in lower recall for this group. Additionally, since the model is trained on historical data, it may not generalize well to new or evolving customer behaviors. While Random Forest is accurate, it can be less interpretable compared to simpler models like Decision Trees, which could make it harder to explain the logic behind its predictions. Moreover, the default probability threshold (0.5) might not be ideal for this specific problem, and adjusting it could improve recall for responders. Finally, the accuracy of predictions depends on the quality of input data. If there is noise or if important variables influencing customer response are missing, the model's performance could be limited. Despite these limitations, the model provides actionable insights that can help decision-makers make informed choices while keeping its constraints in mind.