# Mini Project 1 – Banking Campaign Output Prediction

**Md Hasibul Haque Zahid**

**Student ID: 2302302**

## Introduction

This project applies machine learning techniques to predict if a client will subscribe to a bank term deposit. The data used is related to the bank's direct marketing campaigns, which often involve multiple phone calls to the same client. The data includes various attributes such as the client's personal information, details about the last contact of the current campaign, other attributes, and social and economic context attributes. The goal is to predict the binary outcome ('yes' or 'no') based on these attributes.

Two machine learning models, Decision Tree and Random Forest, are used to solve this classification problem. The aim is to build a model that can accurately predict the subscription outcome for a new client given the same input variables.

The significance of this problem lies in its potential to help the bank more effectively allocate resources towards potential clients who are more likely to subscribe to a term deposit, thereby improving the efficiency of their marketing campaigns. This study also contributes to the broader field of predictive analytics in marketing.

## Features for the Models:

The features used for the models are all the input variables provided in the dataset. These include client data (age, job, marital status, education, default, housing, loan), data related to the last contact of the current campaign (contact,month, day_of_week, duration), other attributes (campaign, pdays, previous, poutcome), and social and economic context attributes (emp.var.rate, cons.price.idx, cons.conf.idx,euribor3m,nr.employed).These features were chosen because they provide comprehensive information about the client and the context, which can be useful for predicting whether the client will subscribe to a term deposit.

The data processing involves several steps:

- **Loading the Data**: The data is loaded from a CSV file into a pandas Data Frame.
- **Handling Categorical Variables**: The dataset contains several categorical variables. Machine learning models require numerical input, so these categorical variables are converted into numerical form using label encoding. Label encoding assigns each unique category in a feature to a numerical value, making it possible for the model to process this data.
- **Splitting the Data**: The data is split into features (X) and the target variable (y). The features include all the input variables, while the target variable is the 'y' column, which indicates whether the client subscribed to a term deposit. The data is further split into training and testing sets, with 80% of the data used for training the models and 20% reserved for testing their performance.

```
# Load your data
df = pd.read_csv('bank-additional-full.csv', delimiter=';') # setting the datapath and using delimiter for identification
print (df) #print the dataset from the excel sheet
```

```
       age          job  marital            education  default housing loan  \
0       56    housemaid  married             basic.4y       no      no   no
1       57     services  married          high.school  unknown      no   no
2       37     services  married          high.school       no     yes   no
3       40       admin.  married             basic.6y       no      no   no
4       56     services  married          high.school       no      no  yes
...    ...          ...      ...                  ...      ...     ...  ...
41183   73      retired  married  professional.course       no     yes   no
41184   46  blue-collar  married  professional.course       no      no   no
41185   56      retired  married    university.degree       no     yes   no
41186   44   technician  married  professional.course       no      no   no
41187   74      retired  married  professional.course       no     yes   no

         contact month day_of_week  ...  campaign  pdays  previous  \
0      telephone   may         mon  ...         1    999         0
1      telephone   may         mon  ...         1    999         0
2      telephone   may         mon  ...         1    999         0
3      telephone   may         mon  ...         1    999         0
4      telephone   may         mon  ...         1    999         0
```

**Fig 1: Read the data file**

```
[2]:
# Convert categorical variables to numeric
le = LabelEncoder()
categorical_features = [col for col in df.columns if df[col].dtype == 'object']
for col in categorical_features:
    df[col] = le.fit_transform(df[col])
```

```
[3]:
# Split the data into train and test sets
X = df.drop('y', axis=1)
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

**Fig 2: Splitting the dataset for training**

## Choice of Algorithms

The algorithms chosen for this task were the Decision Tree and Random Forest classifiers. Here's why:

1. **Decision Trees**: Decision Trees are simple to understand and interpret, and they can handle both numerical and categorical data. They're also capable of fitting complex datasets.
2. **Random Forests**: Random Forests are an ensemble of Decision Trees. They aggregate the predictions of multiple Decision Trees to give a more accurate and stable prediction. They're less likely to over fit than a single Decision Tree.

## Improving Model Performance

The performance of the models was evaluated using accuracy as the metric. If the initial performance was not satisfactory, several strategies could be employed to improve it:

1. **Hyper parameter Tuning**: The models' parameters could be tuned using techniques like Grid Search or Random Search.
2. **Feature Engineering**: New features could be created, irrelevant features could be removed, or existing features could be transformed to improve the models' performance.

3. **Handling Class Imbalance**: If the target variable classes were imbalanced, techniques like oversampling, under sampling, or SMOTE could be used.
4. **Using a Different Model**: If the Decision Tree and Random Forest models did not provide satisfactory results, other models like Gradient Boosting or XGBoost could be tried.

```
[4]: # Decision Tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
print("Decision Tree Classifier report: \n\n", classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classifier report:

              precision    recall  f1-score   support

           0       0.94      0.94      0.94      7303
           1       0.51      0.51      0.51       935

    accuracy                           0.89      8238
   macro avg       0.72      0.72      0.72      8238
weighted avg       0.89      0.89      0.89      8238
```

```
[5]: # Random Forest model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Classifier report: \n\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier report:

              precision    recall  f1-score   support

           0       0.94      0.96      0.95      7303
           1       0.64      0.52      0.58       935

    accuracy                           0.91      8238
   macro avg       0.79      0.74      0.76      8238
weighted avg       0.91      0.91      0.91      8238
```

**Fig 3: Decision tree (DT) & Random Forest (RF)**

**Choice of Algorithms**

The algorithms chosen for this project were the Decision Tree and Random Forest classifiers. Here's why:

**Decision Trees**: Decision Trees are simple to understand and interpret, and they can handle both numerical and categorical data. They're also capable of fitting complex datasets.
**Random Forests**: Random Forests are an ensemble of Decision Trees. They aggregate the predictions of multiple Decision Trees to give a more accurate and stable prediction. They're less likely to over fit than a single Decision Tree.

**Achieving Required Accuracy**

In the first training round, the Decision Tree model achieved an accuracy of approximately 88.77%, and the Random Forest model achieved an accuracy of approximately 91.28%. These are quite high accuracy scores, suggesting that the models were able to learn effectively from the training data and generalize well to unseen data. However, as mentioned earlier, other metrics might also be important to consider, depending on the specific problem and dataset.

```
[6]: # Compare the models
print("Test Accuracy of Decision Tree: ", accuracy_score(y_test, y_pred_dt)*100, "%")
print("Test Accuracy of Random Forest: ", accuracy_score(y_test, y_pred_rf)*100, "%")
```

```
Test Accuracy of Decision Tree:  88.77154649186696 %
Test Accuracy of Random Forest:  91.28429230395727 %
```

**Fig 4: Test Accuracy of Both DT & RF**

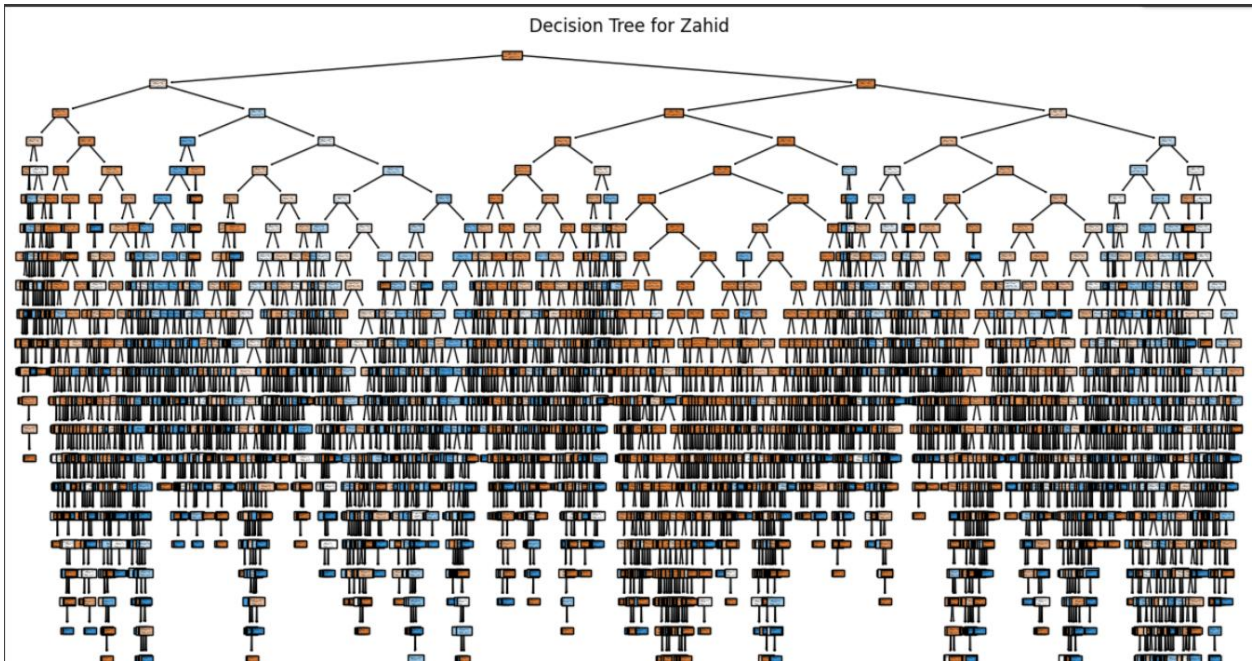


**Fig 5: Visualizing the Decision tree**

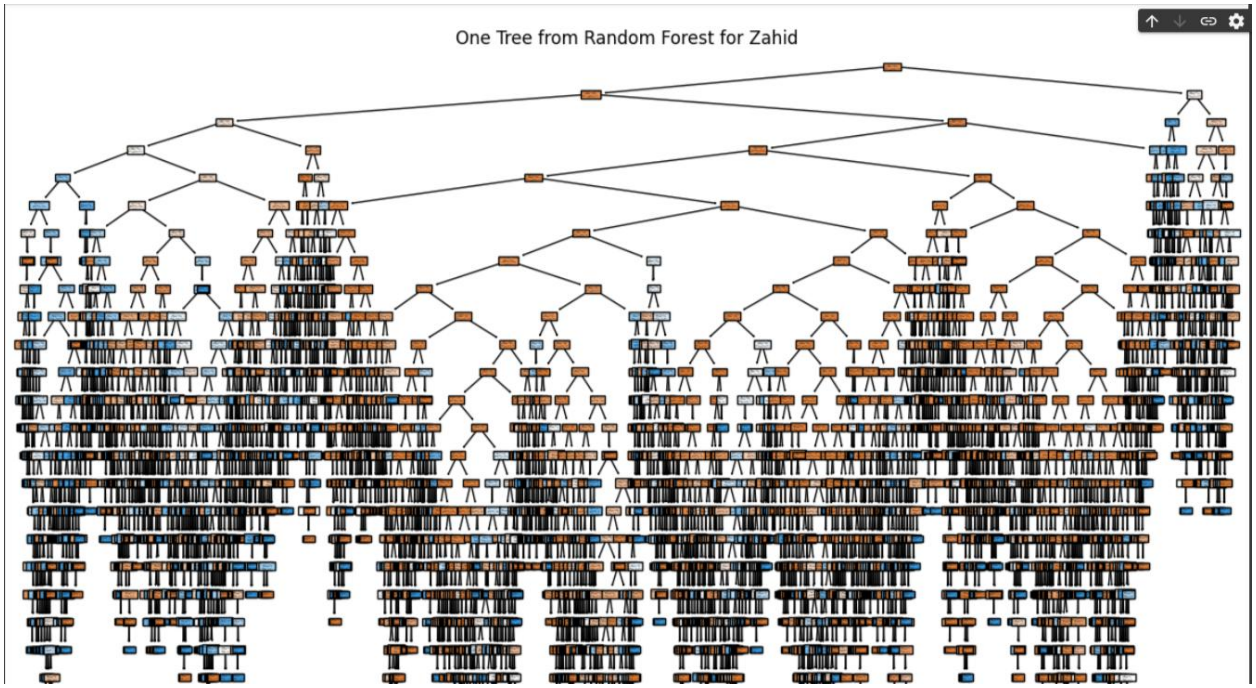


**Fig 6: Visualizing the Random Forest**

# Conclusion

### Scientific Bottlenecks

The primary scientific bottlenecks encountered during this project were:

1. **Data Preprocessing**: The dataset contained both numerical and categorical variables, which required appropriate preprocessing before they could be used in the machine learning models. This was addressed by using label encoding to convert categorical variables into a numerical form that the models could process.
2. **Model Selection and Evaluation**: Choosing the right model and the right evaluation metric for this specific problem was another challenge. We addressed this by trying out two different models (Decision Tree and Random Forest) and evaluating their performance using accuracy as the metric. However, depending on the problem and the data, other models and metrics might be more appropriate.
3. **Over fitting**: Decision Trees are prone to over fitting, which means they can perform well on the training data but poorly on unseen data. This was mitigated by using a Random Forest, which is an ensemble of Decision Trees and tends to be less prone to over fitting.

### Overcoming the Bottlenecks

These bottlenecks were overcome through a combination of data preprocessing, model selection, and performance evaluation. Categorical variables were converted into numerical form using label encoding, and two different models were trained and evaluated on the data. The performance of the models was then compared based on their accuracy scores.

### Better Algorithm

In terms of accuracy, the Random Forest model performed better than the Decision Tree model. The Random Forest model achieved an accuracy of approximately 91.28%, while the Decision Tree model

achieved an accuracy of approximately 88.77%. This suggests that the Random Forest model was better able to generalize from the training data to unseen data.

However, it's important to note that the "best" model can depend on various factors, including the specific characteristics of the data, the problem at hand, and the relevant evaluation metrics. While the Random Forest model had higher accuracy in this case, there might be situations where a Decision Tree or another model could be more appropriate.

In conclusion, this project demonstrated the application of Decision Tree and Random Forest models to a marketing prediction problem. Despite some challenges, the models were able to achieve high accuracy scores, providing valuable insights that could potentially improve the efficiency of future marketing campaigns.

**Warm Regards,**

**Md Hasibul Haque Zahid**
**Master's Degree Programme in Information Technology: Computer Engineering (Turku)**
**Student number: 2302302**