Introduction

The goal of this project was to assist a Portuguese banking institution in predicting whether a client will subscribe to a term deposit based on direct marketing campaigns conducted through phone calls.

To achieve this overarching goal, we have broken it down into the following smaller objectives:

1. **Identify Key Factors**: Determine which client attributes (e.g., age, job type, marital status) are most influential in the decision to subscribe to a term deposit.
2. **Data Analysis and Exploration**: Explore the dataset to gain insights into client characteristics and analyse the relationships between different variables.
3. **Predictive Modelling**: Build models to predict whether a client will subscribe to a term deposit based on the given features.
4. **Evaluate Model Performance**: Compare different models (Logistic Regression, Decision Tree, Random Forest) in terms of accuracy, precision, recall, and F1 score, and identify which model performs best for the given dataset.

These objectives will guide the analysis and help us evaluate the performance of direct marketing campaigns, ultimately providing valuable recommendations for the bank.

Dataset Description:

The dataset used in this project was obtained from UCI Machine Learning Repository (UCI Machine Learning Repository, n.d.). It contains information from direct marketing campaigns of a Portuguese banking institution. The dataset comprises 17 attributes, including 7 numerical variables (e.g., age, balance, duration) and 10 categorical variables (e.g., job type, marital status, education).

Key details of the dataset:

- **Total Records**: 45,211 rows
- **Numerical Attributes**:
    - **Age**: Age of the client.
    - **Balance**: Yearly average balance of the client.
    - **Duration**: Last contact duration during the campaign.
    - **Other Attributes**: Include "campaign," "previous contact," etc.
- **Categorical Attributes**:
    - **Job**: Type of job (e.g., management, technician).
    - **Marital Status**: Marital status of the client (e.g., single, married).
    - **Other Attributes**: Include "education," "loan status," "contact type," etc.

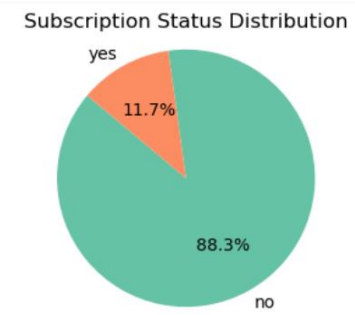
The target variable is whether or not the client subscribed to a term deposit ("yes" or "no"). The dataset is imbalanced, with approximately 88.3% of responses being "no" and 11.7% being "yes."

Data Cleanup

The first step in our modeling process was the investigation into our dataset and verify that our data was sound and of good quality. The first step in our process was to calculate summary statistics for our dataset. Through this process we were able to identify that our dataset contained seven numerical values and ten categorical values. We checked for the null values in our dataset and saw that there were no missing values in our dataset. We then checked for duplicate rows in our dataset and saw that there wasn’t anything duplicate rows. Then we checked for basic statistics. Below are the results:

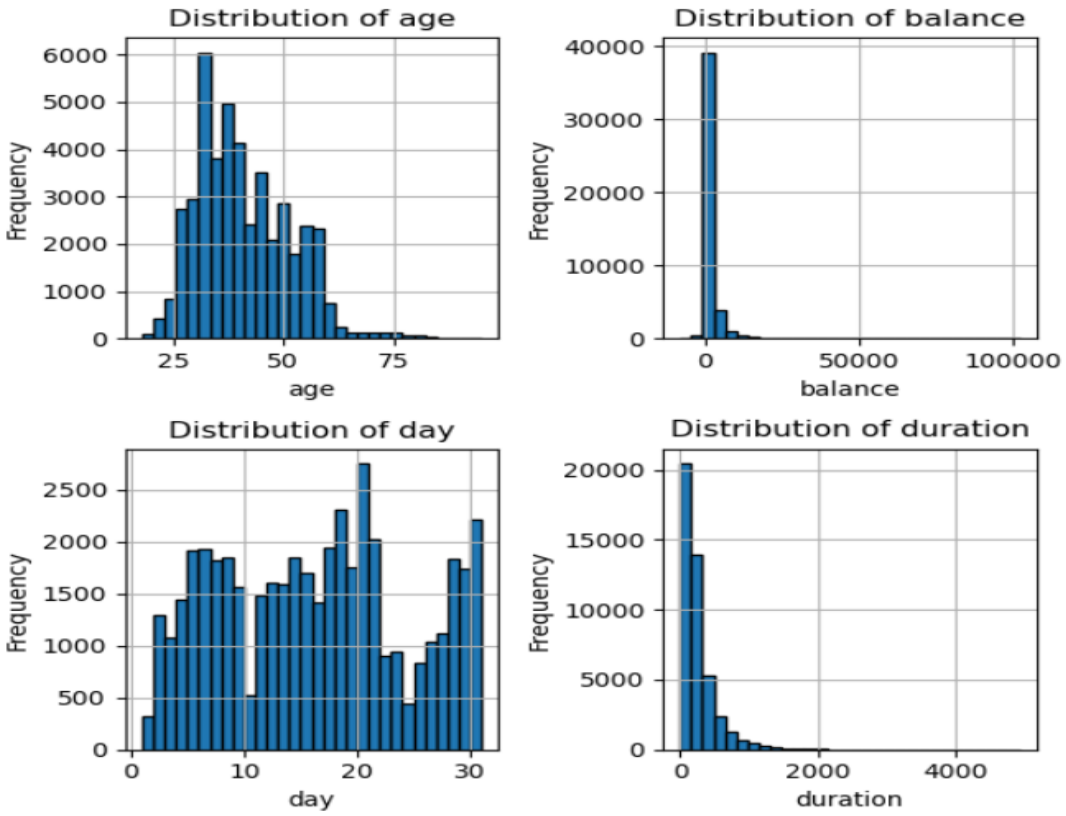
```
Basic Statistics:
                age        balance           day      duration      campaign  \
count  45211.000000   45211.000000  45211.000000  45211.000000  45211.000000
mean      40.936210    1362.272058     15.806419    258.163080      2.763841
std       10.618762    3044.765829      8.322476    257.527812      3.098021
min       18.000000   -8019.000000      1.000000      0.000000      1.000000
25%       33.000000      72.000000      8.000000    103.000000      1.000000
50%       39.000000     448.000000     16.000000    180.000000      2.000000
75%       48.000000    1428.000000     21.000000    319.000000      3.000000
max       95.000000  102127.000000     31.000000   4918.000000     63.000000

              pdays      previous
count  45211.000000  45211.000000
mean      40.197828      0.580323
std      100.128746      2.303441
min       -1.000000      0.000000
25%       -1.000000      0.000000
50%       -1.000000      0.000000
75%       -1.000000      0.000000
max      871.000000    275.000000
```

We can see that the mean age of the people in the dataset is approximately 40 years and the maximum age is 95 years. The mean average yearly balance is approximately 1362. The maximum last contact duration is approximately 4918 seconds which is approximately 81 mins and the average call duration is approximately 258 seconds which is approximately 4 minutes.

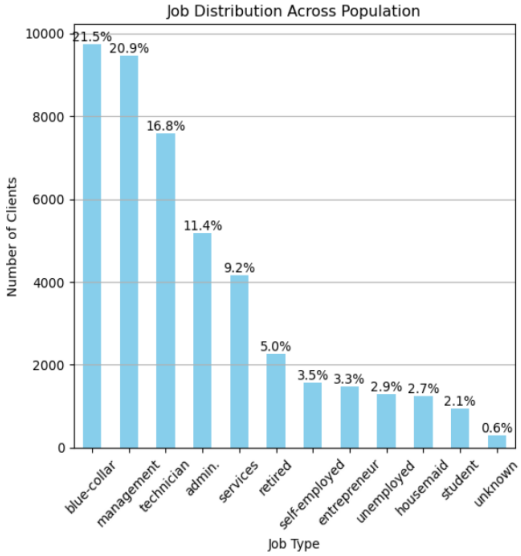


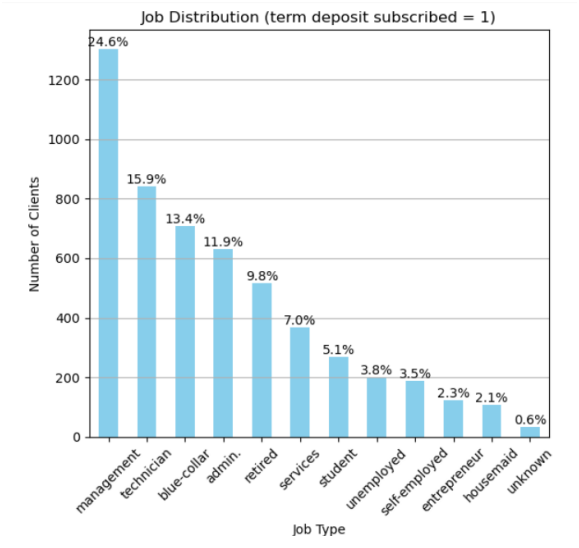
Then we checked the distribution of dependent variable which was has the client subscribed a term deposit? We can see that our dataset has 88.3% No and 11.7% Yes. Our dataset is imbalanced dataset.

After this, we checked the distribution of few numerical variables and results were as follows:



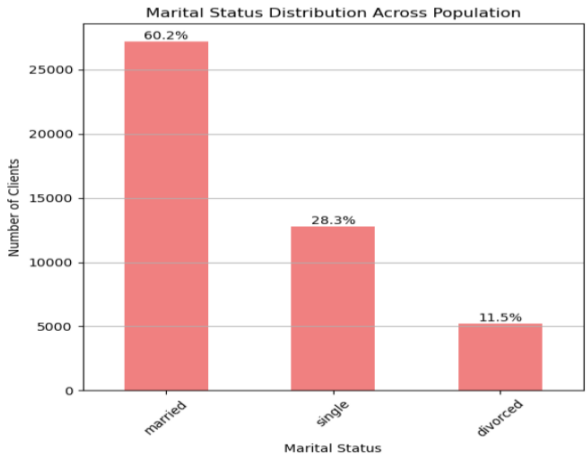
We can see from the above graphs that variables like age, balance and duration are positively skewed. Let us see the distribution of different jobs across population and how does it look among people who have bought term deposits:

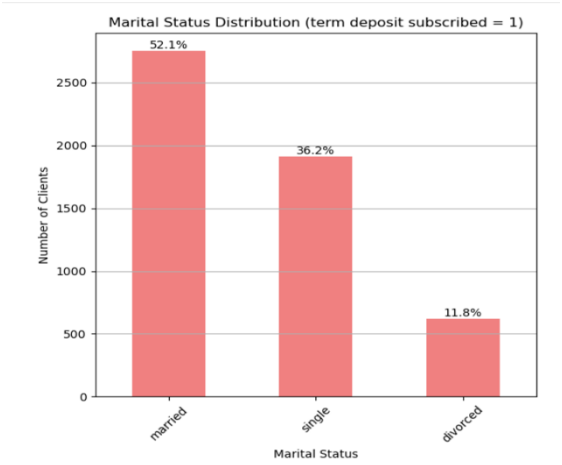




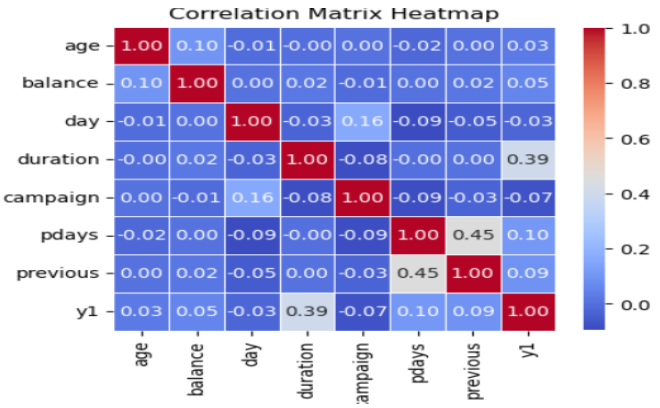
We can see that overall people in management job are 21.5% whereas in the population of people who subscribed to term deposit are 24.6%. Rest all more or less similarly distributed as the population.

Now let us see the distribution of marital status across population and among those who have subscribed to term deposit. The distribution is as follows:





We can see that the percentage of singles in the population is 28.3% whereas in the population where people have subscribed to a term deposit is 36.2%. to each other. After this we saw the correlation between all the numerical variables in the dataset. The resulting plot can be found in figure 1. Here we can see that there are no variables that are strongly correlated with each other. This implies that we may not need to minimize these variables to create the model.



Correlation plot of all numerical variables

Baseline Model Results

Since our response variable was categorical we chose Logistic Regression, Decision Tree and Random Forest

The initial baseline models—Logistic Regression, Decision Tree, and Random Forest—provided insights into the behaviour of the dataset and its predictive potential. Each model was built without hyperparameter tuning, aiming to serve as a foundation for further improvements. Before diving into the modelling, our focus was on data preparation, which included handling outliers, encoding categorical variables, and feature scaling. The outliers in variables such as pdays and balance were handled using the IQR method to ensure extreme values did not disproportionately influence the models' performance. Categorical variables were transformed using OneHotEncoding. One hot encoding is a method used to convert categorical variables into numerical values for use in machine learning models (GeeksforGeeks, 2024a). Then scaling was applied using StandardScaler to ensure all features contributed equally during model training (Bhandari, 2024).

The baseline results showed that the Random Forest model performed the best with an accuracy of **90.15%** and a precision of **66.13%**, followed by Logistic Regression and Decision Tree. Despite having the highest accuracy, Random Forest, like the other models, showed a relatively low recall (37.58%), indicating that many potential subscribers were not being identified. For a marketing campaign aimed at increasing term deposits, high recall is crucial to ensure that a larger pool of potential customers is reached, which would ultimately increase the chance of success.

The Decision Tree model showed a more balanced performance between precision and recall, albeit at a lower overall accuracy compared to Random Forest. Logistic Regression, while

providing an accuracy of **89.76%**, also had low recall, making it less reliable for capturing all positive cases.

Overall, the low recall across all models suggests the need for a more nuanced approach to model tuning and possibly feature engineering. Moving forward, our focus will be on tuning hyperparameters, engineering new features, and implementing sampling techniques to handle class imbalance. Improving recall will be a primary objective to ensure that we are identifying as many potential subscribers as possible while maintaining a reasonable precision.

The results of our model are as follows:

Technique Used	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Logistic Regression	*89.76%*	*64.01%*	*34.56%*	*44.88%*
Decision Tree	*87.13%*	*46.62%*	*46.20%*	*46.41%*
Random Forest	*90.15%*	*66.13%*	*37.58%*	*47.93%*

Important Variables as per Logistic Regression:

Feature	**Coefficient**	**Abs_Coefficient**
remainder__pdays	1.582334	1.582334
remainder__duration	1.252646	1.252646
cat__contact_unknown	-0.698933	0.698933
cat__poutcome_success	0.421746	0.421746
cat__housing_yes	-0.331088	0.331088
cat__month_jul	-0.286971	0.286971
cat__pdays_cat_Not Contacted	-0.273311	0.273311
cat__month_nov	-0.238333	0.238333
cat__month_jan	-0.213449	0.213449
cat__month_aug	-0.199143	0.199143

In our logistic regression analysis, 'pdays' is the most impactful predictor, strongly enhancing subscription chances through recent contacts. 'Duration' also plays a key role; longer discussions tend to lead to higher subscription rates. On the other hand, an 'unknown contact type'

significantly reduces the likelihood of a subscription, highlighting the importance of clear communication. 'Successful prior outcomes' positively affect future subscriptions, demonstrating the power of customer relationships. Seasonal timing, especially in months like July, November, January, and August, shows reduced subscription rates, likely due to financial or holiday-related factors. Additionally, clients with 'housing loans' seem less willing to subscribe, perhaps reflecting financial caution.

Important Variables as per Decision Tree:

Feature	Importance
remainder__duration	0.254448
remainder__balance	0.116487
remainder__age	0.106845
remainder__day	0.095257
cat__poutcome_success	0.090028
remainder__campaign	0.033886
cat__contact_unknown	0.017114
cat__month_jun	0.016805
cat__housing_yes	0.016351
cat__marital_married	0.014508

In the Decision Tree model, 'duration' is the most decisive factor, with longer calls strongly indicating likely subscriptions. 'Balance' and 'age' are also significant, hinting at the importance of a client's financial health and background. 'Day of contact' plays a crucial role, possibly tied to cash flow cycles. The model also shows that 'successful previous campaign outcomes' positively impact future decisions, emphasizing the need for sustained customer engagement. While less impactful, contact type and certain months like June play a moderate role in influencing decisions.

Important Variables as per Random Forest:

Feature	Importance
remainder__duration	0.253427
remainder__age	0.110471
remainder__balance	0.107056
remainder__day	0.099616
cat__poutcome_success	0.052403
remainder__campaign	0.043299
cat__housing_yes	0.024474
cat__contact_unknown	0.014822
cat__education_secondary	0.014691
cat__marital_married	0.013743

The Random Forest model highlights 'duration' as the leading influencer, confirming the importance of engagement depth. 'Age' and 'balance' are also highly predictive, reflecting the relevance of demographic and financial stability. Both the day of contact and campaign intensity strongly affect outcomes, underscoring the strategic value of timing. Successful prior engagements boost subscription likelihood, while financial commitments like 'housing loans' might reduce interest. 'Educational background' and 'marital status' play smaller roles but still influence the decision-making process.

Hyper tuned Model Results

We optimized our baseline models using **hyperparameter tuning** to enhance precision and recall, ensuring the models could handle data complexity more effectively.

- For **Logistic Regression**, we tuned parameters like **regularization strength (C)** and the **solver type** to optimize the model's complexity and convergence.
- For **Decision Tree**, parameters such as **tree depth**, **minimum samples required to split a node**, and **leaf size** were fine-tuned to prevent overfitting while improving recall.

- For **Random Forest**, we adjusted the **number of trees**, **tree depth**, and **features considered at each split** to achieve a more robust and stable model.
- The hyper-tuned models were evaluated using the same metrics as the baseline—**accuracy, precision, recall, and F1 score**.
- **Results After Hyperparameter Tuning:**
    - The **Logistic Regression** model saw a slight improvement in precision and recall, though the increase was marginal, suggesting that the default hyperparameters were already close to optimal.
    - The **Decision Tree** model showed improvements, particularly in **reducing overfitting** and achieving a more balanced performance between precision and recall. This was reflected in an increased **F1 score**.
    - The **Random Forest model** showed a significant boost in **recall**, which was a primary challenge in the baseline version. Adjustments to parameters like the number of features considered at each split made the model more robust in identifying positive cases.
- However, despite the hyperparameter tuning, the overall gains in accuracy and recall were not substantial compared to the baseline. In fact, some metrics, like accuracy for Random Forest, experienced slight declines. This outcome suggests that the default settings of the models were well-suited for the problem and additional complexity may have led to overfitting or over-adjustment.

Technique Used	Accuracy	Precision	Recall	F1 Score
Logistic Regression	*89.76%*	*64.05%*	*34.46%*	*44.82%*
Decision Tree	*87.96%*	*50.11%*	*43.08%*	*46.33%*
Random Forest	*90.01%*	*65.21%*	*36.94%*	*47.16%*

Surprisingly, tuning didn't improve the models. In fact, it slightly reduced accuracy and recall. The default settings were already well-suited for this model.

Conclusion

According to the performance metrics, Random Forest stands out as the key model for predicting client subscription. The results demonstrated the highest accuracy at 90.15% and an impressive F1 score, positioning it as the optimal selection, even though it exhibited a lower recall. The analysis of feature importance in Random Forest underscores the significant elements that drive the model's predictions. The length of the final interaction stands out as the key factor, reflecting client involvement during extended conversations. Additional significant elements consist of Age, Balance, Day of the last contact, and Outcome of previous campaigns. The analysis of these variables reveals important patterns in customer behavior, indicating that older clients, individuals with larger balances, and those who previously engaged effectively in campaigns show a greater likelihood of subscribing.

References

- Bhandari, A. (2024, October 9). Feature Scaling: Engineering, normalization, and Standardization (Updated 2024). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

- Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

- GeeksforGeeks. (2024a, March 21). One hot encoding in machine learning. GeeksforGeeks. https://www.geeksforgeeks.org/ml-one-hot-encoding/

- Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

- Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness, and correlation. Journal of Machine Learning Technologies, 2(1), 37–63.

- UCI Machine Learning Repository. (n.d.). https://archive.ics.uci.edu/dataset/222/bank+marketing

- Udemy (2024). Complete Machine Learning & Data Science Bootcamp 2024. Retrieved from https://www.udemy.com

- Udemy (2024). Python for Data Science and Machine Learning Bootcamp. Retrieved from https://www.udemy.com

- Zhang, Z., & Zhan, D. (2017). A review of hyperparameter tuning methods in machine learning. IEEE Access, 5, 21224–21232.

Appendix A: Baseline Model Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# loading the cleaned dataset from an Excel file.
data = pd.read_excel('/Users/darpanradadiya/Downloads/cleaned_data.xlsx')


# This function detects outliers in numerical columns and caps them using the IQR method to
avoid their negative effect on model performance.
def treat_outliers_iqr(df, cols):
    for col in cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # For each value in the column, cap values that exceed the upper or lower bound
        df[col] = df[col].apply(lambda x: upper_bound if x > upper_bound else (lower_bound if x <
lower_bound else x))
    return df


# List of numerical columns where we want to apply the outlier treatment
numerical_cols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
# Apply the outlier treatment to the specified numerical columns
data = treat_outliers_iqr(data, numerical_cols)


# Splitting the data into features (X) and the target (y)
# We are separating the target column 'y1' from the rest of the features for training the model.
X = data.drop(['y1', 'y'], axis=1)  # Drop 'y1' and 'y' columns (target variables) from the features
y = data['y1']  # 'y1' will be our target label (what we want to predict)
```

```
# List of categorical columns to encode
# Categorical columns cannot be directly used by machine learning models, so we need to convert them to numerical form.
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'pdays_cat']

# Applying OneHotEncoding to categorical features
# This step transforms the categorical columns into binary vectors, which models can process.
column_transformer = ColumnTransformer(
    transformers=[('cat', OneHotEncoder(drop='first'), categorical_cols)],  # OneHotEncoding, with drop='first' to avoid multicollinearity
    remainder='passthrough'  # Leave the numerical columns as they are
)


# Applying the transformation to our dataset
# The transformed features (X_transformed) now have categorical columns encoded into numerical form.
X_transformed = column_transformer.fit_transform(X)


# Splitting the dataset into training and testing sets (80% training, 20% testing)
# This split allows us to train the model on one part of the data and test its performance on the other part.
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.2, random_state=42)


# Applying StandardScaler to normalize feature values
# Standardizing the features ensures that the scale of the features doesn’t affect model performance (e.g., features with larger values won't dominate).
scaler = StandardScaler(with_mean=False)  # We set with_mean=False because the transformed data is sparse (many zeros)
X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform the training data
X_test_scaled = scaler.transform(X_test)  # Transform the test data using the same scaling parameters


### Logistic Regression without Hyperparameter Tuning
# We initialize a Logistic Regression model with default settings.
log_reg = LogisticRegression(random_state=42, max_iter=5000)
log_reg.fit(X_train_scaled, y_train)  # Fit the model on the training data
y_pred_log_reg = log_reg.predict(X_test_scaled)  # Predict on the test data


# Evaluating Logistic Regression performance
```

```
# We calculate the accuracy, precision, recall, and F1 score for the Logistic Regression model.
accuracy_log_no_tuning = accuracy_score(y_test, y_pred_log_reg)
precision_log_no_tuning = precision_score(y_test, y_pred_log_reg)
recall_log_no_tuning = recall_score(y_test, y_pred_log_reg)
f1_log_no_tuning = f1_score(y_test, y_pred_log_reg)


### Decision Tree without Hyperparameter Tuning
# We initialize a Decision Tree classifier with default settings.
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)  # Fit the model on the training data
y_pred_dt = dt.predict(X_test)  # Predict on the test data

# Evaluating Decision Tree performance
# We calculate the accuracy, precision, recall, and F1 score for the Decision Tree model.
accuracy_dt_no_tuning = accuracy_score(y_test, y_pred_dt)
precision_dt_no_tuning = precision_score(y_test, y_pred_dt)
recall_dt_no_tuning = recall_score(y_test, y_pred_dt)
f1_dt_no_tuning = f1_score(y_test, y_pred_dt)

### Random Forest without Hyperparameter Tuning
# We initialize a Random Forest classifier with default settings.
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)  # Fit the model on the training data
y_pred_rf = rf.predict(X_test)  # Predict on the test data

# Evaluating Random Forest performance
# We calculate the accuracy, precision, recall, and F1 score for the Random Forest model.
accuracy_rf_no_tuning = accuracy_score(y_test, y_pred_rf)
precision_rf_no_tuning = precision_score(y_test, y_pred_rf)
recall_rf_no_tuning = recall_score(y_test, y_pred_rf)
f1_rf_no_tuning = f1_score(y_test, y_pred_rf)

# Displaying performance metrics for all models without tuning
# We collect the results of all three models and their performance metrics (accuracy, precision,
recall, F1 score).
results_no_tuning = {
   'Logistic Regression (No Tuning)': {'Accuracy': accuracy_log_no_tuning, 'Precision':
precision_log_no_tuning, 'Recall': recall_log_no_tuning, 'F1 Score': f1_log_no_tuning},
   'Decision Tree (No Tuning)': {'Accuracy': accuracy_dt_no_tuning, 'Precision':
precision_dt_no_tuning, 'Recall': recall_dt_no_tuning, 'F1 Score': f1_dt_no_tuning},
   'Random Forest (No Tuning)': {'Accuracy': accuracy_rf_no_tuning, 'Precision':
precision_rf_no_tuning, 'Recall': recall_rf_no_tuning, 'F1 Score': f1_rf_no_tuning}
}

print("Results for models without hyperparameter tuning:")
```

```
print(results_no_tuning)
```

Appendix B: Hyper tuned model code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the cleaned dataset from an Excel file, so we can begin analysis.
data = pd.read_excel('/Users/darpanradadiya/Downloads/cleaned_data.xlsx')

# Outlier treatment function using IQR (Interquartile Range)
# This function identifies and caps outliers based on the IQR method
# It prevents extreme values from impacting our models.
def treat_outliers_iqr(df, cols):
    for col in cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df[col] = df[col].apply(lambda x: upper_bound if x > upper_bound else (lower_bound if x <
lower_bound else x))
    return df


# Numerical columns that will undergo outlier treatment.
numerical_cols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
data = treat_outliers_iqr(data, numerical_cols)


# Splitting the data into features (X) and target (y).
# Here we are separating our features and the target variable 'y1'.
X = data.drop(['y1', 'y'], axis=1)  # Drop target and additional 'y' column
y = data['y1']  # Target variable
```

```
# List of categorical columns for OneHotEncoding
# We need to convert these categorical columns into a format machine learning models can work with.
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'pdays_cat']


# Encoding categorical columns using OneHotEncoding
# We use OneHotEncoder to transform categorical features into binary vectors.
# ColumnTransformer helps us apply this transformation only to the categorical columns while leaving the others intact.
column_transformer = ColumnTransformer(
    transformers=[('cat', OneHotEncoder(drop='first'), categorical_cols)],  # Avoid multicollinearity by dropping the first category
    remainder='passthrough'  # Keep numerical columns as they are
)


# Apply the transformations to the feature set.
X_transformed = column_transformer.fit_transform(X)


# Splitting the data into training and testing sets (80% train, 20% test)
# This allows us to evaluate the performance of our models on unseen data.
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.2, random_state=42)


# Scaling the features using StandardScaler
# Standardization is important, especially for models like Logistic Regression, which assume normally distributed data.
# with_mean=False is used because our transformed data is sparse (many 0s).
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


### Logistic Regression with Hyperparameter Tuning
# We define a dictionary of hyperparameters to tune for our logistic regression model.
param_dist_log_reg = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularization strength
    'solver': ['liblinear', 'saga'],  # Solvers to find the optimal solution
    'penalty': ['l1', 'l2'],  # Type of regularization (L1 for sparse data, L2 for ridge regression)
    'max_iter': [5000]  # Maximum iterations to ensure convergence
}
```

```
# RandomizedSearchCV tries different combinations of hyperparameters randomly.
# It helps find the best-performing model by testing a few combinations instead of all possibilities.
log_reg_random_search = RandomizedSearchCV(LogisticRegression(random_state=42), param_distributions=param_dist_log_reg, n_iter=10, cv=3, scoring='f1', n_jobs=-1, random_state=42)
log_reg_random_search.fit(X_train_scaled, y_train)

# Making predictions on the test set using the best Logistic Regression model.
y_pred_log_reg_tuned = log_reg_random_search.predict(X_test_scaled)


# Evaluating the performance of Logistic Regression using accuracy, precision, recall, and F1 score.
accuracy_log = accuracy_score(y_test, y_pred_log_reg_tuned)
precision_log = precision_score(y_test, y_pred_log_reg_tuned)
recall_log = recall_score(y_test, y_pred_log_reg_tuned)
f1_log = f1_score(y_test, y_pred_log_reg_tuned)


### Decision Tree with Hyperparameter Tuning
# Define hyperparameters to tune for the decision tree model.
param_dist_dt = {
    'max_depth': [10, 20, 30, 40, 50, None],  # Depth of the tree to prevent overfitting
    'min_samples_split': [2, 5, 10],  # Minimum samples needed to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples at each leaf node
    'max_features': ['sqrt', 'log2', None],  # Number of features to consider when looking for the best split
    'criterion': ['gini', 'entropy']  # Criterion for node splitting
}


# Tuning the Decision Tree using RandomizedSearchCV, similar to the logistic regression.
dt_random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42), param_distributions=param_dist_dt, n_iter=50, cv=5, scoring='f1', n_jobs=-1, random_state=42)
dt_random_search.fit(X_train, y_train)


# Making predictions on the test set using the best Decision Tree model.
y_pred_dt_tuned = dt_random_search.predict(X_test)


# Evaluating Decision Tree performance using accuracy, precision, recall, and F1 score.
accuracy_dt = accuracy_score(y_test, y_pred_dt_tuned)
precision_dt = precision_score(y_test, y_pred_dt_tuned)
```

```
recall_dt = recall_score(y_test, y_pred_dt_tuned)
f1_dt = f1_score(y_test, y_pred_dt_tuned)


### Random Forest with Hyperparameter Tuning
# Define hyperparameters for Random Forest tuning.
param_dist_rf = {
    'n_estimators': [100, 200, 300, 400, 500],  # Number of trees in the forest
    'max_depth': [10, 20, 30, 40, 50, None],  # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples at each leaf node
    'max_features': ['sqrt', 'log2']  # Number of features to consider for the best split
}


# Tuning the Random Forest model with RandomizedSearchCV.
rf_random_search = RandomizedSearchCV(RandomForestClassifier(random_state=42),
param_distributions=param_dist_rf, n_iter=50, cv=5, scoring='f1', n_jobs=-1, random_state=42)
rf_random_search.fit(X_train, y_train)

# Making predictions on the test set using the best Random Forest model.
y_pred_rf_tuned = rf_random_search.predict(X_test)

# Evaluating Random Forest performance using accuracy, precision, recall, and F1 score.
accuracy_rf = accuracy_score(y_test, y_pred_rf_tuned)
precision_rf = precision_score(y_test, y_pred_rf_tuned)
recall_rf = recall_score(y_test, y_pred_rf_tuned)
f1_rf = f1_score(y_test, y_pred_rf_tuned)

# Storing results of all models into a dictionary for easy comparison.
results = {
    'Logistic Regression': {'Accuracy': accuracy_log, 'Precision': precision_log, 'Recall':
recall_log, 'F1 Score': f1_log},
    'Decision Tree': {'Accuracy': accuracy_dt, 'Precision': precision_dt, 'Recall': recall_dt, 'F1
Score': f1_dt},
    'Random Forest': {'Accuracy': accuracy_rf, 'Precision': precision_rf, 'Recall': recall_rf, 'F1
Score': f1_rf}
}

print("Results for models with hyperparameter tuning:")
print(results)
```

Code Results

Results for models without hyperparameter tuning:
{'Logistic Regression (No Tuning)': {'Accuracy': 0.8976003538648678, 'Precision': 0.6400679117147708, 'Recall': 0.34555453712190654, 'F1 Score': 0.4488095238095238}, 'Decision Tree (No Tuning)': {'Accuracy': 0.8712816543182572, 'Precision': 0.4662349676225717, 'Recall': 0.461961503208066, 'F1 Score': 0.46408839779005523}, 'Random Forest (No Tuning)': {'Accuracy': 0.9014707508570164, 'Precision': 0.6612903225806451, 'Recall': 0.3758020164986251, 'F1 Score': 0.4792518994739918}}

Results for models with hyperparameter tuning:
{'Logistic Regression': {'Accuracy': 0.8976003538648678, 'Precision': 0.6405451448040886, 'Recall': 0.3446379468377635, 'F1 Score': 0.44815256257449343}, 'Decision Tree': {'Accuracy': 0.8795753621585757, 'Precision': 0.5010660980810234, 'Recall': 0.4307974335472044, 'F1 Score': 0.4632824051256777}, 'Random Forest': {'Accuracy': 0.9001437576025655, 'Precision': 0.6521035598705501, 'Recall': 0.3693858845096242, 'F1 Score': 0.4716208308952604}}