BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604 Machine Learning

Coursework Report

RGU Student ID	**:**	**2409662**
IIT Student ID	**:**	**20230214**
Student Name	**:**	**Gagani Kulathilaka**

GitHub Repository Link:

GaganiKulathilaka/Customer-Subscription-Prediction

Table of Contents

1 Introduction

This report presents the methods utilized for addressing the coursework's classification issue, which involved predicting, using the provided dataset, whether the client would subscribe for a bank term deposit. Machine learning algorithms such as random forest classification and neural networks were to be used to make the prediction. This study provides a justification for the implementation of machine learning techniques, feature selection, and data preprocessing to address the issue. The prerequisites of implementing this program is as below :

1. Python programming
2. Exploratory Data Analysis (EDA)
3. Data pre-processing techniques
4. Data cleaning techniques
5. Data imputation methods
6. Feature selection and engineering techniques

2 Corpus Preparation

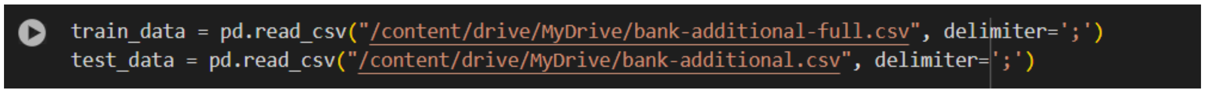
2.1 Downloading the dataset

The hyperlink provided in the coursework specification led to the website (`https://archive.ics.uci.edu/dataset/222/bank%2Bmarketing`) containing a dataset related with marketing campaigns of a banking institution. The zip file included two sub folders called 'bank' folder contained an older version of the dataset while the 'bank-additional' contained the full dataset. The files contained in the 'bank-additional' folder is as below :

1. bank-additional - The testing dataset
2. bank-additional-full - The training dataset
3. bank-additional-names - A description of the dataset

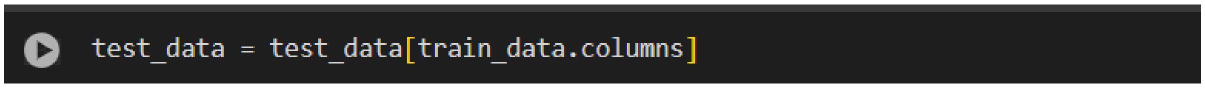
2.2 Creating Data Frames

The 'bank-additional-full' and 'bank-additional' datasets were allocated to the variables 'train_data' and 'test_data', respectively, in order to provide data frames for the given problem. The.csv files were read into the application using the 'pandas' library.



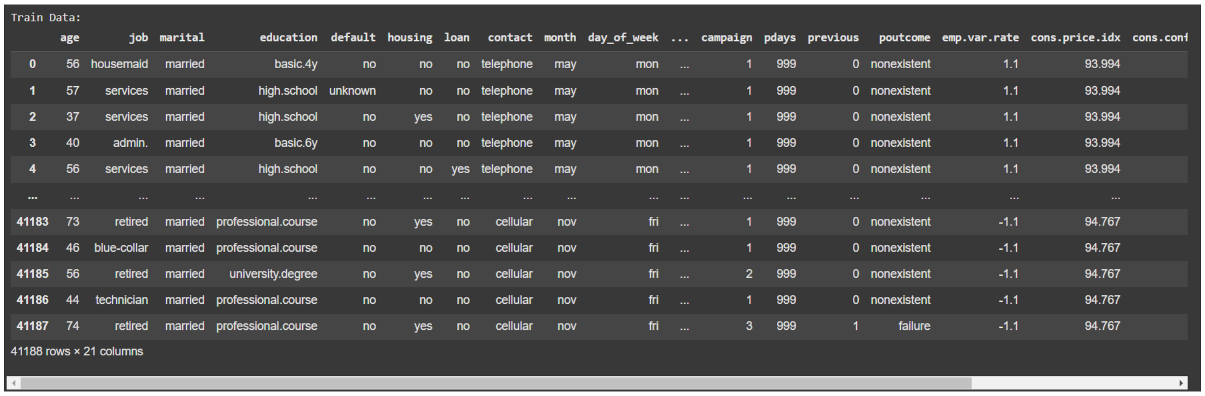
```
train_data = pd.read_csv("/content/drive/MyDrive/bank-additional-full.csv", delimiter=';')
test_data = pd.read_csv("/content/drive/MyDrive/bank-additional.csv", delimiter=';')
```

The following line of code was run to see if the columns in the two datasets are identical and aligned.



```
test_data = test_data[train_data.columns]
```

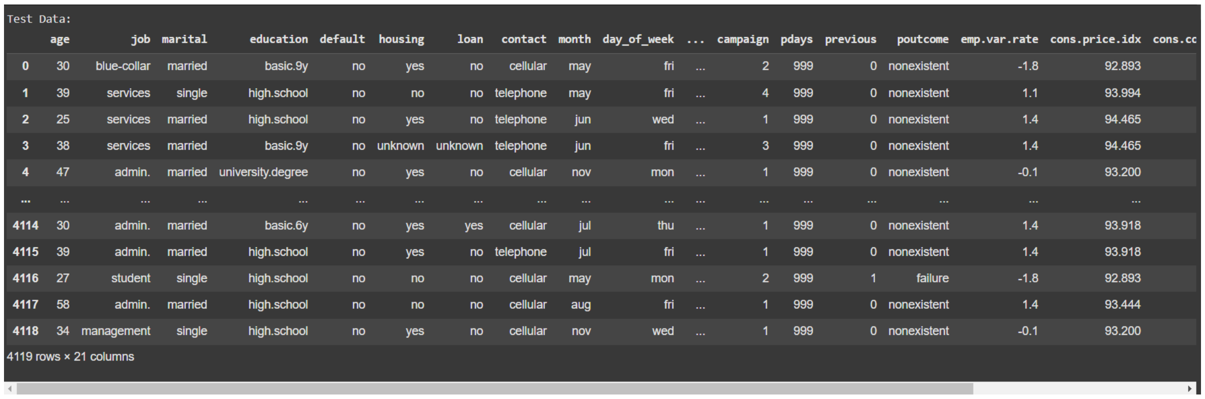
The data sets are then printed to verify that the indexing and the number of rows and columns are accurate.



Train Data:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0	nonexistent	-1.1	94.767	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1	failure	-1.1	94.767	

41188 rows × 21 columns

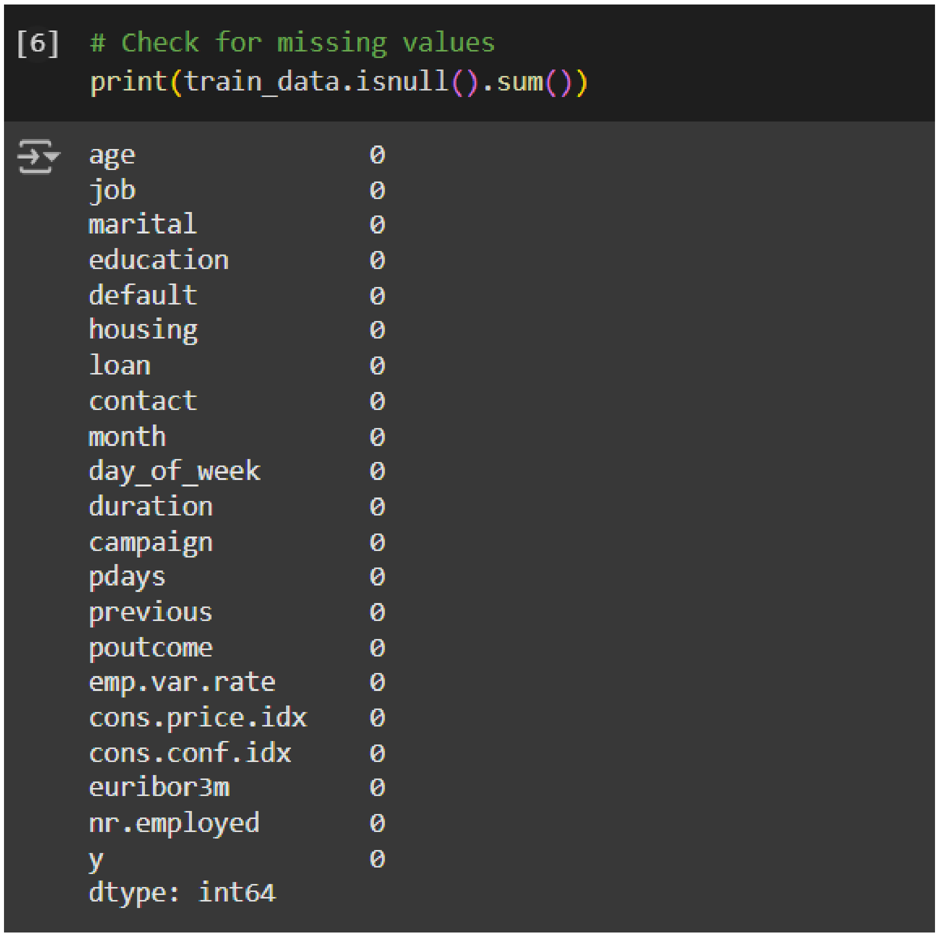


Test Data:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.cc
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	...	2	999	0	nonexistent	-1.8	92.893	
1	39	services	single	high.school	no	no	no	telephone	may	fri	...	4	999	0	nonexistent	1.1	93.994	
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	...	1	999	0	nonexistent	1.4	94.465	
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	...	3	999	0	nonexistent	1.4	94.465	
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	...	1	999	0	nonexistent	-0.1	93.200	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	...	1	999	0	nonexistent	1.4	93.918	
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	...	1	999	0	nonexistent	1.4	93.918	
4116	27	student	single	high.school	no	no	no	cellular	may	mon	...	2	999	1	failure	-1.8	92.893	
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	...	1	999	0	nonexistent	1.4	93.444	
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	...	1	999	0	nonexistent	-0.1	93.200	

4119 rows × 21 columns

The dataset is then examined for null values.



```
[6] # Check for missing values
    print(train_data.isnull().sum())
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

2.3 Exploratory Data Analysis

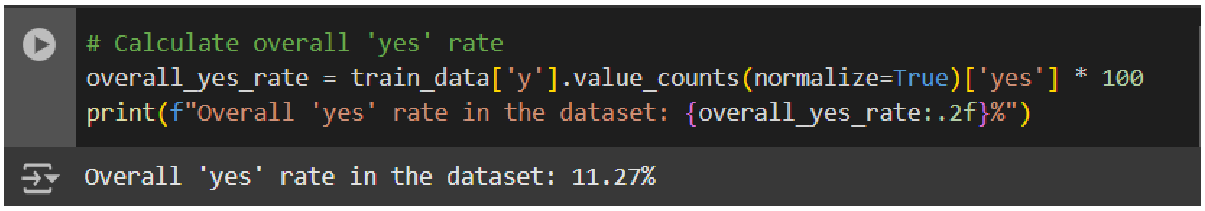
Exploratory Data Analysis (EDA) is the process of analyzing and summarizing datasets to uncover patterns, relationships, anomalies, and other insights, often using statistical and visual techniques.

2.3.1 Handling missing values

This data set is examined for data with the value 'unknown' since the output above shows no null values.

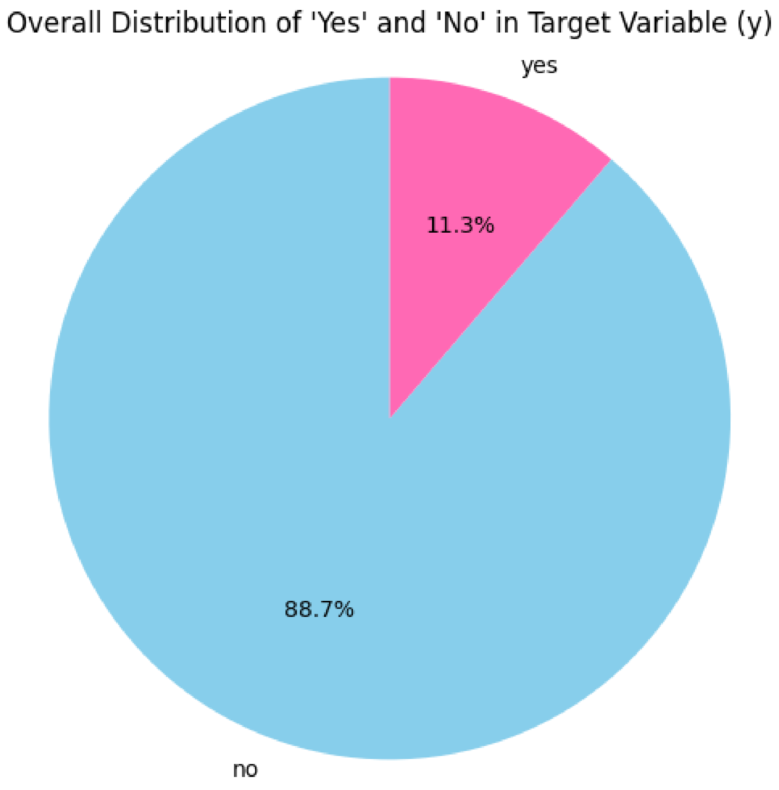
Comparing the impact of the columns with null values on the target variable (y) was the methodology employed to deal with these data.

The overall 'yes' rate for the target variable is first determined.



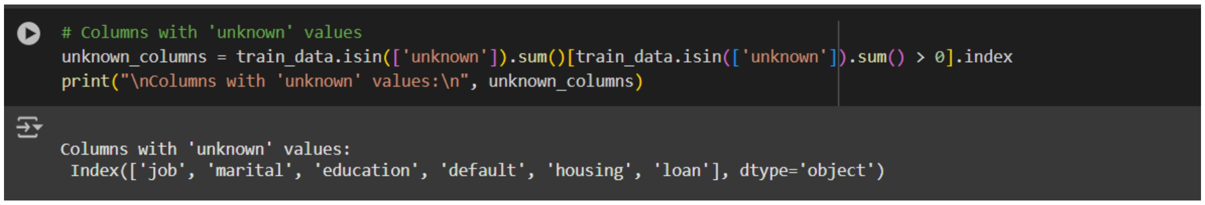
```
# Calculate overall 'yes' rate
overall_yes_rate = train_data['y'].value_counts(normalize=True)['yes'] * 100
print(f"Overall 'yes' rate in the dataset: {overall_yes_rate:.2f}%")
```

```
Overall 'yes' rate in the dataset: 11.27%
```



The output above indicates that the total ”yes” rate is 11.27%.The pie chart shows a severe imbalance of the outcome towards one category which could be handled by dropping the columns with the data with the values ’unknown’ or imputing them suing simple imputation methods.

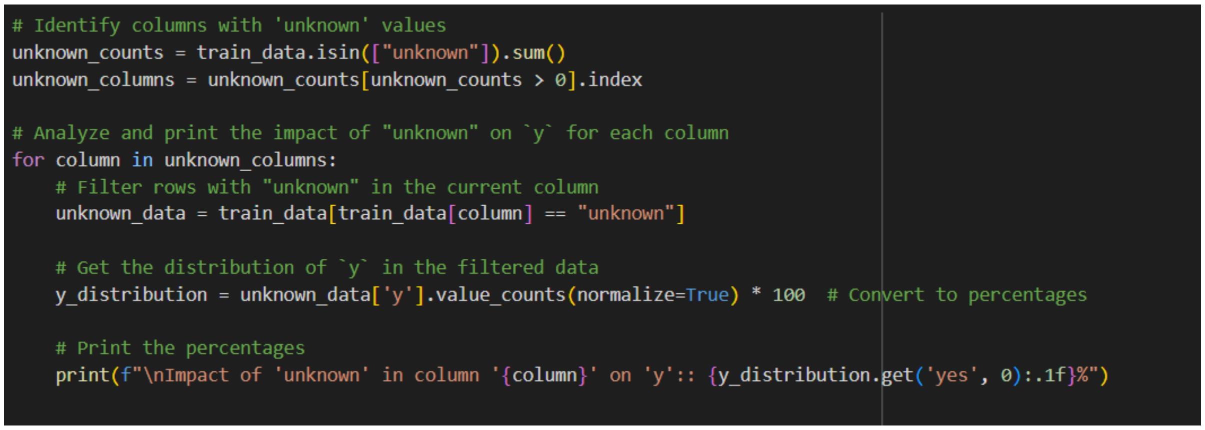
The impact of the column on the target variable is then computed once the columns containing data with the value ”unknown” are located.



```
# Columns with 'unknown' values
unknown_columns = train_data.isin(['unknown']).sum()[train_data.isin(['unknown']).sum() > 0].index
print("\nColumns with 'unknown' values:\n", unknown_columns)
```

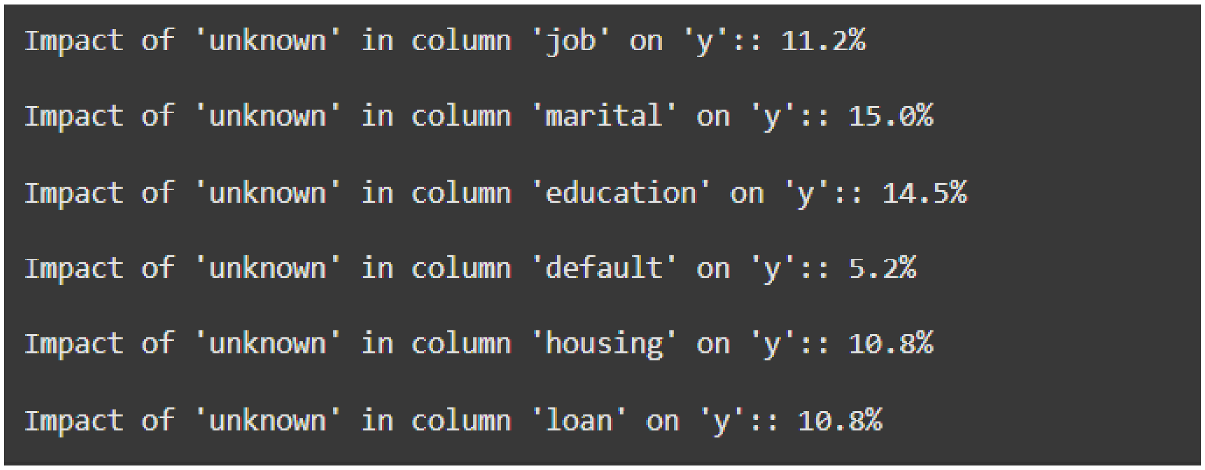
```
Columns with 'unknown' values:
 Index(['job', 'marital', 'education', 'default', 'housing', 'loan'], dtype='object')
```

The software determines the columns’ impact on the target variable’s result after identifying them using a for loop.



```
# Identify columns with 'unknown' values
unknown_counts = train_data.isin(["unknown"]).sum()
unknown_columns = unknown_counts[unknown_counts > 0].index

# Analyze and print the impact of "unknown" on `y` for each column
for column in unknown_columns:
    # Filter rows with "unknown" in the current column
    unknown_data = train_data[train_data[column] == "unknown"]

    # Get the distribution of `y` in the filtered data
    y_distribution = unknown_data['y'].value_counts(normalize=True) * 100  # Convert to percentages

    # Print the percentages
    print(f"\nImpact of 'unknown' in column '{column}' on 'y':: {y_distribution.get('yes', 0):.1f}%")
```

Following is the obtained results by running the above code snippet :



```
Impact of 'unknown' in column 'job' on 'y':: 11.2%

Impact of 'unknown' in column 'marital' on 'y':: 15.0%

Impact of 'unknown' in column 'education' on 'y':: 14.5%

Impact of 'unknown' in column 'default' on 'y':: 5.2%

Impact of 'unknown' in column 'housing' on 'y':: 10.8%

Impact of 'unknown' in column 'loan' on 'y':: 10.8%
```

We may conclude from the analysis of the above results that the influence of the column towards the target variable is between 5% and 15%.

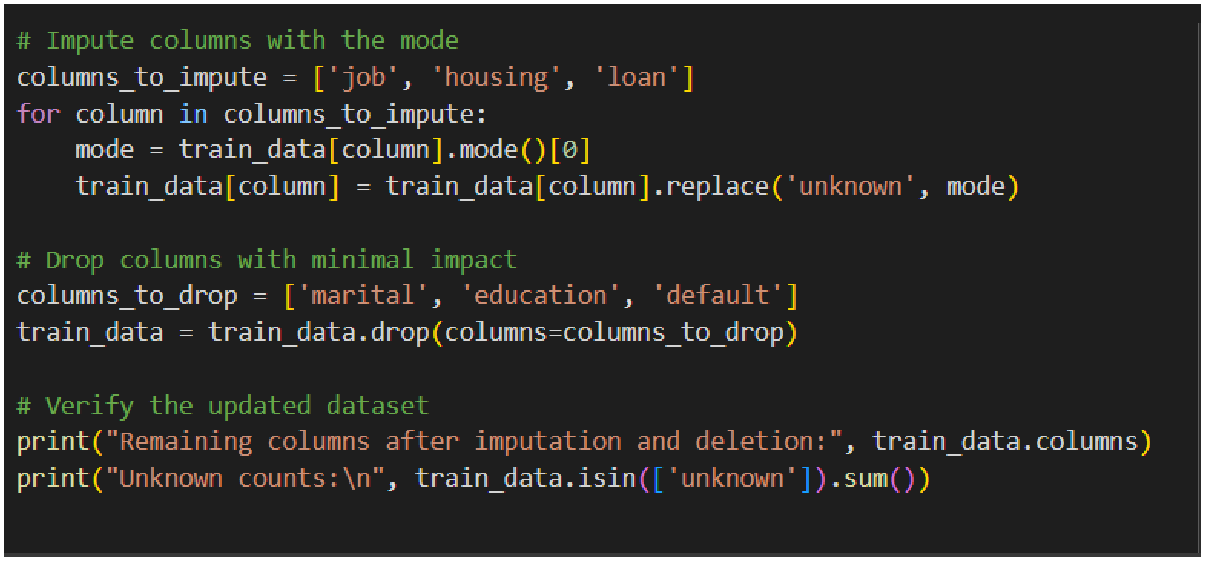
The methodology utilized for dealing with these 'unknown' values is to compare the percentage obtained to the overall 'yes' rate of the target variable.

If the columns that received values that were fairly similar to the target variable's overall "yes" value (11.27%) are imputed because they might have hidden relationships with the target variable, while the columns that deviate from the target variable's overall "yes" value are removed from the dataset.

The "marital," "education," and "default" columns that deviate from the target variable's overall "yes" value of 11.27% were removed based on the approach used and results obtained, while the "job," "housing," and "loan" columns that have values identical to 11.27% were imputed.

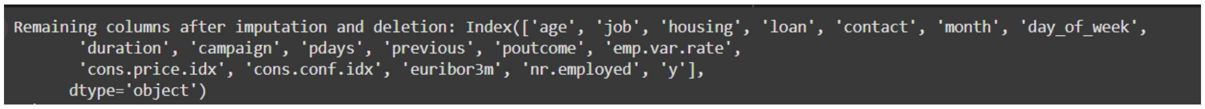
The process of replacing missing values in a dataset so that it can be utilized efficiently for modeling or analysis is known as imputation. Imputation techniques include utilizing the mean, mode, and median to replace the missing data.

Since the columns needed to be imputed in our data set contains of categorical data, imputation with mode was utilized. This involves replacing the missing values of a column with the most frequently occurring value (the mode) of that column.

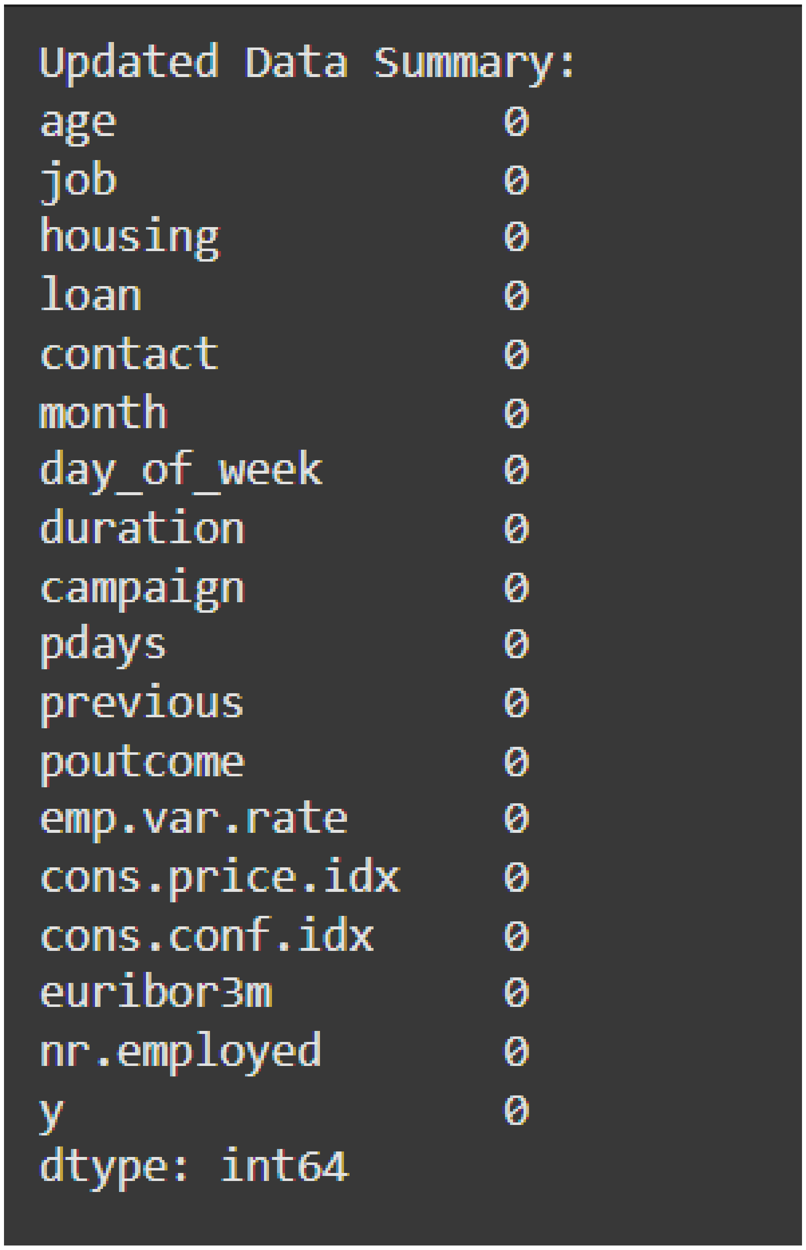


```
# Impute columns with the mode
columns_to_impute = ['job', 'housing', 'loan']
for column in columns_to_impute:
    mode = train_data[column].mode()[0]
    train_data[column] = train_data[column].replace('unknown', mode)

# Drop columns with minimal impact
columns_to_drop = ['marital', 'education', 'default']
train_data = train_data.drop(columns=columns_to_drop)

# Verify the updated dataset
print("Remaining columns after imputation and deletion:", train_data.columns)
print("Unknown counts:\n", train_data.isin(['unknown']).sum())
```

Below are the remaining columns of the dataset after handing the null and missing values.

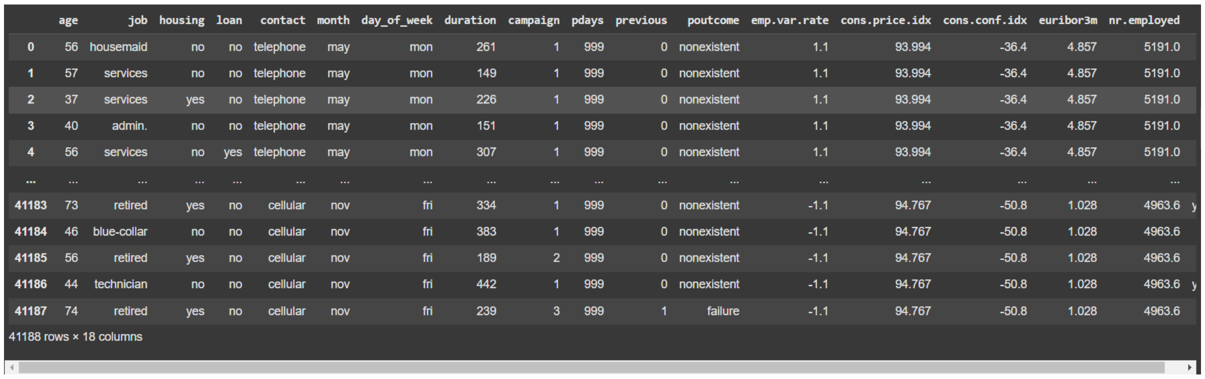


```
Remaining columns after imputation and deletion: Index(['age', 'job', 'housing', 'loan', 'contact', 'month', 'day_of_week',
       'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```



```
Updated Data Summary:
age               0
job               0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

The dataset appears as follow after the columns have been deleted and imputated.



	age	job	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	
0	56	housemaid	no	no	telephone	may	mon	261	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	
1	57	services	no	no	telephone	may	mon	149	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	
2	37	services	yes	no	telephone	may	mon	226	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	
3	40	admin.	no	no	telephone	may	mon	151	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	
4	56	services	no	yes	telephone	may	mon	307	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	yes	no	cellular	nov	fri	334	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	y
41184	46	blue-collar	no	no	cellular	nov	fri	383	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	
41185	56	retired	yes	no	cellular	nov	fri	189	2	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	
41186	44	technician	no	no	cellular	nov	fri	442	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	y
41187	74	retired	yes	no	cellular	nov	fri	239	3	999	1	failure	-1.1	94.767	-50.8	1.028	4963.6	

41188 rows × 18 columns

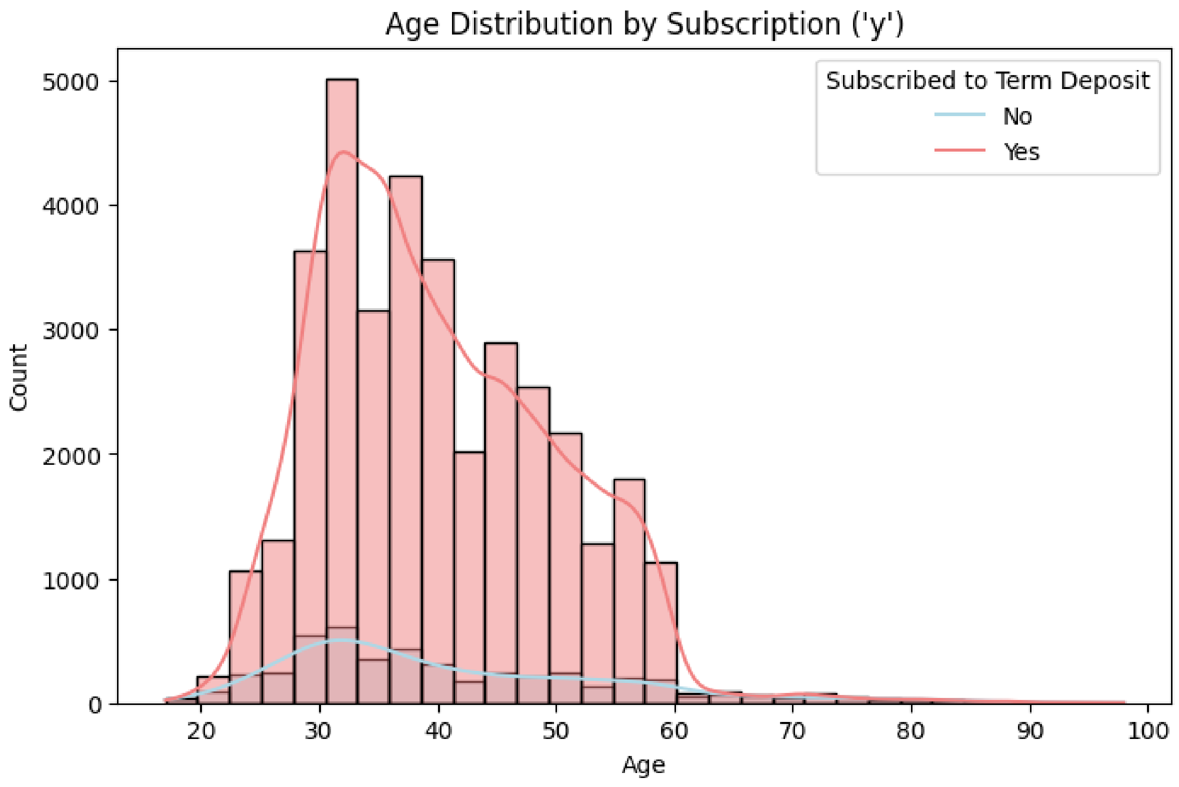
As visualized the number of columns have decreased to 18 but the number of rows is the same.

2.3.2 Graphical Representations

After handling the null and missing values plot are drawn to visualize relationships within the data and distributions of variables.

Impact of Age on Subscription

The first plot drawn is a histogram to showing the distribution of 'age' segmented by the target variable 'y'. This will help us understand how age is distributed and its impact on the likelihood of subscription.



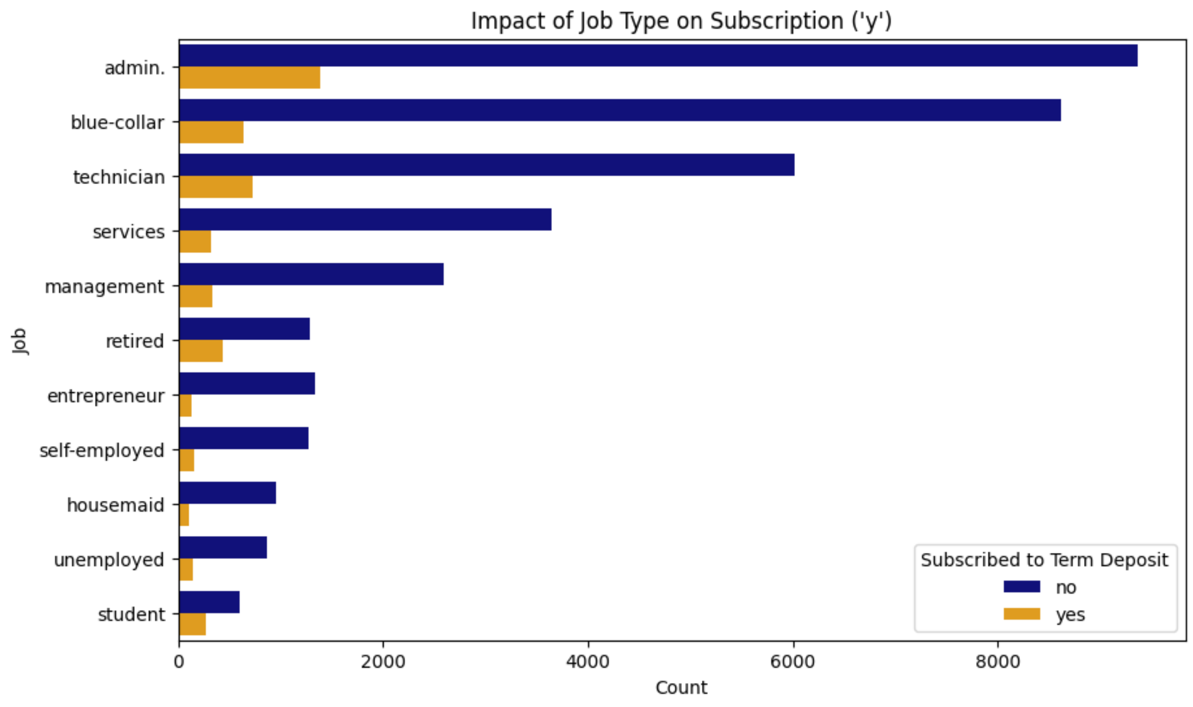
When analyzing the above histogram we could conclude that the distribution gets a bell-shaped curve.

The histogram shows a higher density of non-subscribers ('No') across all age groups, peaking between ages 30 to 40.

The overlaid density plots highlight that subscribers ('Yes') are proportionally higher in older age groups, particularly between 30 and 60, suggesting that age is a potential factor influencing subscription decisions.

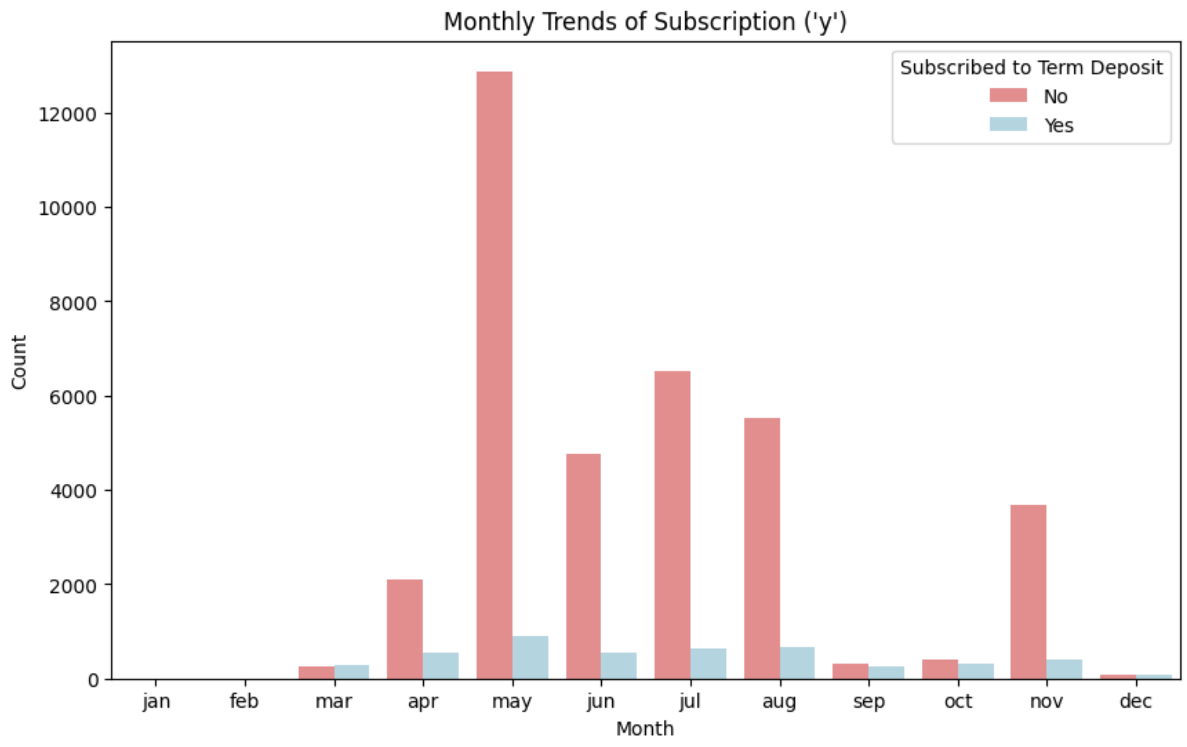
Impact of Job Type on Subscription

The second plot drawn is a bar chart to showing the distribution of job type' segmented by the target variable 'y'. This will help us understand the job type impact on the likelihood of subscription.



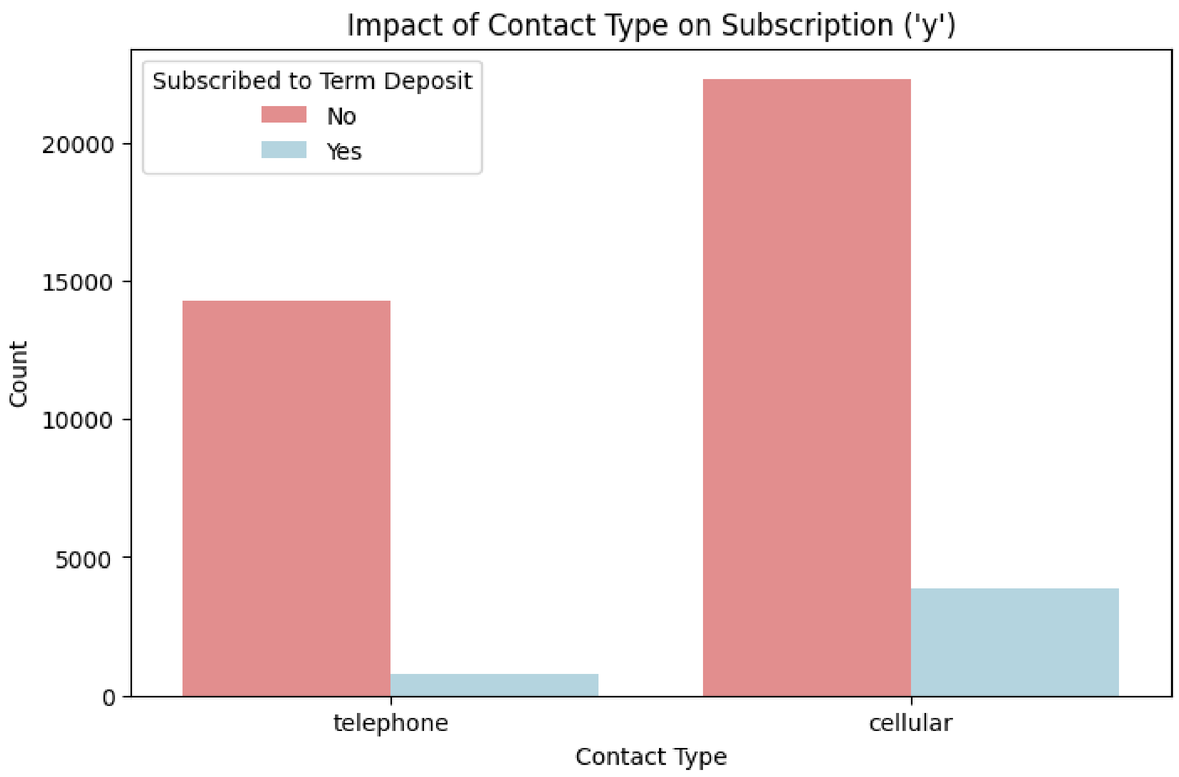
Analyzing the figure above, we can observe that subscription rates differ by type of job, with management, student, and retired categories displaying a comparatively greater percentage of yes outcomes than other job categories like housemaid and blue-collar.

Impact of month on Subscription



The above plot illustrates the monthly distribution of term deposit subscriptions. Non-subscribers dominate across all months, with a significant spike in May, likely indicating seasonality or targeted campaigns. Subscriptions are relatively more consistent across months, though they also see slight increases in May and other months such as August and November, reflecting possible trends in customer engagement during those periods.

Impact of contact type on Subscription



The number of subscribers is comparatively higher when the contact is made via cellular than when it is made via telephone, allowing us to clearly observe a difference in subscription rates when examining the plot above.

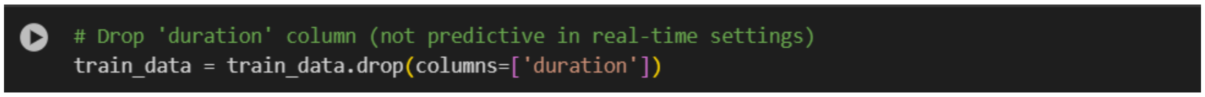
In conclusion, all these attributes have a significant impact on the target variable and analysing the patterns between these attributes and the target variable will give more insight about the distribution of the target variable.

2.4 Feature Selection and Engineering

Feature selection is the process of selecting the most important and pertinent variables (features) from the dataset to incorporate into the model. This seeks to improve the model's performance and lessen overfitting. This process also includes removing attributes which might negatively affect the outcome of the target variable.

As stated in the text file, the column ”duration” should be eliminated because its value is unknown prior to the execution of the program, making it unsuitable for use in developing a realistic prediction model.

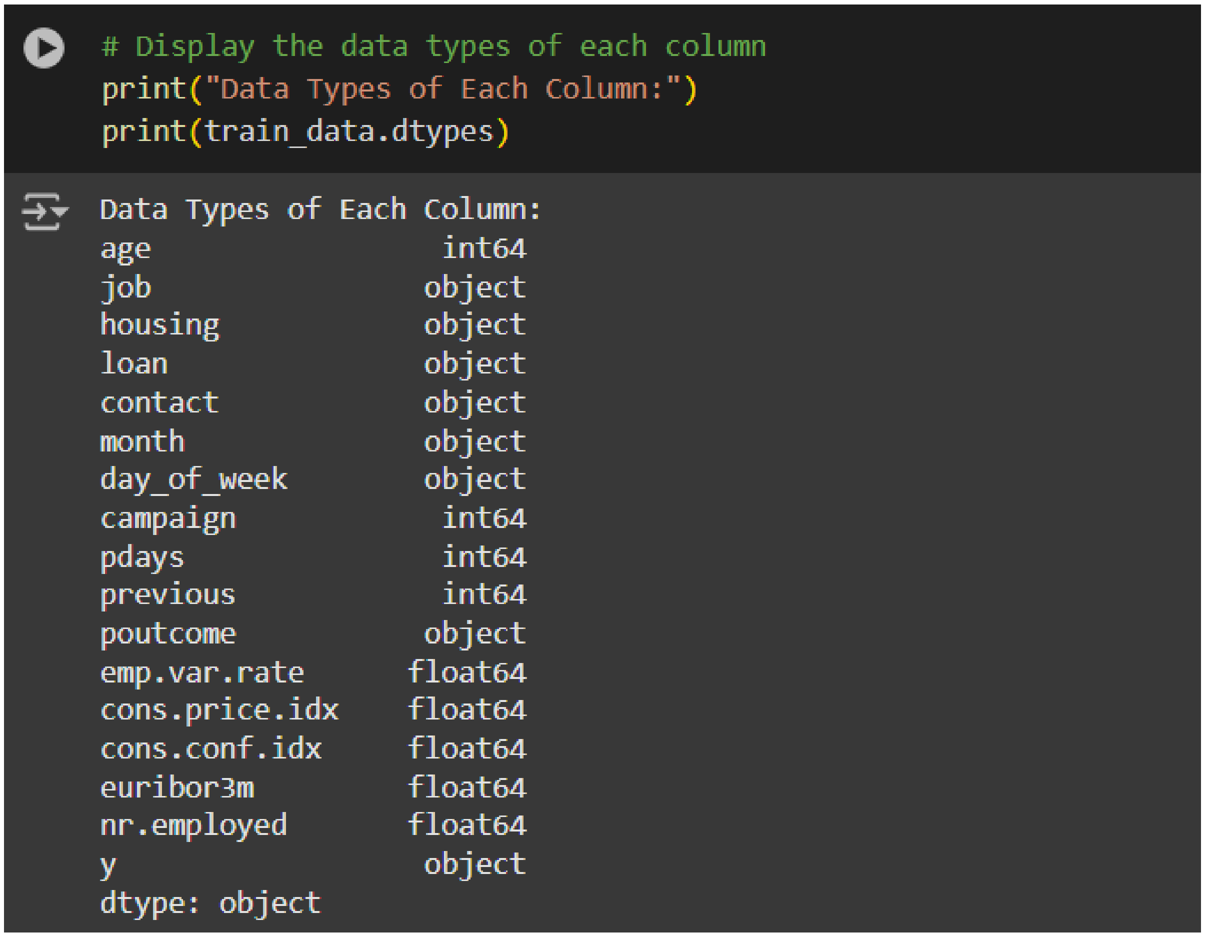
The code snippet below was run to eliminate the column from the dataset.



```
# Drop 'duration' column (not predictive in real-time settings)
train_data = train_data.drop(columns=['duration'])
```

Following the removal of the ”duration” column, the remaining columns are analyzed to determine their impact on the target variable's result.

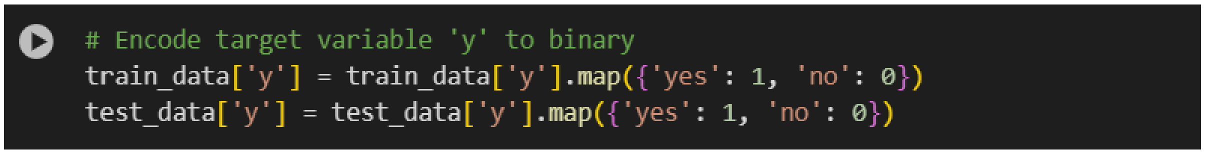
To obtain this information, we must first determine the data type of each column, for which the code snippet below is executed.



```
# Display the data types of each column
print("Data Types of Each Column:")
print(train_data.dtypes)
```

```
Data Types of Each Column:
age                 int64
job                object
housing            object
loan               object
contact            object
month              object
day_of_week        object
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
```

The data type ’object’ is referred for categorical variables while the data type ’int64’ and ’float64’ is referred for numerical attributes.
After identifying the data types the values of the target variable is encoded. A dictionary is used to perform this task. ’yes’ value is mapped to 1 while ’no’ is mapped to 0.

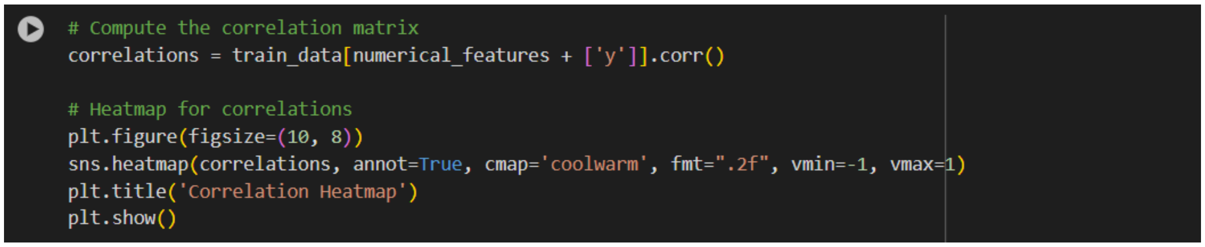


```
# Encode target variable 'y' to binary
train_data['y'] = train_data['y'].map({'yes': 1, 'no': 0})
test_data['y'] = test_data['y'].map({'yes': 1, 'no': 0})
```

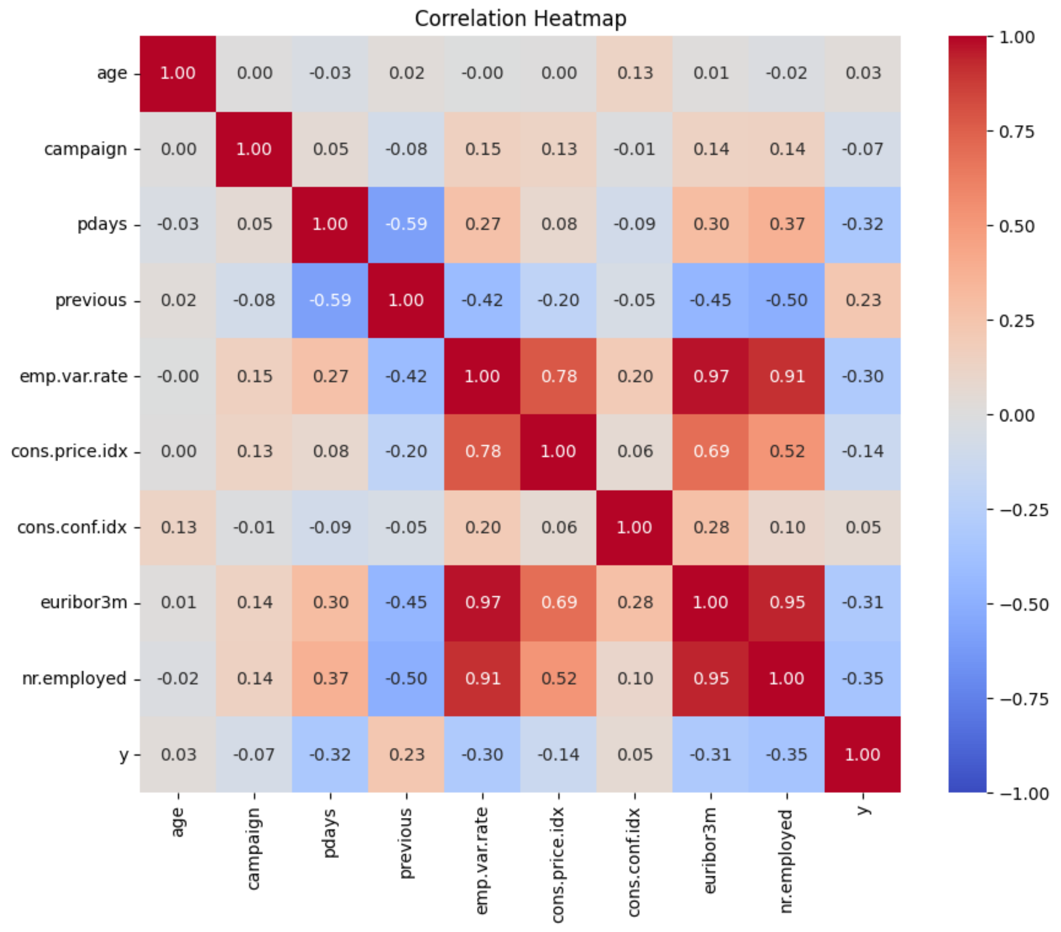
The binary encoding is essential for the machine learning models implemented as that interpret the target variable as a numerical value to calculate probabilities and optimize objectives.

2.4.1 Correlation Analysis

Prior to performing the correlation analysis on the numerical columns, a heat map was generated to visualize the correlation matrix.



```python
# Compute the correlation matrix
correlations = train_data[numerical_features + ['y']].corr()

# Heatmap for correlations
plt.figure(figsize=(10, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
age	1.00	0.00	-0.03	0.02	-0.00	0.00	0.13	0.01	-0.02	0.03
campaign	0.00	1.00	0.05	-0.08	0.15	0.13	-0.01	0.14	0.14	-0.07
pdays	-0.03	0.05	1.00	-0.59	0.27	0.08	-0.09	0.30	0.37	-0.32
previous	0.02	-0.08	-0.59	1.00	-0.42	-0.20	-0.05	-0.45	-0.50	0.23
emp.var.rate	-0.00	0.15	0.27	-0.42	1.00	0.78	0.20	0.97	0.91	-0.30
cons.price.idx	0.00	0.13	0.08	-0.20	0.78	1.00	0.06	0.69	0.52	-0.14
cons.conf.idx	0.13	-0.01	-0.09	-0.05	0.20	0.06	1.00	0.28	0.10	0.05
euribor3m	0.01	0.14	0.30	-0.45	0.97	0.69	0.28	1.00	0.95	-0.31
nr.employed	-0.02	0.14	0.37	-0.50	0.91	0.52	0.10	0.95	1.00	-0.35
y	0.03	-0.07	-0.32	0.23	-0.30	-0.14	0.05	-0.31	-0.35	1.00

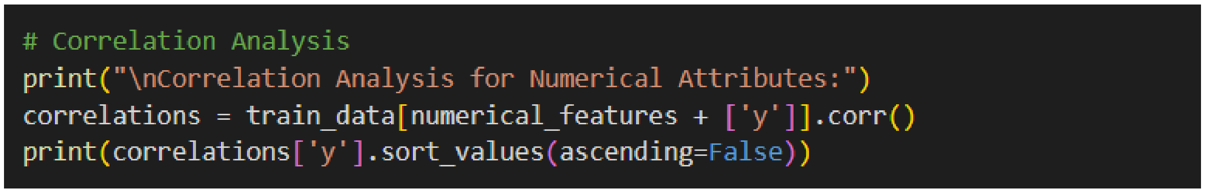
In the above heat map, Strong positive correlation are indicated by warmer colors like red while strong negative correlation are indicated by cooler colors like blue. Neutral correlation are white or light-colored.

Variables with strong correlations to the target (y) are more likely to impact predictions and are eeded for feature selection.

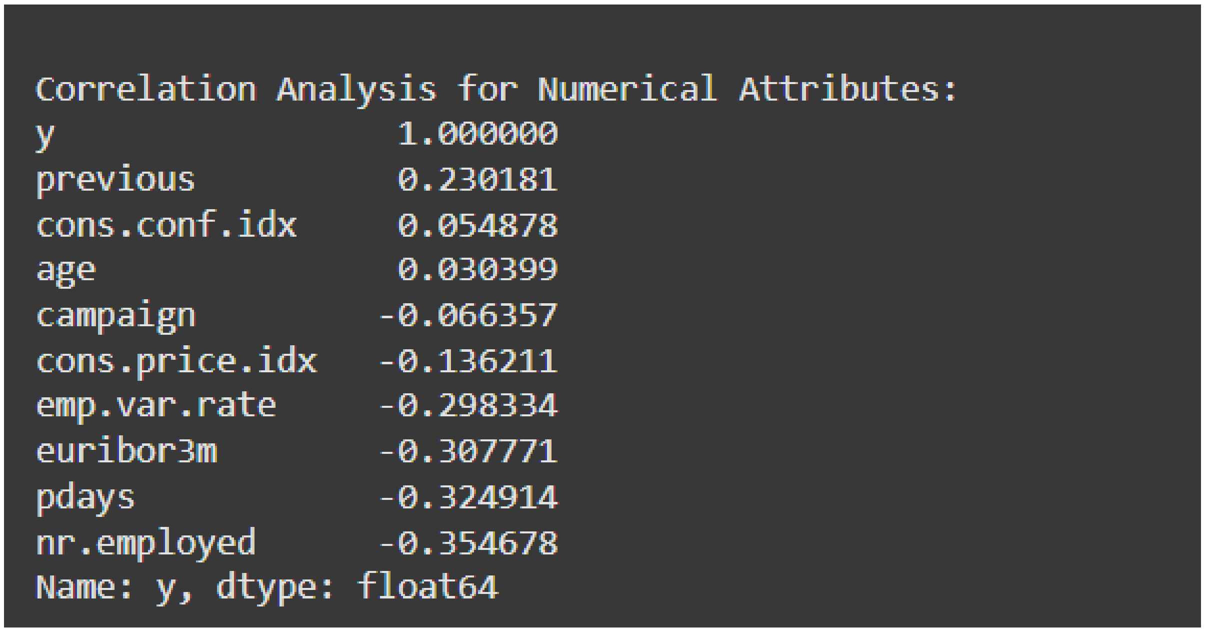
Correlation analysis measures the statistical relationship between two variables.

The correlation coefficient ranges from -1 to 1:

- 1: Perfect positive correlation
- -1: Perfect negative correlation
- 0: No correlation



```
# Correlation Analysis
print("\nCorrelation Analysis for Numerical Attributes:")
correlations = train_data[numerical_features + ['y']].corr()
print(correlations['y'].sort_values(ascending=False))
```



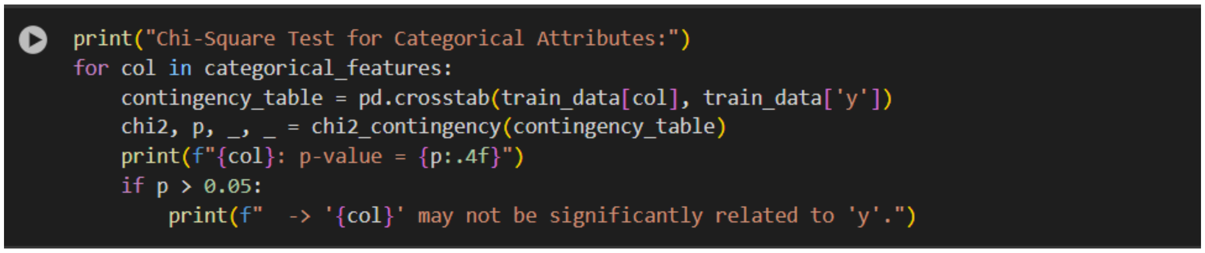
```
Correlation Analysis for Numerical Attributes:
y                 1.000000
previous          0.230181
cons.conf.idx     0.054878
age               0.030399
campaign         -0.066357
cons.price.idx   -0.136211
emp.var.rate     -0.298334
euribor3m        -0.307771
pdays            -0.324914
nr.employed      -0.354678
Name: y, dtype: float64
```

Features with very low correlation should be removed as they do not have a significant impact on the outcome. Therefore, features with —correlation— ¡ 0.05 are removed. As a result the ’age’ column is dropped from the dataset.

2.4.2 Chi-square test

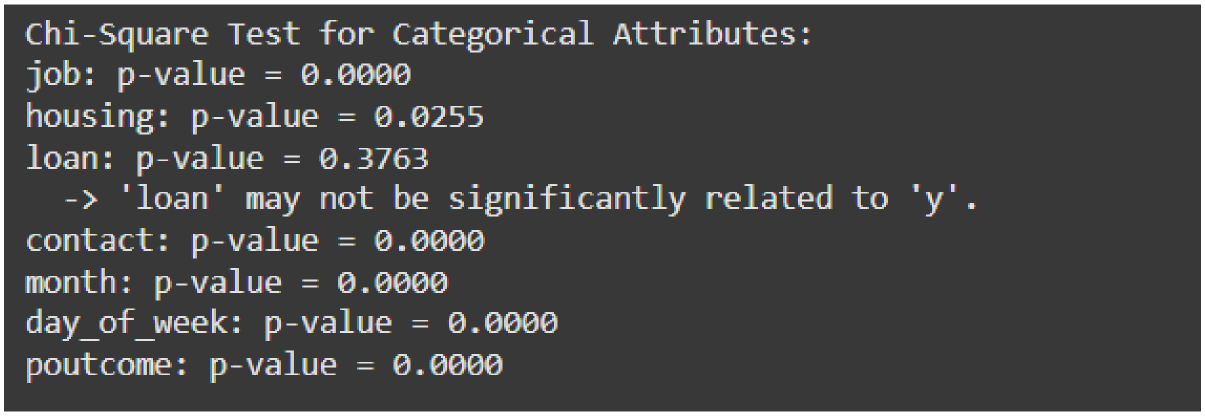
Chi-Square is a statistical technique used to determine whether two categorical variables significantly correlate with one another. It helps in determining whether one variable's distribution is independent of another.

This helps assess which categorical features are significantly related to the target variable.



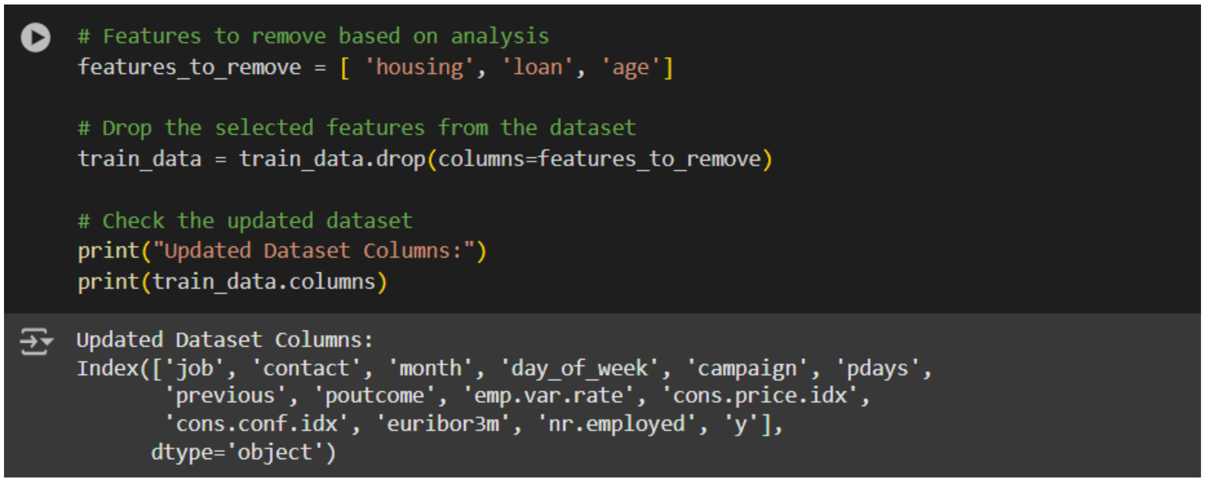
```
print("Chi-Square Test for Categorical Attributes:")
for col in categorical_features:
    contingency_table = pd.crosstab(train_data[col], train_data['y'])
    chi2, p, _, _ = chi2_contingency(contingency_table)
    print(f"{col}: p-value = {p:.4f}")
    if p > 0.05:
        print(f"  -> '{col}' may not be significantly related to 'y'.")
```

The common threshold used when performing chi-square test is p-value = 0.05. If the p-value ¿ 0.05, the null hypothesis cannot be rejected, implying that the column is likely not significantly related to the target variable hence such columns should be removed from the dataset.



```
Chi-Square Test for Categorical Attributes:
job: p-value = 0.0000
housing: p-value = 0.0255
loan: p-value = 0.3763
  -> 'loan' may not be significantly related to 'y'.
contact: p-value = 0.0000
month: p-value = 0.0000
day_of_week: p-value = 0.0000
poutcome: p-value = 0.0000
```

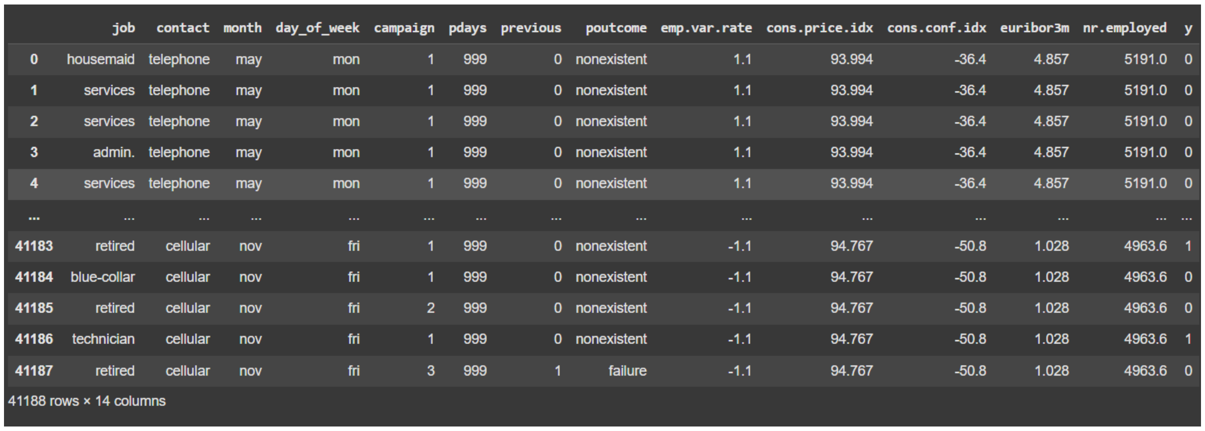
Therefore the columns 'housing' and 'loan' are removed.



```
# Features to remove based on analysis
features_to_remove = [ 'housing', 'loan', 'age']

# Drop the selected features from the dataset
train_data = train_data.drop(columns=features_to_remove)

# Check the updated dataset
print("Updated Dataset Columns:")
print(train_data.columns)
```

```
Updated Dataset Columns:
Index(['job', 'contact', 'month', 'day_of_week', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

The dataset is visualized as below after removing the columns identifed from performing correlation analysis and chi-square test.



	job	contact	month	day_of_week	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	housemaid	telephone	may	mon	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	0
1	services	telephone	may	mon	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	0
2	services	telephone	may	mon	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	0
3	admin.	telephone	may	mon	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	0
4	services	telephone	may	mon	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	retired	cellular	nov	fri	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	1
41184	blue-collar	cellular	nov	fri	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	0
41185	retired	cellular	nov	fri	2	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	0
41186	technician	cellular	nov	fri	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	1
41187	retired	cellular	nov	fri	3	999	1	failure	-1.1	94.767	-50.8	1.028	4963.6	0

41188 rows × 14 columns

2.5 Data Transformation Techniques

describe what is data transformation in a few line Data transformation is the process of converting data into a suitable format for analysis. This includes tasks like scaling, normalization and encoding. This improve the compatibility and effectiveness of machine learning models, making the data more accurate and reliable.

2.5.1 One Hot Encoding

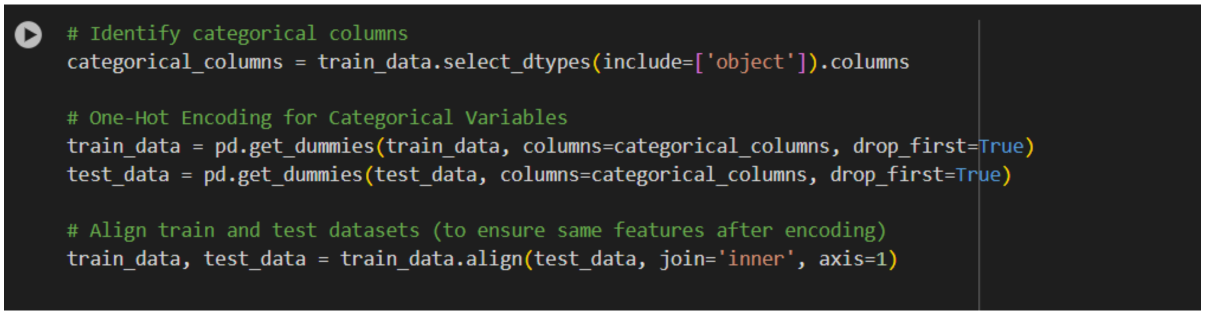
One-hot encoding is a method used to convert categorical data into a binary matrix.

For each unique category in a categorical feature, it creates a new binary column indicating the presence (1) or absence (0) of that category in the observation.

For example, the column 'poutcome' contains 3 unique categories ("failure","nonexistent","success"), one-hot encoding will create three new columns: "failure","nonexistent","success", each containing 1 or 0.

This process allows machine learning models to interpret categorical data, as most algorithms cannot work with non-numerical data.

The below code snippet is executed to perform one-hot encoding on the categorical columns.



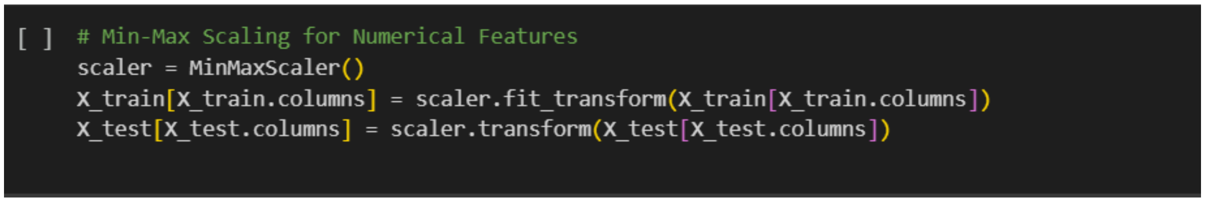
```
# Identify categorical columns
categorical_columns = train_data.select_dtypes(include=['object']).columns

# One-Hot Encoding for Categorical Variables
train_data = pd.get_dummies(train_data, columns=categorical_columns, drop_first=True)
test_data = pd.get_dummies(test_data, columns=categorical_columns, drop_first=True)

# Align train and test datasets (to ensure same features after encoding)
train_data, test_data = train_data.align(test_data, join='inner', axis=1)
```

2.5.2 Min-max Scaling

Min-Max Scaling is a normalization technique used to scale numerical data to a fixed range, typically [0, 1].

Min-Max Scaling standardizes numerical features, making them comparable in scale and suitable for the neural netowrk model which is sensitive to feature magnitudes.

The below code snippet is executed to perform min-max scaling on the numerical columns.



```
[ ] # Min-Max Scaling for Numerical Features
    scaler = MinMaxScaler()
    X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])
    X_test[X_test.columns] = scaler.transform(X_test[X_test.columns])
```

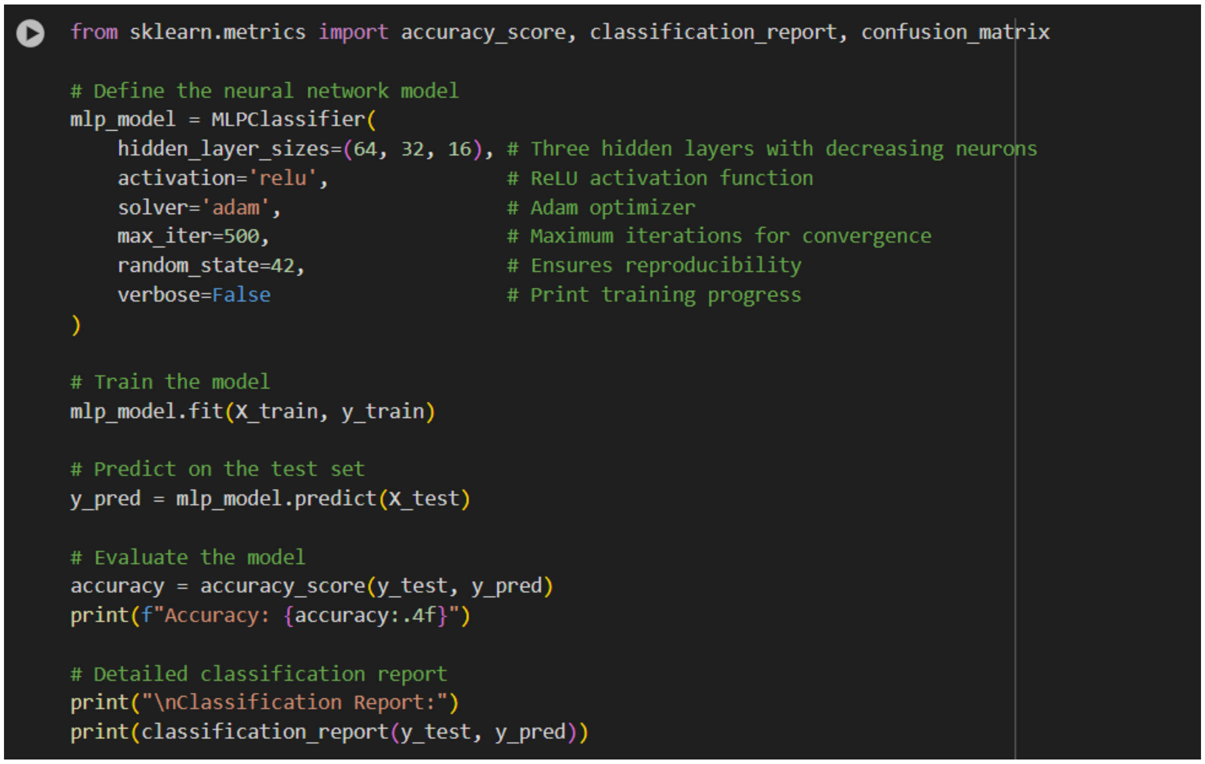
3 Solution Methodology

3.1 Neural Network Model

Neural Network Model is composed of layers of interconnected nodes (neurons), where each connection has a weight that adjusts during training to minimize prediction error.

Three primary layer types make up neural networks: input (which receives data), hidden (which processes data using non-linear transformations), and output (which generates predictions). Every node determines its output by applying an activation function.

The below code was executed to produce a neaural network model to predict the accuracy of the given dataset.



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

# Define the neural network model
mlp_model = MLPClassifier(
    hidden_layer_sizes=(64, 32, 16), # Three hidden layers with decreasing neurons
    activation='relu',               # ReLU activation function
    solver='adam',                   # Adam optimizer
    max_iter=500,                    # Maximum iterations for convergence
    random_state=42,                 # Ensures reproducibility
    verbose=False                    # Print training progress
)

# Train the model
mlp_model.fit(X_train, y_train)

# Predict on the test set
y_pred = mlp_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

This code demonstrates how to solve a classification problem by implementing a neural network classifier using ‘MLPClassifier‘ from ‘scikit-learn‘. In order to gradually acquire abstract and sophisticated features from the data, the model is first defined using three hidden layers, each of which has 64, 32, and 16 neurons.

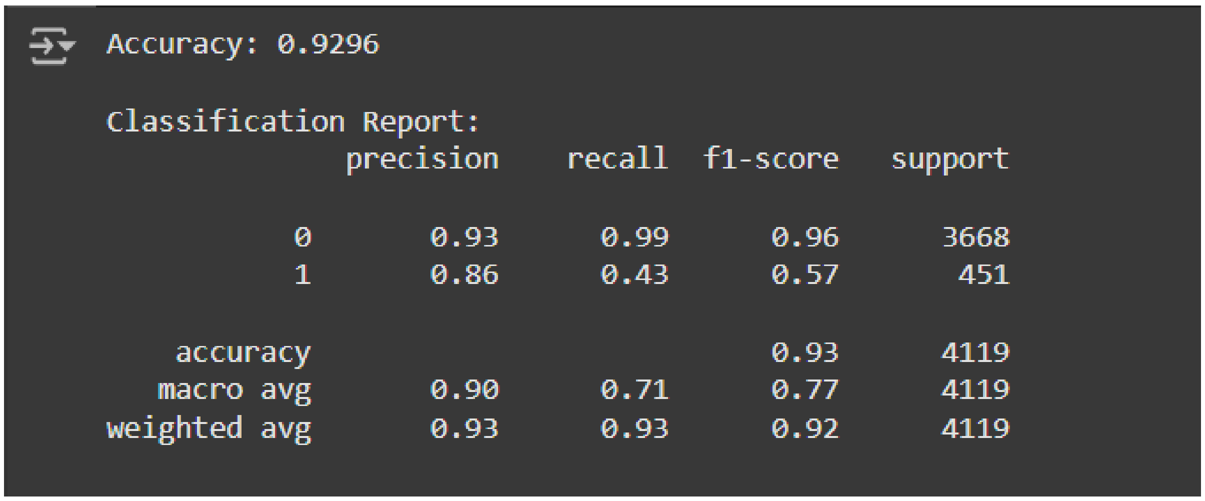
Because of its effectiveness and capacity to manage non-linear data, the Rectified Linear Unit (ReLU) activation function is utilized. To speed up training and attain faster convergence, the model uses the Adam optimizer, an adaptive optimization technique that combines the benefits of momentum and RMSProp.

By setting the maximum number of training iterations to 500, the ‘max_iter‘ option makes sure the model has enough chances to modify its weights for increased accuracy. To guarantee consistent results across runs, a random seed (‘random_state=42‘) is set, and verbose output is disabled to maintain a clean console.

The ‘fit()‘ method is used to train the model, which uses back propagation to change the weights of its neurons to learn patterns from the training dataset (‘X_train‘, ‘y_train‘). The ‘predict()‘ method, which generates predictions (‘y_pred‘) based on the test set’s characteristics, is used to test the model on unknown data (‘X_test‘) once it has been trained.

The ‘accuracy_score()‘ function provides a broad measure of model success by calculating the ratio of correct predictions to the total number of test samples.

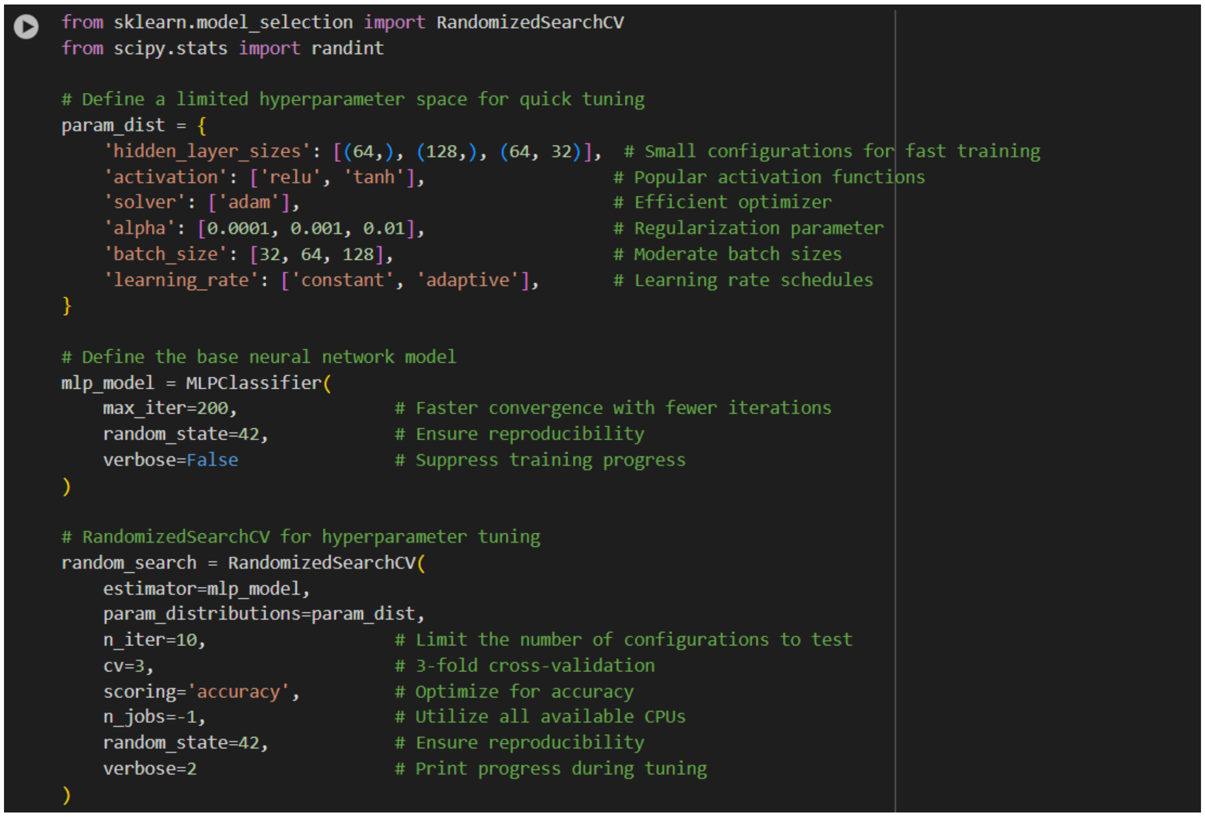
To provide a more detailed understanding of the model’s performance, ‘classification_report()‘ is used to generate a classification report. F1-score, a balanced measure of precision and recall, precision (the ratio of correctly predicted positive observations to total predicted positives), recall (the ratio of correctly predicted positive observations to all actual positives), and support (the number of actual occurrences of each class in the dataset) are all included in this report.



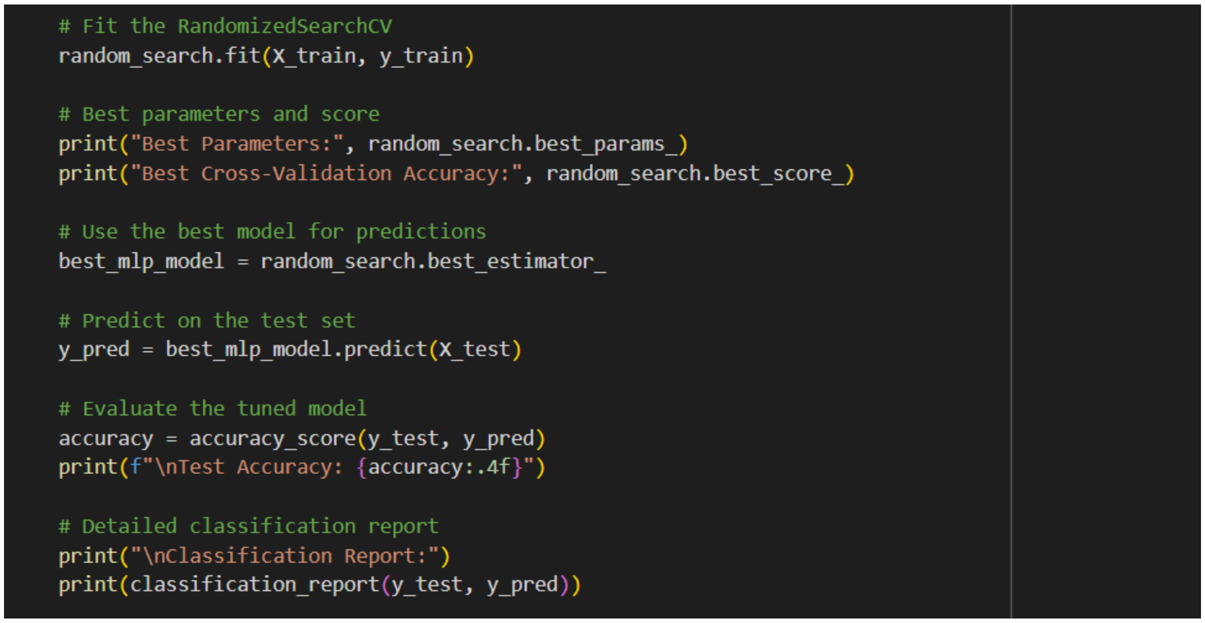
```
Accuracy: 0.9296

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      3668
           1       0.86      0.43      0.57       451

    accuracy                           0.93      4119
   macro avg       0.90      0.71      0.77      4119
weighted avg       0.93      0.93      0.92      4119
```

The model obtained an accuracy of 0.9296 proving that this model is quite an accurate model to be used for the prediction purpose.

Hyperparameter tuning



```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define a limited hyperparameter space for quick tuning
param_dist = {
    'hidden_layer_sizes': [(64,), (128,), (64, 32)],  # Small configurations for fast training
    'activation': ['relu', 'tanh'],                   # Popular activation functions
    'solver': ['adam'],                               # Efficient optimizer
    'alpha': [0.0001, 0.001, 0.01],                   # Regularization parameter
    'batch_size': [32, 64, 128],                      # Moderate batch sizes
    'learning_rate': ['constant', 'adaptive'],        # Learning rate schedules
}

# Define the base neural network model
mlp_model = MLPClassifier(
    max_iter=200,                # Faster convergence with fewer iterations
    random_state=42,             # Ensure reproducibility
    verbose=False                # Suppress training progress
)

# RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(
    estimator=mlp_model,
    param_distributions=param_dist,
    n_iter=10,                   # Limit the number of configurations to test
    cv=3,                        # 3-fold cross-validation
    scoring='accuracy',          # Optimize for accuracy
    n_jobs=-1,                   # Utilize all available CPUs
    random_state=42,             # Ensure reproducibility
    verbose=2                    # Print progress during tuning
)
```


```
# Fit the RandomizedSearchCV
random_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", random_search.best_params_)
print("Best Cross-Validation Accuracy:", random_search.best_score_)

# Use the best model for predictions
best_mlp_model = random_search.best_estimator_

# Predict on the test set
y_pred = best_mlp_model.predict(X_test)

# Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy: {accuracy:.4f}")

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

By methodically exploring through a preset hyperparameter space, this code optimizes the neural network's design for better performance on the classification task.

The number of neurons in the hidden layers (hidden_layer_sizes), activation functions (activation), regularization strength (alpha), batch size (batch_size), and learning rate schedules (learning_rate) are among the few hyper parameters that can be adjusted using the ‘param_dist‘ dictionary.

In order to determine the optimal configuration, this implementation experiments with different combinations of these hyper parameters in comparison to the default neural network model.

While this code tests several configurations like (64,), (128,), and (64, 32) for hidden layers, as well as multiple batch sizes like 32, 64, and 128; the default model has fixed parameters like a single ‘hidden_layer_sizes=(64, 32, 16) and batch_size=default‘.

Ten hyper parameter combinations (n_iter=10‘) are randomly selected from the given space via the RandomizedSearchCV process, which then uses 3-fold cross-validation (cv=3) to assess each configuration.

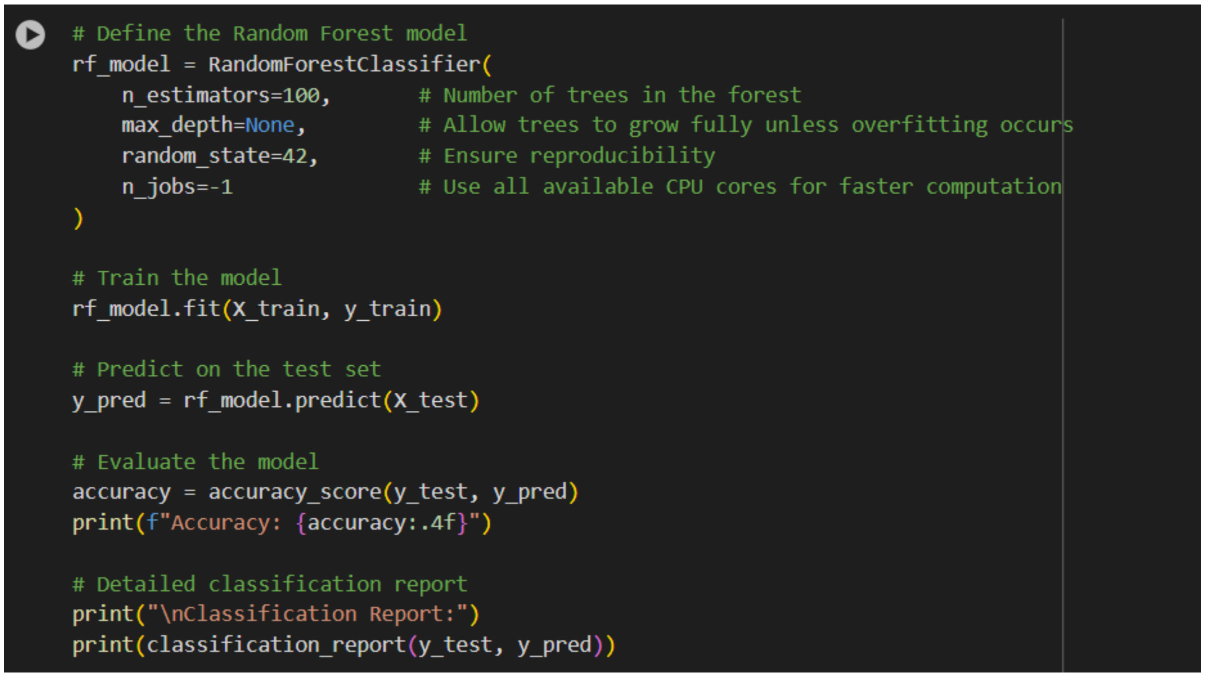
To ensure that the optimal configuration maximizes accuracy, the evaluation metric is set to accuracy (scoring='accuracy'). For efficiency, the training process is parallelized across all available CPUs (n_jobs=-1), and verbose=2 is used to indicate progress

Following the fitting of the training data (X_train, y_train), 'random_search.best_params_' is used to determine the optimal set of hyper parameters, and random_search.best_score_ displays the associated cross-validation accuracy. random_search.best_estimator_ is used to extract the best model, which is then applied to the test set (X_test) to generate predictions.

3.2 Random Forest Classification Model

A Random Forest is an ensemble machine learning model that uses multiple decision trees to classify data.

To promote generalization and reduce over fitting, each tree is constructed using a random subset of features and a random portion of training data. The model makes the ultimate choice during prediction by combining the outputs from every tree.



```
# Define the Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=100,       # Number of trees in the forest
    max_depth=None,         # Allow trees to grow fully unless overfitting occurs
    random_state=42,        # Ensure reproducibility
    n_jobs=-1               # Use all available CPU cores for faster computation
)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

This code demonstrates the implementation of a Random Forest model using ‘RandomForestClassifier‘ from ‘scikit-learn‘. First, hyperparameters are defined for the Random Forest model.

In contrast to a single decision tree, the model more resilient and less likely to overfit when an ensemble of 100 decision trees is used, as indicated by the ‘n_estimators=100‘ option. The individual trees can develop to their full potential with the ‘max_depth=None‘ parameter until all of their leaves are pure or contain fewer samples than what is needed for a split. By regulating the randomness in feature sampling and tree construction, the ‘random_state=42‘ guarantees reproducible results. The ‘n_jobs=-1‘ parameter makes use of every CPU core.

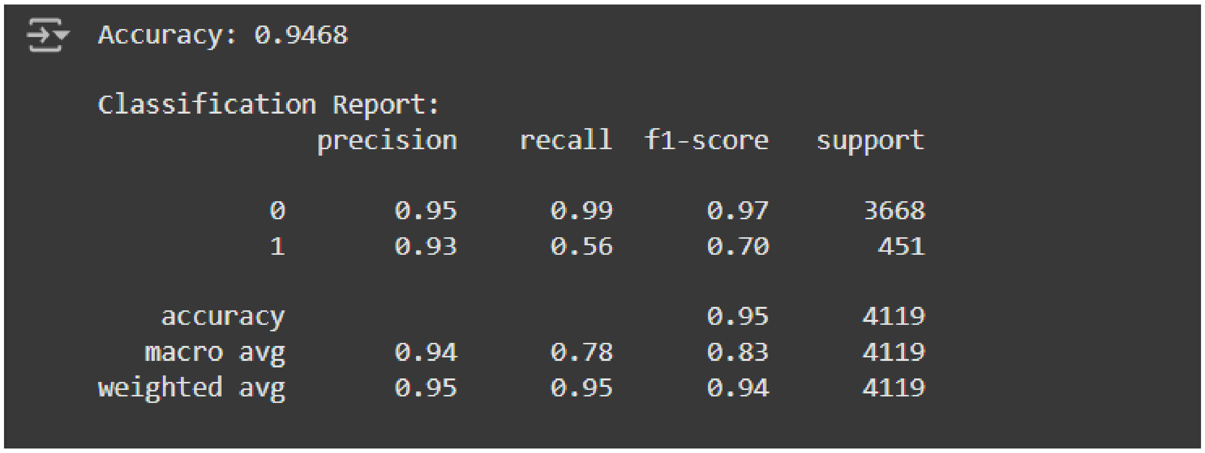
The model learns patterns from the training data (‘X_train‘, ‘y_train‘) using the ‘fit()‘ method, which starts the training process. To ensure variation across the trees, the Random Forest builds many decision trees during this phase, each trained on a random part of the data and a random subset of the features.

After training, the model is tested with unseen data (’X_test‘) using the ‘predict()‘ method, which aggregates the outcomes of all decision trees by majority voting to produce predictions (‘y_pred‘).

The ‘accuracy_score()‘ function determines the proportion of accurate predictions among all test samples in order to assess the model’s performance, offering a broad indication of the model’s performance. A thorough analysis of performance indicators, such as precision, recall, F1-score, and support for each class, is then displayed using the ‘classification_report()‘.

When there is an imbalance in the dataset, the F1-score offers a balanced measure by taking the harmonic mean of precision and recall. Precision measures the proportion of true positive predictions out of all positive predictions, while recall measures the proportion of true positives out of all actual positives.

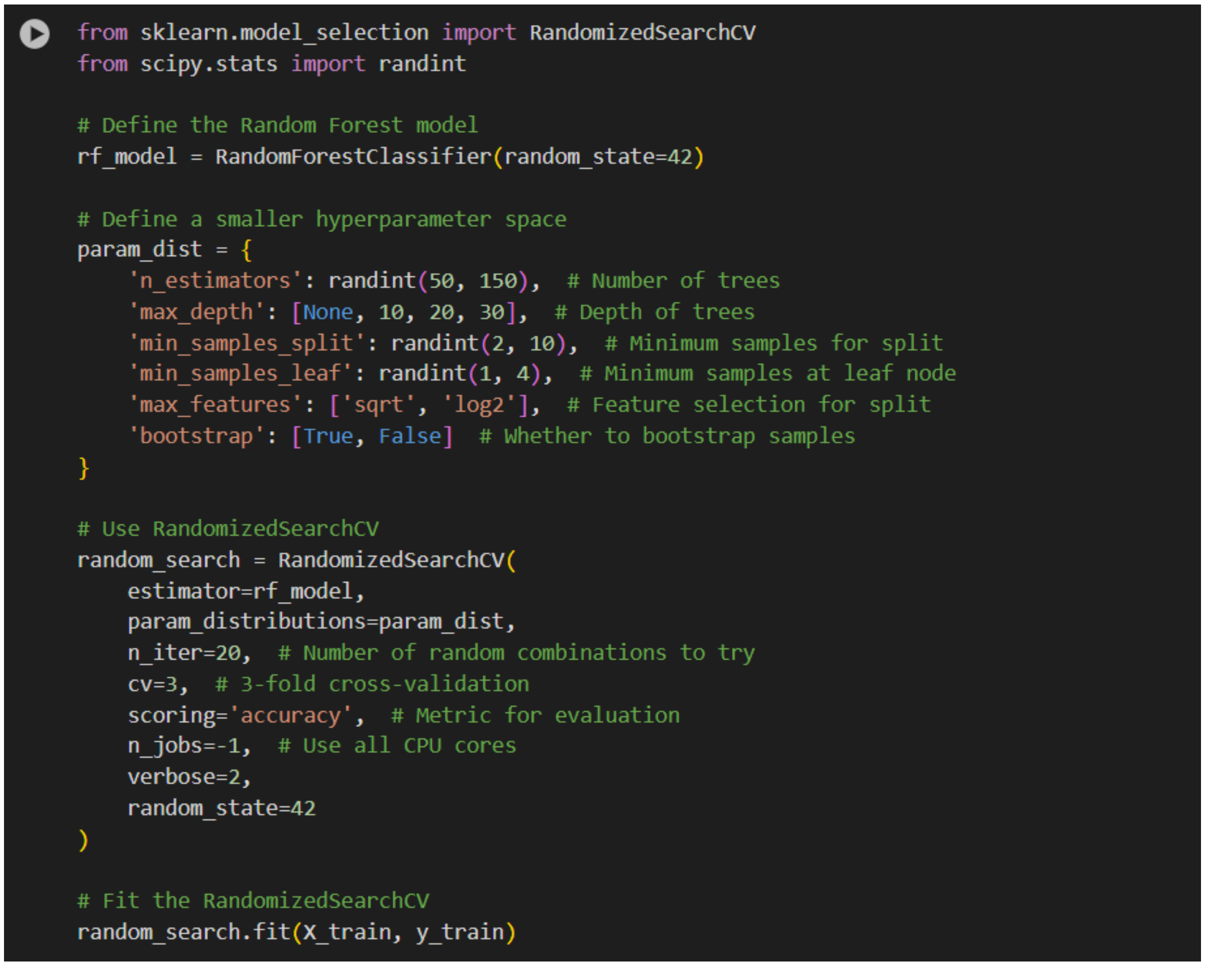
The number of real instances of each class in the test set is shown by the support metric.



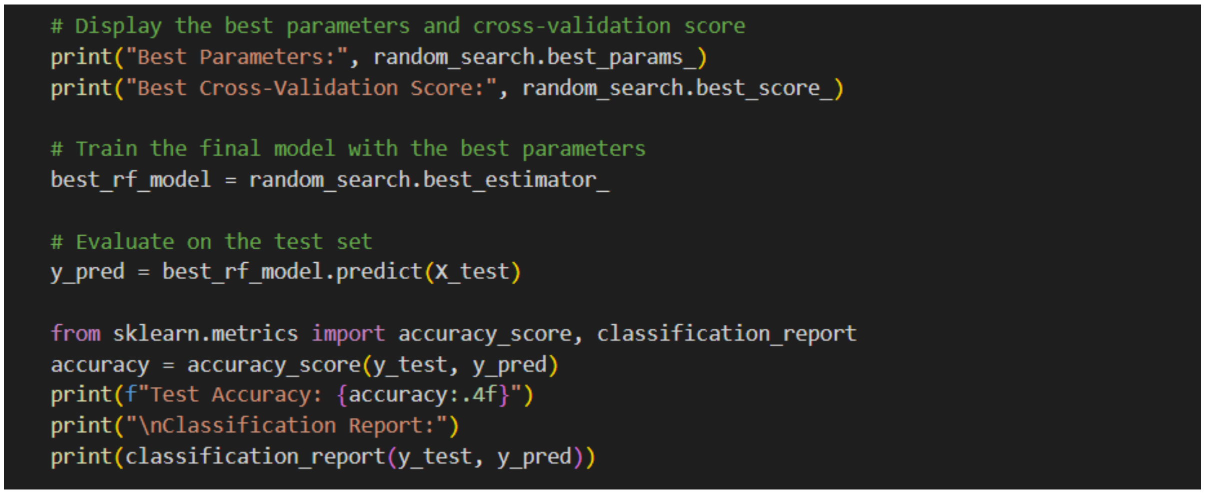
```
Accuracy: 0.9468

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      3668
           1       0.93      0.56      0.70       451

    accuracy                           0.95      4119
   macro avg       0.94      0.78      0.83      4119
weighted avg       0.95      0.95      0.94      4119
```

The model obtained an accuracy of 0.9468 proving that this model is quite an accurate model to be used for the prediction purpose.

Hyper parameter Tuning for the Random Forest Classification Model



```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define a smaller hyperparameter space
param_dist = {
    'n_estimators': randint(50, 150),  # Number of trees
    'max_depth': [None, 10, 20, 30],  # Depth of trees
    'min_samples_split': randint(2, 10),  # Minimum samples for split
    'min_samples_leaf': randint(1, 4),  # Minimum samples at leaf node
    'max_features': ['sqrt', 'log2'],  # Feature selection for split
    'bootstrap': [True, False]  # Whether to bootstrap samples
}

# Use RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_dist,
    n_iter=20,  # Number of random combinations to try
    cv=3,  # 3-fold cross-validation
    scoring='accuracy',  # Metric for evaluation
    n_jobs=-1,  # Use all CPU cores
    verbose=2,
    random_state=42
)

# Fit the RandomizedSearchCV
random_search.fit(X_train, y_train)
```



```
# Display the best parameters and cross-validation score
print("Best Parameters:", random_search.best_params_)
print("Best Cross-Validation Score:", random_search.best_score_)

# Train the final model with the best parameters
best_rf_model = random_search.best_estimator_

# Evaluate on the test set
y_pred = best_rf_model.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

In contrast to the default Random Forest model, which uses fixed hyper parameters this code defines a search space (‘param_dist‘) with ranges and options for a variety of hyperparameters, including ’n_estimators’ (number of trees), ’max_depth’ (maximum depth of trees), ’min_samples_split’ (minimum samples required to split a node), ’min_samples_leaf’ (minimum samples required in leaf nodes), ’max_features; (number of features considered for splits), and ’bootstrap’ (whether to use bootstrapping).

Twenty combinations of these hyper parameters are randomly sampled using the ‘RandomizedSearchCV‘ (n_iter=20‘), and each combination is assessed using three-fold cross-validation (‘cv=3‘).

For efficiency, the process is parallelized using all available CPU cores (n_jobs=-1), and the evaluation metric is set to accuracy (scoring=’accuracy’).

The model with this configuration is trained on the training data once the optimal set of hyper parameters has been found.

The model’s performance is assessed using accuracy and a comprehensive classification report that includes metrics like precision, recall, and F1-score. Predictions are performed on the test set (‘X_test‘).

Instead of using preset hyperparameter values (e.g., n_estimators=100, max_depth=None) like in the default Random Forest model, this code dynamically selects the best-performing configuration based on cross-validation after exploring a range of values.

4 Model Analysis and Evaluation

4.1 Model Evaluation for Class 0 Results

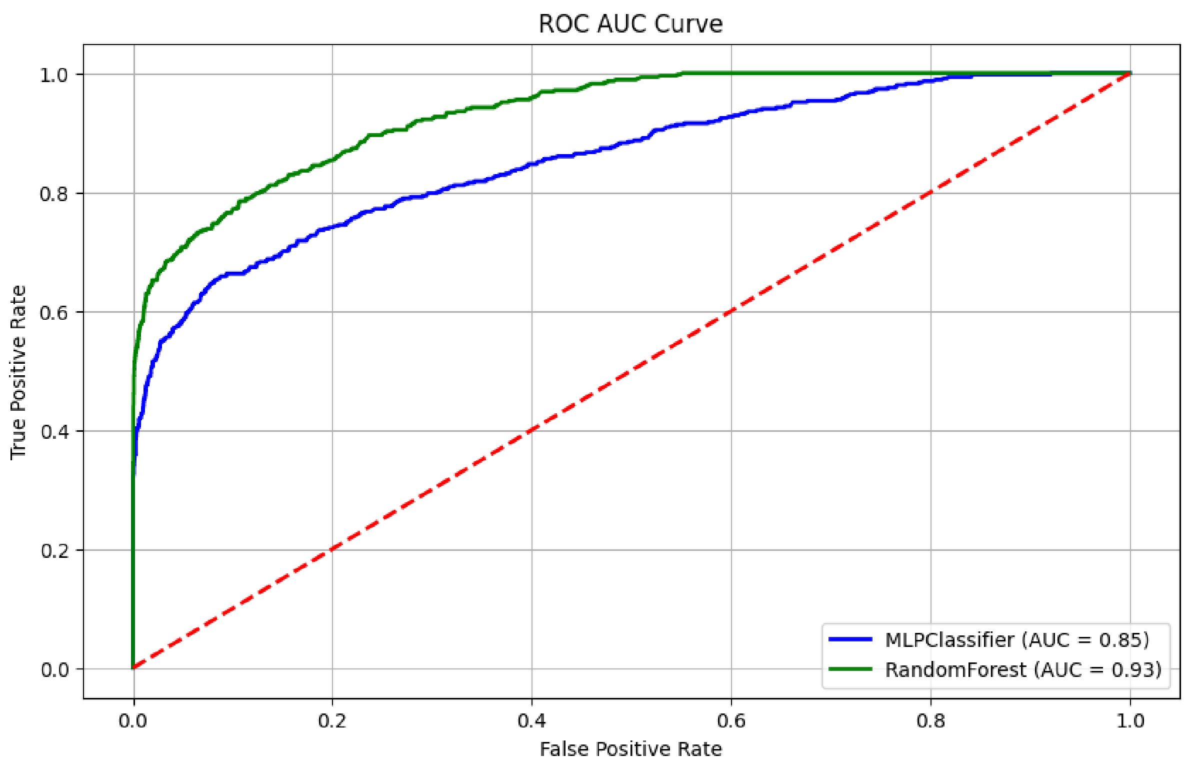
MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE	HYPERPARAMETE
Neural Network	0.9296	0.93	0.99	0.96	hidden_layer_sizes=(64, 32, 16), activation='relu', solver='adam', max_iter=500
Random Forest Classification	0.9468	0.95	0.99	0.97	n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1
Neural Network (Hyperparameter Tuned)	0.9090	0.92	0.98	0.95	hidden_layer_sizes=(128,), activation='tanh', solver='adam', alpha=0.001, batch_size=64, learning_rate='adaptive'
Random Forest (Hyperparameter Tuned)	0.9102	0.92	0.99	0.95	n_estimators=120, max_depth=30, min_samples_split=5, min_samples_leaf=2, bootstrap=True

4.2 Model Evaluation for Class 1 Results

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE	HYPERPARAMETE
Neural Network	0.9296	0.86	0.43	0.57	hidden_layer_sizes=(64, 32, 16), activation='relu', solver='adam', max_iter=500
Random Forest Classification	0.9468	0.93	0.56	0.70	n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1
Neural Network (Hyperparameter Tuned)	0.9090	0.70	0.30	0.42	hidden_layer_sizes=(128,), activation='tanh', solver='adam', alpha=0.001, batch_size=64, learning_rate='adaptive'
Random Forest (Hyperparameter Tuned)	0.9102	0.74	0.28	0.40	n_estimators=120, max_depth=30, min_samples_split=5, min_samples_leaf=2, bootstrap=True

4.3 ROC-AUC Curve

The ROC-AUC Curve (Receiver Operating Characteristic - Area Under the Curve) is a graphical representation and performance metric used to evaluate the effectiveness of binary classification models.



This plot compares the performance of two models, Random Forest and MLPClassifier, in predicting term deposit subscriptions using an ROC curve.

The Random Forest model (AUC = 0.93) significantly outperforms the MLPClassifier (AUC = 0.85), demonstrating its superior ability to distinguish between subscribers and non-subscribers.

The steep ascent of the Random Forest curve indicates higher true positive rates at lower false positive rates, making it a better choice for this classification task.

5 Appendix

5.1 Exploratory Data Analysis Code

Import the necessary libraries

```
import pandas as pd import matplotlib.pyplot as plt import seaborn as sns
```

Load the dataset

```
train_data = pd.read_csv(”/content/drive/MyDrive/bank-additional-full.csv”, delimiter=’;’)
```

Display the data types of each column

```
print(”Data Types of Each Column:”) print(train_data.dtypes)
```

The columns with the data type ’object’ are the categorical data while the columns with the data type ’int64’ and ’float64’ are the numberical data.

Check for null values

```
print(train_data.isnull().sum())
```

According to the above output the dataset does not consist of any null values.

Handling missing values

Calculating the overall ’yes’ rate of the target variable using a pie chart

```
# Calculate the distribution of ’yes’ and ’no’ in the target variable

y_distribution = train_data[’y’].value_counts(normalize=True)

# Plot a pie chart

plt.figure(figsize=(6, 6))

plt.pie(
y_distribution,
labels=y_distribution.index,
autopct=’%1.1f%
startangle=90,
colors=[’skyblue’, ’#FF69B4’]
)

plt.title(”Overall Distribution of ’Yes’ and ’No’ in Target Variable (y)”)
plt.axis(’equal’)  Ensures the pie is drawn as a circle

plt.show()
```

Identifying columns with data with the value ’unknown’

```
#Identify columns with ’unknown’ values

unknown_counts = train_data.isin([”unknown”]).sum()

unknown_columns = unknown_counts[unknown_counts ¿ 0].index

# Analyze and print the impact of ”unknown” on ‘y‘ for each column

for column in unknown_columns:
# Filter rows with ”unknown” in the current column
unknown_data = train_data[train_data[column] == ”unknown”]
# Get the distribution of ‘y‘ in the filtered data
y_distribution = unknown_data[’y’].value_counts(normalize=True) * 100 # Convert to percentages
# Print the percentages
print(f”of ’unknown’ in column ’column’ on ’y’: y_distribution.get(’yes’, 0):.1f%”)

# Impute columns with the mode

columns_to_impute = [’job’, ’housing’, ’loan’]

for column in columns_to_impute:
mode = train_data[column].mode()[0]
train_data[column] = train_data[column].replace(’unknown’, mode)

#Drop columns with minimal impact

columns_to_drop = [’marital’, ’education’, ’default’]

train_data = train_data.drop(columns=columns_to_drop)

#Verify the updated dataset

print(”Remaining columns after imputation and deletion:”, train_data.columns)

print(”Unknown counts:”, train_data.isin([’unknown’]).sum())
```

Printing the preprocessed dataset

```
df = train_data
```

print(df)

Graphical Representations

Age Distribution and Subscription

plt.figure(figsize=(8, 5))

sns.histplot(data=train_data, x='age', hue='y', kde=True, palette=['lightcoral', 'lightblue'], bins=30)

plt.title("Age Distribution by Subscription ('y')")

plt.xlabel("Age")

plt.ylabel("Count")

plt.legend(title="Subscribed to Term Deposit", labels=['No', 'Yes'])

plt.show()

Relationship between 'job' and Subscription

plt.figure(figsize=(10, 6))

sns.countplot(y='job', hue='y', data=train_data, palette=['#00008B', '#FFA500'], order=train_data['job'].value_counts().index)

plt.title("Impact of Job Type on Subscription ('y')")

plt.xlabel("Count")

plt.ylabel("Job")

plt.legend(title="Subscribed to Term Deposit")

plt.show()

Monthly Trends (month) of Subscription

plt.figure(figsize=(10, 6))

sns.countplot(data=train_data, x='month', hue='y', palette=['lightcoral', 'lightblue'], order=['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec'])

plt.title("Monthly Trends of Subscription ('y')") plt.xlabel("Month")

plt.ylabel("Count")

plt.legend(title="Subscribed to Term Deposit", labels=['No', 'Yes'])

plt.show()

Impact of Contact Type (contact) on Subscription

plt.figure(figsize=(8, 5))

sns.countplot(data=train_data, x='contact', hue='y', palette=['lightcoral', 'lightblue'])

plt.title("Impact of Contact Type on Subscription ('y')")

plt.xlabel("Contact Type")

plt.ylabel("Count")

plt.legend(title="Subscribed to Term Deposit", labels=['No', 'Yes'])

plt.show()

Overall Summary of the dataset

Columns with less significant effect on the target variable such as 'marital', 'education' and 'default' were removed to prevent the model from overfitting to irrelevant data.

There is a strong imbalance in the dataset with the majority of clients not subscribing to the term deposit.

In expectation of more data on the dataset and potential correlations between attributes, a few more bar plots were created.

5.2 Machine Learning Algorithms Code

Import the necessary libraries

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import chi2_contingency

import numpy as np
```

Load the training and testing datasets into the notebook

```
train_data = pd.read_csv(”/content/drive/MyDrive/bank-additional-full.csv”, delimiter=’;’)

test_data = pd.read_csv(”/content/drive/MyDrive/bank-additional.csv”, delimiter=’;’)
```

Check if the column names and order matches between the two datasets

```
test_data = test_data[train_data.columns]
```

Print the non-processed training dataset

```
print(”Train Data:”)

df = train_data

df
```

Print the non-processed testing dataset

```
print(”Test Data:”)

df = test_data
```

```
df
```

Check for the null values in the dataset

```
# Check for missing values

print(train_data.isnull().sum())
```

Even though there aren't any null values in the dataset, upon careful observation we can see data with the value 'unknown' which should be handled.

Prior to handling that we need to know the overall 'yes' proportion for target variable 'y'.

```
# Calculate overall 'yes' rate

overall_yes_rate = train_data['y'].value_counts(normalize=True)['yes'] * 100

print(f"Overall 'yes' rate in the dataset: overall_yes_rate:.2f%")

# Columns with 'unknown' values

unknown_columns = train_data.isin(['unknown']).sum()[train_data.isin(['unknown']).sum() ¿ 0].index

print("with 'unknown' values:", unknown_columns)
```

Delete the columns with 'yes' proportion which varies from the overall 'yes' percentage.

Impute the columns with the 'yes' proportion similar to or slightly different from the overall 'yes' rate as those attributes might have hidden relationships with the target variable 'y'.

```
# Impute columns with the mode

columns_to_impute = ['job', 'housing', 'loan']

for column in columns_to_impute:
mode = train_data[column].mode()[0]
train_data[column] = train_data[column].replace('unknown', mode)

# Drop columns with minimal impact

columns_to_drop = ['marital', 'education', 'default'] train_data = train_data.drop(columns=columns_to_drop)

# Verify the updated dataset

print("Remaining columns after imputation and deletion:", train_data.columns)
```

# Check the updated data

print(”Updated Data Summary:”)

print(train_data.isin([’unknown’]).sum())

Feature Selection

Since the values of the duration column is unavailable until after the outcome it cannot be used in a real-time predictive model.

To address this issue the duration column was dropped from the dataset.

# Drop ’duration’ column (not predictive in real-time settings)

train_data = train_data.drop(columns=[’duration’])

# Display the data types of each column

print(”Data Types of Each Column:”)

print(train_data.dtypes)

# Split attributes into categorical and numerical

categorical_features = [’job’, ’housing’, ’loan’, ’contact’, ’month’, ’day_of_week’, ’poutcome’]

numerical_features = [’age’, ’campaign’, ’pdays’, ’previous’, ’emp.var.rate’, ’cons.price.idx’, ’cons.conf.idx’, ’euribor3m’, ’nr.employed’]

The target column ’y’ was encoded into binary values (yes = 1, no = 0).

# Encode target variable ’y’ to binary

train_data[’y’] = train_data[’y’].map(’yes’: 1, ’no’: 0)

test_data[’y’] = test_data[’y’].map(’yes’: 1, ’no’: 0)

Generating a correlation heatmap to understand relationships between numerical features and the target variable.

# Compute the correlation matrix

correlations = train_data[numerical_features + [’y’]].corr()

```
# Heatmap for correlations

plt.figure(figsize=(10, 8))

sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)

plt.title('Correlation Heatmap')

plt.show()
```

To determine the relationship between a continuous variable and a binary variable, correlation analysis is performed.

```
# Correlation Analysis

print("Analysis for Numerical Attributes:")

correlations = train_data[numerical_features + ['y']].corr() print(correlations['y'].sort_values(ascending=False))
```

Chi-square tests were performed on categorical attributes to evaluate their significance concerning the target.

```
print("Chi-Square Test for Categorical Attributes:")

for col in categorical_features:
contingency_table = pd.crosstab(train_data[col], train_data['y'])
chi2, p,_, _ = chi2_contingency(contingency_table) print(f"col: p-value = p:.4f")
if p   0.05:
print(f"   'col' may not be significantly related to 'y'.")
```

Features with very low correlation should be removed.

Based on the correlation analysis performed 'age' column should be removed.

Features that have a (p-value   0.05) might not be strongly correlated with y and should be eliminated.

Based on the chi-square test 'housing' and 'loan' should be removed.

```
# Features to remove based on analysis

features_to_remove = [ 'housing', 'loan', 'age']

# Drop the selected features from the dataset

train_data = train_data.drop(columns=features_to_remove)

# Check the updated dataset
```

```
print(”Updated Dataset Columns:”)

print(train_data.columns)

df = train_data

df
```

One Hot Encoding was used to generate binary columns for every categorical value

```
# Identify categorical columns

categorical_columns = train_data.select_dtypes(include=[’object’]).columns

# One-Hot Encoding for Categorical Variables

train_data = pd.get_dummies(train_data, columns=categorical_columns, drop_first=True)

test_data = pd.get_dummies(test_data, columns=categorical_columns, drop_first=True)

# Align train and test datasets (to ensure same features after encoding)

train_data, test_data = train_data.align(test_data, join=’inner’, axis=1)

X_train = train_data.drop(columns=[”y”]) # Feature set for the training data

y_train = train_data[”y”] # Extract the target variable from the training data

X_test = test_data.drop(columns=[”y”]) # Feature set for the testing data

y_test = test_data[”y”] # Extract the target variable from the testing data
```

To ensures that all features contribute equally to the model training process, Min-Max Scaling was used to scale data to a fixed range, between 0 and 1.

```
# Min-Max Scaling for Numerical Features

scaler = MinMaxScaler()

X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])

X_test[X_test.columns] = scaler.transform(X_test[X_test.columns])
```

Ensure that both the training and testing datasets have the same number of features after preprocessing (encoding and scaling) print(f"Number of features in X_train: X_train.shape[1]")

print(f"Number of features in X_test: X_test.shape[1]")

Based on the above outcome we can concloude that after pre-processing, both the training and testing datasets have the same number of features.

# Display the datasets after encoding and scaling

print("Train Dataset After One-Hot Encoding and Scaling:")

X_train

print("Dataset After One-Hot Encoding and Scaling:")

X_test

Final data frame can be viewed as above.

Machine Learning Algorithms

1. Nueral Network Model

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Define the neural network model

```
mlp_model = MLPClassifier(
hidden_layer_sizes=(64, 32, 16), # Three hidden layers with decreasing neurons
activation='relu', # ReLU activation function
solver='adam', # Adam optimizer
max_iter=500, # Maximum iterations for convergence
random_state=42, # Ensures reproducibility
verbose=False # Print training progress
)
```

# Train the model

mlp_model.fit(X_train, y_train)

# Predict on the test set

y_pred = mlp_model.predict(X_test)

# Evaluate the model

```
accuracy = accuracy_score(y_test, y_pred)

print(f”Accuracy: accuracy:.4f”)

# Detailed classification report

print(”Report:”)

print(classification_report(y_test, y_pred))
```

2. Random Forest Classification Model

```
# Define the Random Forest model

rf_model = RandomForestClassifier(
n_estimators=100, # Number of trees in the forest
max_depth=None, # Allow trees to grow fully unless overfitting occurs
random_state=42, # Ensure reproducibility
n_jobs=-1 # Use all available CPU cores for faster computation
)

# Train the model

rf_model.fit(X_train, y_train)

# Predict on the test set

y_pred = rf_model.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f”Accuracy: accuracy:.4f”)

# Detailed classification report

print(”Report:”)

print(classification_report(y_test, y_pred))
```

After analysing the above two models we can conclude the followings :

In general, class zero has superior precision, recall, and f1-score than class one.

Despite this, it shows a balance in performance across classes because the weighted average and the macro average are rather similar.

Therefore, it may be concluded that the model is neither overfitting nor underfitting.

Further Improvements

Hyperparameter Tuning is one of the highly effective and essential strategy for improving model performance.

Hyperparameter Tuning for Neural Network Model

```
from sklearn.model_selection import RandomizedSearchCV

from scipy.stats import randint

# Define a limited hyperparameter space for quick tuning

param_dist =
’hidden_layer_sizes’: [(64,), (128,), (64, 32)], # Small configurations for fast training
’activation’: [’relu’, ’tanh’], # Popular activation functions
’solver’: [’adam’], # Efficient optimizer
’alpha’: [0.0001, 0.001, 0.01], # Regularization parameter
’batch_size’: [32, 64, 128], # Moderate batch sizes
’learning_rate’: [’constant’, ’adaptive’], # Learning rate schedules


# Define the base neural network model

mlp_model = MLPClassifier(
max_iter=200, # Faster convergence with fewer iterations
random_state=42, # Ensure reproducibility
verbose=False # Suppress training progress
)

# RandomizedSearchCV for hyperparameter tuning

random_search = RandomizedSearchCV(
estimator=mlp_model,
param_distributions=param_dist,
n_iter=10, # Limit the number of configurations to test
cv=3, # 3-fold cross-validation
scoring=’accuracy’, # Optimize for accuracy
n_jobs=-1, # Utilize all available CPUs
random_state=42, # Ensure reproducibility
```

verbose=2 # Print progress during tuning
)

# Fit the RandomizedSearchCV

random_search.fit(X_train, y_train)

# Best parameters and score

print(”Best Parameters:”, random_search.best_params_)

print(”Best Cross-Validation Accuracy:”, random_search.best_score_)

# Use the best model for predictions

best_mlp_model = random_search.best_estimator_

# Predict on the test set

y_pred = best_mlp_model.predict(X_test)

# Evaluate the tuned model

accuracy = accuracy_score(y_test, y_pred)

print(f”Accuracy: accuracy:.4f”)

# Detailed classification report

print(”Report:”)

print(classification_report(y_test, y_pred))

Hyperparameter Tuning for Random Forest

from sklearn.model_selection import RandomizedSearchCV

from scipy.stats import randint

# Define the Random Forest model

rf_model = RandomForestClassifier(random_state=42)

# Define a smaller hyperparameter space

param_dist =

```
'n_estimators': randint(50, 150), # Number of trees
'max_depth': [None, 10, 20, 30], # Depth of trees
'min_samples_split': randint(2, 10), # Minimum samples for split
'min_samples_leaf': randint(1, 4), # Minimum samples at leaf node
'max_features': ['sqrt', 'log2'], # Feature selection for split
'bootstrap': [True, False] # Whether to bootstrap samples


# Use RandomizedSearchCV

random_search = RandomizedSearchCV(

estimator=rf_model,
param_distributions=param_dist,
n_iter=20, # Number of random combinations to try
cv=3, # 3-fold cross-validation
scoring='accuracy', # Metric for evaluation
n_jobs=-1, # Use all CPU cores
verbose=2,
random_state=42
)

# Fit the RandomizedSearchCV

random_search.fit(X_train, y_train)

# Display the best parameters and cross-validation score

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

print("Best Cross-Validation Score:", random_search.best_score_)

# Train the final model with the best parameters

best_rf_model = random_search.best_estimator_

# Evaluate on the test set y_pred = best_rf_model.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report

accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: accuracy:.4f")

print("Report:")
print(classification_report(y_test, y_pred))
```

When comparing the accuracy of my neural network and hyperparameter-tuned neural network models, the original neural network achieved an accuracy of 0.9296, while the tuned model only reached 0.9090

Similarly, for the random forest classifier, the original model attained an accuracy of 0.9468, whereas the tuned version achieved 0.9102.

Since the default models consistently perform above 90% accuracy, it may be more effective to avoid hyperparameter tuning in this case.

AUC-ROC Curve

This curve plots the true positive value against the false positive value.

```
# Define the MLPClassifier and RandomForestClassifier models

mlp_model = MLPClassifier(

hidden_layer_sizes=(64, 32, 16),
activation='relu',
solver='adam',
max_iter=500,
random_state=42
)

rf_model = RandomForestClassifier(

n_estimators=100,
random_state=42,
n_jobs=-1
)

# Train the models

mlp_model.fit(X_train, y_train)

rf_model.fit(X_train, y_train)

# Generate ROC AUC Curve

from sklearn.metrics import roc_curve, auc

# MLPClassifier ROC AUC
```

```
mlp_y_pred_proba = mlp_model.predict_proba(X_test)[:, 1]

fpr_mlp, tpr_mlp, _ = roc_curve(y_test, mlp_y_pred_proba)

roc_auc_mlp = auc(fpr_mlp, tpr_mlp)

# RandomForestClassifier ROC AUC

rf_y_pred_proba = rf_model.predict_proba(X_test)[:, 1]

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_y_pred_proba)

roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plotting

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.plot(fpr_mlp, tpr_mlp, label=f"MLPClassifier (AUC = roc_auc_mlp:.2f)", lw=2, color='blue')

plt.plot(fpr_rf, tpr_rf, label=f"RandomForest (AUC = roc_auc_rf:.2f)", lw=2, color='green')

plt.plot([0, 1], [0, 1], color='red', linestyle='–', lw=2)

plt.title('ROC AUC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc='lower right')

plt.grid(True)

plt.show()
```

Ranks of the models based on the plot :

1. Random Forest Classification
2. Neural Network

6 References

1. Data Science Stack Exchange. (n.d.). metric - How to interpret classification report of scikit-learn? [online] Available at: `https://datascience.stackexchange.com/questions/64441/how-to-interpret-classification-report-of-scikit-learn`.

2. Ganji, L. (2019). One Hot Encoding in Machine Learning. [online] GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/ml-one-hot-encoding/`.

3. GeeksforGeeks (2021). What is Exploratory Data Analysis ? [online] GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/`.

4. GeeksForGeeks (2023). Hyperparameter tuning. [online] GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/hyperparameter-tuning/`.

5. GeeksforGeeks (2021). Data Pre-Processing wit Sklearn using Standard and Min-max scaler. [online] GeeksforGeeks. Available at: `https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/`.

6. Simplilearn (2022). Introduction to Data Imputation — Simplilearn. [online] Simplilearn.com. Available at: `https://www.simplilearn.com/data-imputation-article`.