```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
term_deposit_df =pd.read_excel("/content/project.xlsx", sheet_name='Sheet1')
```

```
term_deposit_df.shape
```

```
(45211, 17)
```

```
display(pd.DataFrame(term_deposit_df ))
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	
...	...	...	...	...	...	...	...	...	...	...	...	
45206	51			ertiary	no	825	no	no	cellular	17	nov	
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	

Disk: 24.42 GB/107.72 GB

45211 rows × 17 columns

```
term_deposit_df =pd.read_excel("/content/project1.xlsx", sheet_name='Sheet1')
```

```
display(pd.DataFrame(term_deposit_df ))
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	
...	...	...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	

45211 rows × 17 columns

```
# Get the unique values of each column
# Iterate through each column and print unique values
for column in term_d
    print(f"Value counts in '{column}':\n{term_deposit_df[column].value_counts()}\n")
```



```
Value counts in 'age':
age
32    2085
31    1996
33    1972
34    1930
35    1894
      ...
93       2
90       2
95       2
88       2
94       1
Name: count, Length: 77, dtype: int64

Value counts in 'job':
job
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: count, dtype: int64

Value counts in 'marital':
marital
married     27214
single      12790
divorced     5207
Name: count, dtype: int64

Value counts in 'education':
education
```

```
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: count, dtype: int64

Value counts in 'default':
default
no     44396
yes      815
Name: count, dtype: int64

Value counts in 'balance':
balance
 0        3514
 1         195
 2         156
 4         139
```

```
term_deposit_df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

```
# Converting categorical columns to numeric
# Making the mapping                                nverting all responses to lowercase first
term_deposit_df['y'] =                              .str.lower().map({'yes': 1, 'no': 0})
term_deposit_df['marital'] = term_deposit_df['marital'].str.lower().map({'married': 1, 'single': 2,'divorce


# Define bins as 0-18, 19-60, 61-100
bins = [0, 18, 60, 100]

# Numbers for the groups instead of names
group_nums = [1, 2, 3]  # For example, 1=Young, 2=Adult, 3=Senior
# Categorizing the data
term_deposit_df['AgeGroup'] = pd.cut(term_deposit_df['age'], bins, labels=group_nums)


# Get count of unique values in each column
unique_counts = term_deposit_df.nunique()

print(unique_counts)
```

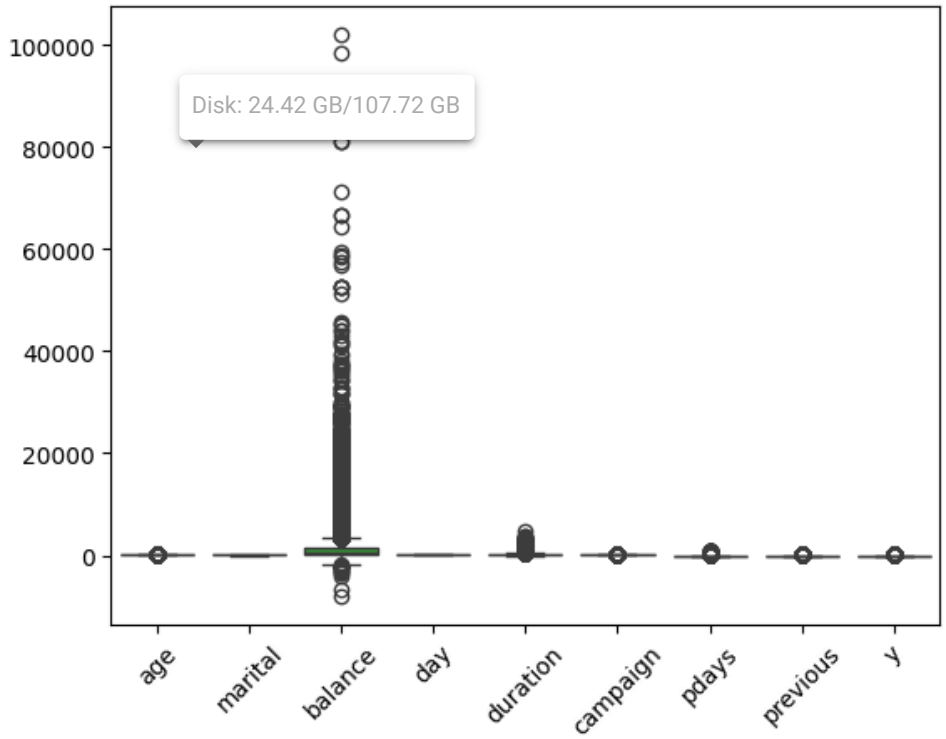
```
age            77
job            12
marital         3
education       4
default         2
balance      7168
housing         2
loan            2
contact         3
day            31
month          12
duration     1573
campaign       48
pdays         559
previous       41
poutcome        4
y               2
AgeGroup        3
dtype: int64
```

```
# Function to replace NaN values
def replace_nan_with_mean_mode(df):
    for column in df.columns:
        # Check if the column is numeric
        if df[column].dtype in ['int64', 'float64']:
            mean_value = df[column].mode()[0]
            df[column].fillna(mean_value, inplace=True)
        else:  # Assuming non-numeric columns are categorical
            mode_value = df[column].mode()[0]  # mode() returns a Series, get the first element
            df[column].fillna(mode_value, inplace=True)
    return df

# Apply the function to your DataFrame
term_deposit_df = replace_nan_with_mean_mode(term_deposit_df)


# Creates box plots for all numeric columns in the DataFrame
import seaborn as sns


sns.boxplot(data=term_deposit_df)
plt.xticks(rotation=45)  # Useful if you have many columns and the labels overlap
plt.show()
```




```
# Define a function to replace outliers with the median
def replace_outliers_with_median(df, column_name):
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    median = df[column_name].median()

    # Replace outliers with median
    df.loc[(df[column_name] < lower_bound) | (df[column_name] > upper_bound), column_name] = median

    return df
```

```
import numpy as np

# Apply the function to your DataFrame
replace_outliers_with_median = replace_outliers_with_median(term_deposit_df, 'balance')


# Assuming df is your DataFrame and 'column' is your column of interest
q_low = term_deposit_df['duration'].quantile(0.01)
q_hi  = term_deposit_df['duration'].quantile(0.99)

term_deposit_df['duration'] = term_deposit_df['duration'].clip(lower=q_low, upper=q_hi)
# term_deposit_df['duration'] = np.log(term_deposit_df['duration'] + 1)  # Adding 1 to avoid log(0)


# Or using the `columns` parameter
term_deposit_df.drop(columns=['pdays','previous'], inplace=True)


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Data preprocessing
# Example: Encoding and scaling
# Assuming 'target' is your binary target variable and the rest are features

X = term_deposit_df.
y = term_deposit_df[
```

Disk: 24.42 GB/107.72 GB

```
# Encoding categorical variables if necessary (example with pandas)
X = pd.get_dummies(X)

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Choose a model for classification
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)


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

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Evaluate the model
# Predicting the Test set results
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9039035718235099
Confusion Matrix:
 [[7721  231]
```

```
 [ 638  453]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      7952
           1       0.66      0.42      0.51      1091

    accuracy                           0.90      9043
   macro avg       0.79      0.69      0.73      9043
weighted avg       0.89      0.90      0.89      9043
```

```
# Get numerical feature importances
importances = model.feature_importances_

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(X.columns, importances

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: duration            Importance: 0.27
Variable: balance             Importance: 0.1
Variable: age                 Importance: 0.09
Variable: day                 Importance: 0.09
Variable: poutc               Importance: 0.05
Variable: campaign            Importance: 0.04
Variable: marital             Importance: 0.02
Variable: poutcome_unknown    Importance: 0.02
Variable: job_admin.          Importance: 0.01
Variable: job_blue-collar     Importance: 0.01
Variable: job_management      Importance: 0.01
Variable: job_services        Importance: 0.01
Variable: job_technician      Importance: 0.01
Variable: education_primary   Importance: 0.01
Variable: education_secondary Importance: 0.01
Variable: education_tertiary  Importance: 0.01
Variable: education_unknown   Importance: 0.01
Variable: housing_no          Importance: 0.01
Variable: housing_yes         Importance: 0.01
Variable: loan_no             Importance: 0.01
Variable: loan_yes            Importance: 0.01
Variable: contact_cellular    Importance: 0.01
Variable: contact_unknown     Importance: 0.01
Variable: month_apr           Importance: 0.01
Variable: month_aug           Importance: 0.01
Variable: month_feb           Importance: 0.01
Variable: month_jan           Importance: 0.01
Variable: month_jul           Importance: 0.01
Variable: month_jun           Importance: 0.01
Variable: month_mar           Importance: 0.01
Variable: month_may           Importance: 0.01
Variable: month_nov           Importance: 0.01
Variable: month_oct           Importance: 0.01
Variable: month_sep           Importance: 0.01
Variable: poutcome_failure    Importance: 0.01
Variable: poutcome_other      Importance: 0.01
Variable: AgeGroup_2          Importance: 0.01
Variable: AgeGroup_3          Importance: 0.01
Variable: job_entrepreneur    Importance: 0.0
Variable: job_housemaid       Importance: 0.0
Variable: job_retired         Importance: 0.0
Variable: job_self-employed   Importance: 0.0
Variable: job_student         Importance: 0.0
Variable: job_unemployed      Importance: 0.0
Variable: job_unknown         Importance: 0.0
Variable: default_no          Importance: 0.0
Variable: default_yes         Importance: 0.0
```