```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sl
import matplotlib as ml
import bokeh as bk
import plotly as pl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
import statsmodels.api as sm

df = pd.read_csv('/content/bank-full.csv')

# importing required libraries for the analysis. Creating a dataframe to analyze in Pandas
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
# Real world portugal bank data marketing set has been uploaded  URL https://archive.ics.uci.edu/dataset/222/bank+marketing

#Bank Marketing Data
#Donated on 2/13/2012
#The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to pr
# Downloaded from the link shared above and uploaded to Google Colab to analyze, format is CSV
```

```
df = df.dropna()   # removing null values
```

```
Start coding or generate with AI.
```

```
df.drop_duplicates()
df.shape
```

```
(45211, 17)
```

```
print(df.duplicated().sum())   # there are not any duplicate values
```

```
0
```

```
df.info() #info about data types of columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

```
print(df.describe())  # descriptive statistics for each column
```

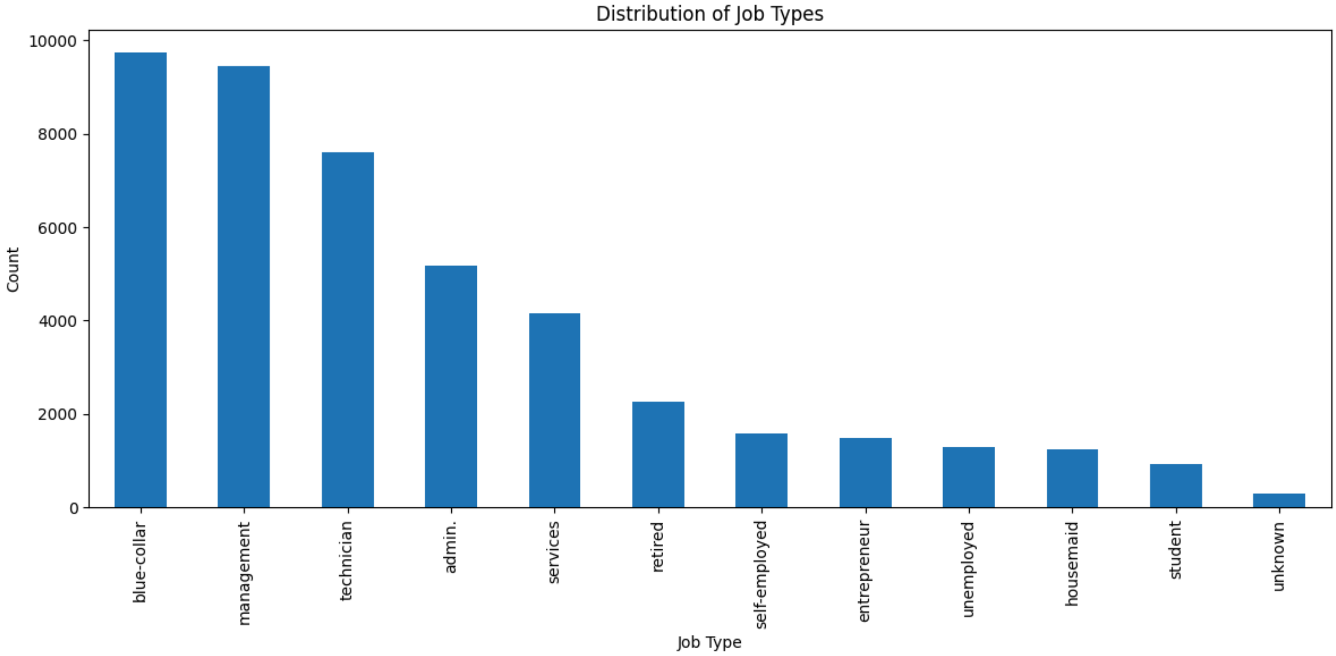
```
                age        balance           day      duration      campaign  \
count  45211.000000   45211.000000  45211.000000  45211.000000  45211.000000
mean      40.936210    1362.272058     15.806419    258.163080      2.763841
std       10.618762    3044.765829      8.322476    257.527812      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
```

```
print(df.head())   #visualization of first files
```

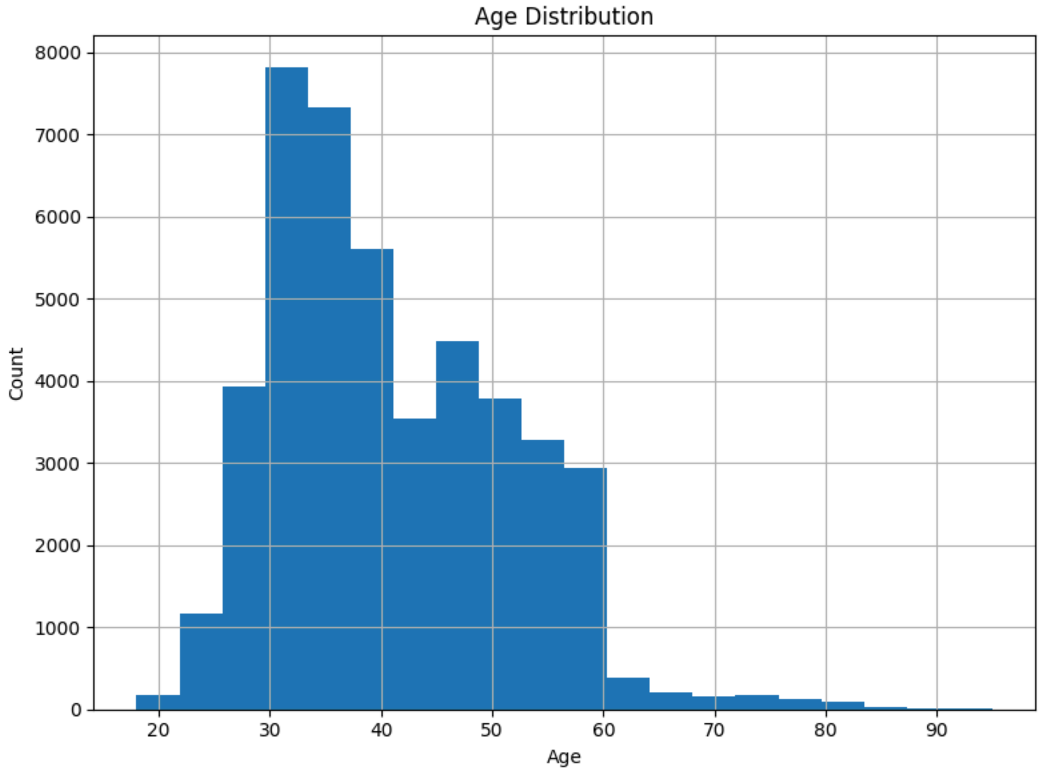
```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  pdays  previous poutcome   y
0  unknown    5   may       261         1     -1         0  unknown  no
1  unknown    5   may       151         1     -1         0  unknown  no
2  unknown    5   may        76         1     -1         0  unknown  no
3  unknown    5   may        92         1     -1         0  unknown  no
4  unknown    5   may       198         1     -1         0  unknown  no
```

```
# VISUALIZATIONS WITH MATPLOTLIB
# Job type distributions

job_counts = df['job'].value_counts()

plt.figure(figsize=(12, 6))
job_counts.plot(kind='bar')
plt.title('Distribution of Job Types')
plt.xlabel('Job Type')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

#age distribution

plt.figure(figsize=(8, 6))
df['age'].hist(bins=20)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

#Marital status distribution

marital_counts = df['marital'].value_counts()

plt.figure(figsize=(8, 6))
marital_counts.plot(kind='pie', autopct='%1.1f%%')
plt.title('Marital Status Distribution')
plt.axis('equal')
plt.tight_layout()
plt.show()

# Education Level distribution
education_counts = df['education'].value_counts()

plt.figure(figsize=(10, 6))
education_counts.plot(kind='bar')
plt.title('Education Distribution')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.tight_layout()
```

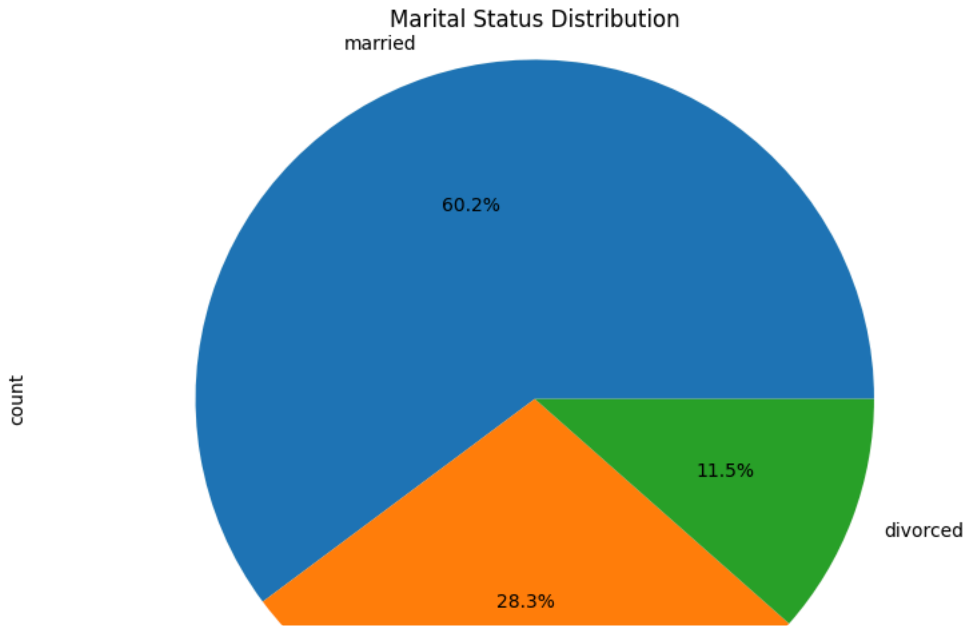
```
plt.show()
```



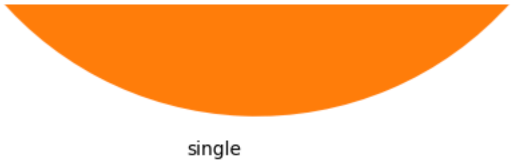
Distribution of Job Types
Count
Job Type
blue-collar
management
technician
admin.
services
retired
self-employed
entrepreneur
unemployed
housemaid
student
unknown

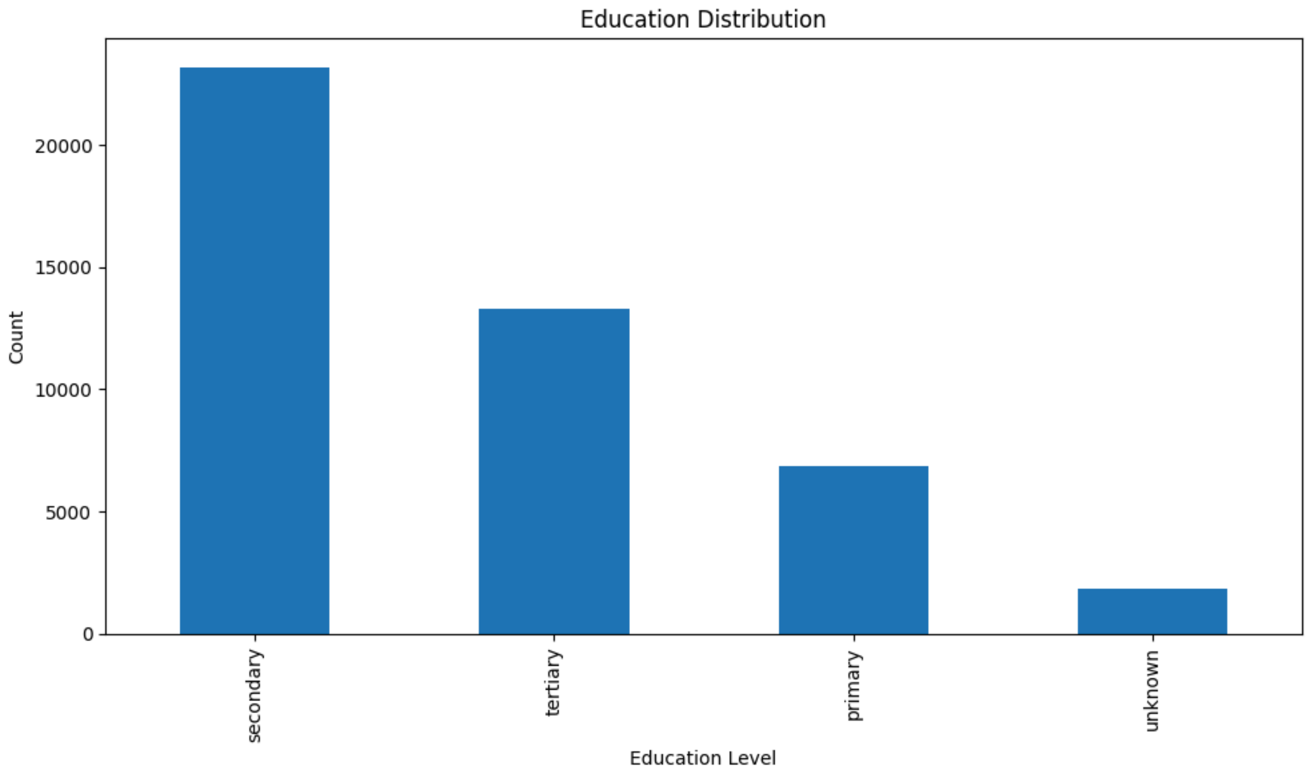


Age Distribution
Count
Age

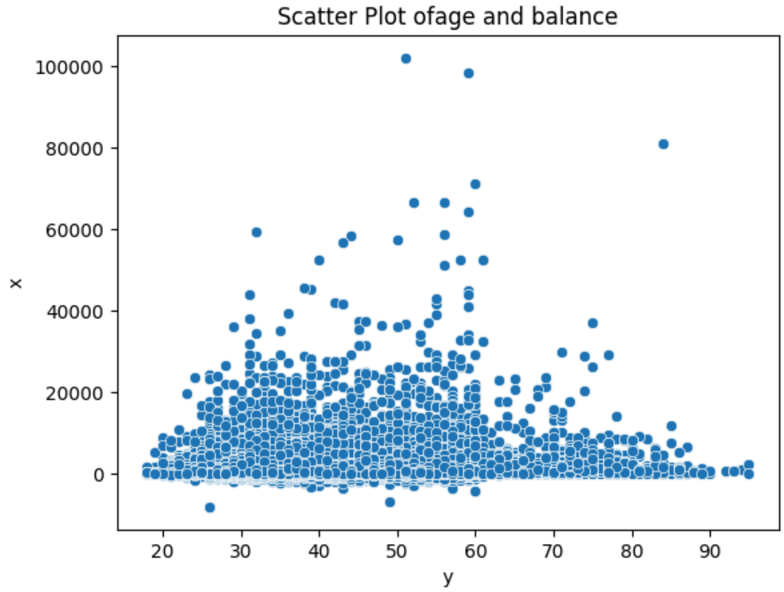


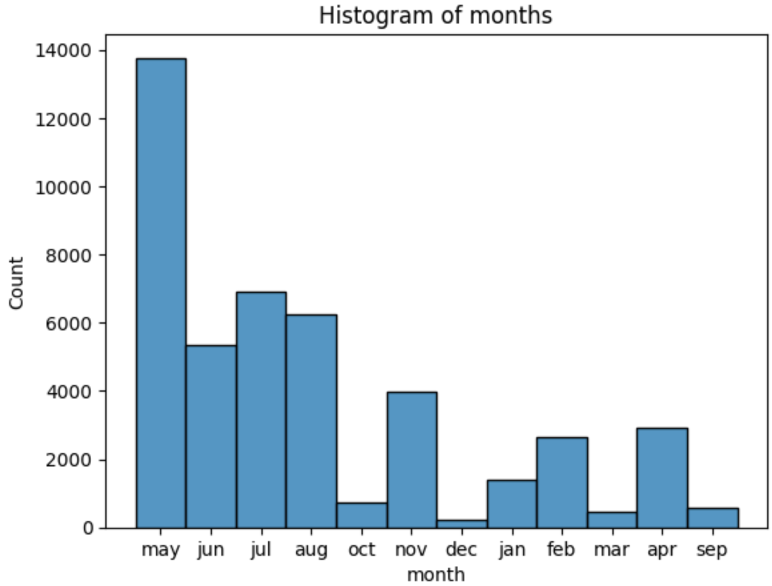
Marital Status Distribution
married
60.2%
28.3%
11.5%
divorced
count

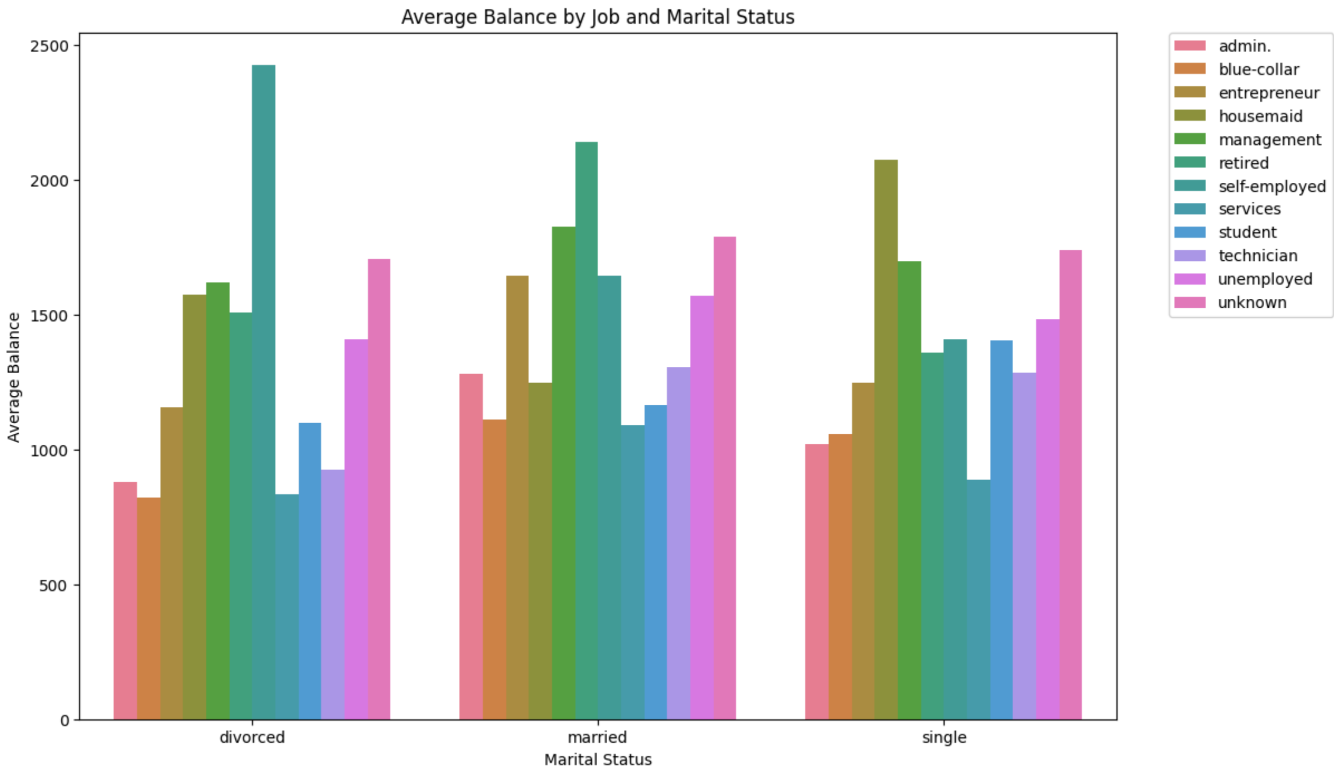

single


Education Distribution
Count
20000
15000
10000
5000
0
secondary
tertiary
primary
unknown
Education Level

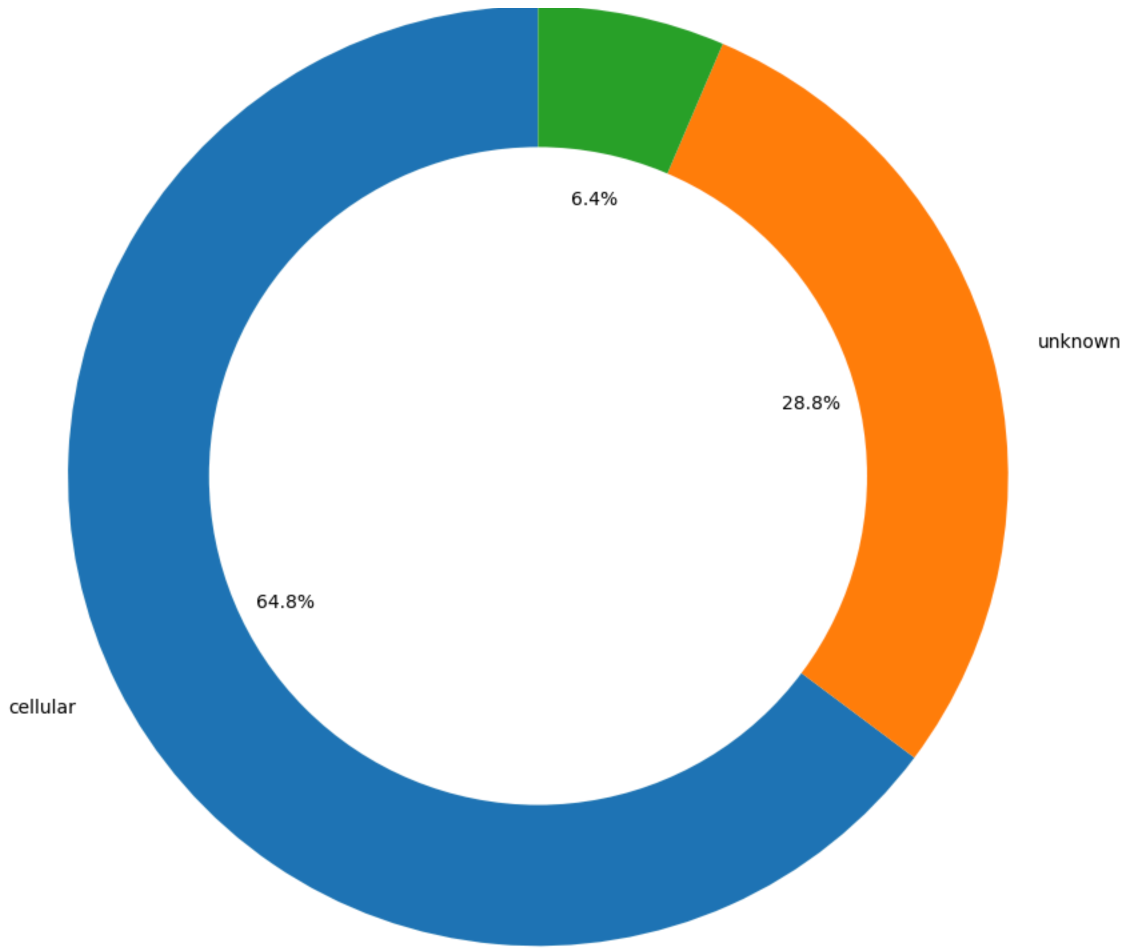
```
# VISUALIZATION WITH SEABORN

# age and balance scatterplot (no distinctive relationship)

sns.scatterplot(x='age', y='balance', data=df)
plt.title('Scatter Plot ofage and balance')
plt.xlabel('y')
plt.ylabel('x')
plt.show()



# histogram of months

sns.histplot(df['month'], bins=20)
plt.title('Histogram of months')
plt.xlabel('month')
plt.ylabel('Count')
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your data is in a DataFrame called 'df'
# and the relevant columns are 'marital_status', 'job', and 'balance'

# Group the data by job and marital status, and calculate the average balance
balance_by_job_marital = df.groupby(['job', 'marital'])['balance'].mean().reset_index()

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 8))

# Create the grouped bar plot
sns.barplot(x='marital', y='balance', hue='job', data=balance_by_job_marital, ax=ax)

# Add labels and title
ax.set_xlabel('Marital Status')
ax.set_ylabel('Average Balance')
ax.set_title('Average Balance by Job and Marital Status')
plt.xticks(rotation=0)
ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.show()

# CONTACT TYPES VISUALIZATION

contact_counts = df['contact'].value_counts()

# Create a donut plot
fig, ax = plt.subplots(figsize=(10, 10))

# Plot the pie chart
total = contact_counts.sum()
sizes = [count / total * 100 for count in contact_counts]
_, _, autotexts = ax.pie(sizes, labels=contact_counts.index, autopct='%1.1f%%', startangle=90)

# Adjust the inner circle to create the donut effect
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
ax.add_artist(centre_circle)

# Set the title and axis labels
ax.set_title('Contact Types', fontsize=16, fontweight='bold')
plt.axis('equal')
plt.show()
```


Scatter Plot ofage and balance
x
y
100000
80000
60000
40000
20000
0
20
30
40
50
60
70
80
90


Histogram of months
Count
month
14000
12000
10000
8000
6000
4000
2000
0
may
jun
jul
aug
oct
nov
dec
jan
feb
mar
apr
sep


Average Balance by Job and Marital Status
Average Balance
Marital Status
2500
2000
1500
1000
500
0
divorced
married
single
admin.
blue-collar
entrepreneur
housemaid
management
retired
self-employed
services
student
technician
unemployed
unknown

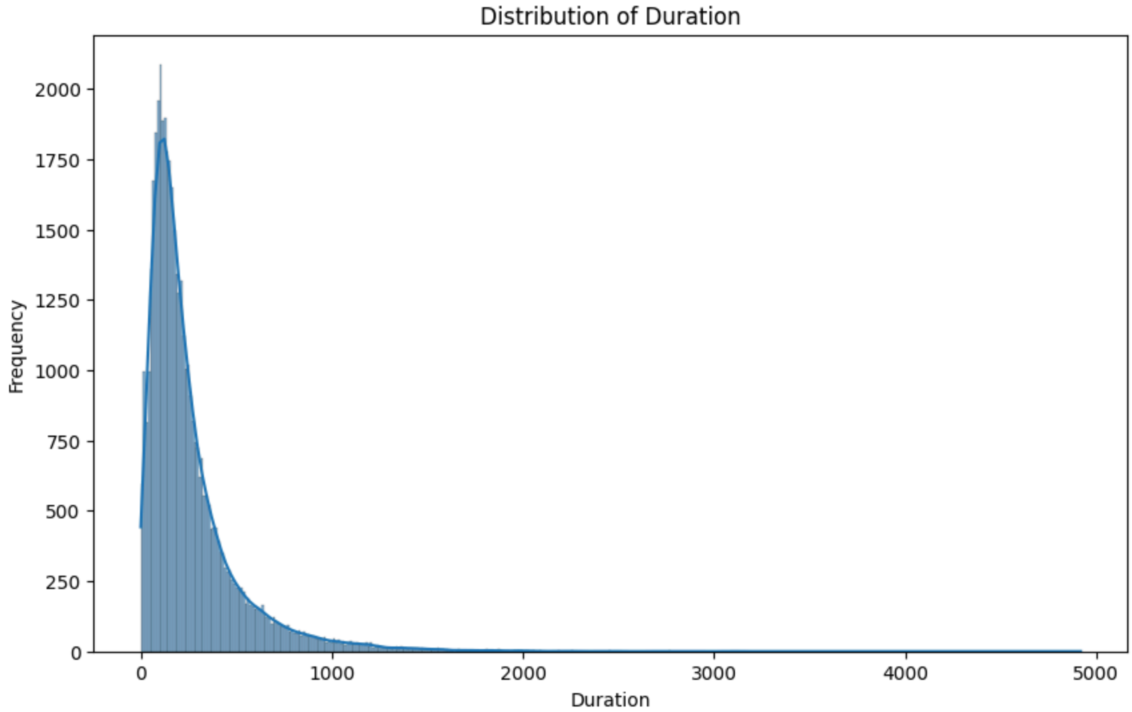

Contact Types
telephone

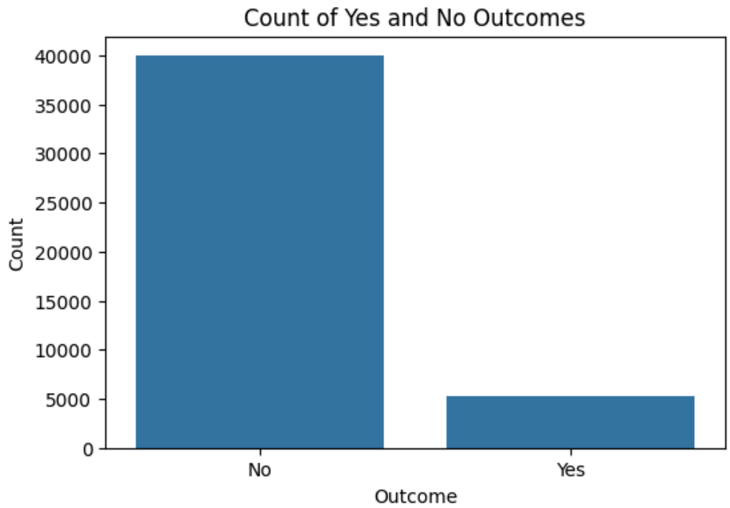

6.4%
unknown
28.8%
64.8%
cellular

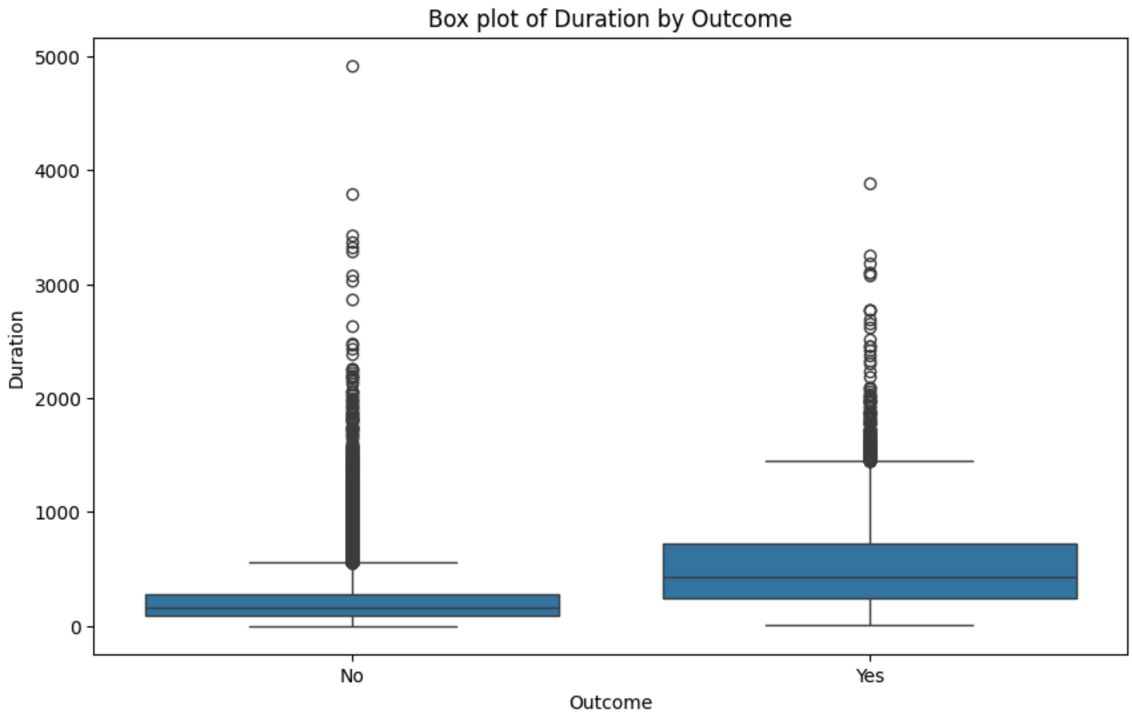
Start coding or generate with AI.

```
#STATISTICAL AND UNIVARIATE-BIVARIATE ANALYSIS

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
import statsmodels.api as sm

# Loading the dataset again
df = pd.read_csv('/content/bank-full.csv')

# Encoding the data in 'y' column: 'yes' -> 1, 'no' -> 0
df['y'] = df['y'].map({'yes': 1, 'no': 0})

# Univariate Analysis for 'duration'
plt.figure(figsize=(10, 6))
sns.histplot(df['duration'], kde=True)
plt.title('Distribution of Duration')
plt.xlabel('Duration')
plt.ylabel('Frequency')
plt.show()

# Univariate Analysis for 'y'
plt.figure(figsize=(6, 4))
sns.countplot(x='y', data=df)
plt.title('Count of Yes and No Outcomes')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.xticks([0, 1], ['No', 'Yes'])
plt.show()

# Bivariate Analysis - Box plot of Duration by Outcome
plt.figure(figsize=(10, 6))
sns.boxplot(x='y', y='duration', data=df)
plt.title('Box plot of Duration by Outcome')
plt.xlabel('Outcome')
plt.ylabel('Duration')
plt.xticks([0, 1], ['No', 'Yes'])
plt.show()

# Split the data into training and testing sets
X = df[['duration']]
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

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

# Detailed output using statsmodels
X_with_const = sm.add_constant(X)  # Add constant term for the intercept
logit_model = sm.Logit(y, X_with_const)
result = logit_model.fit()

# Print the summary of the model
print(result.summary())

# Bivariate Analysis - Logistic Regression Curve
plt.figure(figsize=(10, 6))

# Scatter plot of the actual data points
plt.scatter(X, y, color='blue', label='Data', alpha=0.2)

# Logistic regression curve
X_values = np.linspace(X['duration'].min(), X['duration'].max(), 300)  # Generate 300 points between min and max of duration
X_values_with_const = sm.add_constant(X_values)  # Add constant to the X values for prediction
y_values = result.predict(X_values_with_const)  # Predict using the logistic regression model
```

```
plt.plot(X_values, y_values, color='red', linewidth=2, label='Logistic Regression')

plt.xlabel('Duration')
plt.ylabel('Probability of Yes (y=1)')
plt.title('Logistic Regression between Duration and Loan')
plt.legend()
plt.grid(True)


# Scatter plot of the actual data points
plt.scatter(X, y, color='blue', label='Data')

# Logistic regression curve
X_values = np.linspace(X['duration'].min(), X['duration'].max(), 300)  # Generate 300 points between min and max of duration
X_values_with_const = sm.add_constant(X_values)  # Add constant to the X values for prediction
y_values = result.predict(X_values_with_const)  # Predict using the logistic regression model

plt.plot(X_values, y_values, color='red', linewidth=2, label='Logistic Regression')
```

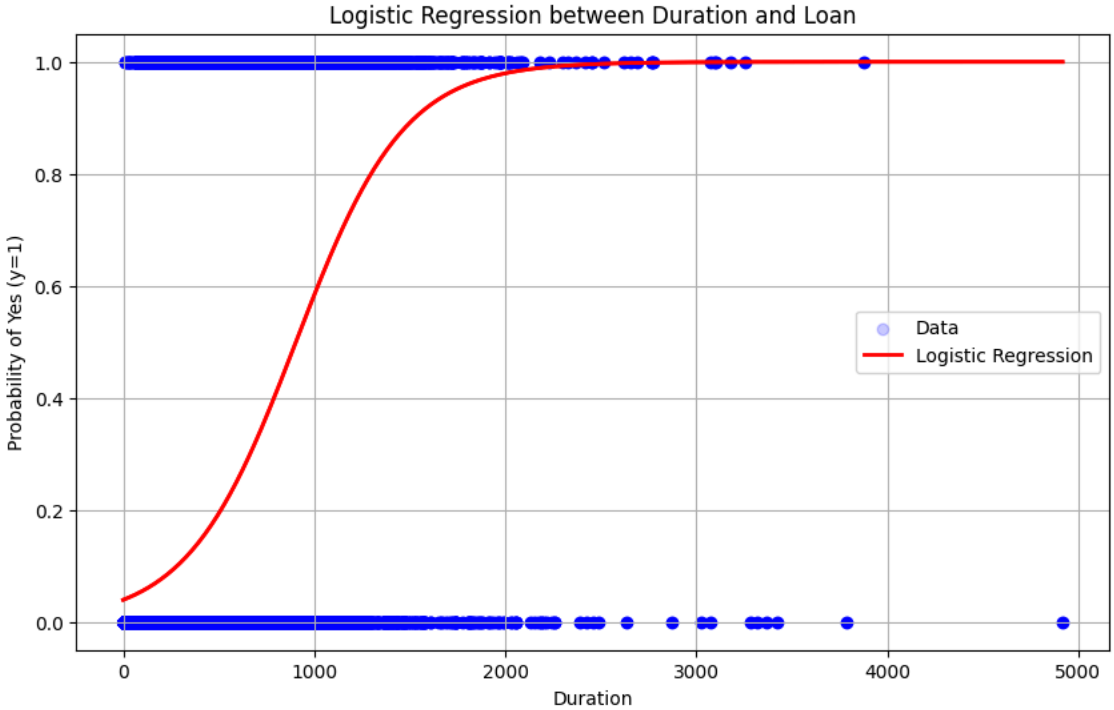







```
Accuracy: 0.8876437629017989
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94     11966
```

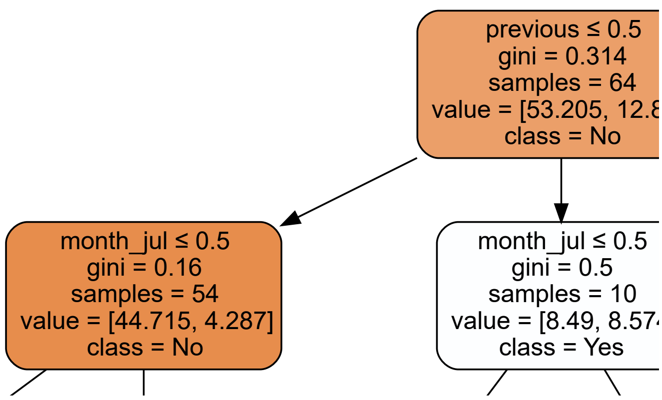
```
           0       0.90      0.98      0.94     11966
           1       0.58      0.16      0.25      1598

    accuracy                           0.89     13564
   macro avg       0.74      0.57      0.60     13564
weighted avg       0.86      0.89      0.86     13564

Optimization terminated successfully.
         Current function value: 0.304148
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                45211
Model:                          Logit   Df Residuals:                    45209
Method:                           MLE   Df Model:                            1
Date:                Mon, 16 Sep 2024   Pseudo R-squ.:                  0.1572
Time:                        08:20:35   Log-Likelihood:                -13751.
converged:                       True   LL-Null:                       -16315.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.1925      0.026   -122.189      0.000      -3.244      -3.141
duration       0.0035   5.48e-05     64.377      0.000       0.003       0.004
==============================================================================
[<matplotlib.lines.Line2D at 0x7b5bda8c6c20>]
```

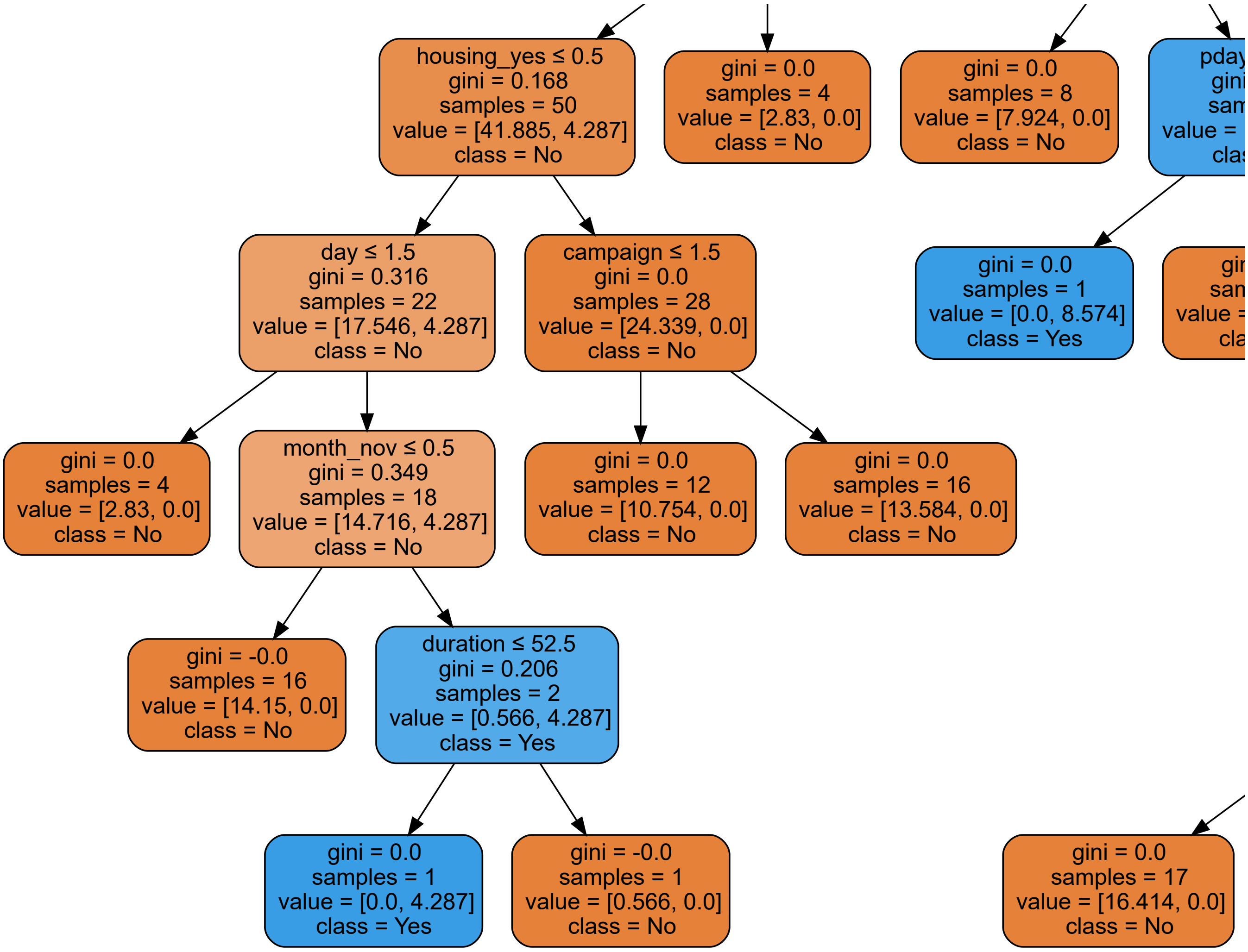


```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# Extract relevant columns for features
features = ['housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous']
X = pd.get_dummies(df[features], drop_first=True)  # One-hot encode categorical features

# Target variable
y = df['y']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and Balance the Random Forest classifier
clf = RandomForestClassifier(random_state=42, class_weight='balanced') # to reduce imbalance in y column

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

# Predict on test data
y_pred = clf.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))    #high accuracy above 0.7
```

```
Accuracy: 0.8987761722205839
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.96      0.94     11966
           1       0.59      0.45      0.51      1598

    accuracy                           0.90     13564
   macro avg       0.76      0.70      0.73     13564
weighted avg       0.89      0.90      0.89     13564
```

```
# create one single tree from this random forest.
single_tree = clf.estimators_[0]

# Visualize the single decision tree
from sklearn.tree import export_graphviz
import graphviz

dot_data = export_graphviz(single_tree, out_file=None,
                         feature_names=X.columns,
                         class_names=['No', 'Yes'],
                         filled=True, rounded=True,
                         special_characters=True)
graph = graphviz.Source(dot_data)
graph
```


previous ≤ 0.5
gini = 0.314
samples = 64
value = [53.205, 12.
class = No
month_jul ≤ 0.5
gini = 0.16
samples = 54
value = [44.715, 4.287]
class = No
month_jul ≤ 0.5
gini = 0.5
samples = 10
value = [8.49, 8.57
class = Yes


housing_yes ≤ 0.5
gini = 0.168
samples = 50
value = [41.885, 4.287]
class = No
gini = 0.0
samples = 4
value = [2.83, 0.0]
class = No
gini = 0.0
samples = 8
value = [7.924, 0.0]
class = No
day ≤ 1.5
gini = 0.316
samples = 22
value = [17.546, 4.287]
class = No
campaign ≤ 1.5
gini = 0.0
samples = 28
value = [24.339, 0.0]
class = No
gini = 0.0
samples = 1
value = [0.0, 8.574]
class = Yes
gini = 0.0
samples = 4
value = [2.83, 0.0]
class = No
month_nov ≤ 0.5
gini = 0.349
samples = 18
value = [14.716, 4.287]
class = No
gini = 0.0
samples = 12
value = [10.754, 0.0]
class = No
gini = 0.0
samples = 16
value = [13.584, 0.0]
class = No
gini = -0.0
samples = 16
value = [14.15, 0.0]
class = No
duration ≤ 52.5
gini = 0.206
samples = 2
value = [0.566, 4.287]
class = Yes
gini = 0.0
samples = 1
value = [0.0, 4.287]
class = Yes
gini = -0.0
samples = 1
value = [0.566, 0.0]
class = No
gini = 0.0
samples = 17
value = [16.414, 0.0]
class = No

```
#grid search

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 300}
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# dt model creation
dt_model = DecisionTreeClassifier()

# hyperparameter grid definition
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# start grid search
grid_search_dt = GridSearchCV(estimator=dt_model, param_grid=param_grid_dt, cv=5)
grid_search_dt.fit(X_train, y_train)

# get the best hyperparameters
print("Best parameters for decision tree are:", grid_search_dt.best_params_)

# get the best dt model
best_dt_model = grid_search_dt.best_estimator_  #Best parameters for decision tree are: {'criterion': 'gini', 'max_depth': 10, 'min_samp
```

```
Best parameters for decision tree are: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 5}
```

```
# best parameters for the random forest

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# create random forest
rf_model = RandomForestClassifier()

# define the grid search
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# and start the grid search
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5)
grid_search_rf.fit(X_train, y_train)

# get the best parameters
print("best parameters for random forest are:", grid_search_rf.best_params_)

# get the best model
best_rf_model = grid_search_rf.best_estimator_
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-15-f9b2b5515c63> in <cell line: 19>()
     17 # and start the grid search
     18 grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5)
---> 19 grid_search_rf.fit(X_train, y_train)
     20
```

```
# new DT

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Extract relevant columns for features
features = ['housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous']
X = pd.get_dummies(df[features], drop_first=True)  # One-hot encode categorical features

# Target variable
y = df['y']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create Decision Tree classifier with best parameters
clf = DecisionTreeClassifier(criterion='gini', max_depth=10, min_samples_leaf=4, min_samples_split=5, random_state=42)

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

# Predict on test data
y_pred = clf.predict(X_test)

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

```
Accuracy: 0.8960483633146564
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.96      0.94     11966
           1       0.59      0.39      0.47      1598
```