Import Library & Data

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

In [ ]:

```
raw_data = pd.read_csv('../DataFolder/trainset.csv')
raw_data.head()
```

Out[ ]:

	age	job	marital	education	housing	loan	contact	month	day_of_week	duration	campaign	p
0	41	blue-collar	divorced	basic.4y	yes	no	telephone	may	mon	1575	1	
1	49	entrepreneur	married	university.degree	yes	no	telephone	may	mon	1042	1	
2	49	technician	married	basic.9y	no	no	telephone	may	mon	1467	1	
3	41	technician	married	professional.course	yes	no	telephone	may	mon	579	1	
4	45	blue-collar	married	basic.9y	yes	no	telephone	may	mon	461	1	

Data Cleaning Process

Data Exploration

In [ ]:

```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29271 entries, 0 to 29270
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          29271 non-null  int64  
 1   job          29271 non-null  object 
 2   marital      29271 non-null  object 
 3   education    29271 non-null  object 
 4   housing      29271 non-null  object 
 5   loan         29271 non-null  object 
 6   contact      29271 non-null  object 
 7   month        29271 non-null  object 
 8   day_of_week  29271 non-null  object 
 9   duration     29271 non-null  int64  
 10  campaign     29271 non-null  int64  
 11  pdays        29271 non-null  int64  
 12  poutcome     29271 non-null  object 
 13  nr.employed  29271 non-null  float64
 14  Subscribed   29271 non-null  object 
dtypes: float64(1), int64(4), object(10)
memory usage: 3.3+ MB
```

In [ ]:

```
# covert unknown Data to np.Nan
raw_data = raw_data.replace('unknown',np.nan)
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29271 entries, 0 to 29270
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          29271 non-null  int64  
 1   job          29011 non-null  object 
 2   marital      29220 non-null  object 
 3   education    28044 non-null  object 
 4   housing      28558 non-null  object 
 5   loan         28558 non-null  object 
 6   contact      29271 non-null  object 
 7   month        29271 non-null  object 
 8   day_of_week  29271 non-null  object 
 9   duration     29271 non-null  int64  
 10  campaign     29271 non-null  int64  
 11  pdays        29271 non-null  int64  
 12  poutcome     29271 non-null  object 
 13  nr.employed  29271 non-null  float64
 14  Subscribed   29271 non-null  object 
dtypes: float64(1), int64(4), object(10)
memory usage: 3.3+ MB
```

In [ ]:
```
raw_data.columns
```

Out[ ]:
```
Index(['age', 'job', 'marital', 'education', 'housing', 'loan', 'contact',
       'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'poutcome',
       'nr.employed', 'Subscribed'],
      dtype='object')
```

In [ ]:
```
# Check Empty Data
raw_data["job"].unique()
raw_data.isna().sum()
```

Out[ ]:
```
age              0
job            260
marital         51
education     1227
housing        713
loan           713
contact          0
month            0
day_of_week      0
duration         0
campaign         0
pdays            0
poutcome         0
nr.employed      0
Subscribed       0
dtype: int64
```

In [ ]:
```
#Check no of unknown
raw_data = raw_data.dropna(subset=["job","marital","education","housing","loan"])
raw_data.isna().sum()
```

Out[ ]:
```
age            0
job            0
marital        0
education      0
housing        0
loan           0
contact        0
month          0
day_of_week    0
duration       0
campaign       0
pdays          0
poutcome       0
nr.employed    0
Subscribed     0
dtype: int64
```

In [ ]:
```
#Data drop is around 7%. The remaining dataset is as follow
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 27178 entries, 0 to 29270
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          27178 non-null  int64  
 1   job          27178 non-null  object 
 2   marital      27178 non-null  object 
 3   education    27178 non-null  object 
 4   housing      27178 non-null  object 
 5   loan         27178 non-null  object 
 6   contact      27178 non-null  object 
 7   month        27178 non-null  object 
 8   day_of_week  27178 non-null  object 
 9   duration     27178 non-null  int64  
 10  campaign     27178 non-null  int64  
 11  pdays        27178 non-null  int64  
 12  poutcome     27178 non-null  object 
 13  nr.employed  27178 non-null  float64
 14  Subscribed   27178 non-null  object 
dtypes: float64(1), int64(4), object(10)
memory usage: 3.3+ MB
```

In [ ]:
```
#Ratio on Subscribed
ratio =  (raw_data['Subscribed'].value_counts()['no']/raw_data['Subscribed'].value_counts().sum())*100
ratio
```

Out[ ]:
```
89.07204356464787
```

In [ ]:
```
#CheckPoint
check_point = raw_data.copy()
```

Data Visualization

Divide the variables into “Categorical” and “Numerical”

In [ ]:
```
data_categorical = check_point.select_dtypes(include='object')
data_numeric = check_point.select_dtypes(include=['int64','float64'])
```

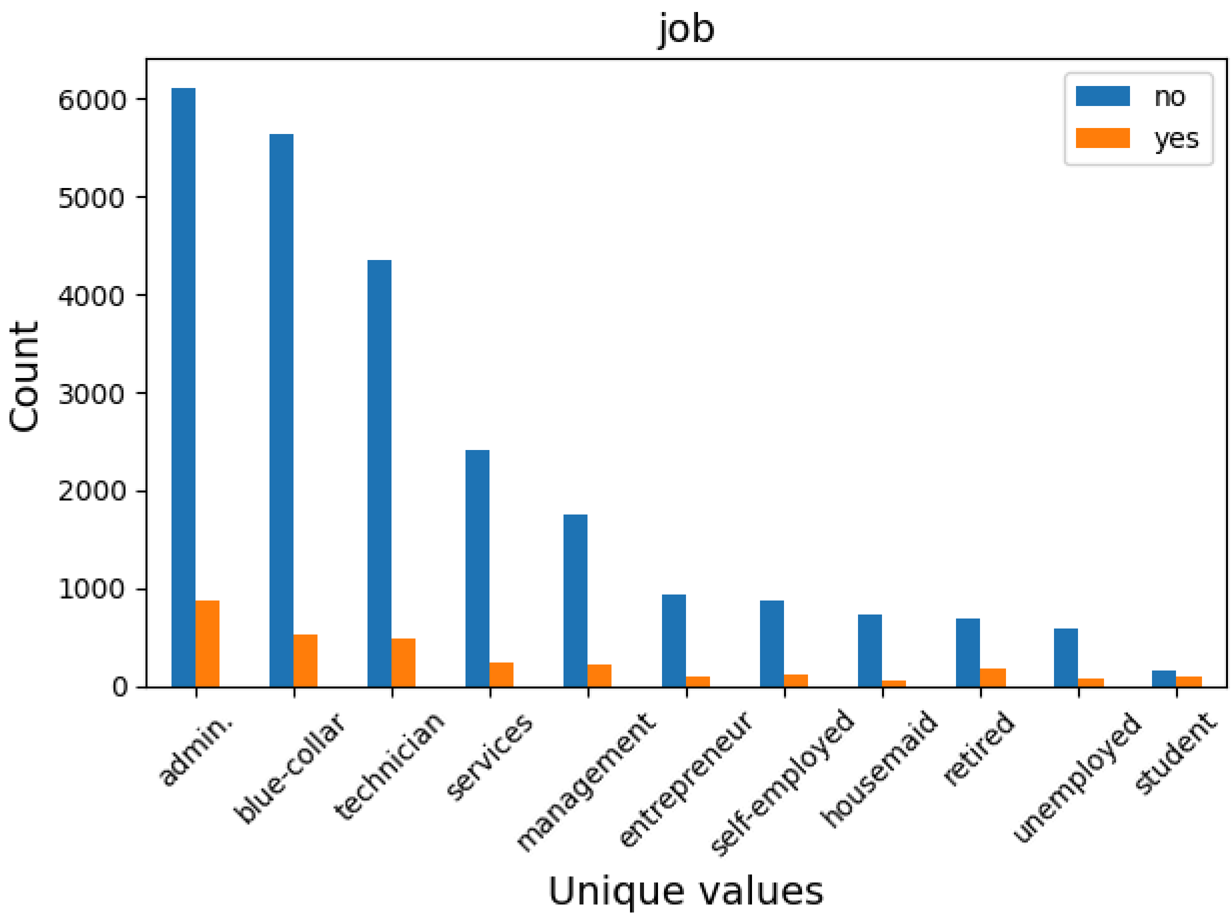
```python
col_cat = data_categorical.columns
col_num = data_numeric.columns
```

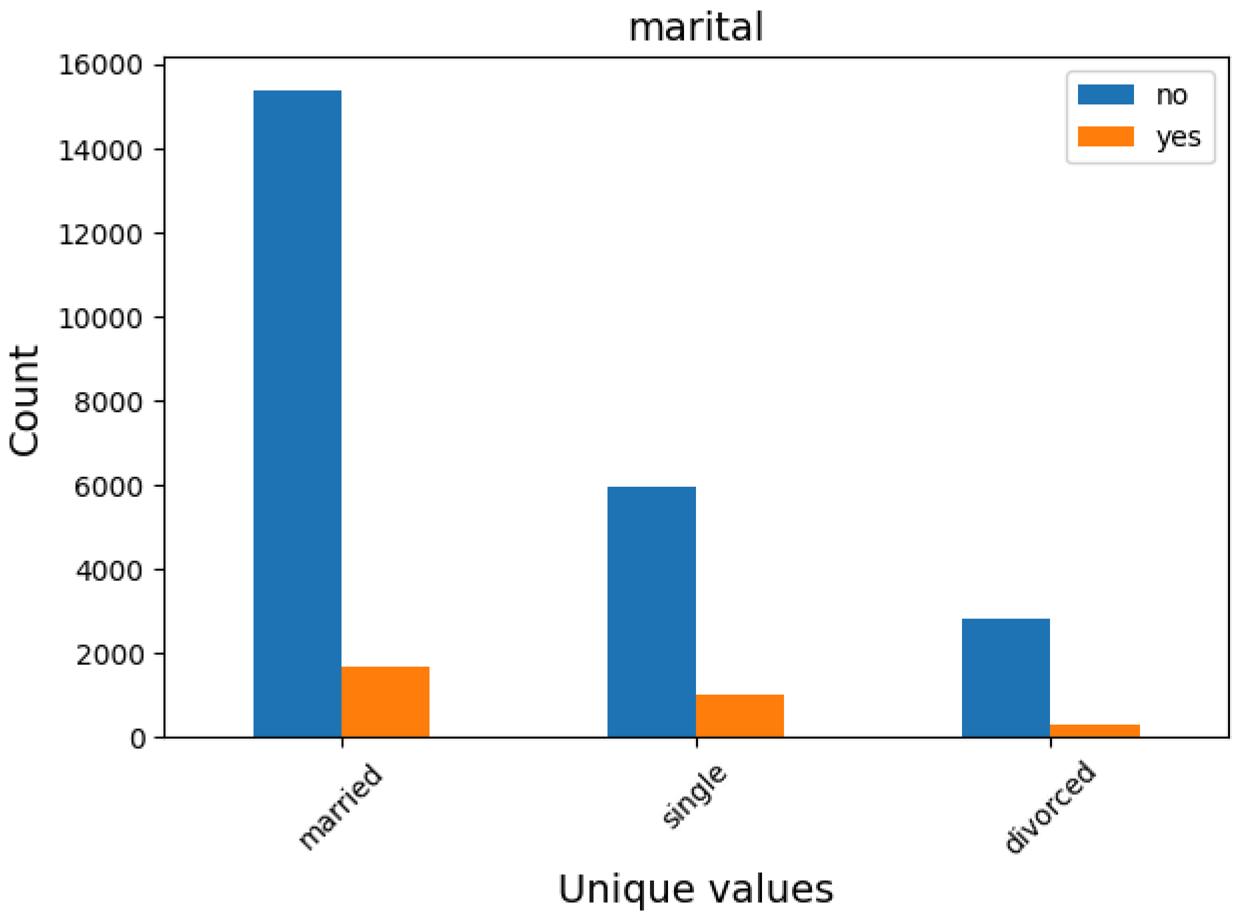
In [ ]:
```python
class UnderstandingData:

    def __init__(self, raw_df):
        self.raw_df = raw_df
        self.raw_df_grouped = raw_df.groupby("Subscribed")
        self.class_name_no = "no"
        self.class_name_yes = "yes"
        self.raw_df_grouped_no = self.raw_df_grouped.get_group(self.class_name_no)
        self.raw_df_grouped_yes = self.raw_df_grouped.get_group(self.class_name_yes)

    def plot_histogram_continuous(self, feature_name, bin_size):
        plt.figure()
        plt.hist(self.raw_df_grouped_no[feature_name], bins=bin_size, label=self.class_name_no)
        plt.hist(self.raw_df_grouped_yes[feature_name], bins=bin_size, label=self.class_name_yes)
        plt.legend()
        plt.title(feature_name, fontsize=14)
        plt.ylabel("Count", fontsize=14)
        plt.xticks(rotation=45)
        plt.xlabel("Unique values", fontsize=14)

    def plot_histogram_categorical(self, feature_name):
        feature_df = pd.DataFrame()
        feature_df["no"] = self.raw_df_grouped_no[feature_name].value_counts()
        feature_df["yes"] = self.raw_df_grouped_yes[feature_name].value_counts()

        feature_df.plot(kind='bar')
        plt.title(feature_name, fontsize=14)
        plt.ylabel("Count", fontsize=14)
        plt.xticks(rotation=45)
        plt.xlabel("Unique values", fontsize=14)
        plt.tight_layout()
```

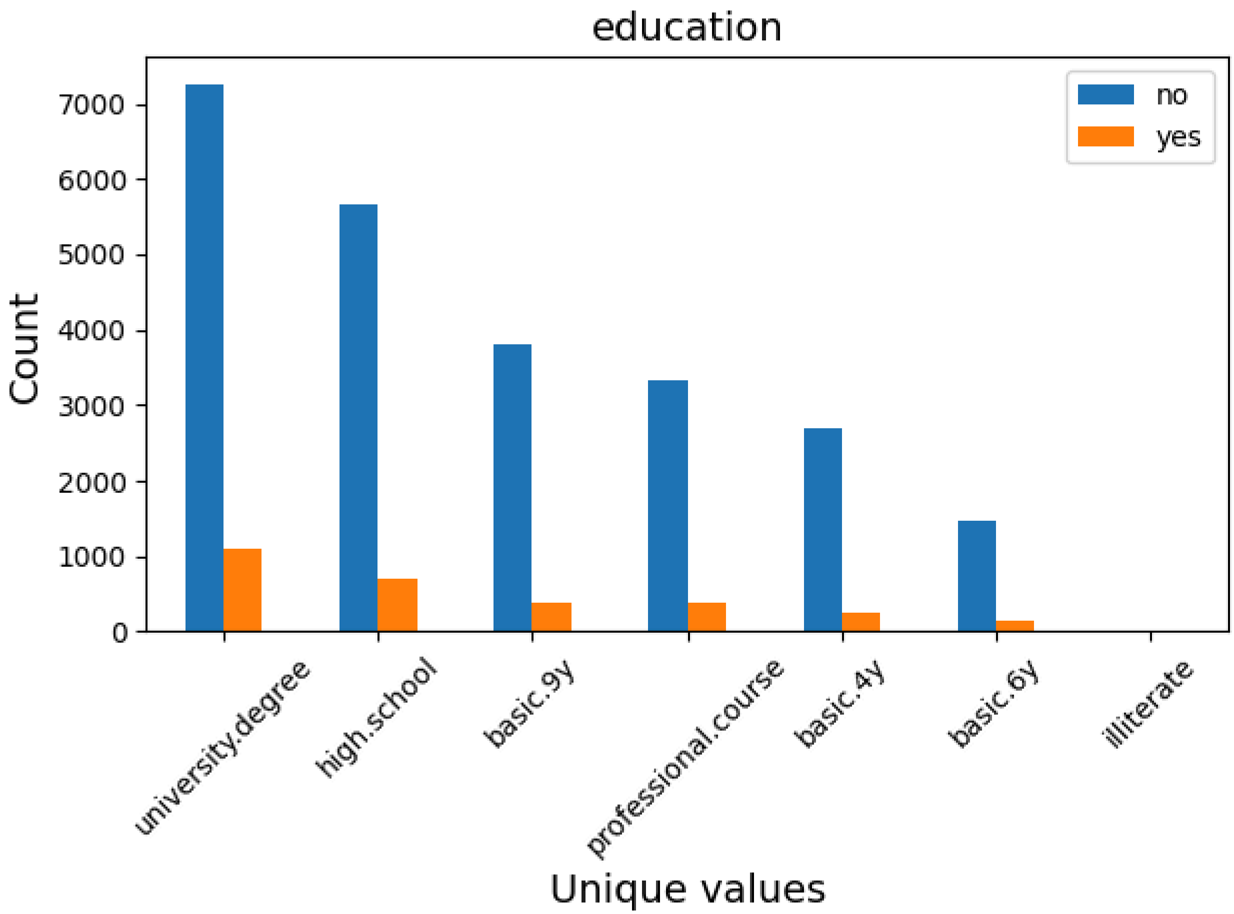
Categorical variables analysis

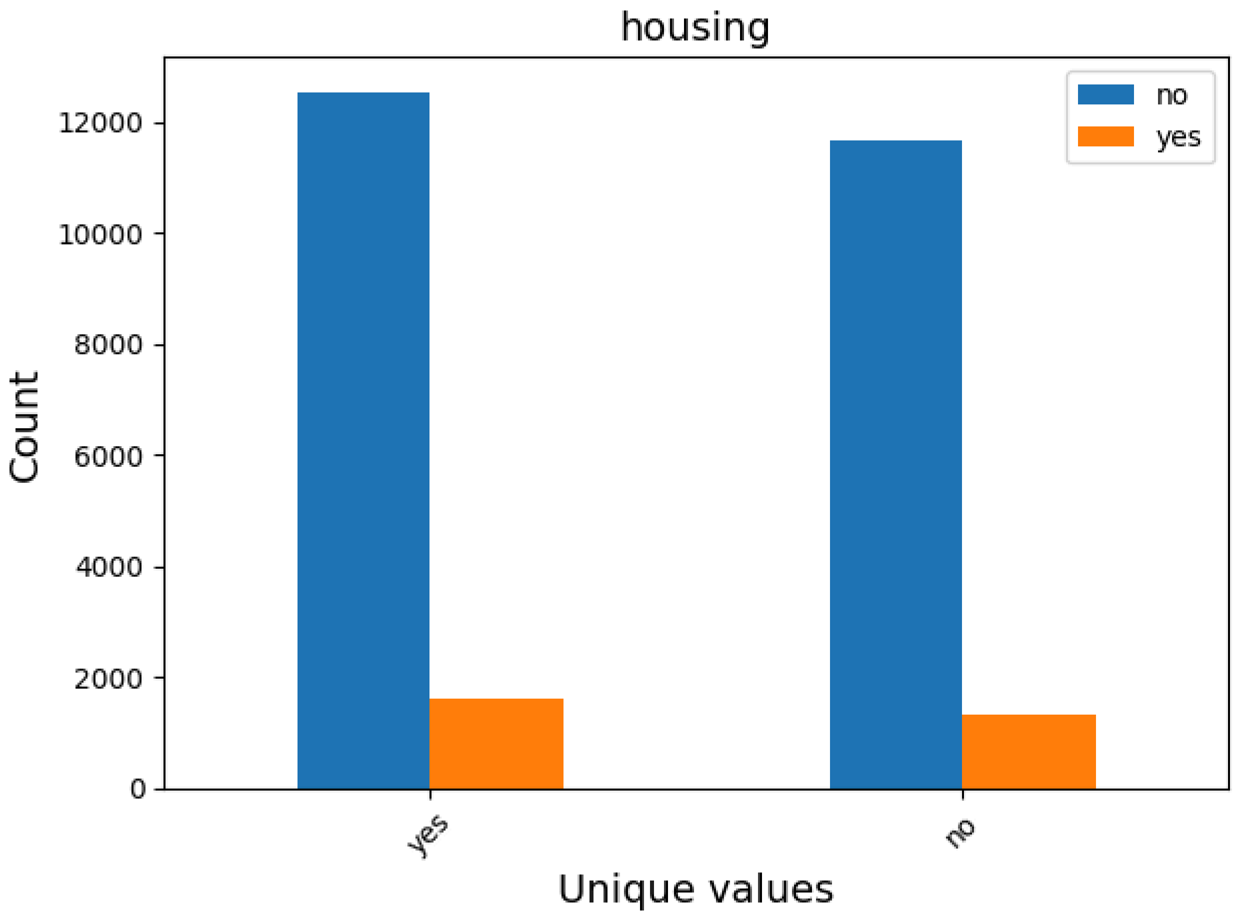
In [ ]:
```python
understanding_data = UnderstandingData(check_point)

understanding_data.plot_histogram_categorical("job")
```

```python
understanding_data.plot_histogram_categorical("marital")
understanding_data.plot_histogram_categorical("education")
understanding_data.plot_histogram_categorical("housing")
understanding_data.plot_histogram_categorical("loan")
understanding_data.plot_histogram_categorical("contact")
understanding_data.plot_histogram_categorical("day_of_week")
understanding_data.plot_histogram_categorical("month")
understanding_data.plot_histogram_categorical("poutcome")
```

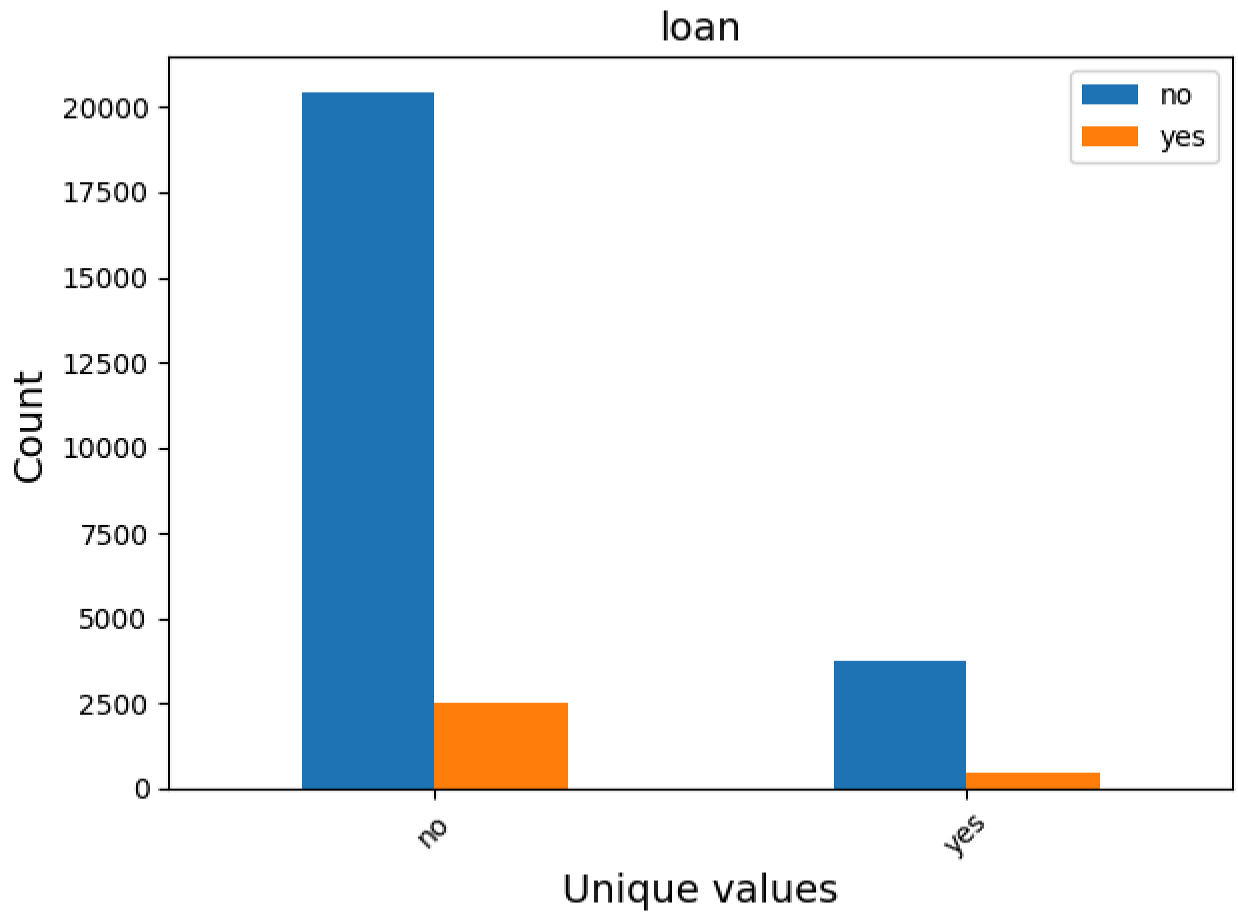


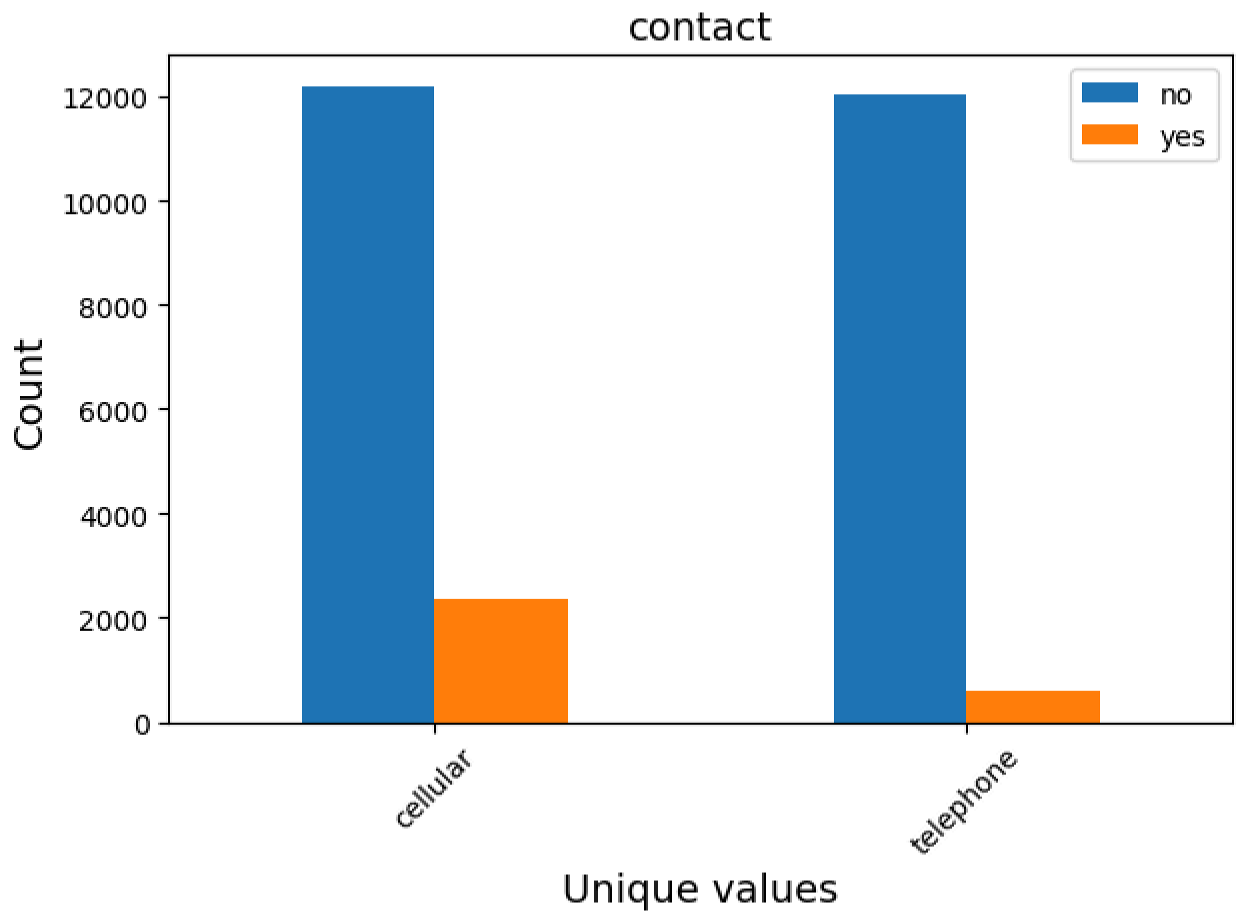

marital
no
yes
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
married
single
divorced
Unique values

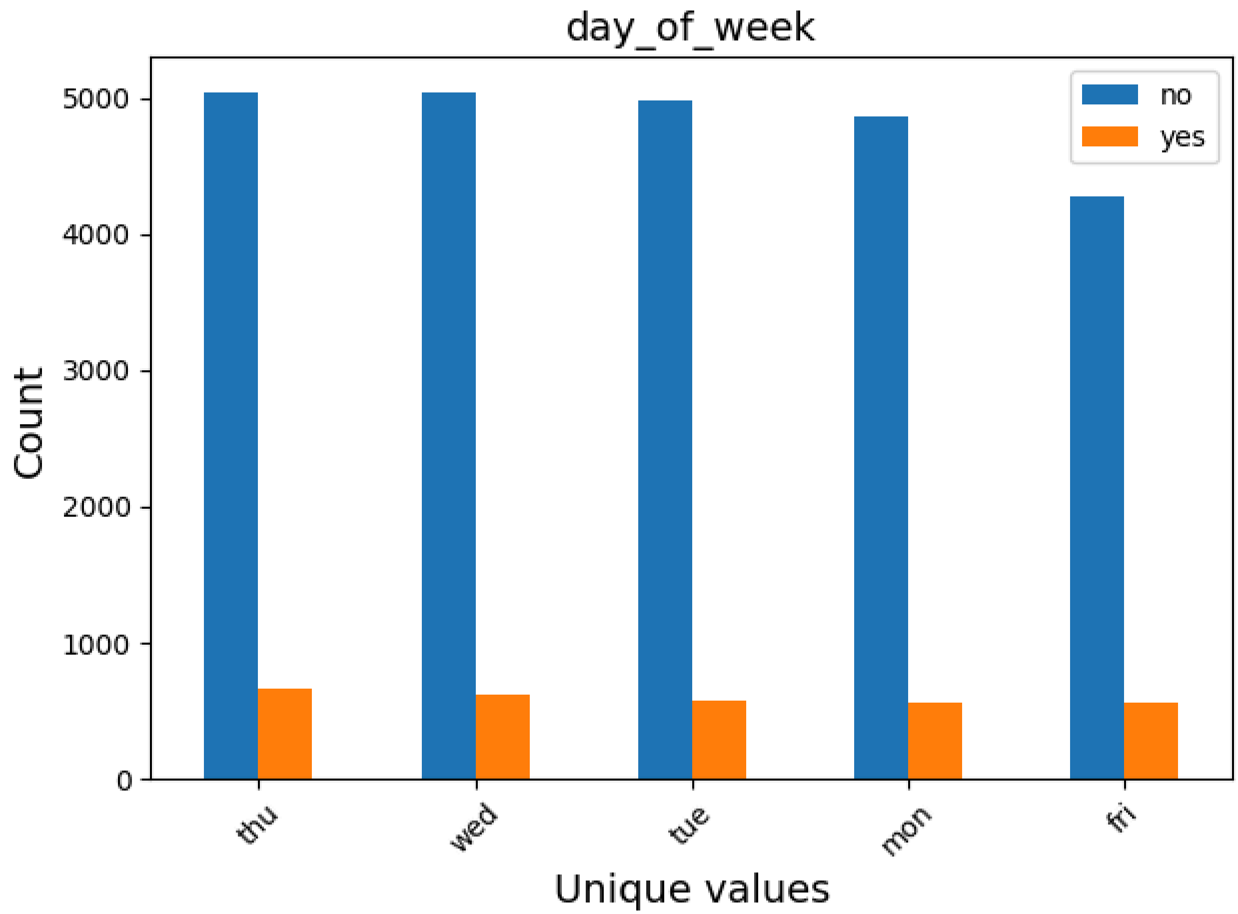

education
Count
Unique values
no
yes
0
1000
2000
3000
4000
5000
6000
7000
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
illiterate

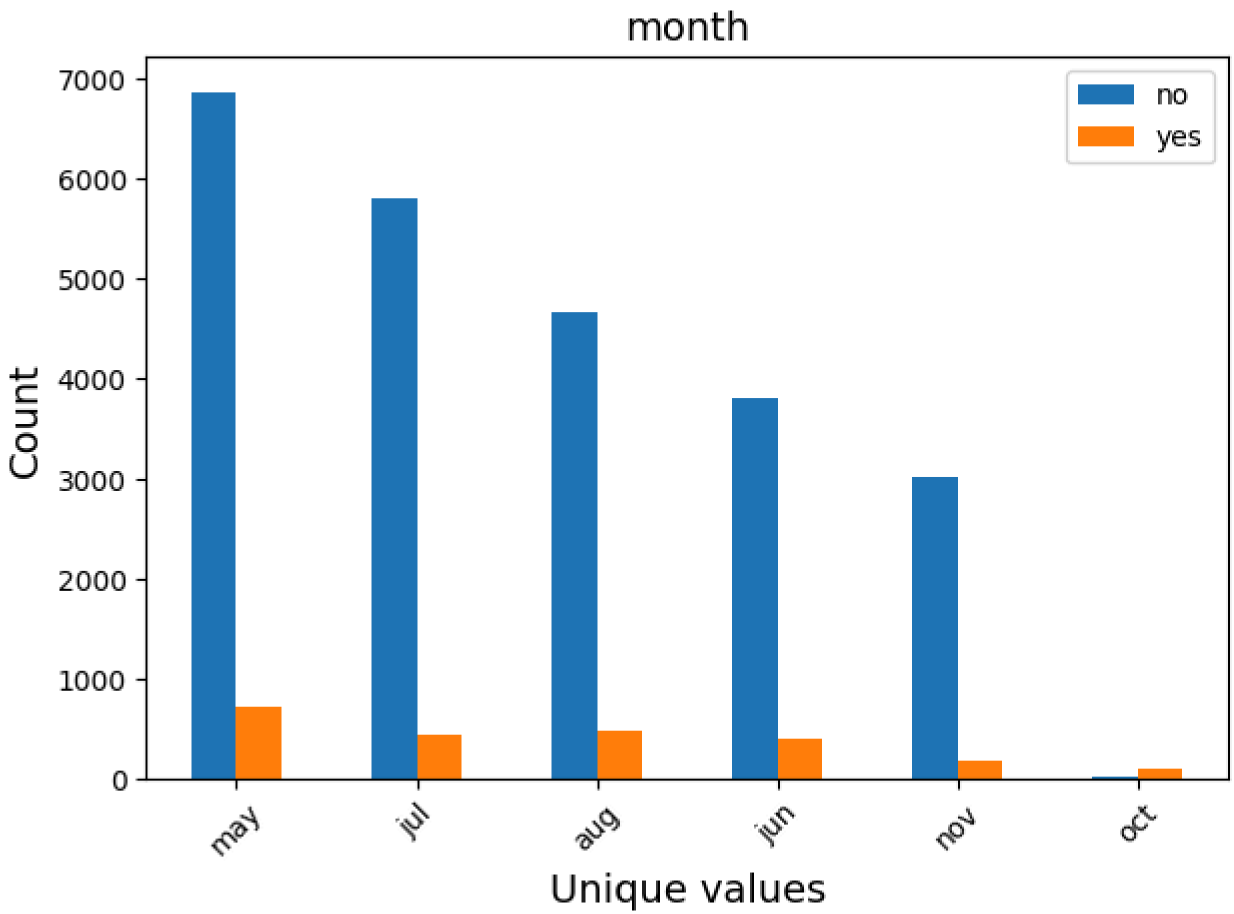


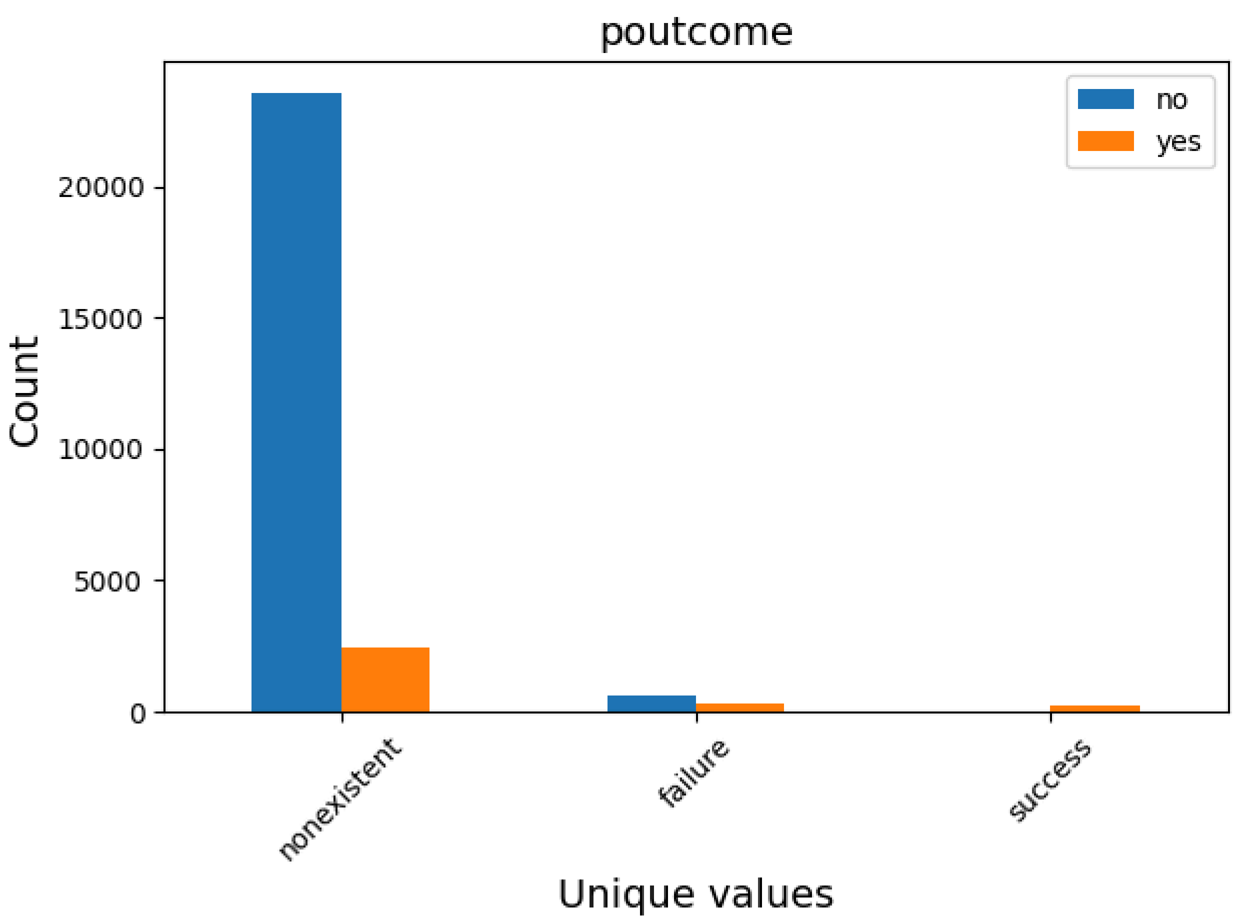
housing
Count
12000
10000
8000
6000
4000
2000
0
no
yes
yes
no
Unique values


loan
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
no
yes
no
yes
Unique values


contact
no
yes
Count
12000
10000
8000
6000
4000
2000
0
cellular
telephone
Unique values

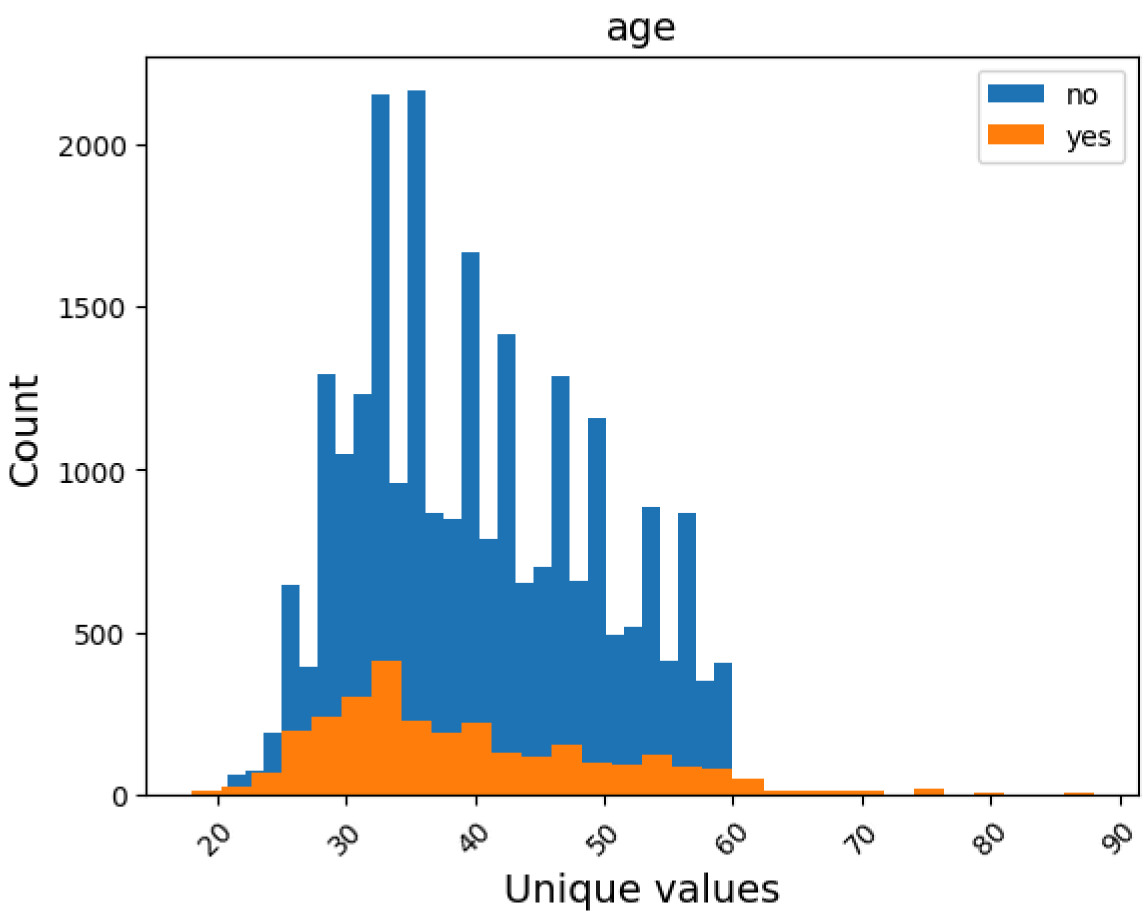

day_of_week
no
yes
Count
5000
4000
3000
2000
1000
0
thu
wed
tue
mon
fri
Unique values


month
7000
6000
5000
4000
3000
2000
1000
0
Count
no
yes
may
jul
aug
jun
nov
oct
Unique values

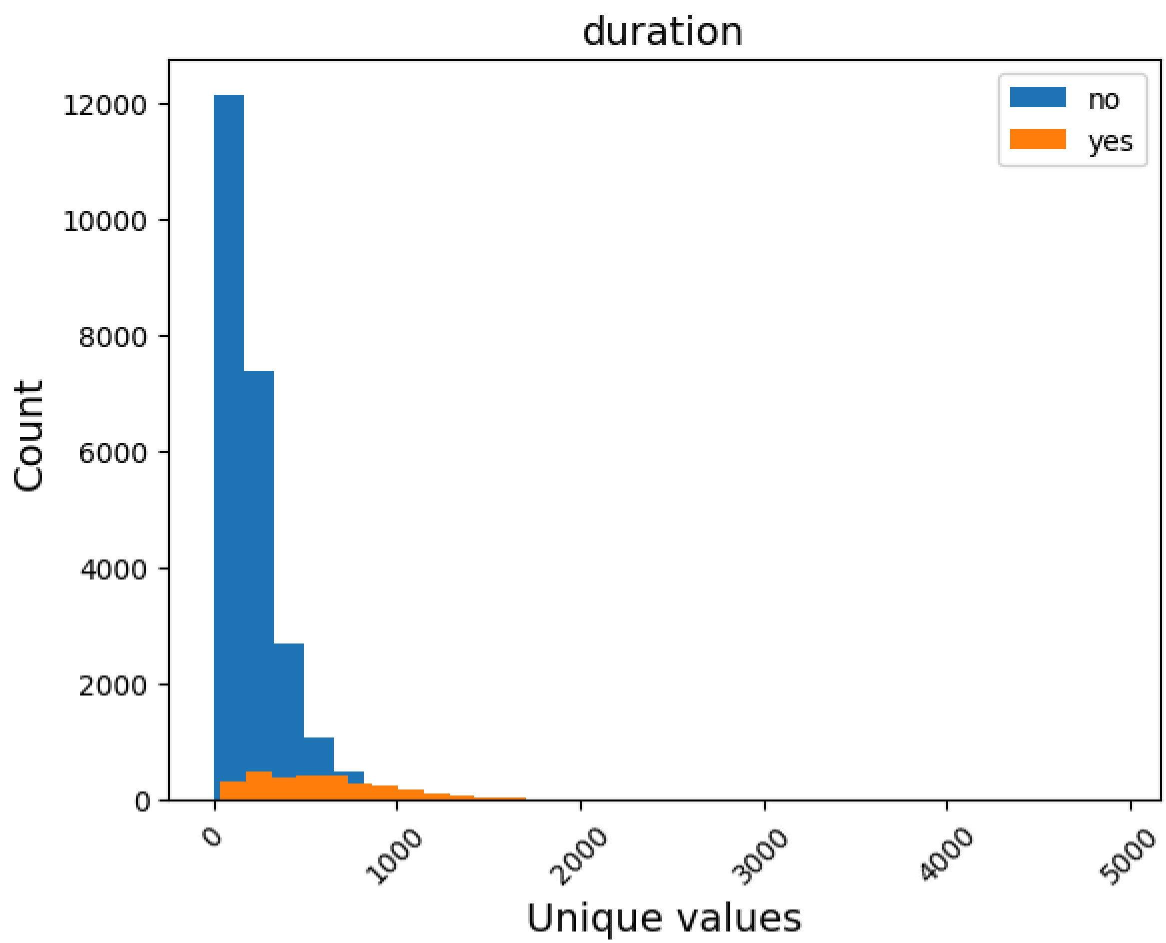


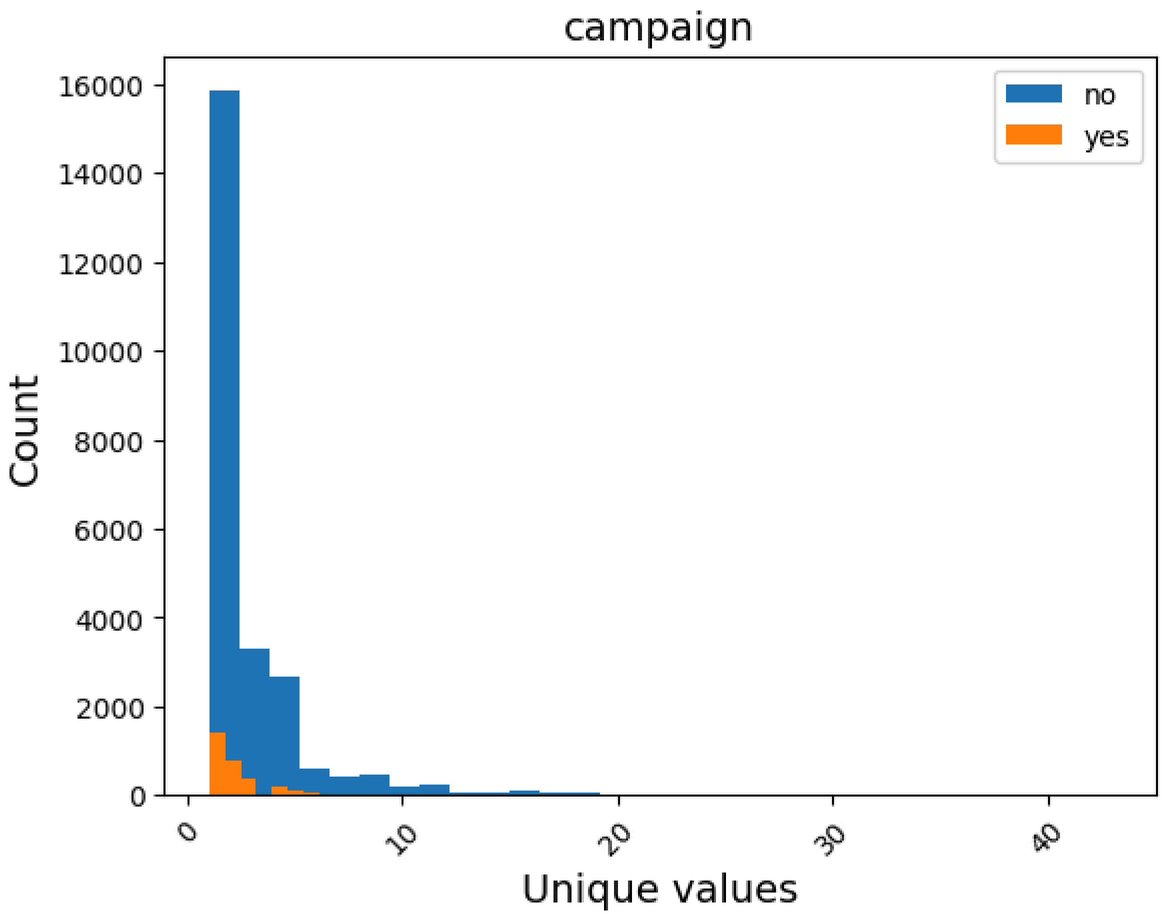
Numerical variables analysis

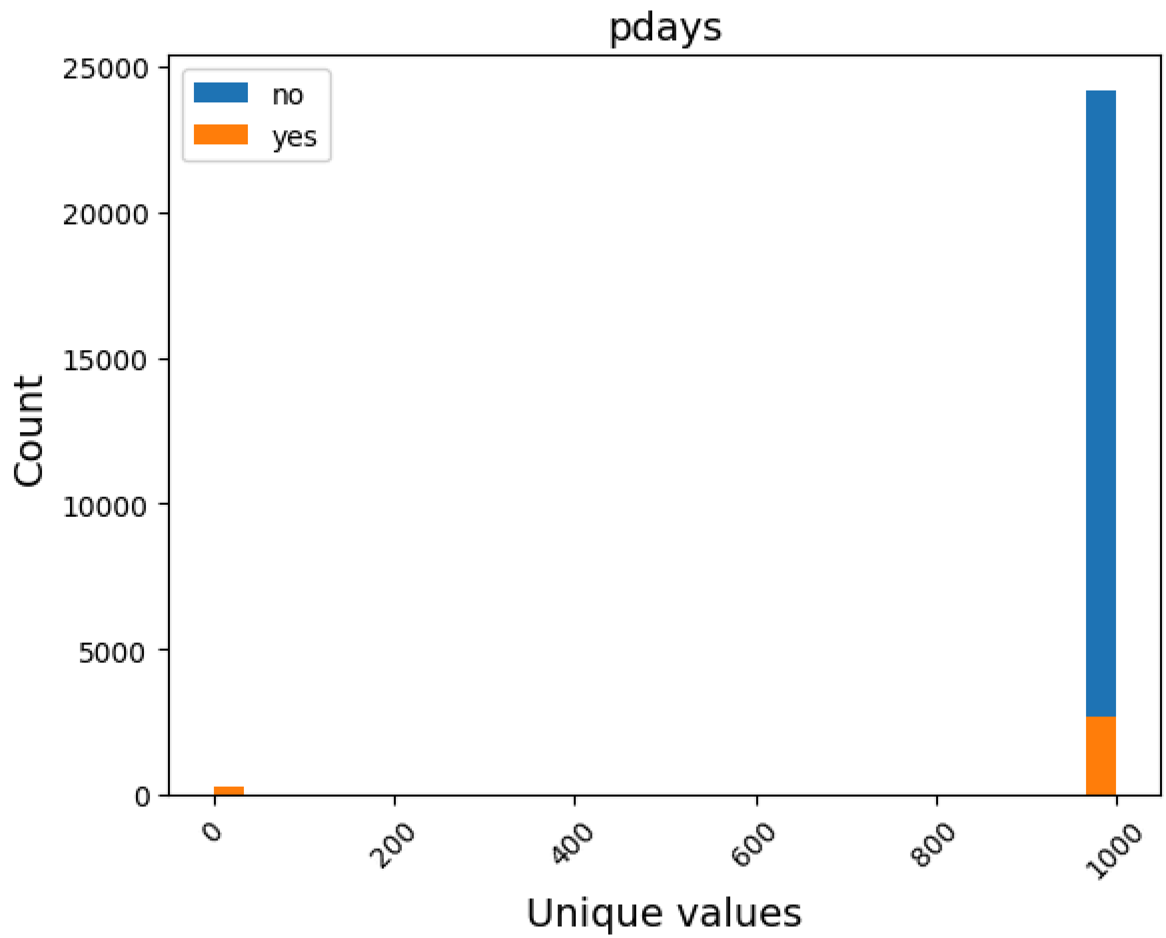
In [ ]:
```
understanding_data = UnderstandingData(check_point)

understanding_data.plot_histogram_continuous("age", 30)
understanding_data.plot_histogram_continuous("duration", 30)
understanding_data.plot_histogram_continuous("campaign", 30)
understanding_data.plot_histogram_continuous("pdays", 30)
understanding_data.plot_histogram_continuous("nr.employed",50)
```

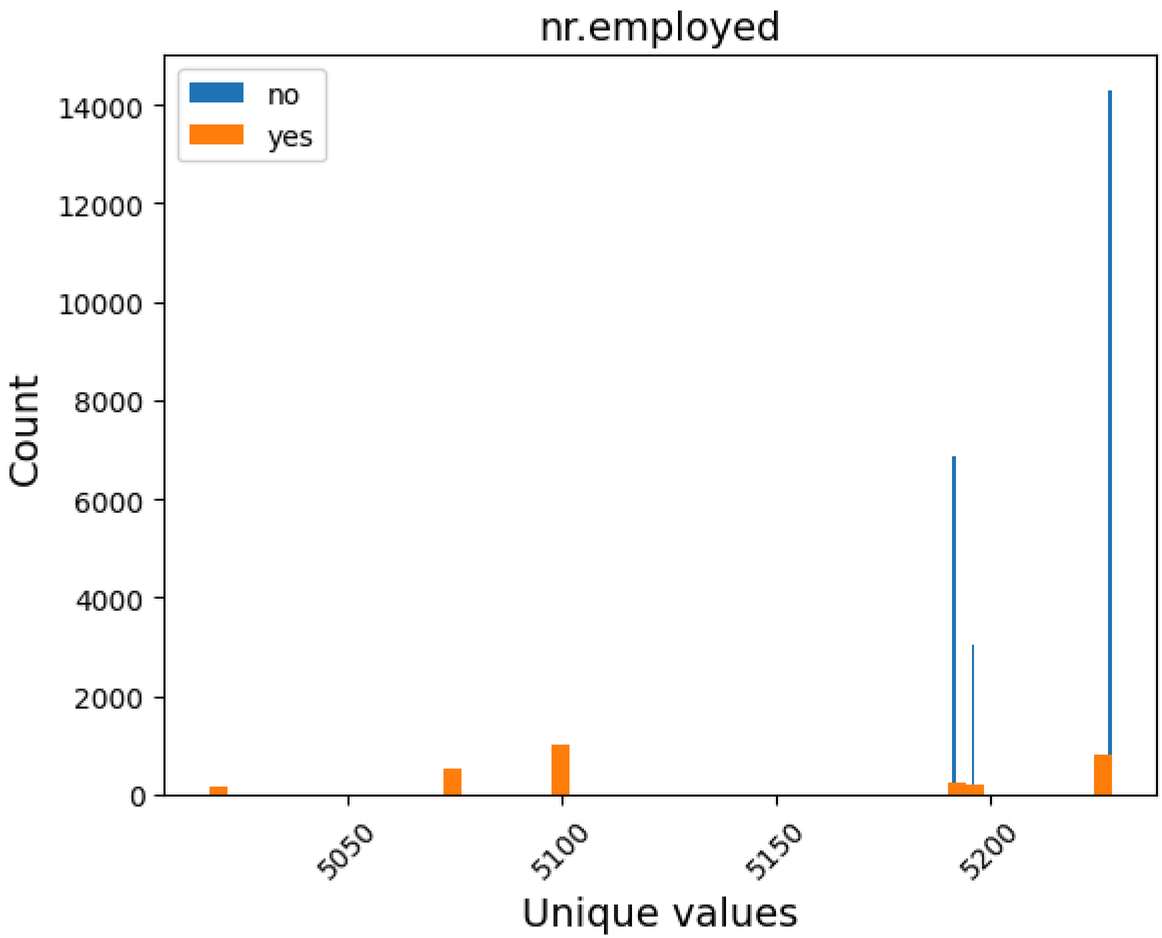


age
no
yes
Count
0
500
1000
1500
2000
20
30
40
50
60
70
80
90
Unique values


duration
no
yes
Count
12000
10000
8000
6000
4000
2000
0
0
1000
2000
3000
4000
5000
Unique values


campaign
no
yes
Count
Unique values
16000
14000
12000
10000
8000
6000
4000
2000
0
0
10
20
30
40

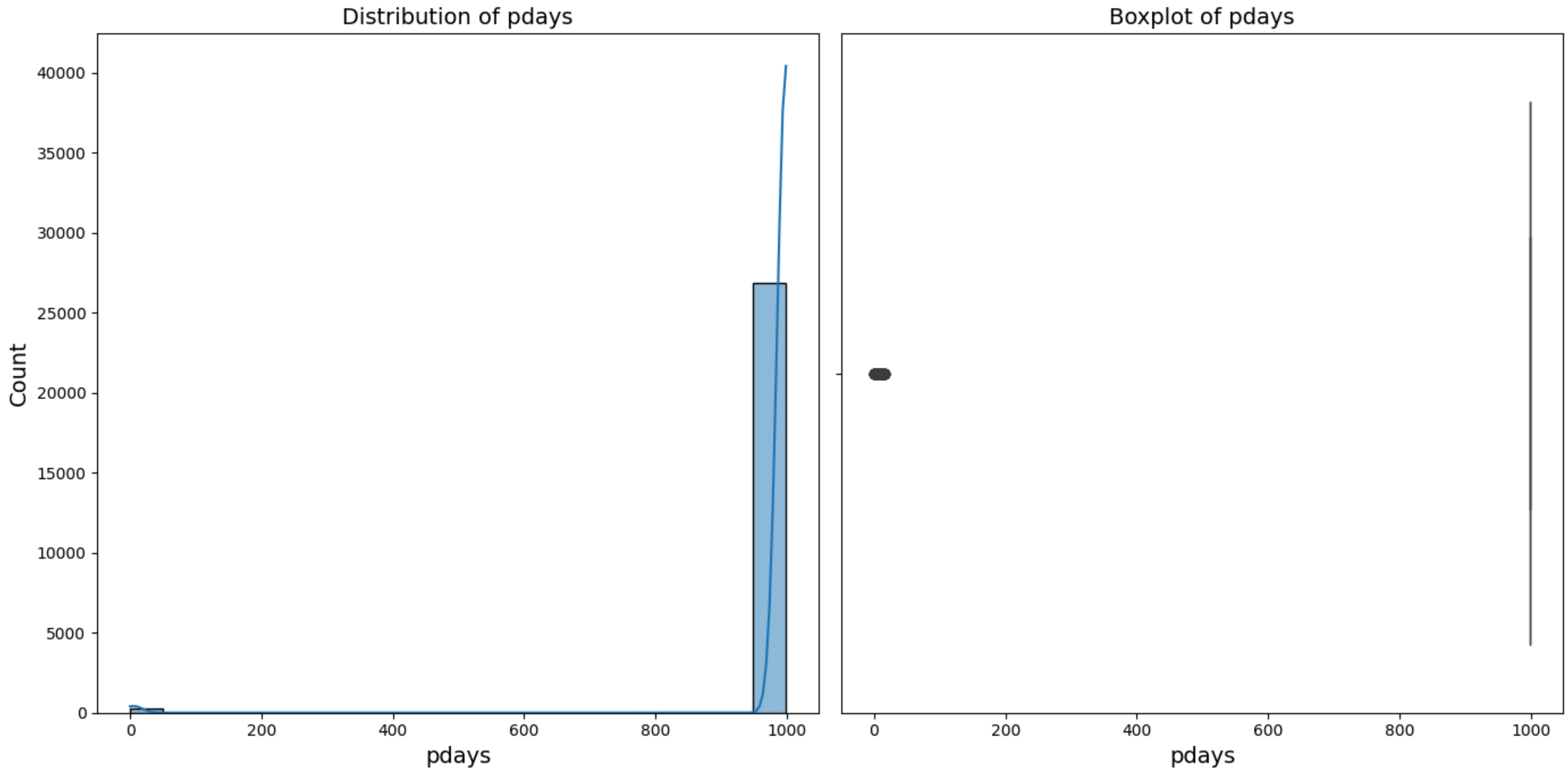

pdays
no
yes
Count
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
Unique values



In [ ]:

```
# Create a subplot grid
fig, axes = plt.subplots(1, 2, figsize=(14, 7), sharey=False)

# Plot distribution (histogram) of 'pdays'
sns.histplot(data=check_point, x='pdays', bins=20, kde=True, ax=axes[0])
axes[0].set_title('Distribution of pdays', fontsize=14)
axes[0].set_ylabel("Count", fontsize=14)
axes[0].set_xlabel("pdays", fontsize=14)

# Plot boxplot of 'pdays'
sns.boxplot(data=check_point, x='pdays', ax=axes[1])
axes[1].set_title('Boxplot of pdays', fontsize=14)
```

```
axes[1].set_xlabel("pdays", fontsize=14)

# Adjust layout
plt.tight_layout()
plt.show()
```



Encode and Heatmap

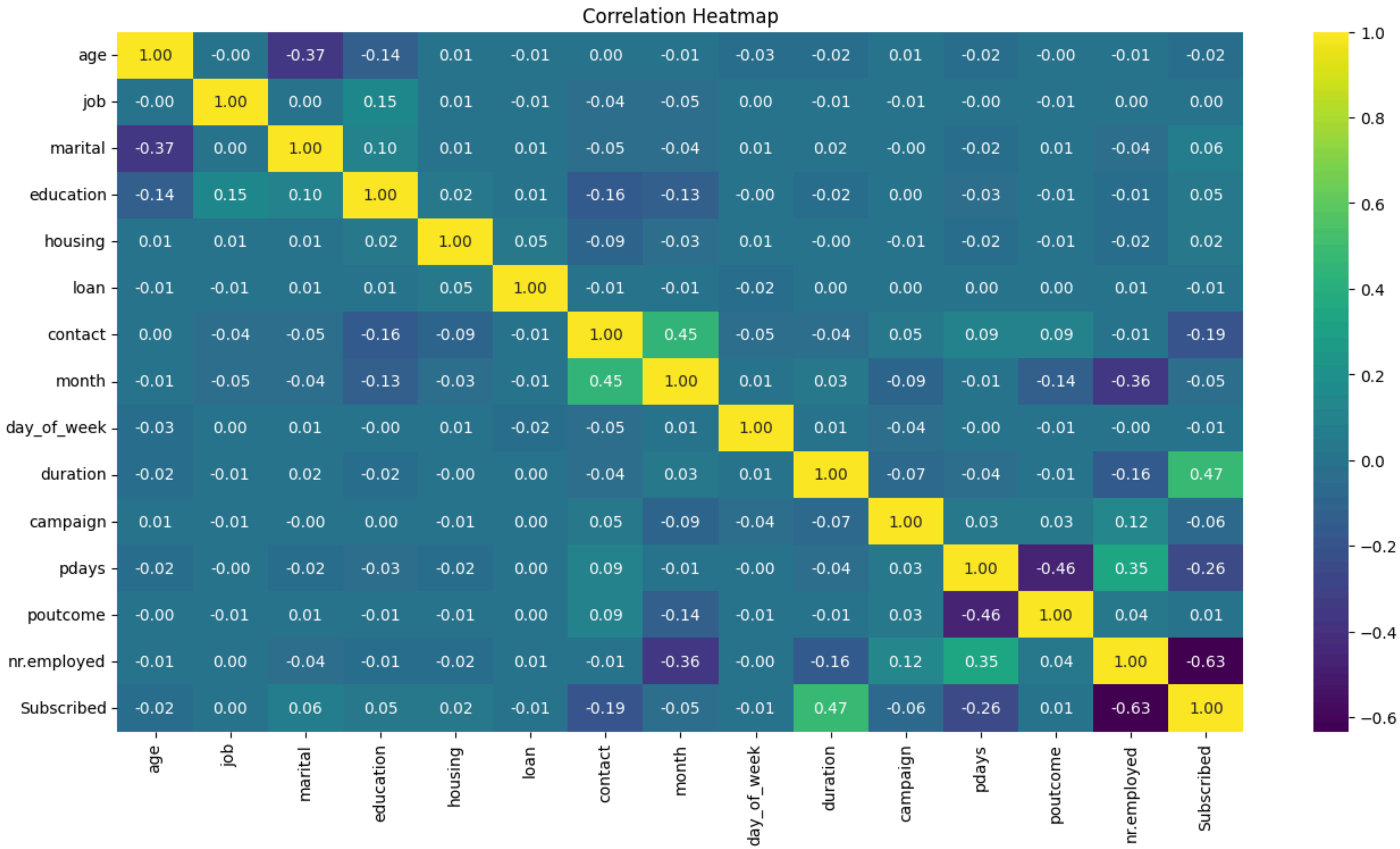
In [ ]:
```
from sklearn.preprocessing import LabelEncoder


encode_point = check_point.copy()

# Initialize LabelEncoder
label_encoder = LabelEncoder()
```

```
# Perform label encoding for each categorical column
for col in data_categorical:
    encode_point[col] = label_encoder.fit_transform(encode_point[col])

# Display the updated dataset with label-encoded categorical variables
print("After Label Encoding:")
print(encode_point.head())
```

```
After Label Encoding:
   age  job  marital  education  housing  loan  contact  month  day_of_week  \
0   41    1        0          0        1     0        1      6            1
1   49    2        1          6        1     0        1      6            1
2   49    9        1          2        0     0        1      6            1
3   41    9        1          5        1     0        1      6            1
4   45    1        1          2        1     0        1      6            1

   duration  campaign  pdays  poutcome  nr.employed  Subscribed
0      1575         1    999         1       5191.0           1
1      1042         1    999         1       5191.0           1
2      1467         1    999         1       5191.0           1
3       579         1    999         1       5191.0           1
4       461         1    999         1       5191.0           1
```

In [ ]:

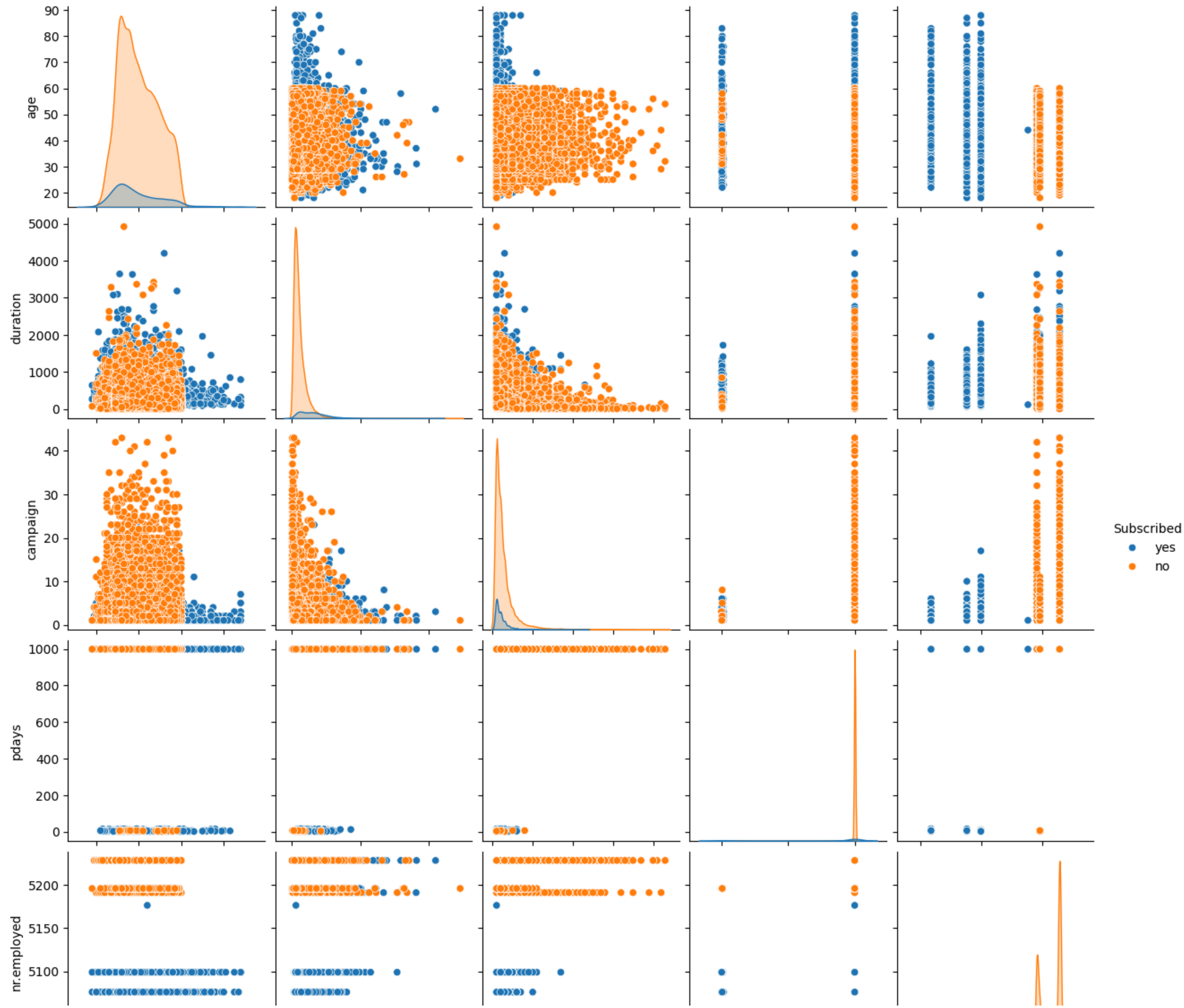
```
corr_matrix = encode_point.corr()

# Set the size of the plot
plt.figure(figsize=(16, 8))

# Create the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='viridis', fmt='.2f')

# Add title
plt.title('Correlation Heatmap')

# Display the plot
plt.show()
```



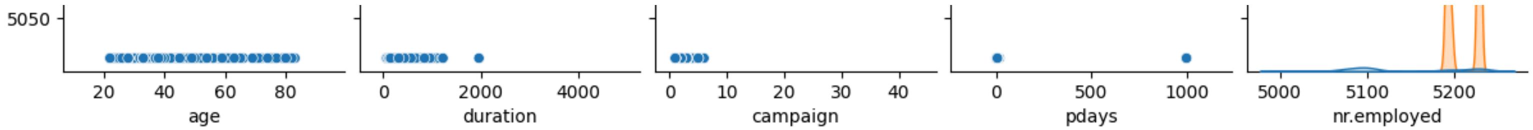
relationship of numertic

In [ ]:
```
sns.pairplot(check_point, hue = 'Subscribed', dropna= True)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x13b929b10>



age
duration
campaign
pdays
nr.employed
Subscribed
yes
no



Data Cleaning

Data Spiting

In [ ]:
```
#copy from check point
x = check_point.iloc[:,:-1]
y = check_point.iloc[:,-1]
```

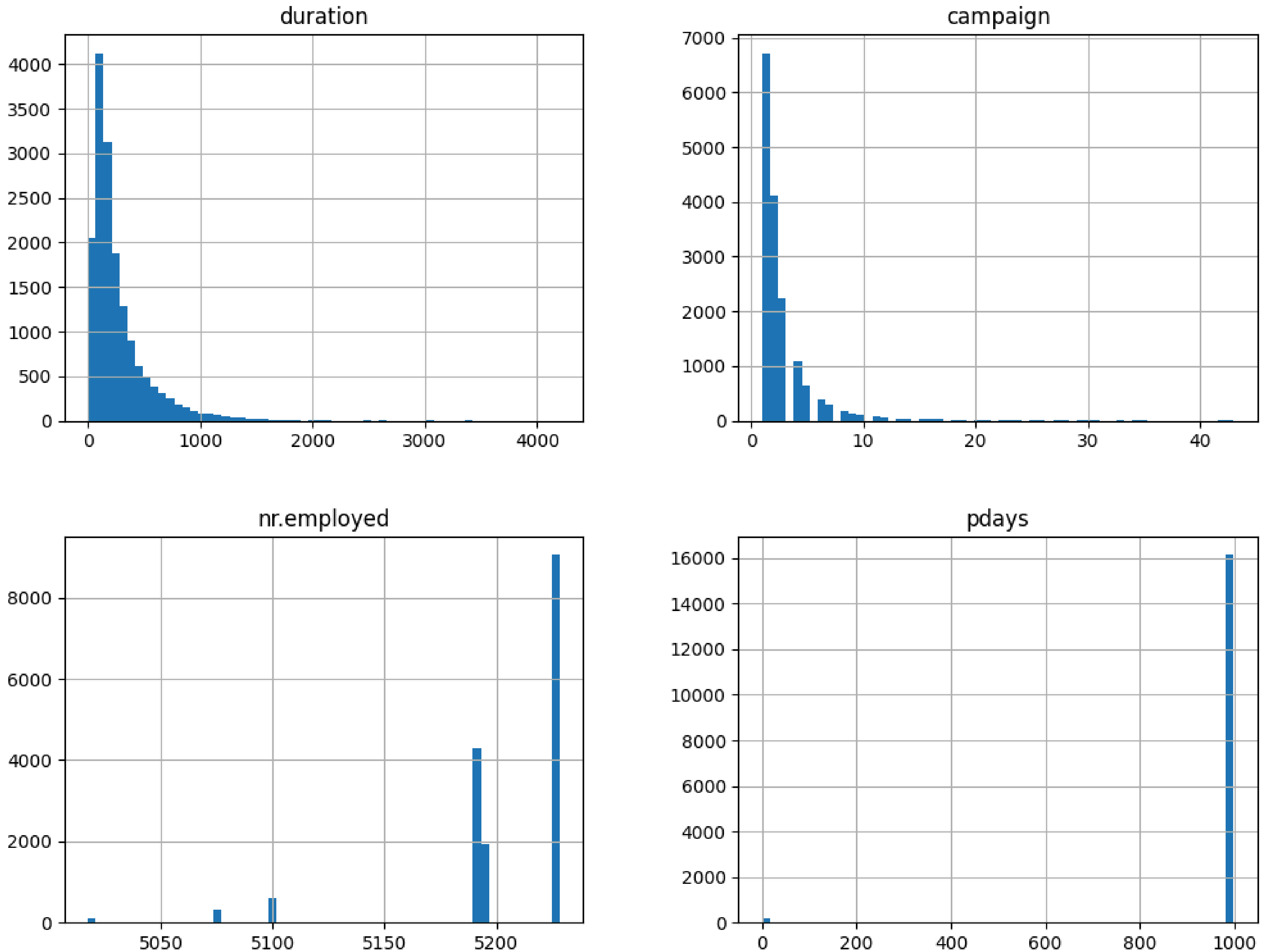
In [ ]:
```
#Regrouping all data to int data,nominal data, ordinary
x = x.loc[:,["duration","campaign","nr.employed","pdays",'education','month','marital','housing','contact']
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 27178 entries, 0 to 29270
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   duration     27178 non-null  int64  
 1   campaign     27178 non-null  int64  
 2   nr.employed  27178 non-null  float64
 3   pdays        27178 non-null  int64  
 4   education    27178 non-null  object 
 5   month        27178 non-null  object 
 6   marital      27178 non-null  object 
 7   housing      27178 non-null  object 
 8   contact      27178 non-null  object 
dtypes: float64(1), int64(3), object(5)
memory usage: 2.1+ MB
```

In [ ]:
```
#Data Splitting in a test size of 30 % and train size 70%
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test  = train_test_split(x,y,test_size= 0.4,random_state=45)
```

Data Handling

In [ ]:
```
#Check Skew
x_train.hist(bins=60,figsize=(12,9))
plt.show()
```




```
In [ ]: # age, duration, campaign, nr.employed need to handle outliers
        int_column =["duration","campaign","nr.employed","pdays"]
```

```python
int_data = x_train[int_column]
int_data
```

Out[ ]:

	duration	campaign	nr.employed	pdays
1593	583	1	5099.1	5
16220	158	1	5228.1	999
20389	254	21	5228.1	999
12969	134	1	5228.1	999
1155	234	1	5195.8	999
...	...	...	...	...
17737	747	1	5228.1	999
14064	207	1	5228.1	999
6568	430	2	5191.0	999
7160	183	7	5191.0	999
25413	111	1	5228.1	999

16306 rows × 4 columns

In [ ]:

```python
#Capping outliner using IQR method
def iqr_capping(column_name):
    int_data[column_name] = int_data[column_name].astype(float)
    Q1 = int_data[column_name].quantile(0.25)
    Q3 = int_data[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Cap the values outside the lower and upper bounds
    int_data.loc[:,column_name] = int_data[column_name].apply(lambda x: lower_bound if x < lower_bound els


    return int_data
```

In [ ]:
```
for column in int_data.columns:
    iqr_capping(column)
```

```
/var/folders/_c/lff8m62n4dgfy2wttc5rsvym0000gn/T/ipykernel_33923/3194890469.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  int_data[column_name] = int_data[column_name].astype(float)
/var/folders/_c/lff8m62n4dgfy2wttc5rsvym0000gn/T/ipykernel_33923/3194890469.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  int_data[column_name] = int_data[column_name].astype(float)
/var/folders/_c/lff8m62n4dgfy2wttc5rsvym0000gn/T/ipykernel_33923/3194890469.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  int_data[column_name] = int_data[column_name].astype(float)
/var/folders/_c/lff8m62n4dgfy2wttc5rsvym0000gn/T/ipykernel_33923/3194890469.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  int_data[column_name] = int_data[column_name].astype(float)
```

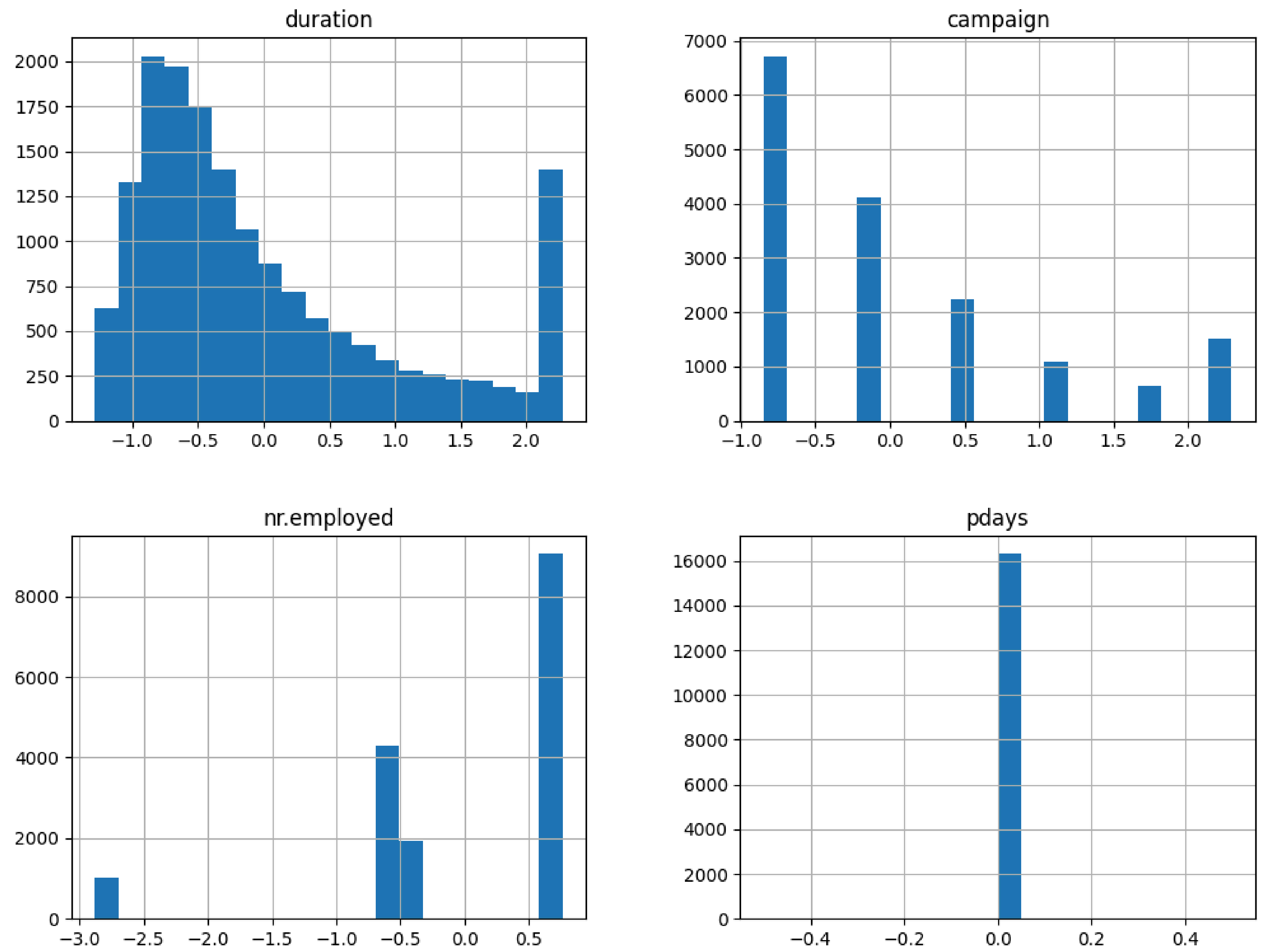
In [ ]:
```
#Standarization
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
std_scaler.fit(int_data)
int_data_scaled = std_scaler.transform(int_data)
df_int_std = pd.DataFrame(int_data_scaled, columns= int_data.columns)
df_int_std.head()
```

Out[ ]:

	duration	campaign	nr.employed	pdays
0	1.776431	-0.846475	-2.881411	0.0
1	-0.456802	-0.846475	0.762704	0.0
2	0.047646	2.290485	0.762704	0.0
3	-0.582914	-0.846475	0.762704	0.0
4	-0.057448	-0.846475	-0.506352	0.0

In [ ]:

```
df_int_std.hist(bins=20,figsize=(12,9))
plt.show()
```



Converting Category index to int. Select Columns which need to encode, select columns with object data type

In [ ]:
```python
#
pd.Series({c: x_test[c].unique() for c in x_test.select_dtypes(include='object').columns})
```

Out[ ]:
```
education    [basic.9y, high.school, university.degree, bas...
month            [jul, apr, jun, may, aug, nov, mar, sep, oct]
marital                             [divorced, single, married]
housing                                               [no, yes]
contact                                   [cellular, telephone]
dtype: object
```

In [ ]:
```python
#Group with Ordinal and Nominal Data
ordinal_column = ['education']
nominal_column = ['month','marital','contact']
cat_ordinal = x_train[ordinal_column]
cat_nominal = x_train[nominal_column]
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16306 entries, 1593 to 25413
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   duration     16306 non-null  int64
 1   campaign     16306 non-null  int64
 2   nr.employed  16306 non-null  float64
 3   pdays        16306 non-null  int64
 4   education    16306 non-null  object
 5   month        16306 non-null  object
 6   marital      16306 non-null  object
 7   housing      16306 non-null  object
 8   contact      16306 non-null  object
dtypes: float64(1), int64(3), object(5)
memory usage: 1.2+ MB
```

In [ ]:
```python
x_train.head()
```

Out[ ]:

	duration	campaign	nr.employed	pdays	education	month	marital	housing	contact
1593	583	1	5099.1	5	basic.9y	apr	married	no	cellular
16220	158	1	5228.1	999	basic.9y	jul	married	no	cellular
20389	254	21	5228.1	999	university.degree	jul	single	no	telephone
12969	134	1	5228.1	999	basic.4y	jun	married	yes	telephone
1155	234	1	5195.8	999	high.school	nov	single	yes	telephone

In [ ]:

```
x_train["education"].unique()
```

Out[ ]:

```
array(['basic.9y', 'university.degree', 'basic.4y', 'high.school',
       'basic.6y', 'professional.course', 'illiterate'], dtype=object)
```

In [ ]:

```
#Converting Ordinary data to array index using OrdinaryEncoder
from sklearn.preprocessing import OrdinalEncoder
ordinal_Encoder = OrdinalEncoder(categories=[['illiterate', 'basic.4y','basic.6y', 'basic.9y','high.school
       'professional.course']])
cat_ordinal_endcoded = ordinal_Encoder.fit_transform(cat_ordinal)
df_ordinal_endcoded = pd.DataFrame(cat_ordinal_endcoded,columns= cat_ordinal.columns)
df_ordinal_endcoded.astype(float)
df_ordinal_endcoded.head()
```

Out[ ]:

	education
0	3.0
1	3.0
2	5.0
3	1.0
4	4.0

In [ ]:

```
#Converting Nominal data to array index using one hot encoder
from sklearn.preprocessing import OneHotEncoder
oneHotEncoder = OneHotEncoder()
oneHotEncoder.fit(cat_nominal)
```

```python
cat_nominal_1hot = oneHotEncoder.transform(cat_nominal)

df_nominal_1hot = pd.DataFrame(cat_nominal_1hot.toarray(), columns=oneHotEncoder.get_feature_names_out())
df_nominal_1hot
```

Out[ ]:

	month_apr	month_aug	month_dec	month_jul	month_jun	month_mar	month_may	month_nov	month_oct	m
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	...	
16301	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
16302	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
16303	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
16304	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
16305	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

16306 rows × 15 columns

In [ ]:

```python
#Converting Nominal data to array index using one hot encoder
YoneHotEncoder = OneHotEncoder()
cat_nominal_before_encode = pd.DataFrame(y_train)
cat_nominal_1hot = YoneHotEncoder.fit_transform(cat_nominal_before_encode)
df_y_train_result = pd.DataFrame(cat_nominal_1hot.toarray(),columns=YoneHotEncoder.categories_)
df_y_train_result.astype(float)
```

Out[ ]:

	no	yes
0	0.0	1.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	0.0	1.0
...	...	...
16301	1.0	0.0
16302	1.0	0.0
16303	1.0	0.0
16304	1.0	0.0
16305	1.0	0.0

16306 rows × 2 columns

In [ ]:

```
# Concat all the standardize data and encoded data
x_train = pd.concat([df_int_std,df_ordinal_endcoded,df_nominal_1hot],axis = 1)
```

In [ ]:

```
x_train.head()
```

Out[ ]:

	duration	campaign	nr.employed	pdays	education	month_apr	month_aug	month_dec	month_jul	month_jun	m
0	1.776431	-0.846475	-2.881411	0.0	3.0	1.0	0.0	0.0	0.0	0.0	
1	-0.456802	-0.846475	0.762704	0.0	3.0	0.0	0.0	0.0	1.0	0.0	
2	0.047646	2.290485	0.762704	0.0	5.0	0.0	0.0	0.0	1.0	0.0	
3	-0.582914	-0.846475	0.762704	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
4	-0.057448	-0.846475	-0.506352	0.0	4.0	0.0	0.0	0.0	0.0	0.0	

Model Selection

In [ ]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(x_train,y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
from sklearn import ensemble
DecidtionTree_model = ensemble.RandomForestClassifier(criterion='gini') # for classification, here you can
# model = tree.DecisionTreeRegressor() for regression
```

In [ ]:
```
test_data_ordinal = ordinal_Encoder.transform(x_test[ordinal_column])
test_data_nominal = oneHotEncoder.transform(x_test[nominal_column])
test_data_int = std_scaler.transform(x_test[int_column])
test_data_result = YoneHotEncoder.transform(pd.DataFrame(y_test))
```

In [ ]:
```
df_test_nominal_1hot = pd.DataFrame(test_data_nominal.toarray(), columns=oneHotEncoder.get_feature_names_ou
df_test_ordinal_endcoded = pd.DataFrame(test_data_ordinal,columns= cat_ordinal.columns)
df_test_data_int = pd.DataFrame(test_data_int, columns= std_scaler.get_feature_names_out())
```

In [ ]:
```
#Concat all Transformed Data
x_test  = pd.concat([df_test_data_int,df_test_ordinal_endcoded,df_test_nominal_1hot],axis = 1)
x_test.head()
```

Out[ ]:

	duration	campaign	nr.employed	pdays	education	month_apr	month_aug	month_dec	month_jul	month_jun	mo
0	0.252577	2.917876	0.762704	0.0	3.0	0.0	0.0	0.0	1.0	0.0	
1	0.226304	-0.846475	0.762704	0.0	4.0	0.0	0.0	0.0	1.0	0.0	
2	3.809986	-0.846475	-4.305660	0.0	5.0	1.0	0.0	0.0	0.0	0.0	
3	-0.971760	2.290485	0.762704	0.0	2.0	0.0	0.0	0.0	0.0	1.0	
4	-0.761573	-0.846475	-4.305660	0.0	6.0	1.0	0.0	0.0	0.0	0.0	

In [ ]:

```
#Compare to train Data
x_train.head()
```

Out[ ]:

	duration	campaign	nr.employed	pdays	education	month_apr	month_aug	month_dec	month_jul	month_jun	m
0	1.776431	-0.846475	-2.881411	0.0	3.0	1.0	0.0	0.0	0.0	0.0	
1	-0.456802	-0.846475	0.762704	0.0	3.0	0.0	0.0	0.0	1.0	0.0	
2	0.047646	2.290485	0.762704	0.0	5.0	0.0	0.0	0.0	1.0	0.0	
3	-0.582914	-0.846475	0.762704	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
4	-0.057448	-0.846475	-0.506352	0.0	4.0	0.0	0.0	0.0	0.0	0.0	

In [ ]:

```
DecidtionTree_model.fit(x_train,y_train)
DecidtionTree_model.score(x_train,y_train)
prediction_tree = DecidtionTree_model.predict(x_test)
prediction_tree
```

Out[ ]:

```
array(['no', 'no', 'yes', ..., 'no', 'no', 'no'], dtype=object)
```

In [ ]:

```
prediction_log = log_reg.predict(x_test)
prediction_log
```

Out[ ]:

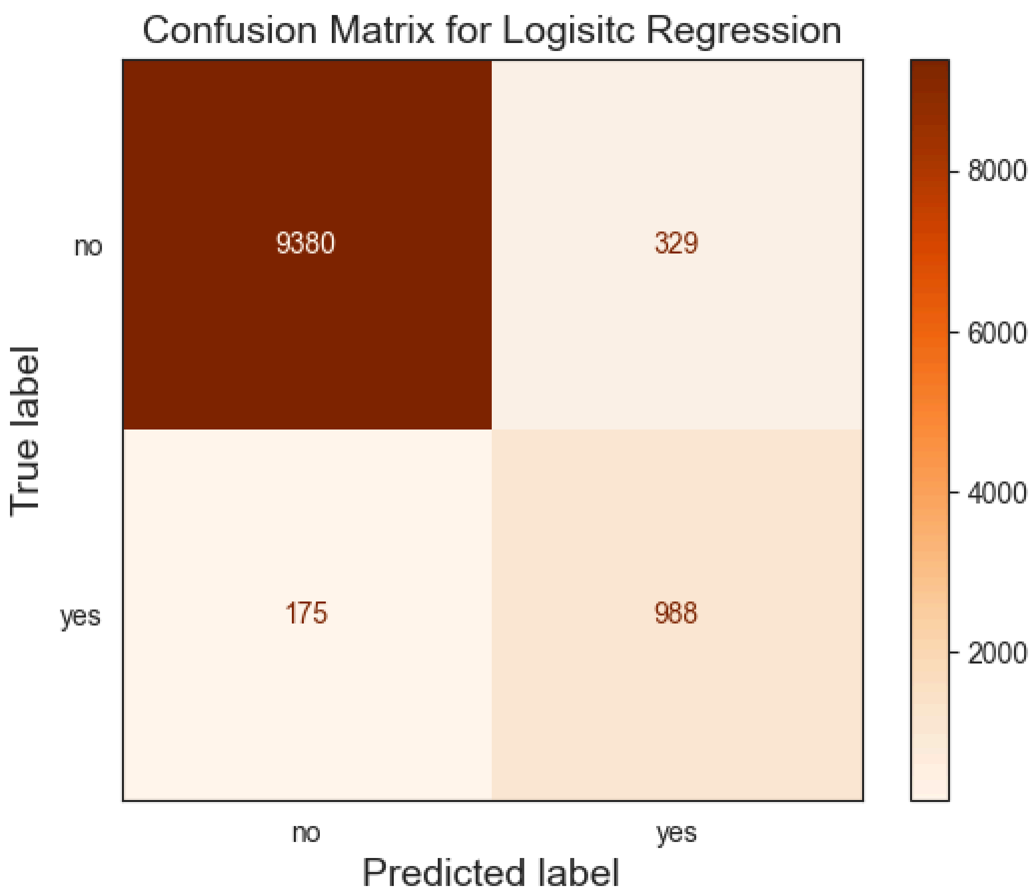
```
array(['no', 'no', 'yes', ..., 'no', 'no', 'no'], dtype=object)
```

Model Evaluation

Logistic Regression

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay
sns.set_style('white')
ConfusionMatrixDisplay.from_estimator(log_reg,x_test,y_test,cmap='Oranges')
plt.title('Confusion Matrix for Logisitc Regression',fontsize=14)
plt.xlabel('Predicted label',fontsize=14)
plt.ylabel('True label',fontsize=14)
plt.show()
```




```
In [ ]: from sklearn.metrics import classification_report
        print(classification_report(y_test,prediction_log))
```

```
              precision    recall  f1-score   support

          no       0.98      0.97      0.97      9709
         yes       0.75      0.85      0.80      1163

    accuracy                           0.95     10872
   macro avg       0.87      0.91      0.89     10872
weighted avg       0.96      0.95      0.95     10872
```

In [ ]:
```
#Accuracy score of the  Train Model
accuracy = log_reg.score(x_train, y_train)
print(f"The accuracy Score is {accuracy}")
```

```
The accuracy Score is 0.9553538574757758
```

In [ ]:
```
#Accuracy score of the Model
accuracy = log_reg.score(x_test, y_test)
print(f"The accuracy Score is {accuracy}")
```

```
The accuracy Score is 0.9536423841059603
```

In [ ]:
```
#Calculate the Precision Score
from sklearn.metrics import precision_score
precision = precision_score(y_test, prediction_log,pos_label='yes')
print(f"The Precision Score is {precision}")
```

```
The Precision Score is 0.7501898253606681
```

In [ ]:
```
#Calculate the Recall Score
from sklearn.metrics import recall_score
recall = recall_score(y_test, prediction_log,pos_label='yes')
print(f"The Recall Score is {recall}")
```

```
The Recall Score is 0.8495270851246776
```

In [ ]:
```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, prediction_log, pos_label='yes')
print("F1 score:", f1)
```

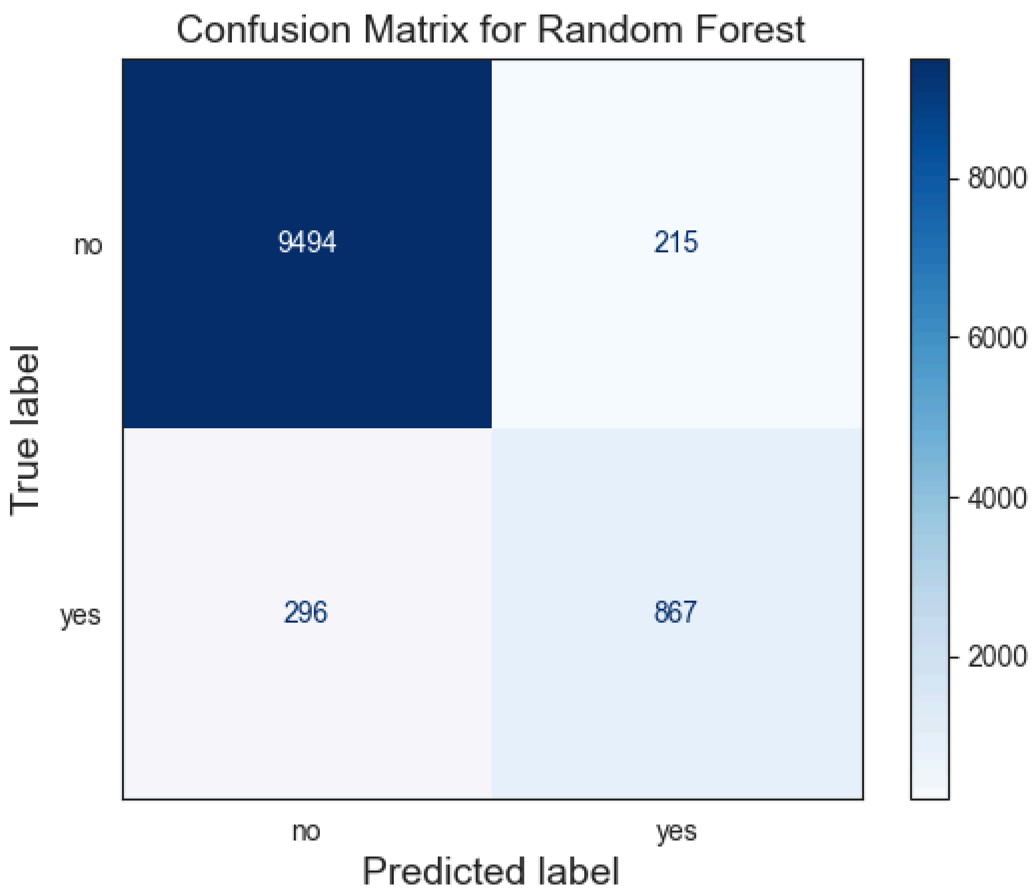
```
F1 score: 0.7967741935483871
```

In [ ]:
```
from sklearn import metrics
y_pred_proba = log_reg.predict_proba(x_test)[:,1]
fpr_log, tpr_log,_= metrics.roc_curve(y_test,  y_pred_proba,pos_label='yes')
roc_auc = metrics.auc(fpr_log, tpr_log)

#create ROC curve
sns.set_style("darkgrid")
plt.figure(figsize=(8, 6))
plt.plot(fpr_log, tpr_log, color='darkorange', lw=2, label='ROC curve (AUC = %0.5f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) for Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

Random Forest Classifier

In [ ]:
```
# Create Confusion Matrix Display
from sklearn.metrics import ConfusionMatrixDisplay
sns.set_style('white')
ConfusionMatrixDisplay.from_estimator(DecidtionTree_model,x_test,y_test,cmap='Blues')
plt.title('Confusion Matrix for Random Forest',fontsize=14)
plt.xlabel('Predicted label',fontsize=14)
plt.ylabel('True label',fontsize=14)
plt.show()
```



In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_tree))
```

```
              precision    recall  f1-score   support

          no       0.97      0.98      0.97      9709
         yes       0.80      0.75      0.77      1163

    accuracy                           0.95     10872
   macro avg       0.89      0.86      0.87     10872
weighted avg       0.95      0.95      0.95     10872
```

In [ ]:
```
#Accuracy score of the Model
accuracy = DecidtionTree_model.score(x_train, y_train)
print(f"The accuracy Score is {accuracy}")
```

```
The accuracy Score is 0.9815405372255611
```

In [ ]:
```
#Accuracy score of the Model
accuracy = DecidtionTree_model.score(x_test, y_test)
print(f"The accuracy Score is {accuracy}")
```

```
The accuracy Score is 0.9529985283296541
```

In [ ]:
```
#Calculate the Precision Score
from sklearn.metrics import precision_score
precision = precision_score(y_test, prediction_tree,pos_label='yes')
print(f"The Precision Score is {precision}")
```

```
The Precision Score is 0.8012939001848429
```

In [ ]:
```
#Calculate the Recall Score
from sklearn.metrics import recall_score
recall = recall_score(y_test, prediction_tree,pos_label='yes')
print(f"The Recall Score is {recall}")
```

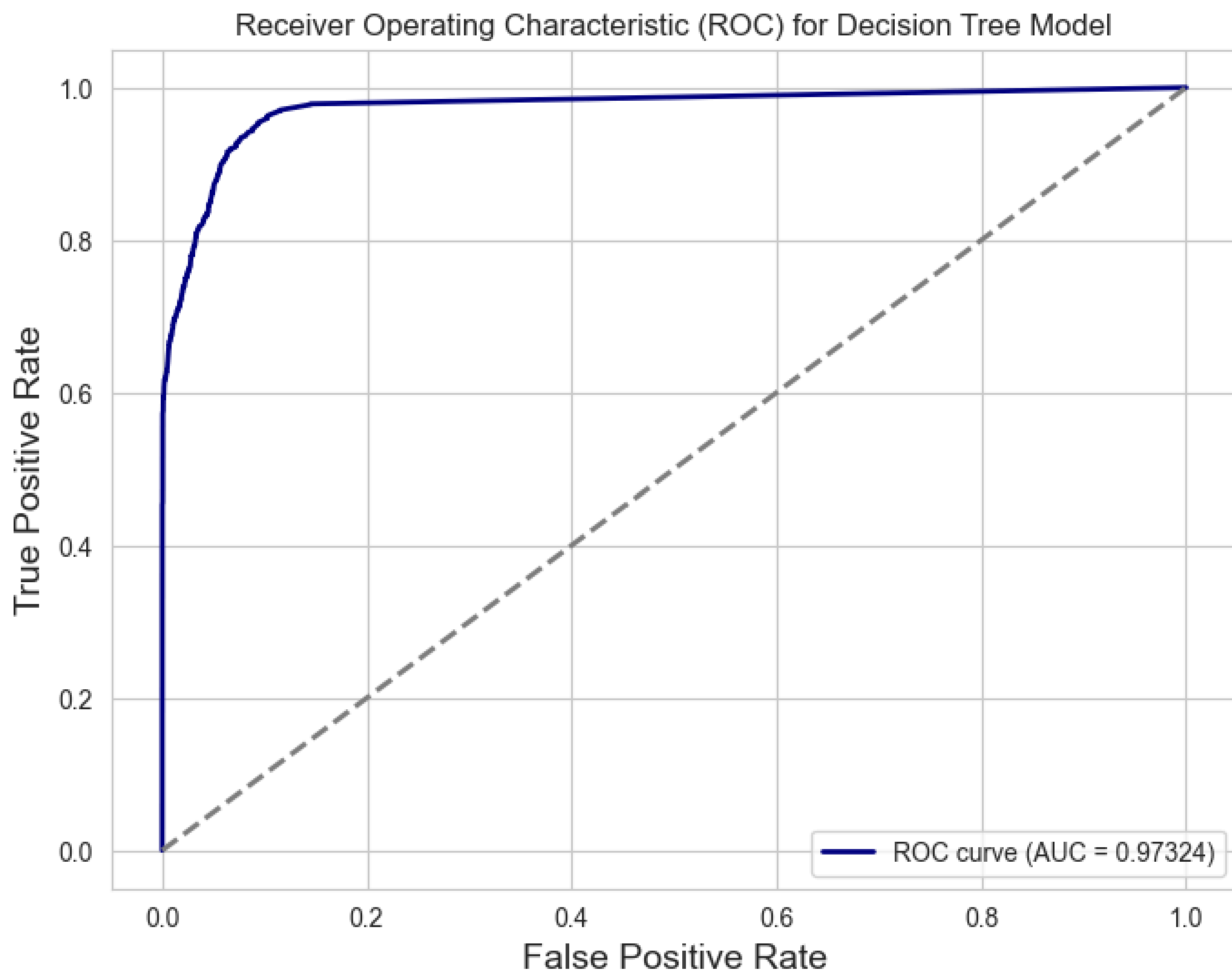
```
The Recall Score is 0.7454858125537404
```

In [ ]:
```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, prediction_tree, pos_label='yes')
print("F1 score:", f1)
```

```
F1 score: 0.7723830734966592
```

In [ ]:

```
y_pred_proba_tree = DecidtionTree_model.predict_proba(x_test)[:,1]
fpr_tree, tpr_tree,_= metrics.roc_curve(y_test,  y_pred_proba_tree,pos_label='yes')
roc_auc_tree = metrics.auc(fpr_tree, tpr_tree)

#create ROC curve
sns.set_style("whitegrid")
plt.figure(figsize=(8, 6))
plt.plot(fpr_tree, tpr_tree, color='navy', lw=2, label='ROC curve (AUC = %0.5f)' % roc_auc_tree)
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlabel('False Positive Rate',fontsize = 14 )
plt.ylabel('True Positive Rate',fontsize = 14)
plt.title('Receiver Operating Characteristic (ROC) for Decision Tree Model')
plt.legend(loc="lower right")
plt.show()
```



Base on the comparison, the Logistic model has a bigger Area under the curve which conclude that Logistic and perform a more accurate result