Exploratory Data Analysis

February 19, 2025

1 importing modules

[904]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from overview import load_bank_variables
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler


pd.set_option('display.max_colwidth', None)  # Show full column content
pd.set_option('display.expand_frame_repr', False)  # Disable line wrapping
pd.set_option('display.max_rows', None)  # Show all rows
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.width', 1000)        # Adjust column width
```

1.1 Loading Data

[905]:
```
bank_df = pd.read_csv("bank.csv")
df = bank_df.copy()
```

[906]:
```
load_bank_variables()
```

[906]:
```
   Variable Name
Description
0           age
Age
1           job  Type of job (e.g., 'admin.','blue-
collar','entrepreneur','housemaid','management','retired','self-
employed','services','student','technician','unemployed','unknown')
2       marital
Marital status (e.g., 'divorced','married','single','unknown'; 'divorced' means
divorced or widowed)
3     education                                Education level (e.g., 'basic.4y
','basic.6y','basic.9y','high.school','illiterate','professional.course','univer
sity.degree','unknown')
4       default
```

```
Has credit in default?
5          balance
Average yearly balance (euros)
6          housing
Has housing loan?
7             loan
Has personal loan?
8          contact
Contact communication type (e.g., 'cellular','telephone')
9      day_of_week
Last contact day of the week
10           month
Last contact month of year (e.g., 'jan', 'feb', 'mar', …, 'nov', 'dec')
11        duration
Last contact duration in seconds (only for benchmarks, discard for real
prediction)
12        campaign
Number of contacts during this campaign (includes last contact)
13           pdays
Number of days since last contact from previous campaign (-1 means not
contacted)
14        previous
Number of contacts before this campaign
15        poutcome
Outcome of previous campaign (e.g., 'failure','nonexistent','success')
16               y
Has the client subscribed to a term deposit?
```

1.2 Data Exploration

[907]:
```
df.head(5)
```

[907]:
```
    age          job   marital  education default  balance housing loan
contact  day month  duration  campaign  pdays  previous poutcome subscribed
0  32.0   technician    single   tertiary      no      392     yes   no
cellular    1   apr       957          2    131         2  failure          no
1  39.0   technician  divorced  secondary      no      688     yes  yes
cellular    1   apr       233          2    133         1  failure          no
2  59.0      retired   married  secondary      no     1035     yes  yes
cellular    1   apr       126          2    239         1  failure          no
3  47.0  blue-collar   married  secondary      no      398     yes  yes
cellular    1   apr       274          1    238         2  failure          no
4  54.0      retired   married  secondary      no     1004     yes   no
cellular    1   apr       479          1    307         1  failure          no
```

[908]:
```
df.tail(5)
```

[908]:
```
        age           job  marital  education default  balance housing loan
 contact  day month  duration  campaign  pdays  previous poutcome subscribed
 1995  20.0       student   single        NaN      no     2785      no   no
 cellular   16   sep       327          2     -1         0      NaN         yes
 1996  28.0        admin.   single  secondary      no      127      no   no
 cellular   16   sep      1334          2     -1         0      NaN         yes
 1997  81.0       retired  married    primary      no     1154      no   no
 telephone   17   sep       231          1     -1         0      NaN         yes
 1998  46.0      services  married    primary      no     4343     yes   no
 NaN   20   sep       185          1     -1         0      NaN         yes
 1999  40.0  entrepreneur  married  secondary      no     6403      no   no
 cellular   22   sep       208          2     -1         0      NaN         yes
```

[909]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1988 non-null   float64
 1   job         1990 non-null   object
 2   marital     2000 non-null   object
 3   education   1896 non-null   object
 4   default     2000 non-null   object
 5   balance     2000 non-null   int64
 6   housing     2000 non-null   object
 7   loan        2000 non-null   object
 8   contact     1809 non-null   object
 9   day         2000 non-null   int64
 10  month       2000 non-null   object
 11  duration    2000 non-null   int64
 12  campaign    2000 non-null   int64
 13  pdays       2000 non-null   int64
 14  previous    2000 non-null   int64
 15  poutcome    1546 non-null   object
 16  subscribed  2000 non-null   object
dtypes: float64(1), int64(6), object(10)
memory usage: 265.8+ KB
```

the dataset countains 2000 rows and 17 columns both numerical and categorical

```
- numerical : age , balance , duration , compaign , pdays , previous
- categorical: job , marital , education , default , housing , loan , contact , month , poutco
```

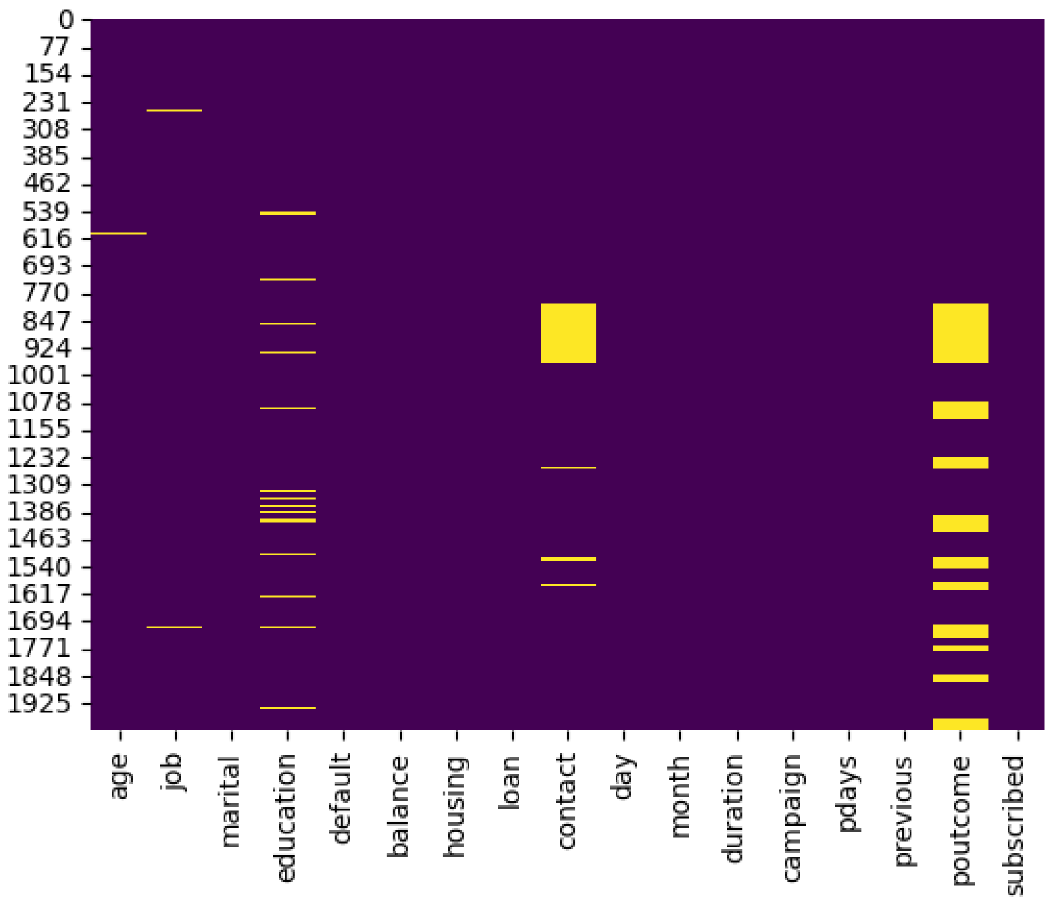
there is some missing values in :

```
- age (12)
- job (10)
- education (104)
```

```
- contact (191)
- poutcome (454)
```

[910]:
```
# Visualizing missing values
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
```

[910]: <Axes: >



1.3 seperate columns by type to plot each

[911]:
```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
categorical_columns = df.select_dtypes(include=['object']).columns
```

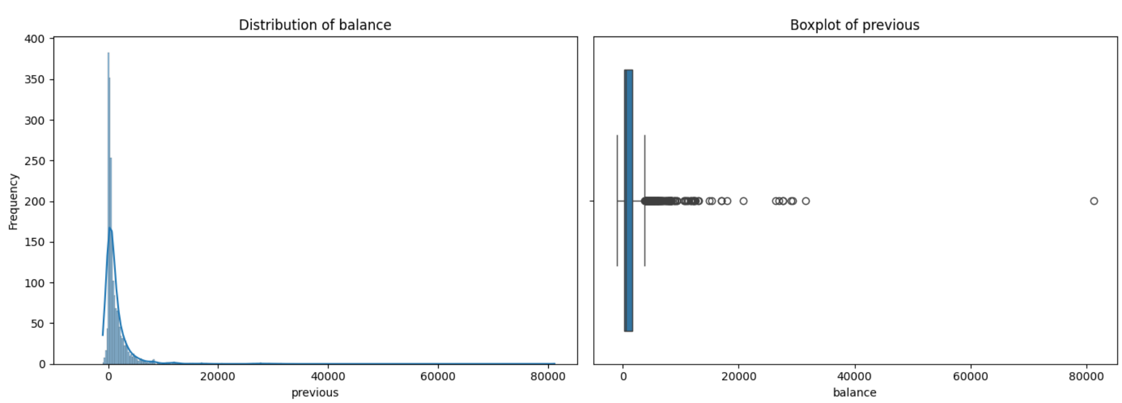
1.4 stats of numerical column

[912]:
```
df[numerical_columns].describe()
```

[912]:
```
                 age       balance          day     duration     campaign  \
pdays     previous
count  1988.000000   2000.000000  2000.000000  2000.000000  2000.000000
2000.000000  2000.000000
mean     41.753018   1413.663500    13.851500   292.020500     1.909500
167.896000     2.561500
std      12.724358   3131.224213     9.712189   221.557295     1.378862
131.754126     3.400735
min      18.000000   -980.000000     1.000000     7.000000     1.000000
-1.000000     0.000000
25%      32.000000    201.500000     5.000000   146.000000     1.000000
75.750000     1.000000
50%      38.000000    551.000000    12.000000   236.000000     1.000000
182.000000     2.000000
75%      50.000000   1644.500000    23.000000   379.000000     2.000000
251.000000     3.000000
max      93.000000  81204.000000    31.000000  1823.000000    11.000000
854.000000    55.000000
```

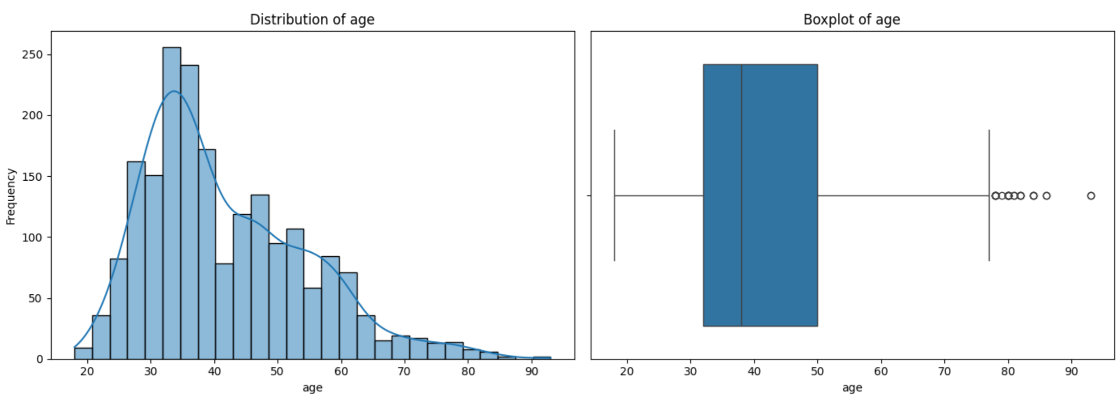
1.5 plotting numerical columns

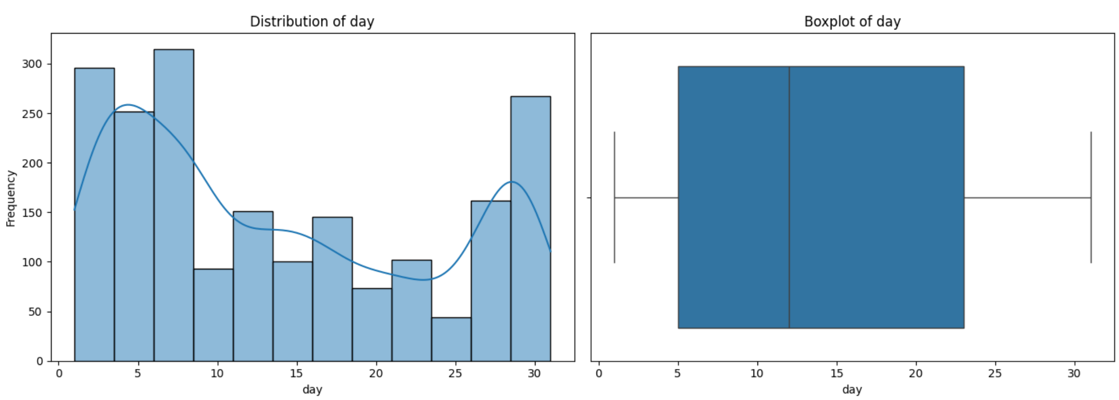
[913]:
```python
 # Distribution Plot
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
sns.histplot(df['balance'], kde=True, ax=axes[0])
axes[0].set_title(f"Distribution of balance")
axes[0].set_xlabel(column)
axes[0].set_ylabel("Frequency")
axes[0].set_xlim(left=-10000)

  # Boxplot
sns.boxplot(x=df["balance"], ax=axes[1])
axes[1].set_title(f"Boxplot of {column}")

plt.tight_layout()
plt.show()
```

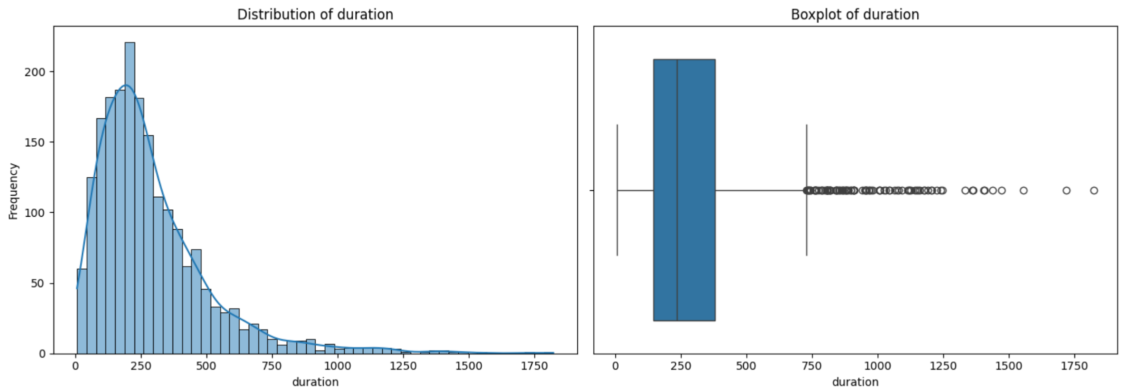


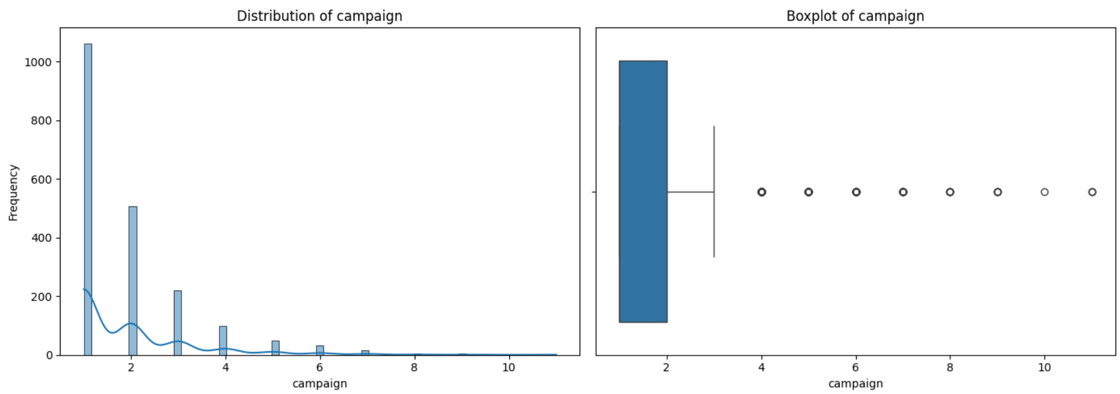
[914]:

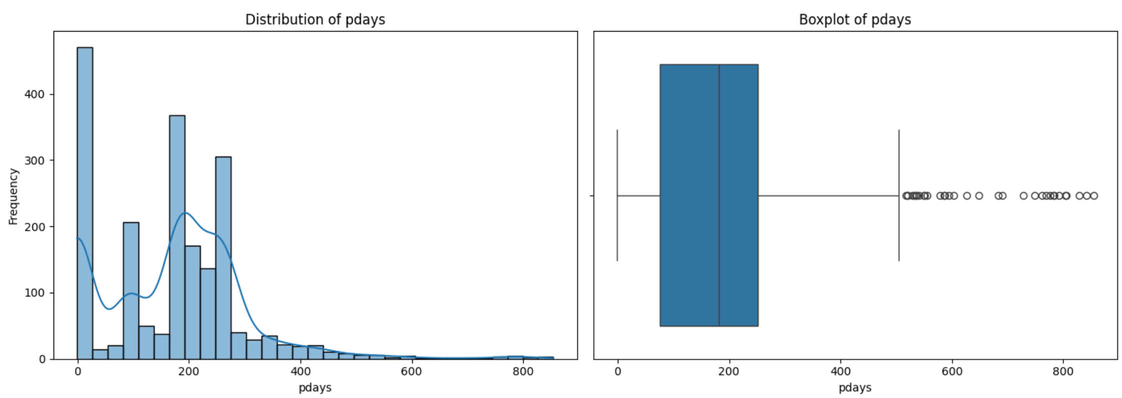
```
for column in numerical_columns:
    if column != 'balance':  # Skip the 'balance' column
        fig, axes = plt.subplots(1, 2, figsize=(14, 5))

        # Distribution Plot
        sns.histplot(df[column].dropna(), kde=True, ax=axes[0])
        axes[0].set_title(f"Distribution of {column}")
        axes[0].set_xlabel(column)
        axes[0].set_ylabel("Frequency")

        # Boxplot
        sns.boxplot(x=df[column], ax=axes[1])
        axes[1].set_title(f"Boxplot of {column}")

        plt.tight_layout()
        plt.show()
```

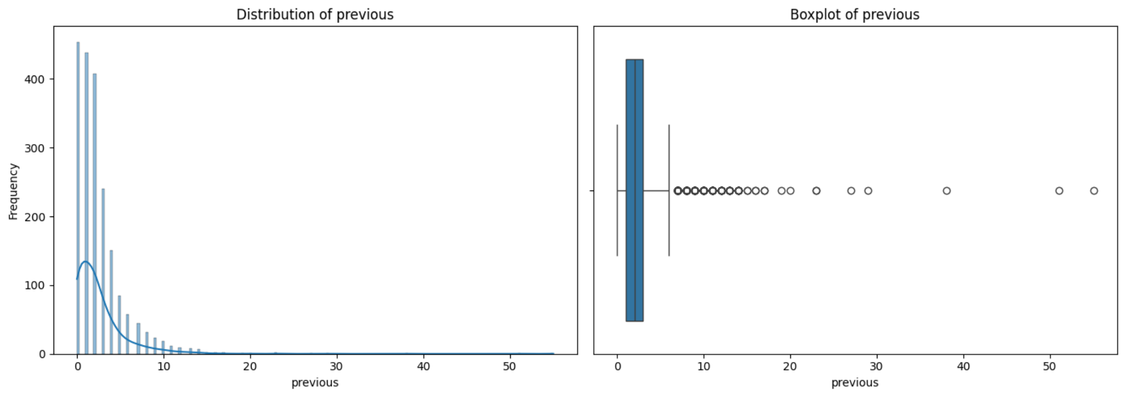





Distribution of duration
Frequency
duration
Boxplot of duration
duration


Distribution of campaign
Frequency
campaign
Boxplot of campaign
campaign


Distribution of pdays
Frequency
pdays
Boxplot of pdays
pdays




```
[ ]:
```

1.6 unique values of categorical variables

```
[915]: for column in df[categorical_columns]:
           print(f"{column} :")
           print(df[column].unique(), "\n")
```

```
job :
['technician' 'retired' 'blue-collar' 'self-employed' 'services'
 'management' 'admin.' 'unemployed' 'student' 'entrepreneur' 'housemaid'
 nan]

marital :
['single' 'divorced' 'married']

education :
['tertiary' 'secondary' nan 'primary']

default :
['no' 'yes']

housing :
['yes' 'no']

loan :
['no' 'yes']

contact :
['cellular' 'telephone' nan]

month :
['apr' 'dec' 'feb' 'jan' 'mar' 'may' 'nov' 'oct' 'aug' 'jul' 'jun' 'sep']
```

```
poutcome :
['failure' 'other' 'success' nan]

subscribed :
['no' 'yes']
```

nan (missing value) exists on these features: * Job * Education * Contact * Poutcome

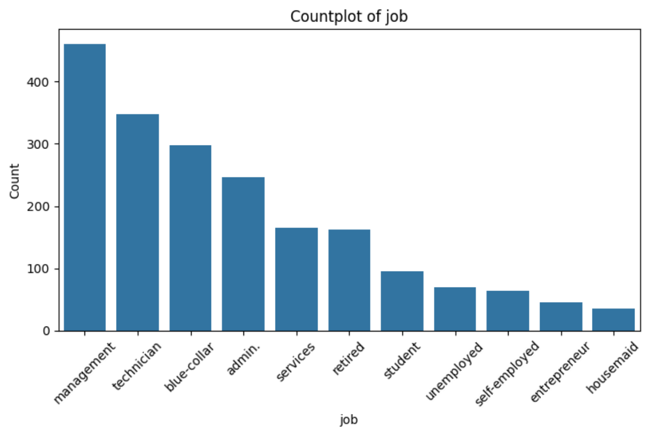
[916]:
```
df[categorical_columns].describe()
```

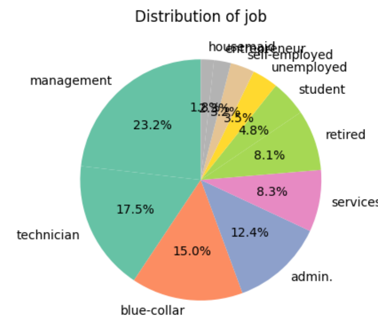
[916]:

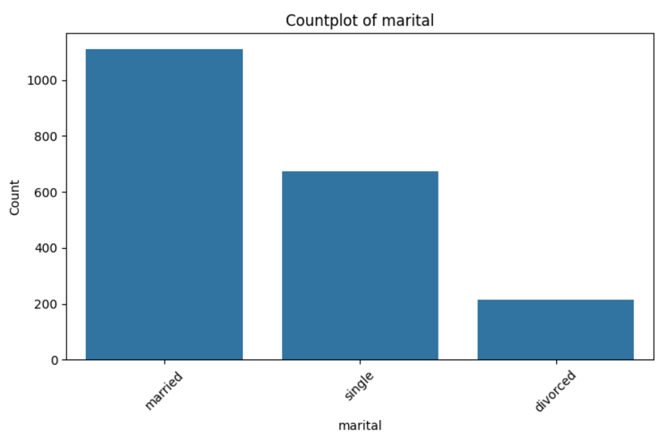
	job	marital	education	default	housing	loan	contact	month	poutcome	subscribed
count	1990	2000	1896	2000	2000	2000	1809	2000	1546	2000
unique	11	3	3	2	2	2	2	12	3	2
top	management	married	secondary	no	no	no	cellular	feb	failure	no
freq	461	1111	995	1985	1037	1750	1663	404	955	1000

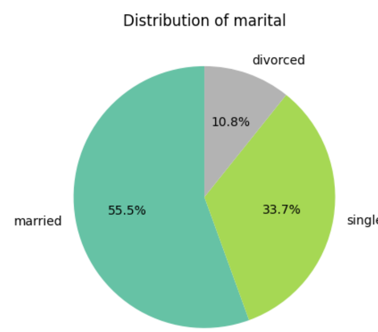
1.7 plotting categorical columns

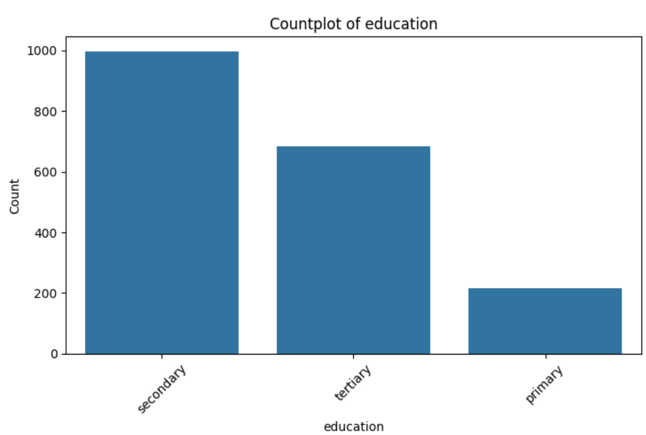
[917]:
```
for column in categorical_columns:
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))  # 1 row, 2 columns

    # Countplot
    sns.countplot(data=df, x=column, order=df[column].value_counts().index,␣
↪ax=axes[0])
    axes[0].set_title(f"Countplot of {column}")
    axes[0].set_xlabel(column)
    axes[0].set_ylabel("Count")
    axes[0].tick_params(axis='x', rotation=45)

    # Pie Chart
    df[column].value_counts().plot.pie(autopct='%1.1f%%', ax=axes[1],␣
↪startangle=90, cmap='Set2')
    axes[1].set_title(f"Distribution of {column}")
    axes[1].set_ylabel('')

    plt.tight_layout()
    plt.show()
```

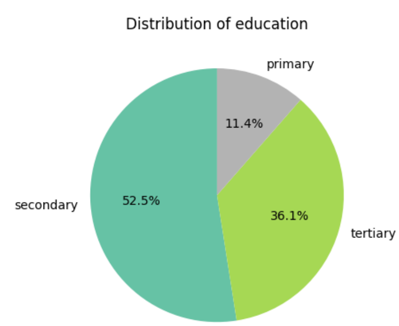

Countplot of job
Count
400
300
200
100
0
management
technician
blue-collar
admin.
services
retired
student
unemployed
self-employed
entrepreneur
housemaid
job

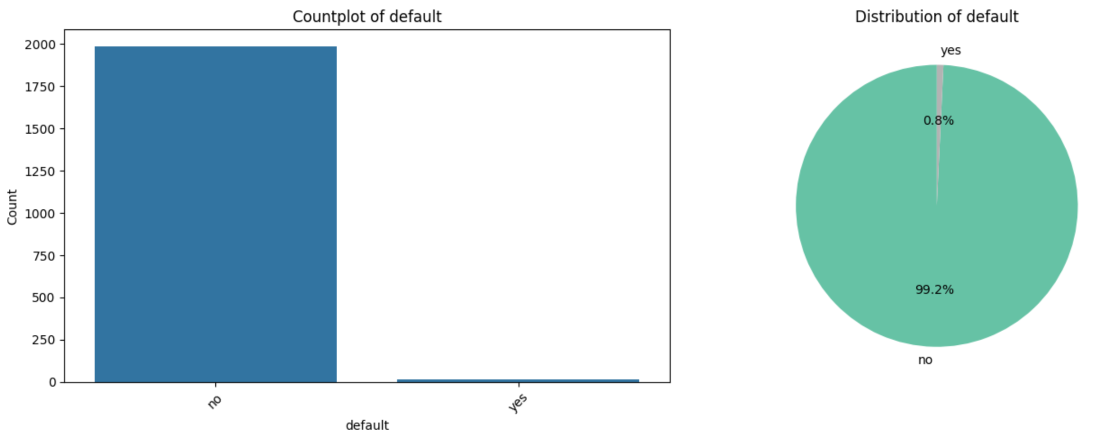

Distribution of job
housemaid
entrepreneur
self-employed
unemployed
student
retired
services
admin.
blue-collar
technician
management
23.2%
17.5%
15.0%
12.4%
8.3%
8.1%
4.8%
3.5%

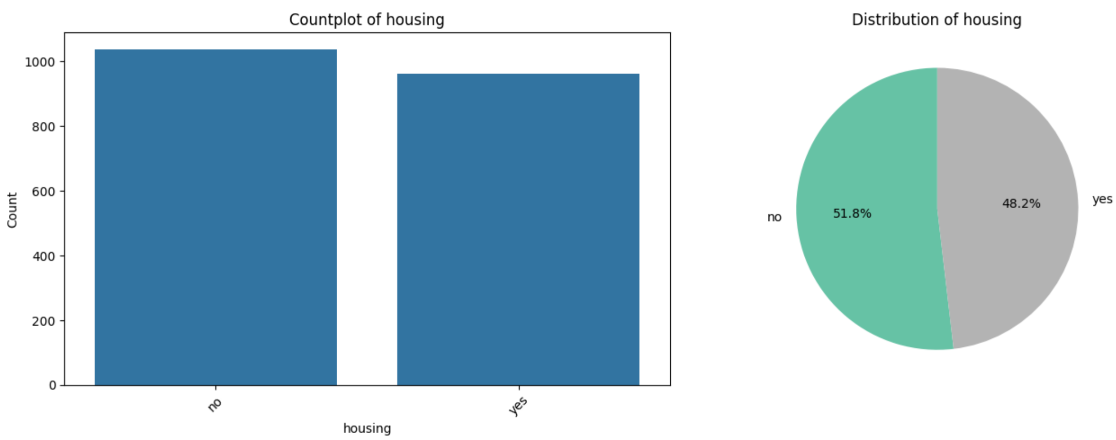

Countplot of marital
Count
1000
800
600
400
200
0
married
single
divorced
marital

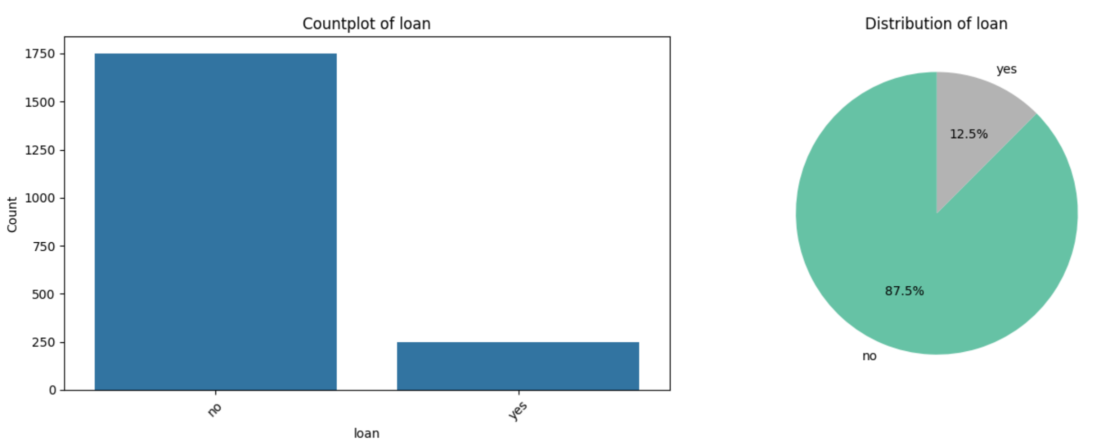

Distribution of marital
divorced
married
single
10.8%
55.5%
33.7%

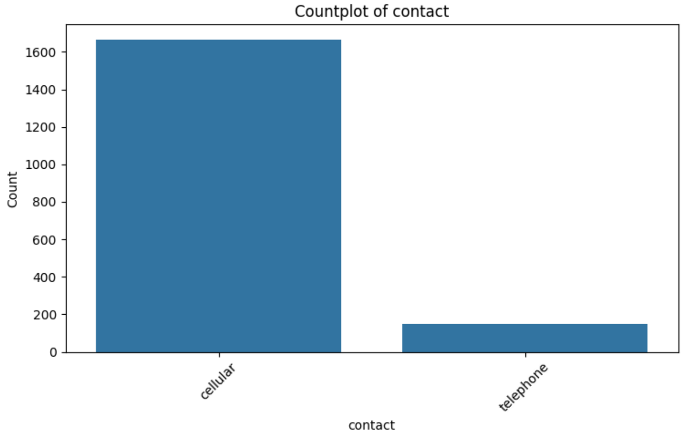

Countplot of education
Count
1000
800
600
400
200
0
secondary
tertiary
primary
education

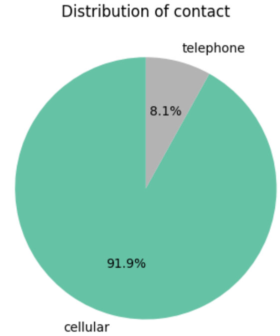

Distribution of education
primary
secondary
tertiary
11.4%
52.5%
36.1%

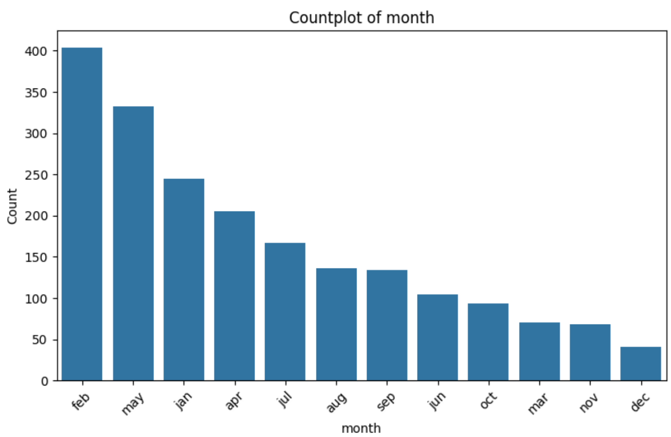

Countplot of default
2000
1750
1500
1250
1000
750
500
250
0
Count
no
yes
default
Distribution of default
yes
0.8%
99.2%
no

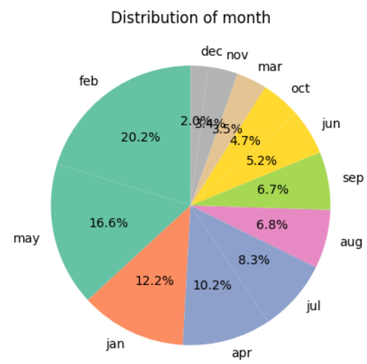

Countplot of housing
1000
800
600
400
200
0
Count
no
yes
housing
Distribution of housing
no
51.8%
48.2%
yes

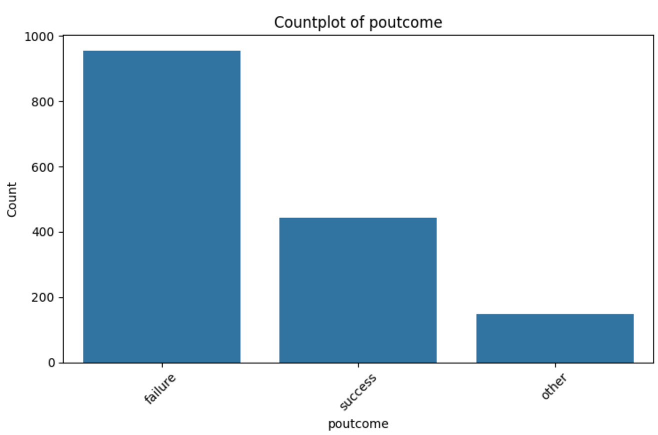

Countplot of loan
1750
1500
1250
1000
750
500
250
0
Count
no
yes
loan
Distribution of loan
yes
12.5%
87.5%
no

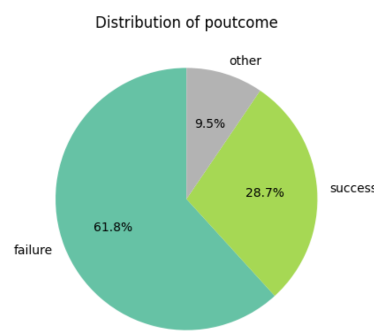

Countplot of contact
Count
1600
1400
1200
1000
800
600
400
200
0
cellular
telephone
contact

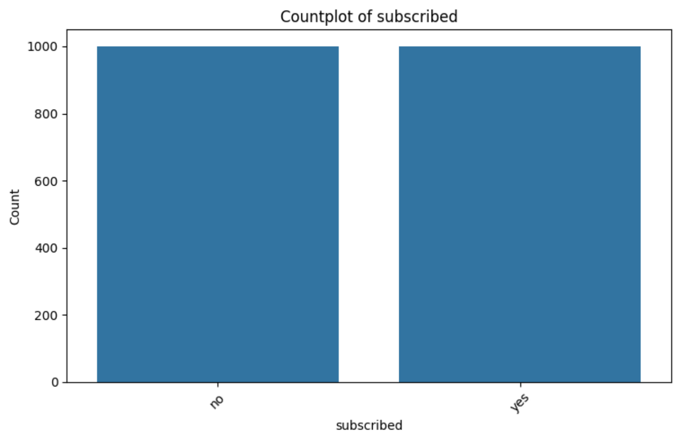

Distribution of contact
telephone
8.1%
91.9%
cellular

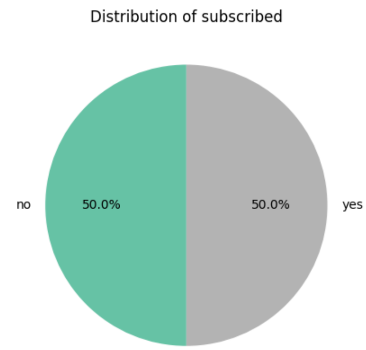

Countplot of month
Count
400
350
300
250
200
150
100
50
0
feb
may
jan
apr
jul
aug
sep
jun
oct
mar
nov
dec
month


Distribution of month
dec
nov
mar
oct
jun
sep
aug
jul
apr
jan
may
feb
20.2%
16.6%
12.2%
10.2%
8.3%
6.8%
6.7%
5.2%
4.7%
3.5%
2.0%


Countplot of poutcome
Count
1000
800
600
400
200
0
failure
success
other
poutcome


Distribution of poutcome
other
9.5%
28.7%
success
61.8%
failure

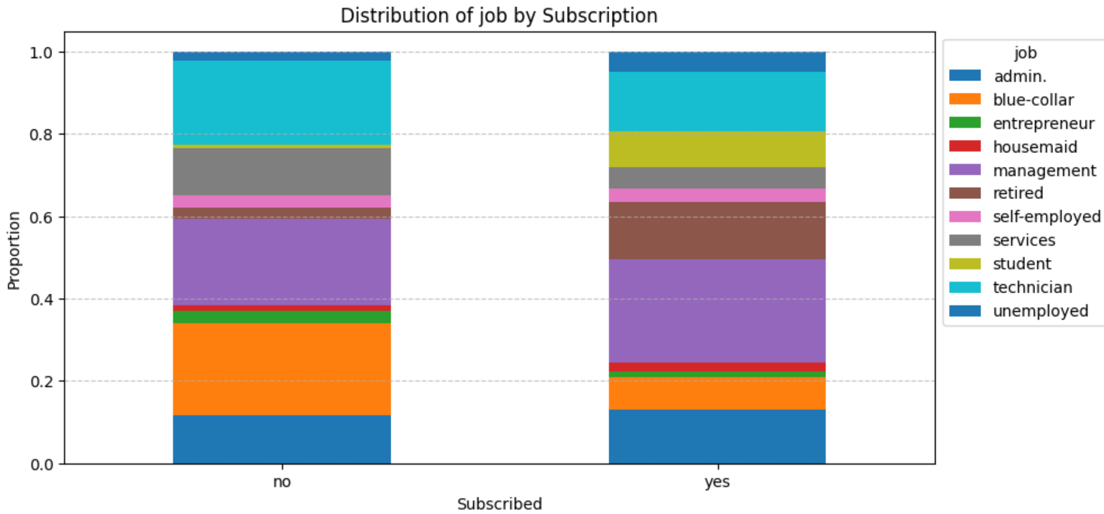


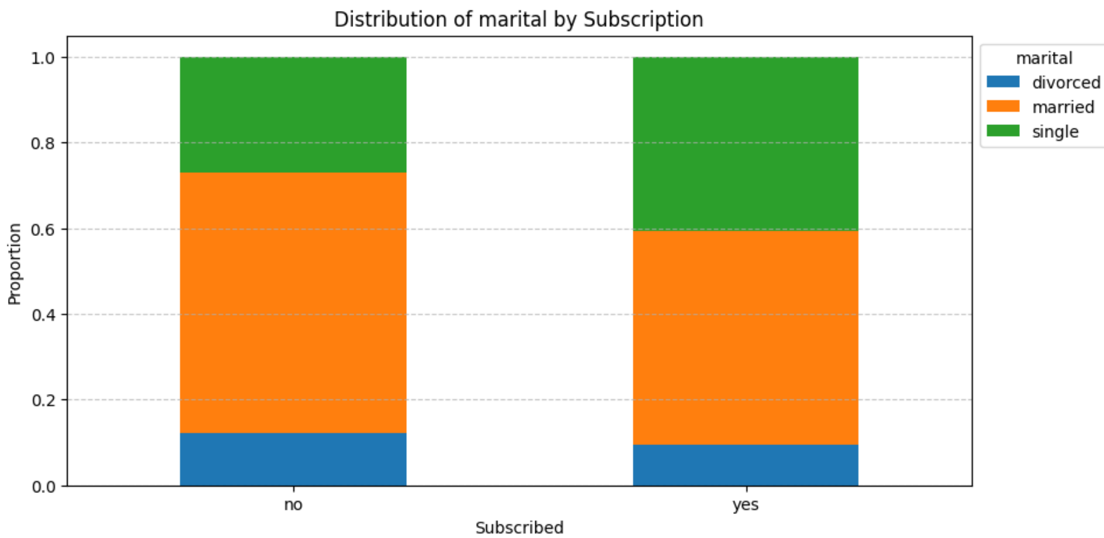


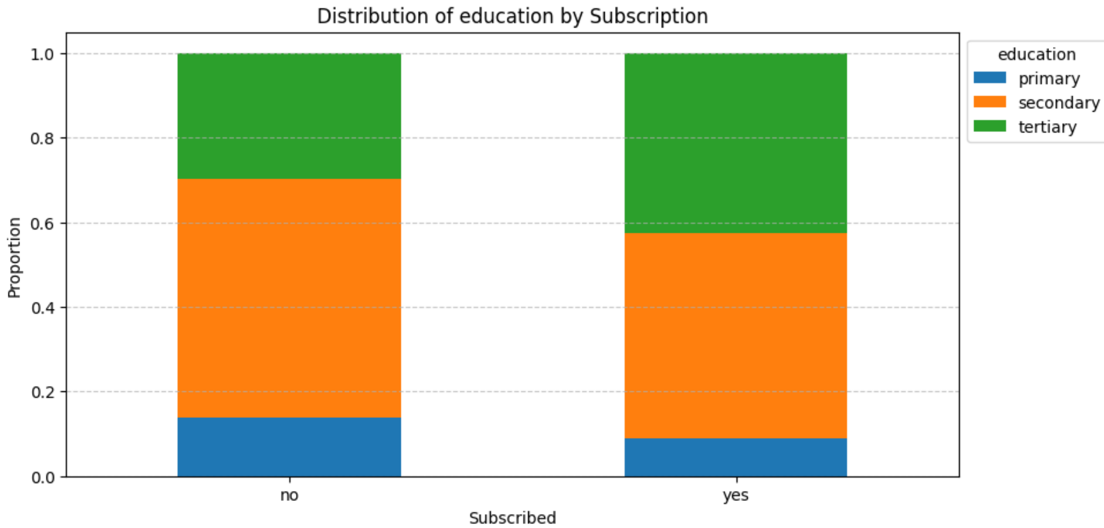
contact , default show low variaty loan show a little variaty but still might contribute to the data

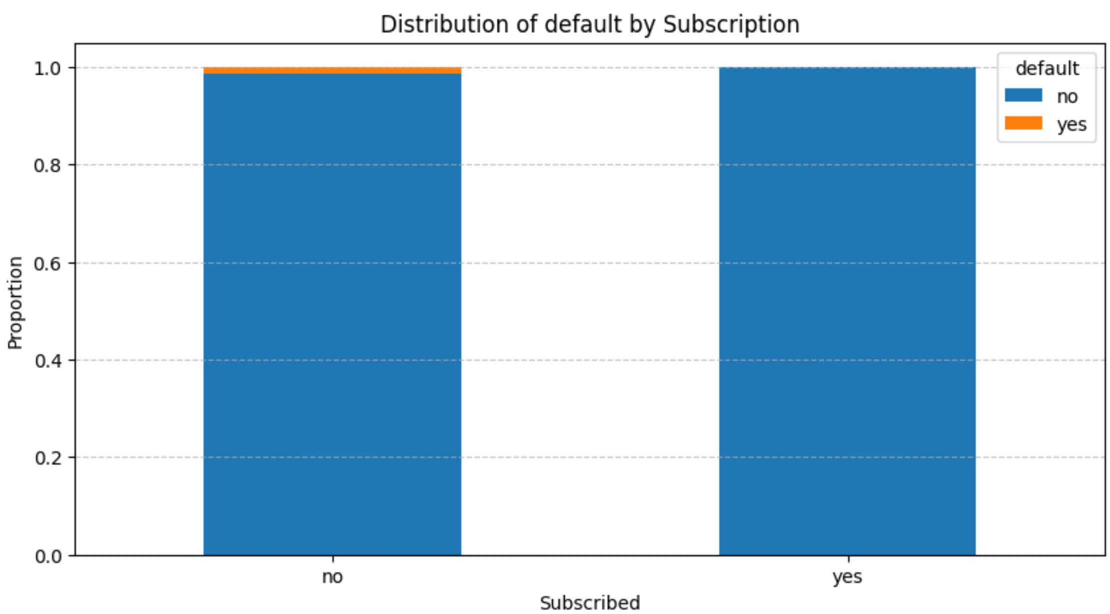
1.8 targeted comparaison

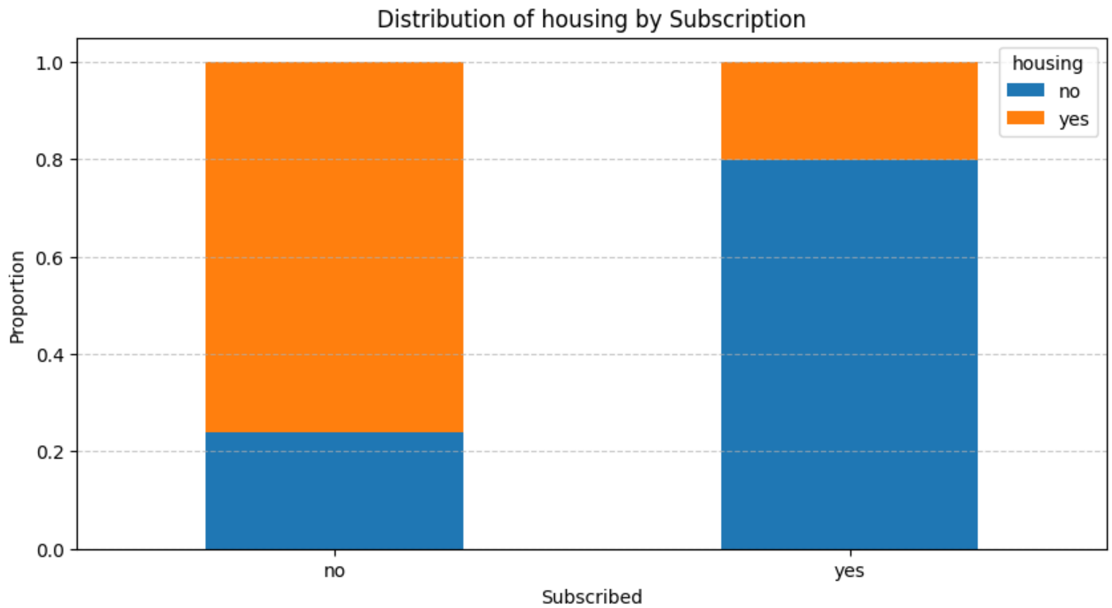
[918]:
```
# Group by 'subscribed' and plot value counts for all categorical columns␣
↪together
for column in categorical_columns:
    if column != 'subscribed':
        grouped = df.groupby('subscribed')[column].value_counts(normalize=True).
↪unstack().fillna(0)

        # Plot as a stacked bar chart
        grouped.plot(kind='bar', stacked=True, figsize=(10, 5))

        plt.title(f"Distribution of {column} by Subscription")
        plt.xlabel("Subscribed")
        plt.ylabel("Proportion")
        plt.legend(title=column, bbox_to_anchor=(1, 1))
        plt.xticks(rotation=0)
        plt.grid(axis="y", linestyle="--", alpha=0.7)

        plt.show()
```

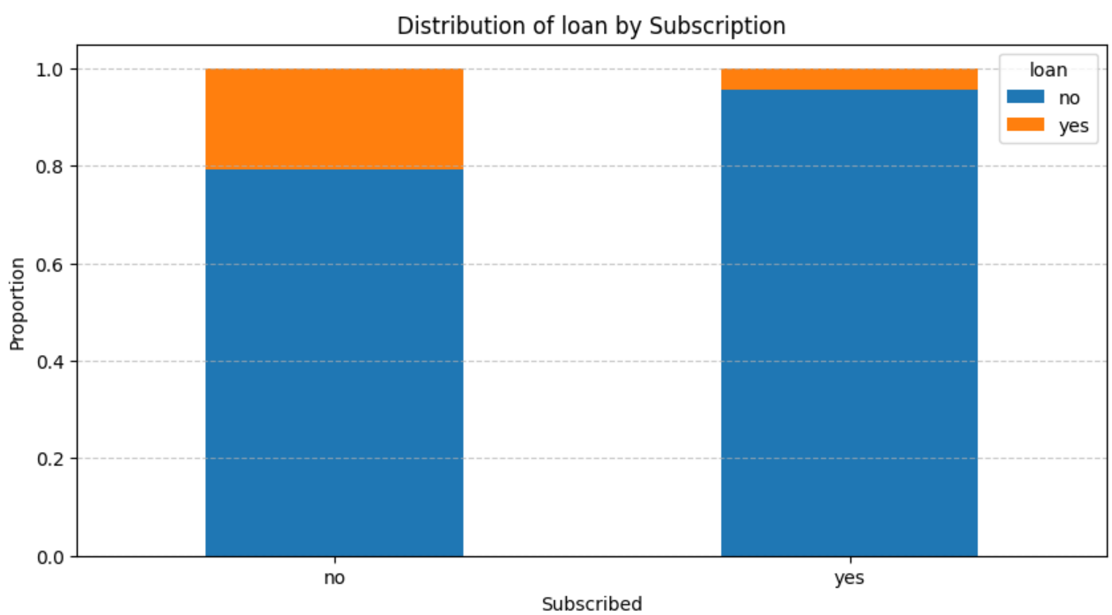

Distribution of job by Subscription
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
no
yes
Subscribed
job
admin.
blue-collar
entrepreneur
housemaid
management
retired
self-employed
services
student
technician
unemployed

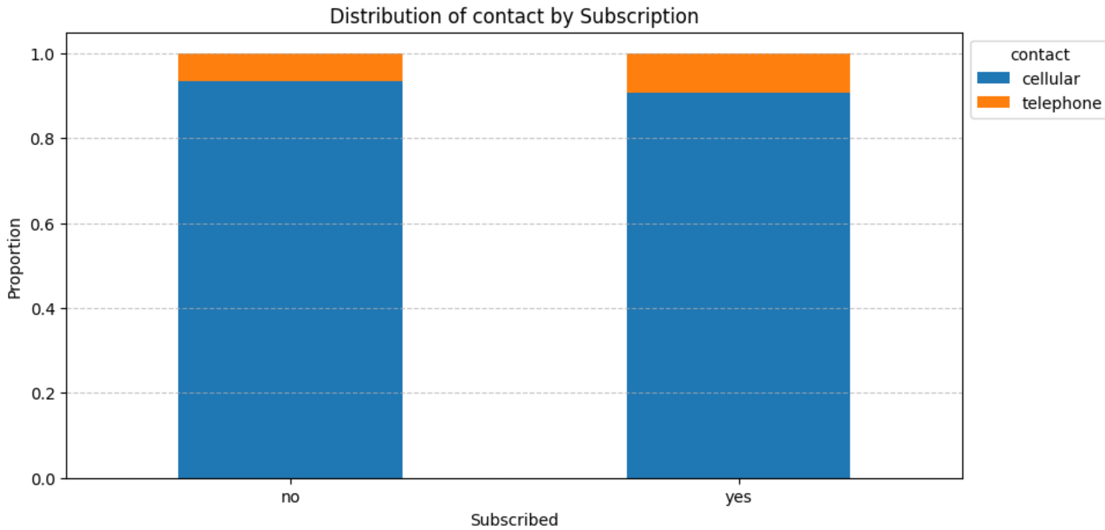

Distribution of marital by Subscription
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
no
yes
Subscribed
marital
divorced
married
single

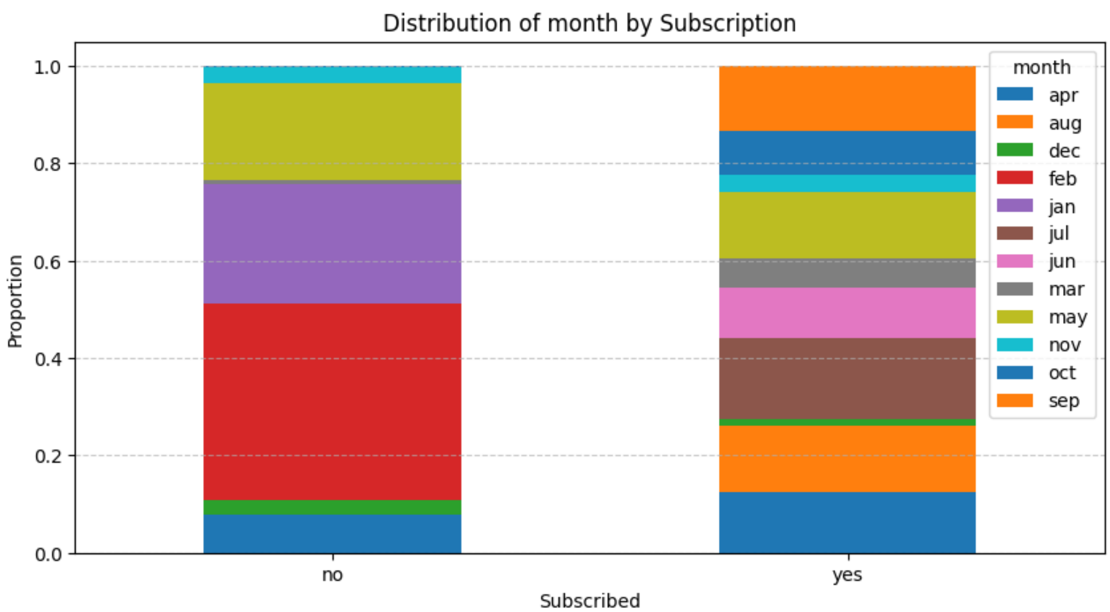

Distribution of education by Subscription
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
no
yes
Subscribed
education
primary
secondary
tertiary

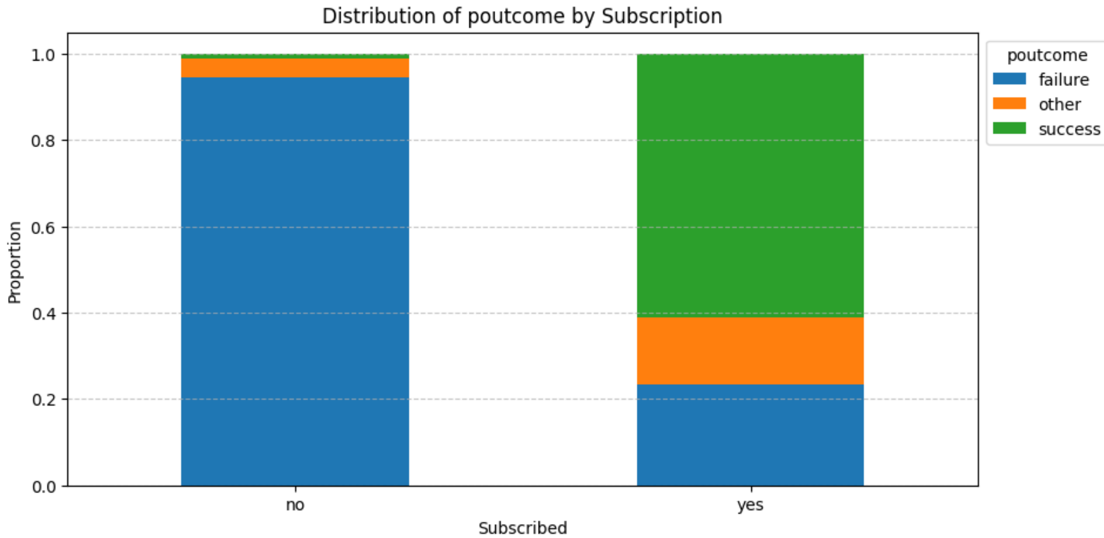

Distribution of default by Subscription
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
no
yes
Subscribed
default
no
yes


Distribution of housing by Subscription
housing
no
yes
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
no
yes
Subscribed


Distribution of loan by Subscription
loan
no
yes
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
no
yes
Subscribed


Distribution of contact by Subscription
contact
cellular
telephone
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
no
yes
Subscribed


Distribution of month by Subscription
month
apr
aug
dec
feb
jan
jul
jun
mar
may
nov
oct
sep
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
no
yes
Subscribed



[919]:
```python
print("--- Numerical Columns Grouped by Subscribed ---")

for column in numerical_columns:
    print(f"\nStatistics for {column}:\n")
    display(df.groupby('subscribed')[column].describe())
```

--- Numerical Columns Grouped by Subscribed ---

Statistics for age:

subscribed	count	mean	std	min	25%	50%	75%	max
no	991.0	40.655903	9.192425	22.0	33.0	39.0	48.0	64.0
yes	997.0	42.843531	15.382656	18.0	31.0	38.0	54.0	93.0

Statistics for balance:

subscribed	count	mean	std	min	25%	50%	75%	max
no	1000.0	942.862	2007.134003	-980.0	114.75	393.0	970.25	26306.0
yes	1000.0	1884.465	3891.864047	-205.0	315.00	875.0	2304.50	81204.0

Statistics for day:

	count	mean	std	min	25%	50%	75%	max
subscribed								

```
no          1000.0  12.364  10.667394  1.0  4.0   8.0  27.25  30.0
yes         1000.0  15.339   8.397893  1.0  9.0  14.0  22.00  31.0
```

```
Statistics for duration:
```

```
             count     mean         std   min    25%    50%     75%     max
subscribed
no          1000.0  206.696  175.152259   7.0   96.0  155.5  256.00  1823.0
yes         1000.0  377.345  230.154246  23.0  224.0  310.0  457.25  1720.0
```

```
Statistics for campaign:
```

```
             count   mean       std  min  25%  50%  75%   max
subscribed
no          1000.0  1.957  1.443341  1.0  1.0  1.0  2.0  11.0
yes         1000.0  1.862  1.310219  1.0  1.0  1.0  2.0  11.0
```

```
Statistics for pdays:
```

```
             count     mean         std  min    25%    50%    75%    max
subscribed
no          1000.0  185.400   99.759611 -1.0  136.0  211.0  259.0  536.0
yes         1000.0  150.392  155.468012 -1.0   -1.0  123.5  185.0  854.0
```

```
Statistics for previous:
```

```
             count   mean       std  min  25%  50%  75%   max
subscribed
no          1000.0  2.362  3.287516  0.0  1.0  2.0  3.0  51.0
yes         1000.0  2.761  3.500590  0.0  0.0  2.0  4.0  55.0
```

1.9 targetting the pdays

[920]:
```
(df['pdays'] == -1).mean()
```

[920]:
```
np.float64(0.227)
```

22.7 % of the clients hasn t been contacted

1.10 targeting the balance

[921]:
```
# How many zero values in `balance?`
print("There are %d account holder or %5f of the total clients who have zero␣
↪balance" % ((df[df['balance']==0]['balance'].count()),
                                                                              ␣
↪          (df[df['balance']==0]['balance'].count())/(df['balance'].count())))
# How many negative values in `balance`?
print("There are %d account holder or %5f of the total clients who owe money" %␣
↪((df[df['balance']<0]['balance'].count()),
                                                              ␣
↪(df[df['balance']<0]['balance'].count())/(df['balance'].count())))
```

```
There are 86 account holder or 0.043000 of the total clients who have zero
balance
There are 93 account holder or 0.046500 of the total clients who owe money
```

[922]:
```
# Is there any of those clients who subscribed to term deposit?
print("There are %d account holder who have zero balance and subscribed term␣
↪deposit" % df[(df['balance']==0) & (df['subscribed']=='yes')]['balance'].
↪count())

print("There are %d account holder who have negative balance and subscribed␣
↪term deposit" % df[(df['balance']<0) & (df['subscribed']=='yes')]['balance'].
↪count())
```

```
There are 42 account holder who have zero balance and subscribed term deposit
There are 7 account holder who have negative balance and subscribed term deposit
```

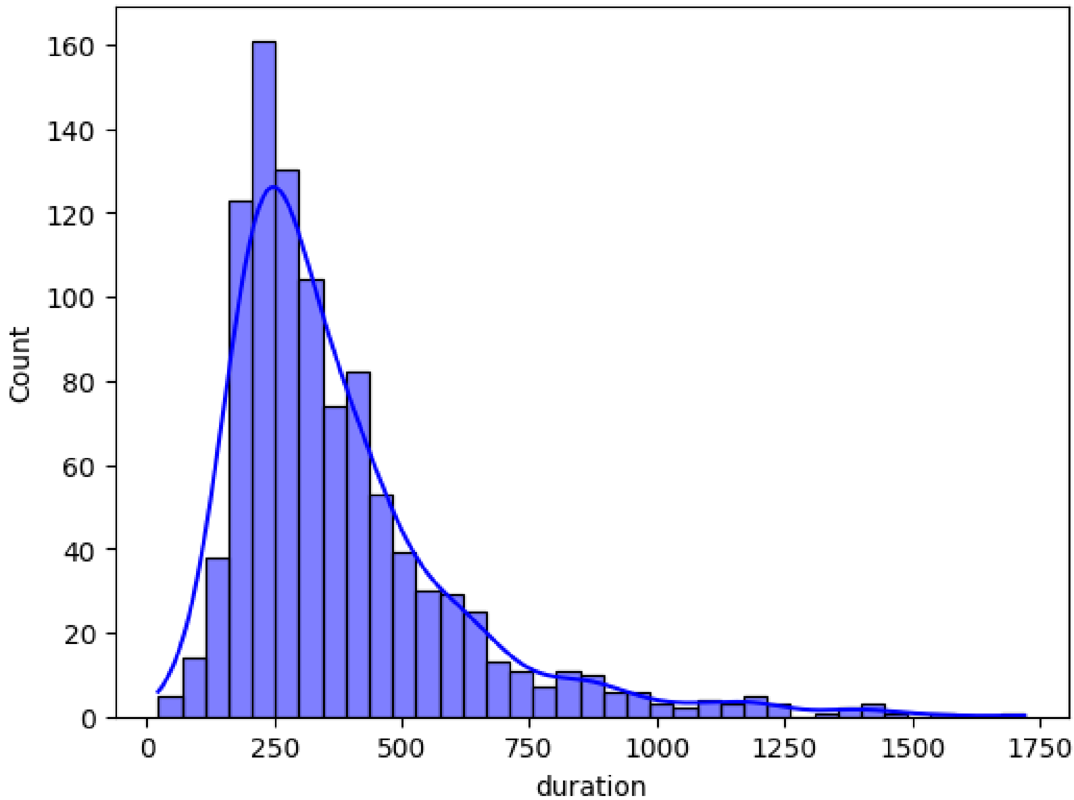
1.11 targeting duration

[923]:
```
# The range of calls duration from clients who subscribed to the term deposit
print("The minimum duration (in seconds) to finalize a deal :",␣
↪df[df['subscribed']=='yes']['duration'].min())
print("The maximum duration (in seconds) to finalize a deal :",␣
↪df[df['subscribed']=='yes']['duration'].max())

# The average of calls duration from clients who subscribed to the term deposit
print("The average duration (in seconds) to finalize a deal :",␣
↪df[df['subscribed']=='yes']['duration'].mean())
```

```
The minimum duration (in seconds) to finalize a deal : 23
The maximum duration (in seconds) to finalize a deal : 1720
The average duration (in seconds) to finalize a deal : 377.345
```

[924]:
```
# The calls duration distribution from clients who subscribed to the term␣
↪deposit
sns.histplot(df[df['subscribed']=='yes']['duration'], color='blue', kde=True)
```

[924]: <Axes: xlabel='duration', ylabel='Count'>



Most subscribing clients had calls lasting between 175 and 375 seconds.

!!!!!!!!! correlation doesn t mean causation

[925]:
```
# show the clients with no contact in  the last campaign but subscribed to the␣
↪term deposit
df[(df['duration']==0) & (df['subscribed']=='yes')]
```

[925]:
```
Empty DataFrame
Columns: [age, job, marital, education, default, balance, housing, loan,
contact, day, month, duration, campaign, pdays, previous, poutcome, subscribed]
Index: []
```

1.12 targeting the poutcome

[926]:
```
# Show Clients who have been contacted before but unknown outcome
df[(df['poutcome'] == '') & (df['previous']!=0)]
```

[926]:
```
Empty DataFrame
Columns: [age, job, marital, education, default, balance, housing, loan,
```

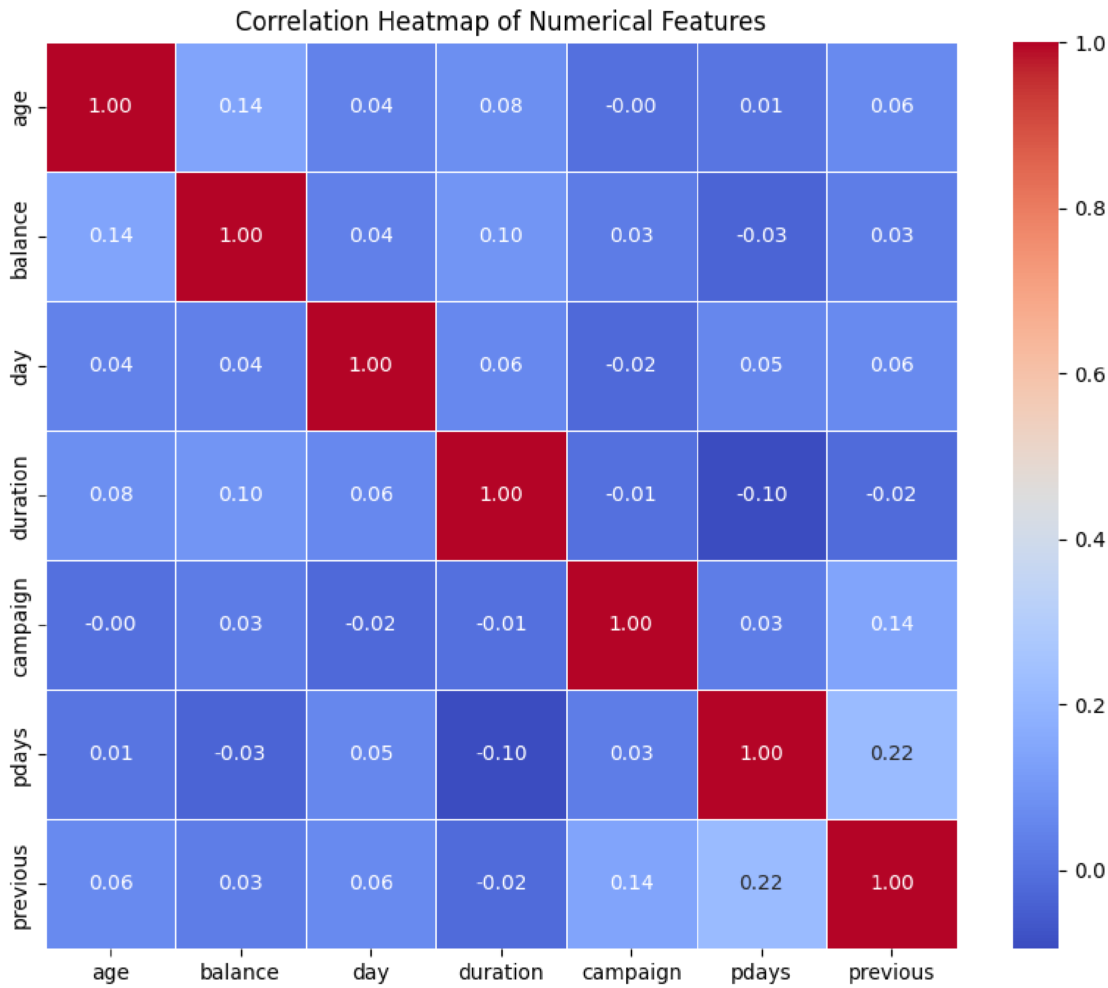
```
contact, day, month, duration, campaign, pdays, previous, poutcome, subscribed]
Index: []
```

[927]:
```
# pdays is -1 => previous is 0 ?
print(df[(df['pdays'] != -1) & (df['previous']==0)]['pdays'].count() ==
     df[(df['pdays'] == -1) & (df['previous'] != 0)]['pdays'].count())
```

```
True
```

1.13 correlation matrix

[928]:
```
# Calculate the correlation matrix
correlation_matrix = df.select_dtypes(include=['int64', 'float64']).corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



there is almost no correlation between any pair of numerical columns

2 summary

`nan (missing value)` exists on these features: * Job * Education * Contact * Poutcome

columns to drop : * contact : Almost all clients contacted via cellular
* default : Almost no clients have defaulted (99.2% "no")
* day : we ll treat the cycle data in month and dispose of this column * duration : has a wide range of values ,might outweigh other important features.

3 DATA preparation

[929]:
```
df.isnull().sum()
```

[929]:
```
age            12
job            10
marital         0
education     104
default         0
balance         0
housing         0
loan            0
contact       191
day             0
month           0
duration        0
campaign        0
pdays           0
previous        0
poutcome      454
subscribed      0
dtype: int64
```

[930]:
```
original_df = df.copy()
```

3.0.1 Droping unnecessary columns

[931]:
```
df.drop(columns=['contact', 'duration', 'default','day'], inplace=True)
```

[932]:
```
df.columns
```

[932]:
```
Index(['age', 'job', 'marital', 'education', 'balance', 'housing', 'loan',
'month', 'campaign', 'pdays', 'previous', 'poutcome', 'subscribed'],
dtype='object')
```

3.0.2 handling missing values

- age : the distribution is positively skewed for that we ll use the median to impute the missing values

[933]:
```
df.fillna({'age': df['age'].median()}, inplace=True)
```

- poutcome : the data isn't missing but rather unknown

[934]:
```
df['poutcome'] = df['poutcome'].replace("", "never")
```

- job : missing values < 0.5 % of the data , we can dispose of those rows

[935]:
```
df = df.dropna(subset=['job'])
```

- education : we create a new category of unknown education level ' unknown '

[936]:
```
df.fillna({'education': 'unknown'}, inplace=True)
```

3.0.3 encoding

[937]:
```
# Convert 'month' into seasonal categories
season_map = {
    'dec': 'Winter', 'jan': 'Winter', 'feb': 'Winter',
    'mar': 'Spring', 'apr': 'Spring', 'may': 'Spring',
    'jun': 'Summer', 'jul': 'Summer', 'aug': 'Summer',
    'sep': 'Fall', 'oct': 'Fall', 'nov': 'Fall'
}

df['season'] = df['month'].map(season_map)
df.drop(columns=['month'], inplace=True)  # Drop original month column

# One-Hot Encoding for 'season' column
df = pd.get_dummies(df, columns=['season'], drop_first=True)

# One-Hot Encoding for Other Nominal Categorical Columns
df = pd.get_dummies(df, columns=['job', 'marital', 'education', 'poutcome'],␣
 ↪drop_first=True)
```

[938]:
```
# Label Encoding for Binary Categorical Columns
binary_cols = ['housing', 'loan']
le = LabelEncoder()

for col in binary_cols:
    df[col] = le.fit_transform(df[col])
```

3.1 Scaling

[ ]:
```
# Define numerical columns to scale
numerical_columns = ['age', 'balance', 'campaign', 'previous', 'pdays']

# Initialize RobustScaler
scaler = RobustScaler()

# Apply scaling
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```