**This dataset is about the direct phone call marketing campaigns, which aim to promote term deposits among existing customers, by a Portuguese banking institution from May 2008 to November 2010.**

Input variables:

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5 - default: has credit in default? (categorical: 'no','yes','unknown')

6 - housing: has housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown') # related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model. # other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success') # social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

17 - cons.price.idx: consumer price index - monthly indicator (numeric)

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

20 - nr.employed: number of employees - quarterly indicator (numeric) Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
data=pd.read_csv('bank-additional-full.csv',sep=';')
```

```
df=data.copy()
```

```
df.sample(5)
```

| | age | job | marital | education | default | housing | loan | contact | mor |
|---|---|---|---|---|---|---|---|---|---|
| **11357** | 31 | technician | married | high.school | no | no | no | telephone | . |
| **33955** | 40 | admin. | single | high.school | no | no | yes | cellular | m |
| **19435** | 49 | housemaid | married | university.degree | no | yes | no | cellular | a |
| **14732** | 38 | self-employed | married | professional.course | no | no | no | cellular | |
| **6726** | 51 | unemployed | married | basic.9y | unknown | yes | no | telephone | m |

```
columns=df.columns
columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

```
int_cols=[]
obj_cols=[]
flo_cols=[]
for column in df.columns:
        if df[column].dtype == 'int64':
          int_cols.append(column)
        elif df[column].dtype=='object':
          obj_cols.append(column)
        elif df[column].dtype=='float64':
          flo_cols.append(column)
```

```
for i in columns:
    unique_values = df[i].unique()
    if len(unique_values) > 30:
        print()
        print(f"{i}: {len(unique_values)}")
    else:
        print()
        print(f"{i}: {unique_values}")
```

```
age: 78
```

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

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

education: ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']

default: ['no' 'unknown' 'yes']

housing: ['no' 'yes' 'unknown']

loan: ['no' 'yes' 'unknown']

contact: ['telephone' 'cellular']

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

day_of_week: ['mon' 'tue' 'wed' 'thu' 'fri']

duration: 1544

campaign: 42

pdays: [999   6   4   3   5   1   0  10   7   8   9  11   2  12  13  14  15  16
  21  17  18  22  25  26  19  27  20]

previous: [0 1 2 3 4 5 6 7]

poutcome: ['nonexistent' 'failure' 'success']

emp.var.rate: [ 1.1  1.4 -0.1 -0.2 -1.8 -2.9 -3.4 -3.  -1.7 -1.1]

cons.price.idx: [93.994 94.465 93.918 93.444 93.798 93.2   92.756 92.843 93.075 92.893
 92.963 92.469 92.201 92.379 92.431 92.649 92.713 93.369 93.749 93.876
 94.055 94.215 94.027 94.199 94.601 94.767]

cons.conf.idx: [-36.4 -41.8 -42.7 -36.1 -40.4 -42.  -45.9 -50.  -47.1 -46.2 -40.8 -33.6
 -31.4 -29.8 -26.9 -30.1 -33.  -34.8 -34.6 -40.  -39.8 -40.3 -38.3 -37.5
 -49.5 -50.8]

euribor3m: 316

nr.employed: [5191.  5228.1 5195.8 5176.3 5099.1 5076.2 5017.5 5023.5 5008.7 4991.6
 4963.6]

y: ['no' 'yes']
```

Got The Unique Values from the dataframe for each columns their is no redundant value with spelling mistake

## Checking For Null Values

```
df.isnull().sum()
```

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

## Checking Duplicate Rows

```
index=df[df.duplicated()].index
```

```
df.drop(index,inplace=True)
```

```
df[df.duplicated()].index
```

```
Index([], dtype='int64')
```

## EDA

**Univariate Analysis**

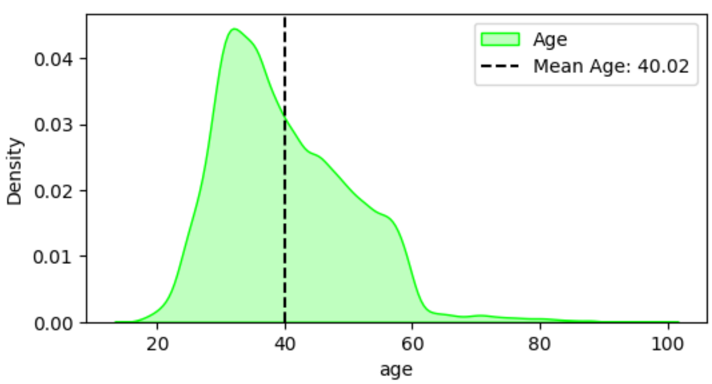
```
k='Black'
```

```
print(int_cols,'\n',flo_cols,'\n',obj_cols)
```

```
['age', 'duration', 'campaign', 'pdays', 'previous']
 ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
 ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'y']
```

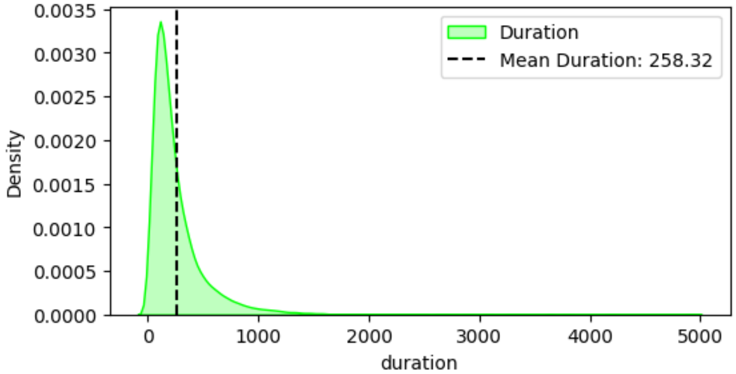
AGE

```
plt.figure(figsize=(6,3))
sns.kdeplot(df['age'], color='lime', shade=True, label='Age')
plt.axvline(x=df['age'].mean(), color='k', linestyle='--', label='Mean Age: {:.2f}'.format(df['age'].mean()))
plt.legend()
plt.show()
```




Duration

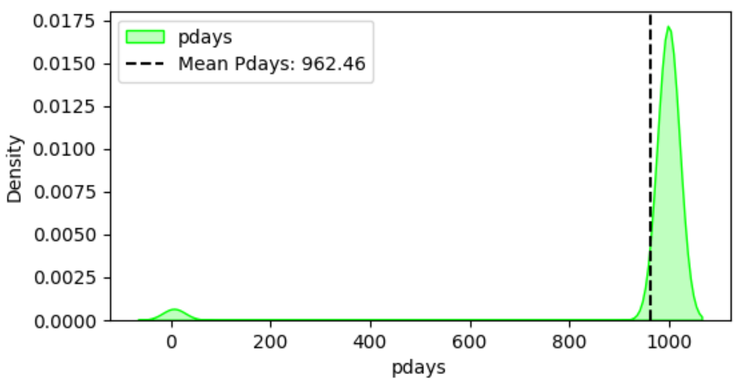
```
plt.figure(figsize=(6,3))
sns.kdeplot(df['duration'], color='lime', shade=True, label='Duration')
plt.axvline(x=df['duration'].mean(), color='k', linestyle='--', label='Mean Duration: {:.2f}'.format(df['duration'].mean()))
plt.legend()
plt.show()
```




This Data is skewed towards right this means most of the clients has been never called before

Pdays

```
plt.figure(figsize=(6,3))
sns.kdeplot(df['pdays'], color='lime', shade=True, label='pdays')
plt.axvline(x=df['pdays'].mean(), color='k', linestyle='--', label='Mean Pdays: {:.2f}'.format(df['pdays'].mean()))
plt.legend()
plt.show()
```

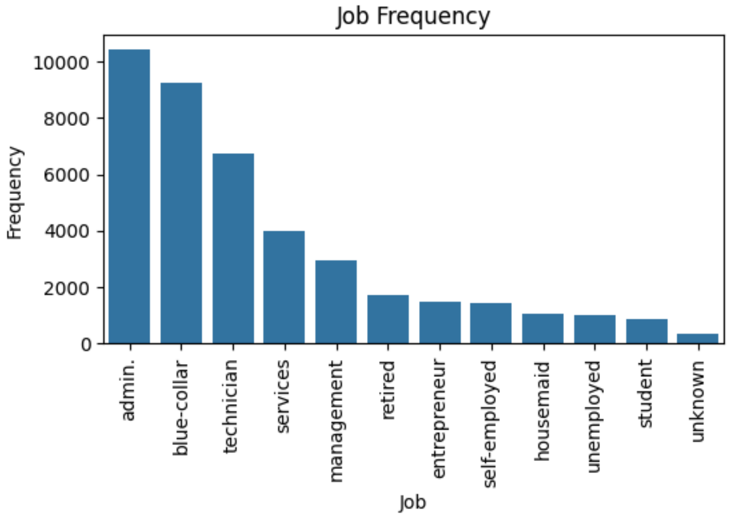



As I told in perivous 'duration plot' most of the clients have never been contacted

This plot also make that statement true as most of the value in this pdays column are 999

Job

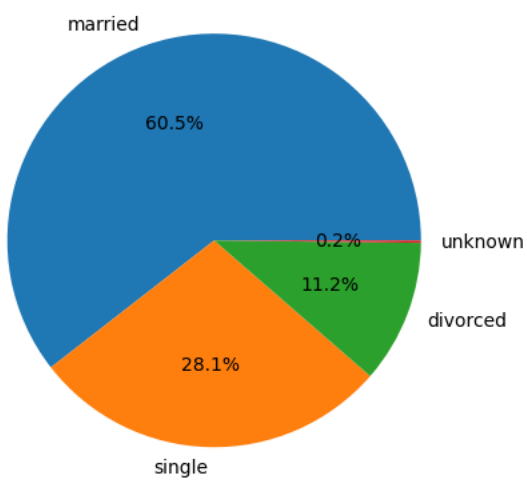
```
plt.figure(figsize=(6,3))

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

sns.barplot(x=job_counts.index, y=job_counts.values)
plt.xticks(rotation=90)

plt.xlabel('Job')
plt.ylabel('Frequency')
plt.title('Job Frequency')
plt.show()
print(job_counts)
```




```
job
admin.           10419
blue-collar       9253
technician        6739
services          3967
management        2924
retired           1718
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: count, dtype: int64
```

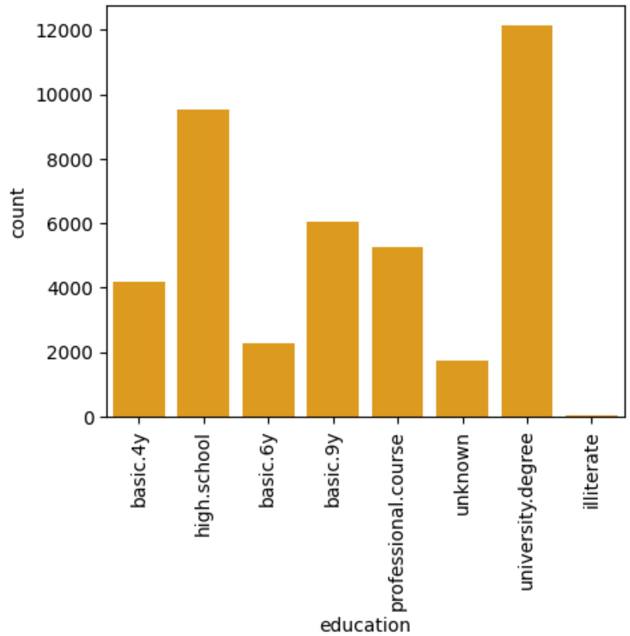
MARITAL

```
plt.figure(figsize=(5,6))
mar=df['marital'].value_counts()
plt.pie(mar,labels=mar.index,autopct='%1.1f%%');
```




EDUCATION

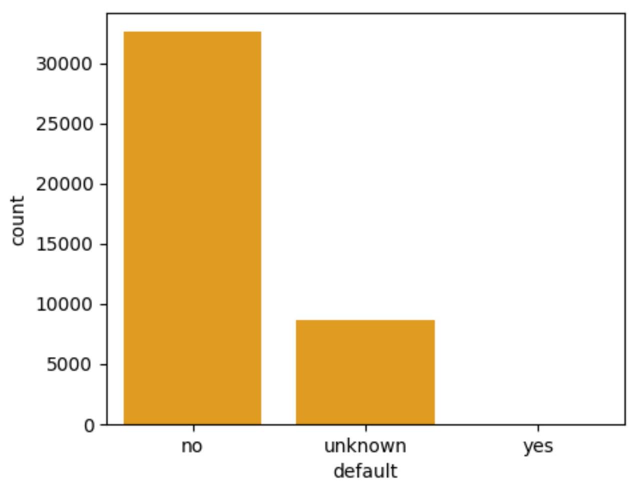
```
edu=df['education'].value_counts()
edu
plt.figure(figsize=(5,4))
sns.countplot(x='education',data=df,color='orange')
plt.xticks(rotation=90);
```




DEFAULT

```
plt.figure(figsize=(5,4))
sns.countplot(x='default', data=df,color='orange')
```

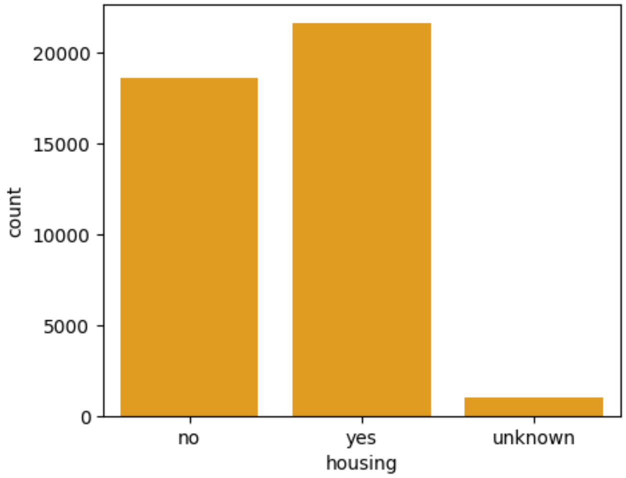
```
<Axes: xlabel='default', ylabel='count'>
```




HOUSING

```
housing=df['housing'].value_counts()
plt.figure(figsize=(5,4))
sns.countplot(x='housing', data=df,color='orange')
print(housing)
```

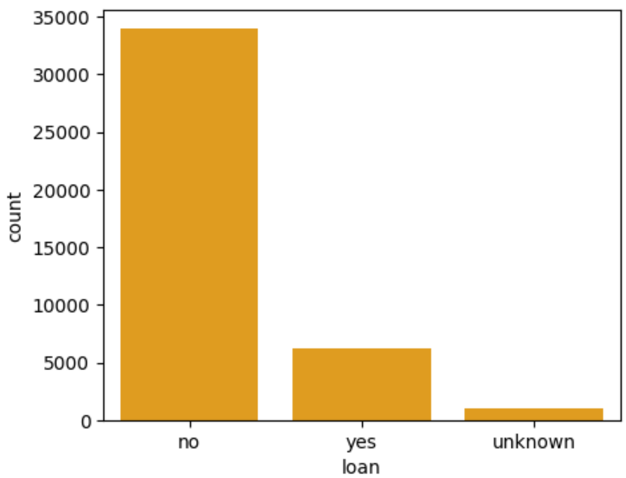
```
housing
yes        21571
no         18615
unknown      990
Name: count, dtype: int64
```




LOAN

```
plt.figure(figsize=(5,4))
sns.countplot(x='loan', data=df,color='orange')
```

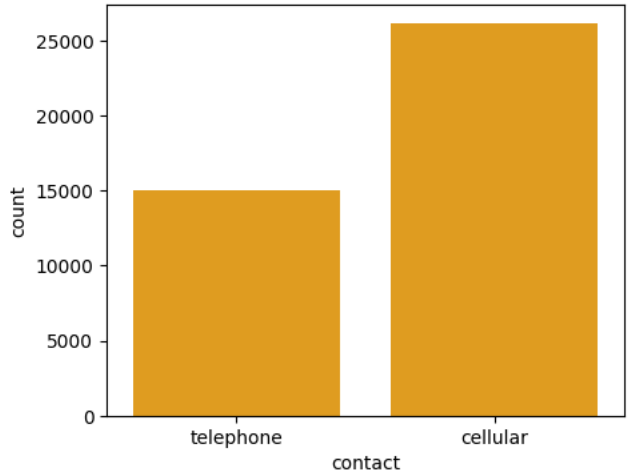
```
<Axes: xlabel='loan', ylabel='count'>
```




CONTACT

```
plt.figure(figsize=(5,4))
sns.countplot(x='contact', data=df,color='orange')
```

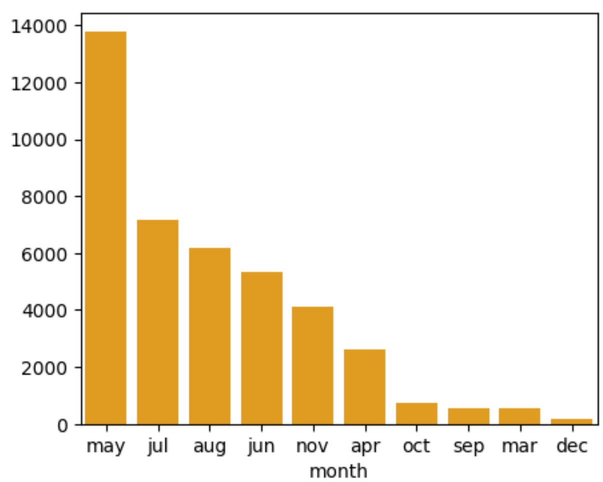
```
<Axes: xlabel='contact', ylabel='count'>
```




MONTH

```
x='month'
temp=df[x].value_counts()
plt.figure(figsize=(5,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
print(temp)
```

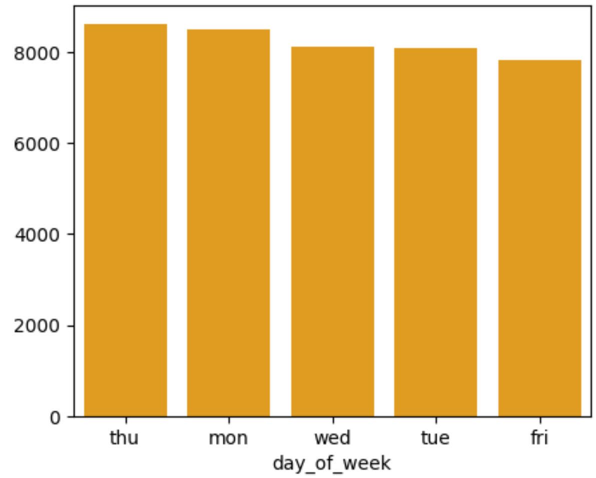
```
month
may    13767
jul     7169
aug     6176
jun     5318
nov     4100
apr     2631
oct      717
sep      570
mar      546
dec      182
Name: count, dtype: int64
```




DAY OF WEEK

```
x='day_of_week'
temp=df[x].value_counts()
plt.figure(figsize=(5,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
print(temp)
```

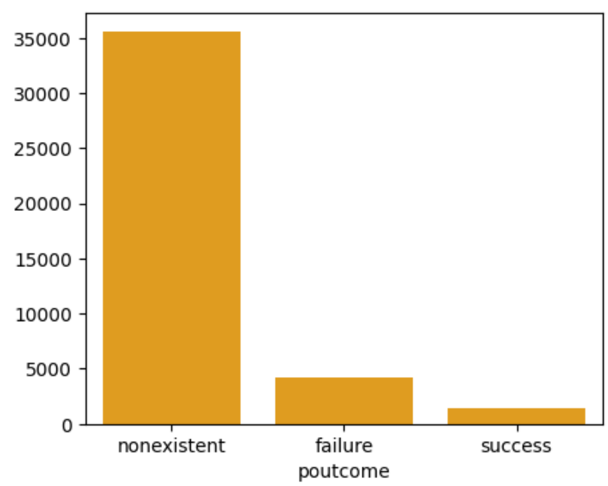
```
day_of_week
thu    8618
mon    8512
wed    8134
tue    8086
fri    7826
Name: count, dtype: int64
```




POUTCOME

```
x='poutcome'
temp=df[x].value_counts()
plt.figure(figsize=(5,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
print(temp)
```

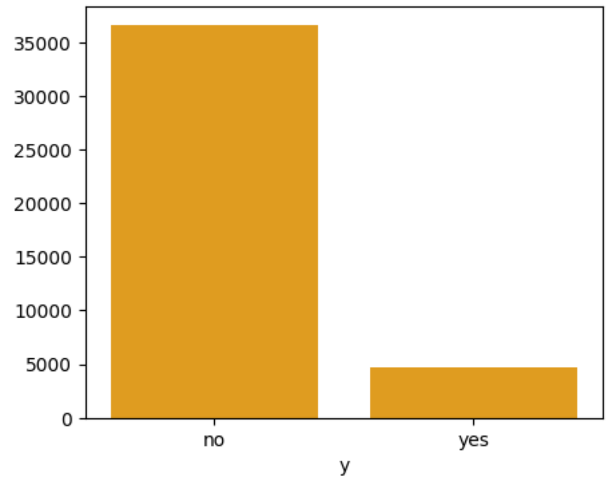
```
poutcome
nonexistent    35551
failure         4252
success         1373
Name: count, dtype: int64
```




TARGET COLUMN

```
x='y'
temp=df[x].value_counts()
plt.figure(figsize=(5,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
print(temp)
```

```
y
no     36537
yes     4639
Name: count, dtype: int64
```



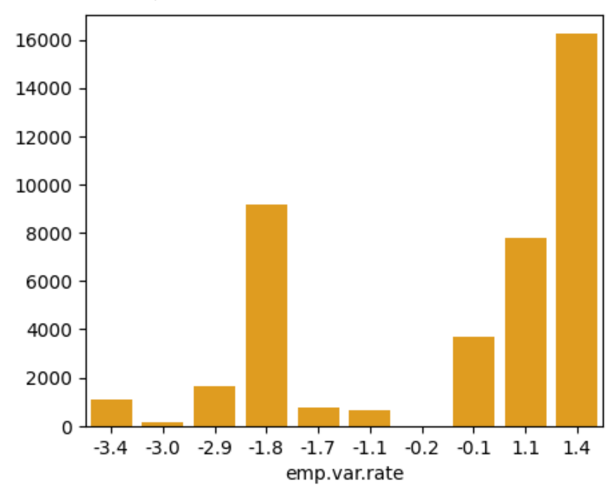

More People Have rejected term loan than people accepted

Data is Uneven we will manage this by techniques like stratified cv

EMPLOYEE VARIABILITY RATE

```
x='emp.var.rate'
temp=df[x].value_counts()
plt.figure(figsize=(5,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
print(temp)
```

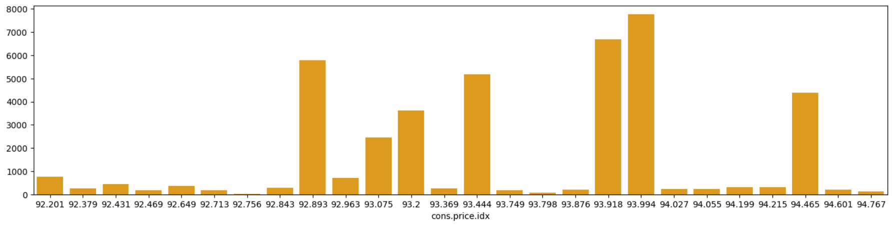
```
emp.var.rate
 1.4    16228
-1.8     9182
 1.1     7762
-0.1     3682
-2.9     1662
-3.4     1070
-1.7      773
-1.1      635
-3.0      172
-0.2       10
Name: count, dtype: int64
```




CONSUMER PRICE INDEX

```
x='cons.price.idx'
temp=df[x].value_counts()
plt.figure(figsize=(18,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
```

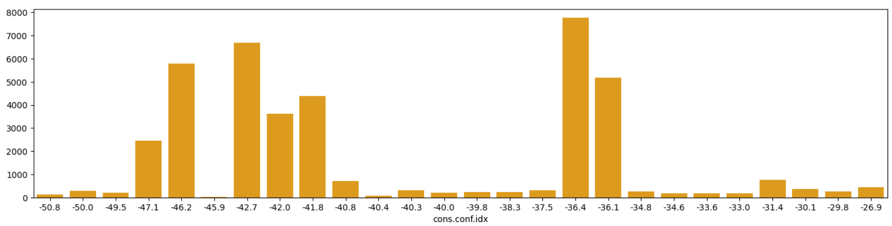
```
<Axes: xlabel='cons.price.idx'>
```



CONSUMER CONFIDENCE INDEX

```
x='cons.conf.idx'
temp=df[x].value_counts()
plt.figure(figsize=(18,4))
sns.barplot(x=temp.index, y=temp.values,color='orange')
```

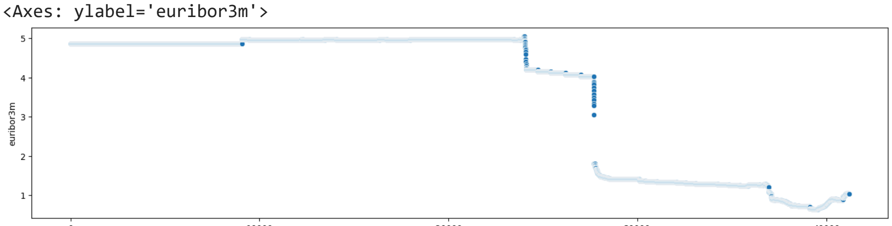
```
<Axes: xlabel='cons.conf.idx'>
```



EUROBOR

```
x='euribor3m'
temp=df[x].value_counts()
plt.figure(figsize=(18,4))
sns.scatterplot(df[x])
```

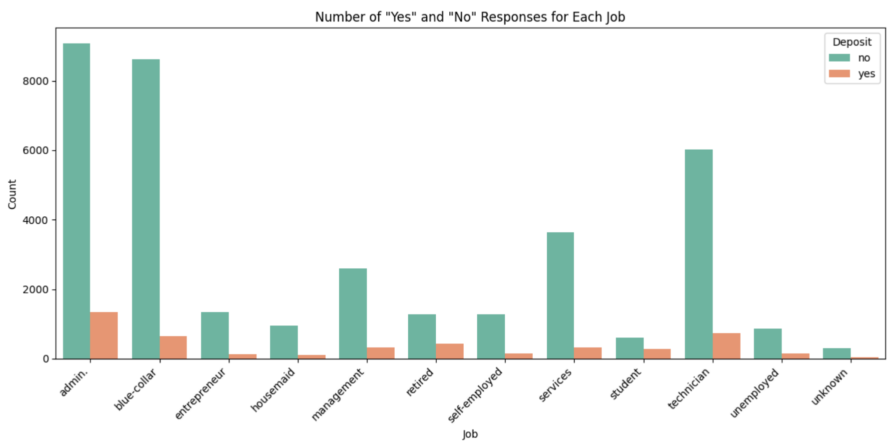
```
<Axes: ylabel='euribor3m'>
```



# BIVARIATE ANALYSIS

JOB--Y

```
# Grouping the DataFrame by 'job' and 'y', then summing the counts
grouped = df.groupby(['job', 'y']).size().unstack(fill_value=0).reset_index()

# Melt the DataFrame to have 'job' as the identifier variable
melted = pd.melt(grouped, id_vars='job', var_name='Response', value_name='Count')

# Plotting grouped bar plot
plt.figure(figsize=(12, 6))
sns.barplot(x='job', y='Count', hue='Response', data=melted, palette='Set2')
plt.title('Number of "Yes" and "No" Responses for Each Job')
plt.xlabel('Job')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Deposit')
plt.tight_layout()
plt.show()
```




We can say on the basis of data that job profile admin have subscirbed to term-deposit more

But Retired People,students and self-employeed people are tend to subscribe more if we see their acceptance ratio

MARITAL--Y

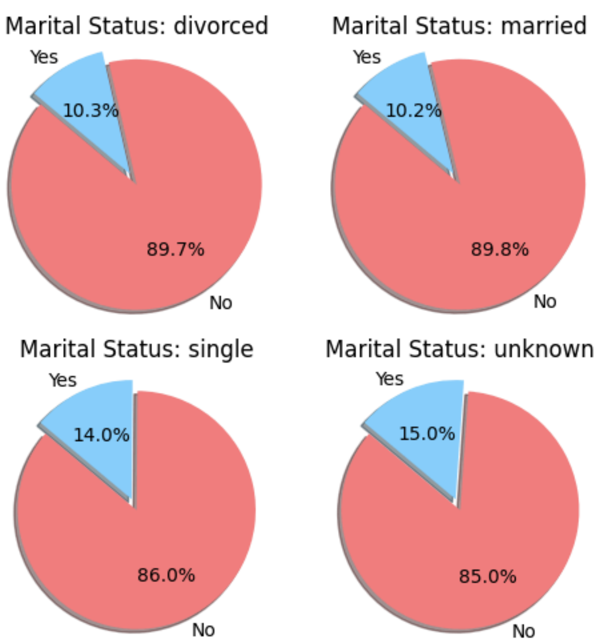
```
gr=df.groupby(['marital', 'y']).size().unstack(fill_value=0).reset_index()
gr
```

| y | marital | no | yes |
|---|---|---|---|
| 0 | divorced | 4135 | 476 |
| 1 | married | 22390 | 2531 |
| 2 | single | 9944 | 1620 |
| 3 | unknown | 68 | 12 |

Next steps: Generate code with gr View recommended plots

divorced,married,single,unknown WITH y

```
import matplotlib.pyplot as plt

# Define colors for the pie slices
colors = ['lightcoral', 'lightskyblue']

# Define explode for the pie slices (to emphasize 'Yes')
explode = (0, 0.1)

labels = ['No', 'Yes']

autopct = '%1.1f%%'
startangle = 140

fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs = axs.flatten()

for i, (_, row) in enumerate(gr.iterrows()):
    counts = [row['no'], row['yes']]

    axs[i].pie(counts, labels=labels, colors=colors, autopct=autopct, startangle=startangle, explode=explode, shadow=True)

    axs[i].set_title(f'Marital Status: {row["marital"]}')
    axs[i].axis('equal')
plt.tight_layout()
plt.show()
```




Based On Data We can say that people with,

Marital Status =Unknown ,Single Have High Subscription ratio

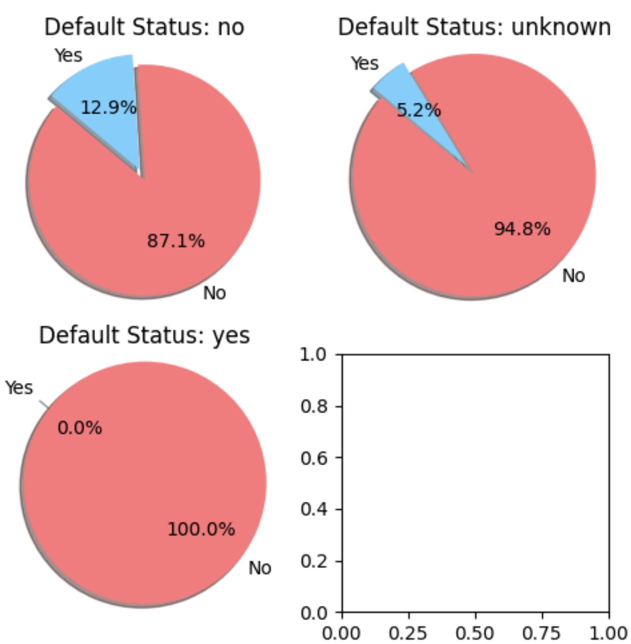
DEFAULT--Y

```
gr1=df.groupby(['default','y']).size().unstack(fill_value=0).reset_index()
gr1
```

| y | default | no | yes |
|---|---|---|---|
| 0 | no | 28381 | 4196 |
| 1 | unknown | 8153 | 443 |
| 2 | yes | 3 | 0 |

Next steps: Generate code with gr1 View recommended plots

```
import matplotlib.pyplot as plt

# Define colors for the pie slices
colors = ['lightcoral', 'lightskyblue']

# Define explode for the pie slices (to emphasize 'Yes')
explode = (0, 0.1)

labels = ['No', 'Yes']

autopct = '%1.1f%%'
startangle = 140

fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs = axs.flatten()

for i, (_, row) in enumerate(gr1.iterrows()):
    counts = [row['no'], row['yes']]

    axs[i].pie(counts, labels=labels, colors=colors, autopct=autopct, startangle=startangle, explode=explode, shadow=True)

    axs[i].set_title(f'Default Status: {row["default"]}')
    axs[i].axis('equal')
plt.tight_layout()
plt.show()
```




People who Have no Default are tend to subscribe term deposite more than the people who have defaulted

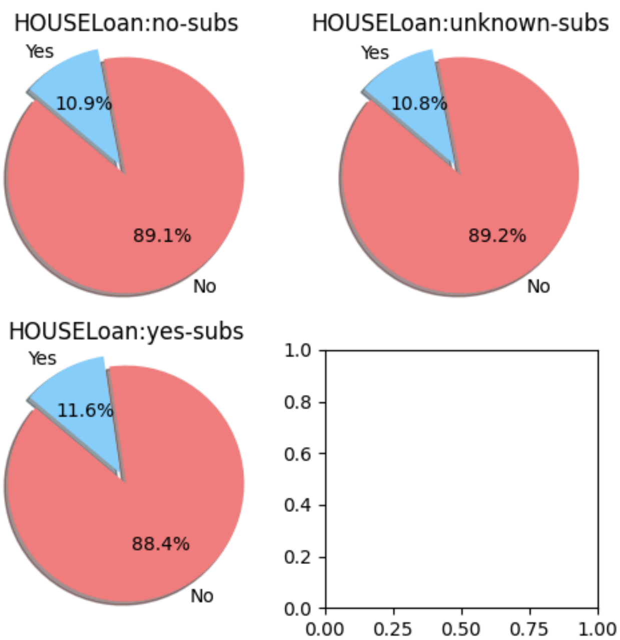
HOOUSING--Y

```
gr2=df.groupby(['housing','y']).size().unstack().reset_index()
gr2
```

| y | housing | no | yes |
|---|---|---|---|
| 0 | no | 16590 | 2025 |
| 1 | unknown | 883 | 107 |
| 2 | yes | 19064 | 2507 |

Next steps: Generate code with gr2 View recommended plots

```
import matplotlib.pyplot as plt

# Define colors for the pie slices
colors = ['lightcoral', 'lightskyblue']

# Define explode for the pie slices (to emphasize 'Yes')
explode = (0, 0.1)

labels = ['No', 'Yes']

autopct = '%1.1f%%'
startangle = 140

fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs = axs.flatten()

for i, (_, row) in enumerate(gr2.iterrows()):
    counts = [row['no'], row['yes']]

    axs[i].pie(counts, labels=labels, colors=colors, autopct=autopct, startangle=startangle, explode=explode, shadow=True)

    axs[i].set_title(f'HOUSELoan:{row["housing"]}-subs')
    axs[i].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

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




There is no or little relationship between housing loan and term deposit

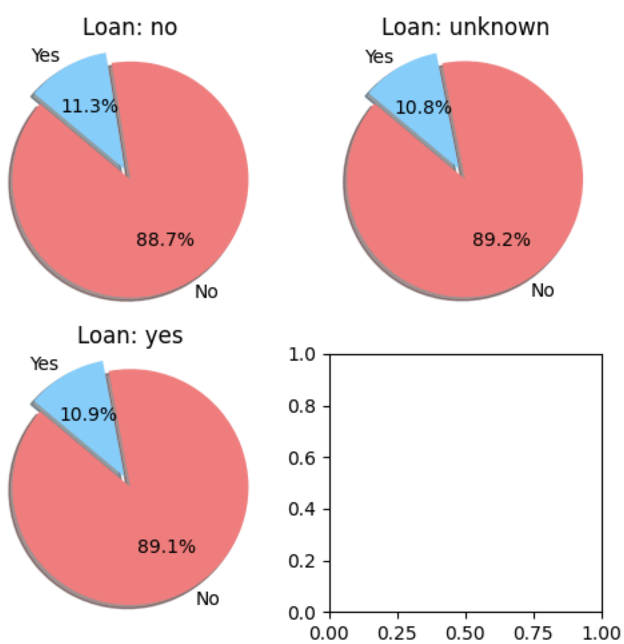
LOAN--Y

```
gr3=df.groupby(['loan','y']).size().unstack().reset_index()
gr3
```

| y | loan | no | yes |
|---|---|---|---|
| 0 | no | 30089 | 3849 |
| 1 | unknown | 883 | 107 |
| 2 | yes | 5565 | 683 |

Next steps: Generate code with gr3 View recommended plots

```
import matplotlib.pyplot as plt

# Define colors for the pie slices
colors = ['lightcoral', 'lightskyblue']

# Define explode for the pie slices (to emphasize 'Yes')
explode = (0, 0.1)

# Define labels for the pie slices
labels = ['No', 'Yes']

autopct = '%1.1f%%'
startangle = 140

# Plotting pie charts
fig, axs = plt.subplots(2, 2, figsize=(5, 5))
axs = axs.flatten()

for i, (_, row) in enumerate(gr3.iterrows()):
    counts = [row['no'], row['yes']]

    axs[i].pie(counts, labels=labels, colors=colors, autopct=autopct, startangle=startangle, explode=explode, shadow=True)

    axs[i].set_title(f'Loan: {row["loan"]}')
    axs[i].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

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




No Conclusion Can be drawn as people with loan ,no loan and unknown all are subscribing term deposite with same ratio

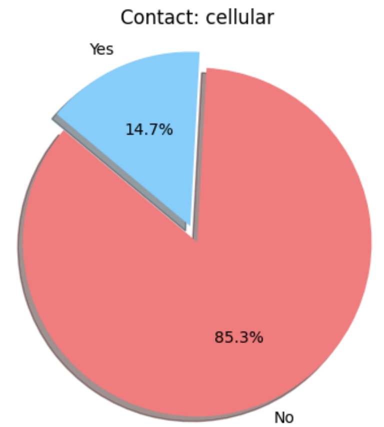
CONTACT--Y

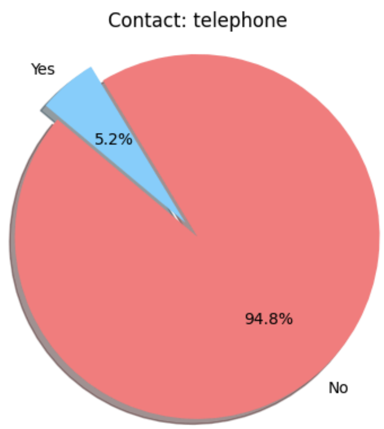
```
gr4=df.groupby(['contact','y']).size().unstack().reset_index()
gr4
```

| y | contact | no | yes |
|---|---|---|---|
| 0 | cellular | 22283 | 3852 |
| 1 | telephone | 14254 | 787 |

Next steps: Generate code with gr4 View recommended plots

```
import matplotlib.pyplot as plt

# Define colors for the pie slices
colors = ['lightcoral', 'lightskyblue']

# Define explode for the pie slices (to emphasize 'Yes')
explode = (0, 0.1)

# Define labels for the pie slices
labels = ['No', 'Yes']

autopct = '%1.1f%%'
startangle = 140

fig, axs = plt.subplots(2, 1, figsize=(8, 8))
axs = axs.flatten()

for i, (_, row) in enumerate(gr4.iterrows()):
    counts = [row['no'], row['yes']]

    axs[i].pie(counts, labels=labels, colors=colors, autopct=autopct, startangle=startangle, explode=explode, shadow=True)

    axs[i].set_title(f'Contact: {row["contact"]}')
    axs[i].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

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




People Who Are Contacted through cellular medium are tend to subscribe for term deposite more than people contacted throug telephone

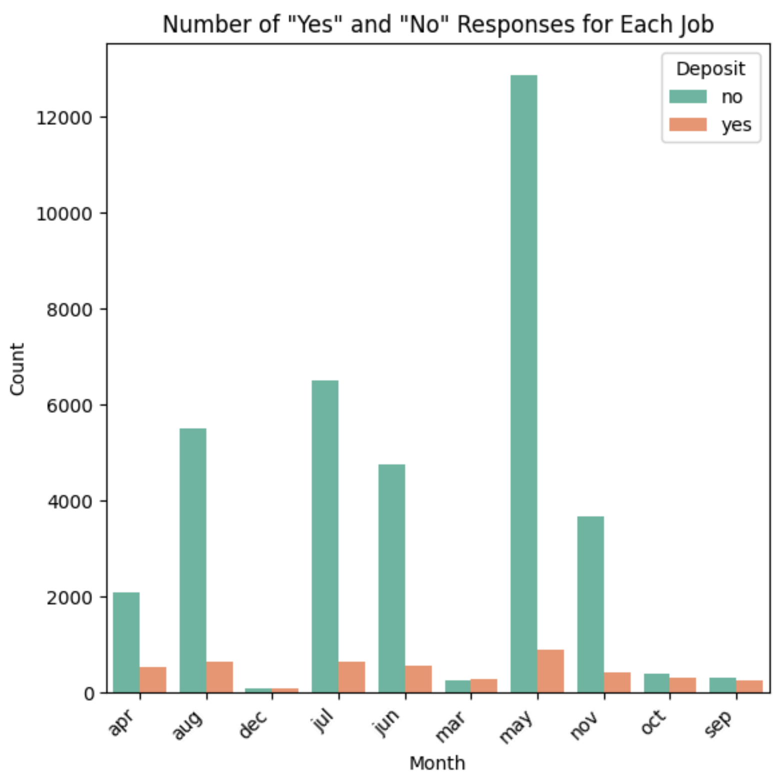
MONTH--Y

```
gr5=df.groupby(['month','y']).size().unstack().reset_index()
gr5
```

| y | month | no | yes |
|---|---|---|---|
| 0 | apr | 2092 | 539 |
| 1 | aug | 5521 | 655 |
| 2 | dec | 93 | 89 |
| 3 | jul | 6521 | 648 |
| 4 | jun | 4759 | 559 |
| 5 | mar | 270 | 276 |
| 6 | may | 12881 | 886 |
| 7 | nov | 3684 | 416 |
| 8 | oct | 402 | 315 |
| 9 | sep | 314 | 256 |

Next steps: Generate code with gr5 View recommended plots

```
melted = pd.melt(gr5, id_vars='month', var_name='Response', value_name='Count')
melted['percent']=round(melted['Count']*100/melted.groupby('month')['Count'].transform('sum'),1)
melted['percent']=melted['percent'].apply(lambda x: '{}%'.format(x))
# Plotting grouped bar plot
plt.figure(figsize=(6, 6))
sns.barplot(x='month', y='Count', hue='Response', data=melted, palette='Set2')
plt.title('Number of "Yes" and "No" Responses for Each Job')
plt.xlabel('Month')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Deposit')
plt.tight_layout()
plt.show()
```

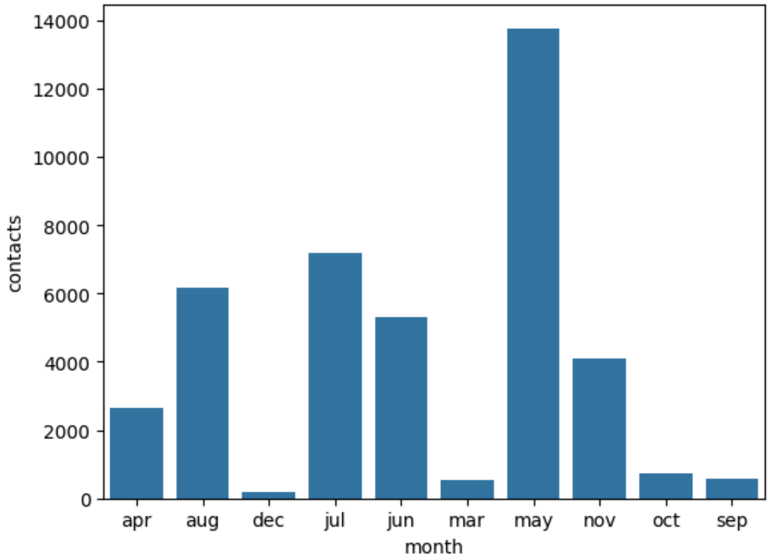



There is more subscription in month of may

But There is good subscription in oct,sep,april,june,july in term of no and yes ratio

```
# Grouping by 'month' and 'contact', then getting value counts
contact_counts = df.groupby(['month', 'contact']).size()

# Resetting the index to make 'month' and 'contact' as columns
contact_counts = contact_counts.reset_index(name='count')

# Summing 'count' for each month
monthly_contact_sum = contact_counts.groupby('month')['count'].sum()

new_df = pd.DataFrame(monthly_contact_sum)

new_df = new_df.rename(columns={'count': 'contacts'})

sns.barplot(x='month',y='contacts',data=new_df)
```

```
<Axes: xlabel='month', ylabel='contacts'>
```




We Can See That Number Of Contact In oct,sep,april,june,july if we increase our contact to people in these month then bank can get high subscription as in these month rejection is less as compared to other months

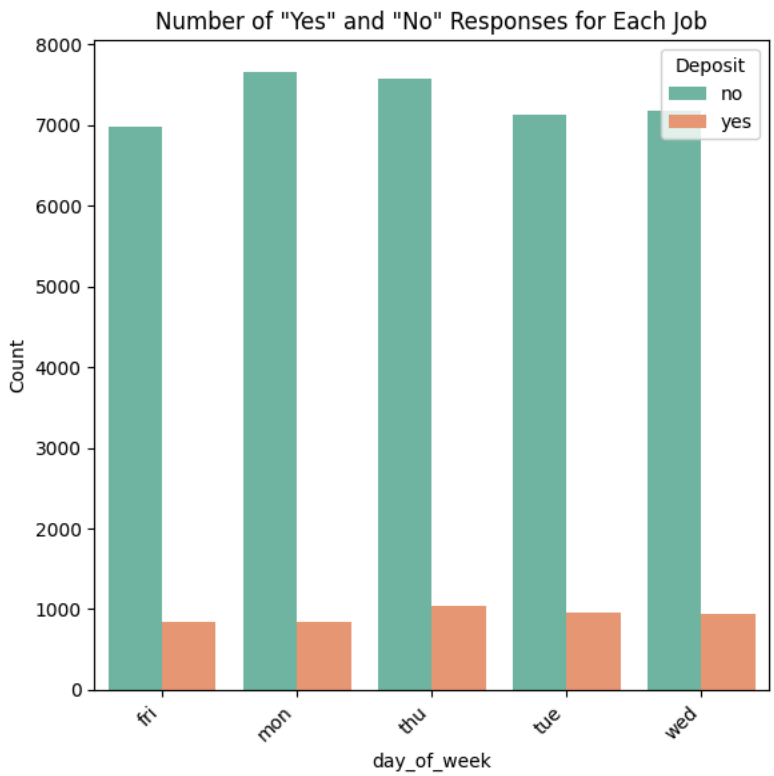
DAY OF WEEK---Y

```
gr6=df.groupby(['day_of_week','y']).size().unstack().reset_index()
gr6
```

| y | day_of_week | no | yes |
|---|---|---|---|
| 0 | fri | 6980 | 846 |
| 1 | mon | 7665 | 847 |
| 2 | thu | 7574 | 1044 |
| 3 | tue | 7133 | 953 |
| 4 | wed | 7185 | 949 |

Next steps: Generate code with gr6 View recommended plots

```
melted = pd.melt(gr6, id_vars='day_of_week', var_name='Response', value_name='Count')
# Plotting grouped bar plot
plt.figure(figsize=(6, 6))
sns.barplot(x='day_of_week', y='Count', hue='Response', data=melted, palette='Set2')
plt.title('Number of "Yes" and "No" Responses for Each Job')
plt.xlabel('day_of_week')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Deposit')
plt.tight_layout()
plt.show()
```

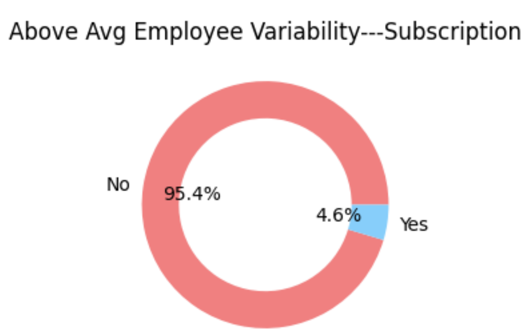



People tend to Subscribe more on thursday

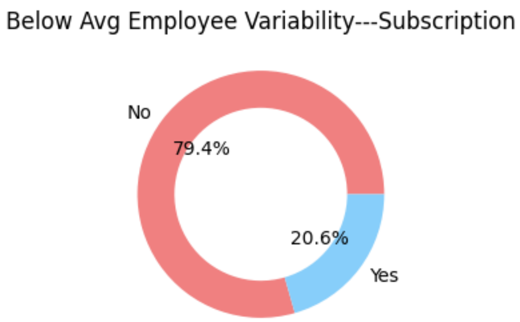
EMPLOYEE VARIABILTY RATE --Y

```
m=df['emp.var.rate'].mean()
above_avg=df[df['emp.var.rate']>m]
av=above_avg.groupby('y').size()
below_avg=df[df['emp.var.rate']<m]
bv=below_avg.groupby('y').size()
```

```
plt.figure(figsize=(3,3))
plt.pie(av, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors);
plt.title('Above Avg Employee Variability---Subscription');
```




```
plt.figure(figsize=(3,3))
plt.pie(bv, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Below Avg Employee Variability---Subscription');
```



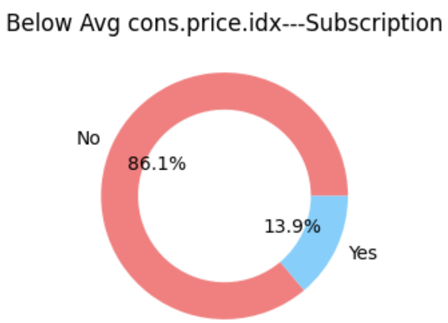

When There is High Employee Variability Subscription Is Less

When There is Low Employee Variability Subscription Is More

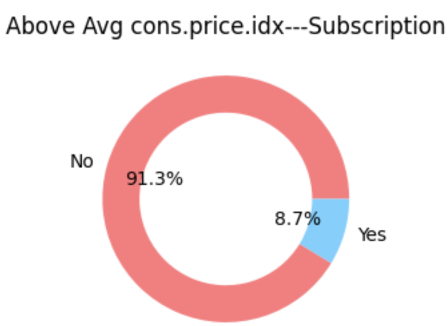
CONSUMER PRICE INDEX----Y

```
m=df['cons.price.idx'].mean()
above_avg=df[df['cons.price.idx']>m]
av=above_avg.groupby('y').size()
below_avg=df[df['cons.price.idx']<m]
bv=below_avg.groupby('y').size()
```

```
plt.figure(figsize=(3,3))
plt.pie(bv, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Below Avg cons.price.idx---Subscription');
```




```
plt.figure(figsize=(3,3))
plt.pie(av, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Above Avg cons.price.idx---Subscription');
```



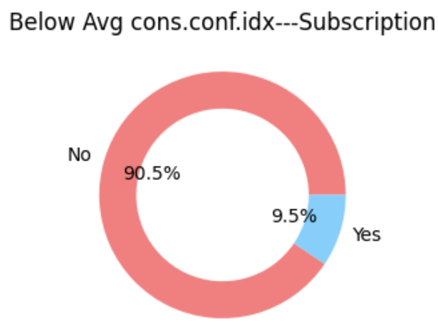

We Can See When Consumer Price Index is Below Avg Subscription Increases

We Can See When Consumer Price Index is Above Avg Subscription Decreases

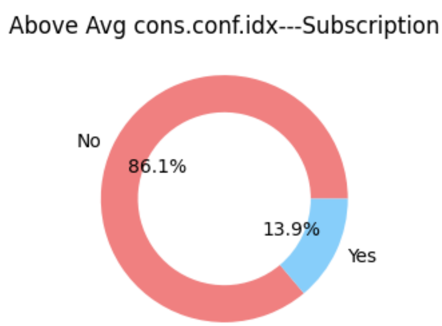
CONSUMER CONFIDENCE INDEX --Y

```
m=df['cons.conf.idx'].mean()
above_avg=df[df['cons.conf.idx']>m]
av=above_avg.groupby('y').size()
below_avg=df[df['cons.conf.idx']<m]
bv=below_avg.groupby('y').size()
```

```
plt.figure(figsize=(3,3))
plt.pie(bv, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Below Avg cons.conf.idx---Subscription');
```




```
plt.figure(figsize=(3,3))
plt.pie(av, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Above Avg cons.conf.idx---Subscription');
```



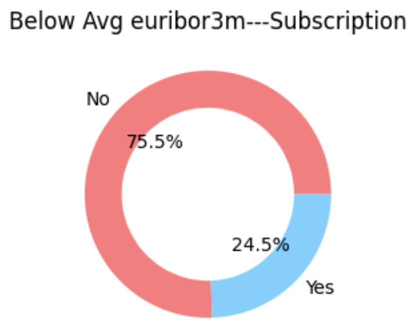

We can see that when consumer confidence index is above avg subscription increases

We can see that when consumer confidence index is below avg subscription decreases

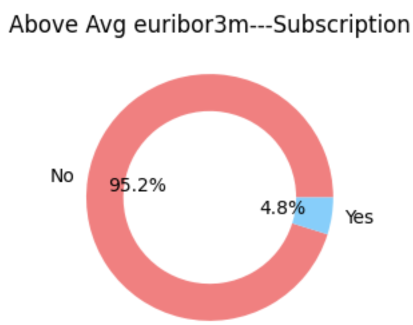
EURO BOR--Y

```
m=df['euribor3m'].mean()
above_avg=df[df['euribor3m']>m]
av=above_avg.groupby('y').size()
below_avg=df[df['euribor3m']<m]
bv=below_avg.groupby('y').size()
```

```
plt.figure(figsize=(3,3))
plt.pie(bv, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Below Avg euribor3m---Subscription');
```




```
plt.figure(figsize=(3,3))
plt.pie(av, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Above Avg euribor3m---Subscription');
```




We can see when the euribor is below avg people tends to subscribe term deposit more than when euribor is above avg..

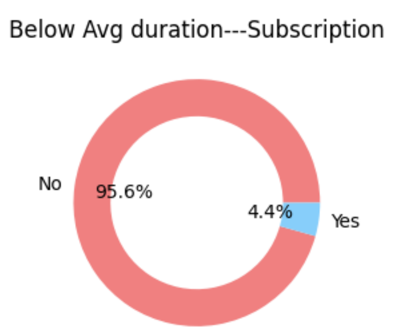
**But Why So???**

Low EURIBOR rates may lead to lower borrowing costs for banks, which could incentivize them to offer more favorable terms on term deposits to attract funding. Banks may also use term deposits as a source of funding for their lending activities, offering competitive rates to attract depositors.

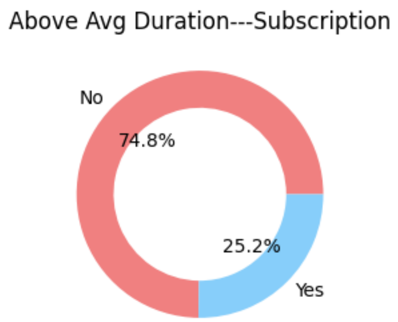
DURATION--Y

```
m=df['duration'].mean()
above_avg=df[df['duration']>m]
av=above_avg.groupby('y').size()
below_avg=df[df['duration']<m]
bv=below_avg.groupby('y').size()
```

```
plt.figure(figsize=(3,3))
plt.pie(bv, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Below Avg duration---Subscription');
```



```
plt.figure(figsize=(3,3))
plt.pie(av, labels=labels, autopct='%1.1f%%', wedgeprops=dict(width=0.3), colors=colors)
plt.title('Above Avg Duration---Subscription');
```



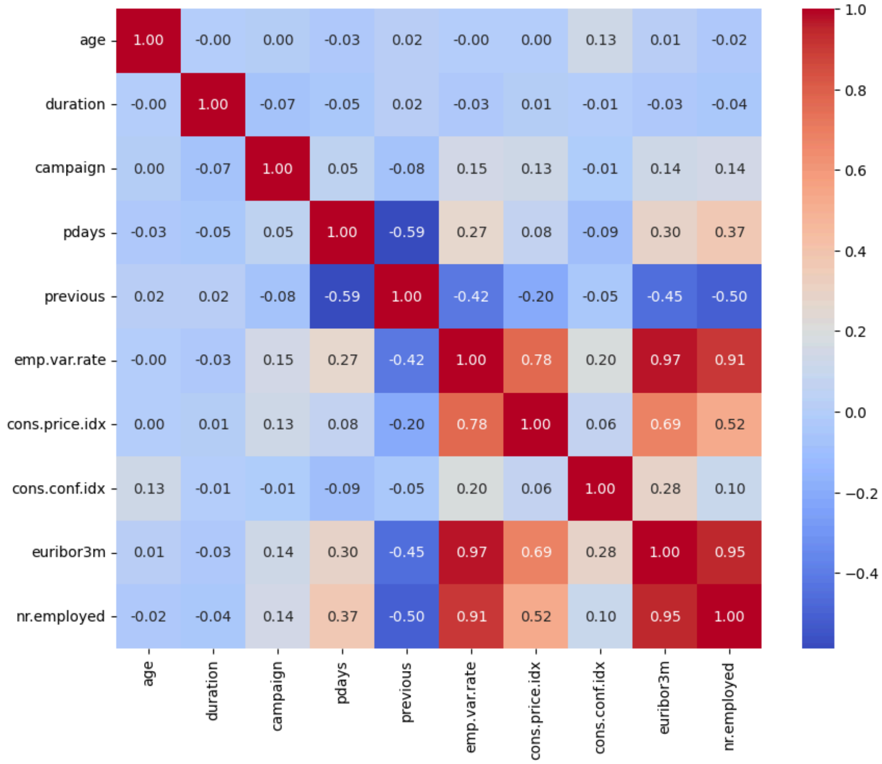

We can Clearly see that there is hogh correlation between duration and Y column

As Duration Increases People Subscription Increases

***But***, we cannot use duration column in our prediction model as we have to predict the outcome will people subscribe to term deposite if yes then only we are going to call them Hence,we will drop duration column while building predictive models

HEATMAP

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

# Select only numeric columns
numeric_df = df.select_dtypes(include=['number'])

# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```




```
print('int_cols:',int_cols)
print('obj_cols:',obj_cols)
print('flo_cols:',flo_cols)
```

```
int_cols: ['age', 'duration', 'campaign', 'pdays', 'previous']
obj_cols: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'y']
flo_cols: ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
```

### ONE HOT ENCODING THE NOMINAL CATEGORICAL COLUMNS

```
columns_categorical = ['job', 'marital',
       'education', 'default',
       'housing', 'loan', 'contact', 'month',
       'day_of_week', 'poutcome']
```

```
df_2=pd.DataFrame(df)
```

```
df_2.head()
```

| | age | job | marital | education | default | housing | loan | contact | month | day_of_\ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 56 | housemaid | married | basic.4y | no | no | no | telephone | may | |
| **1** | 57 | services | married | high.school | unknown | no | no | telephone | may | |
| **2** | 37 | services | married | high.school | no | yes | no | telephone | may | |
| **3** | 40 | admin. | married | basic.6y | no | no | no | telephone | may | |
| **4** | 56 | services | married | high.school | no | no | yes | telephone | may | |

```
# Apply one-hot encoding using pandas.get_dummies()
encoded_df = pd.get_dummies(df[columns_categorical],drop_first=False,dtype=int)
```

```
categorical_enc=list(encoded_df.columns)
categorical_enc
```

```
['job_admin.',
 'job_blue-collar',
 'job_entrepreneur',
 'job_housemaid',
 'job_management',
 'job_retired',
 'job_self-employed',
 'job_services',
 'job_student',
 'job_technician',
 'job_unemployed',
 'job_unknown',
 'marital_divorced',
 'marital_married',
 'marital_single',
 'marital_unknown',
 'education_basic.4y',
 'education_basic.6y',
 'education_basic.9y',
 'education_high.school',
 'education_illiterate',
 'education_professional.course',
 'education_university.degree',
 'education_unknown',
 'default_no',
 'default_unknown',
 'default_yes',
 'housing_no',
 'housing_unknown',
 'housing_yes',
 'loan_no',
 'loan_unknown',
 'loan_yes',
 'contact_cellular',
 'contact_telephone',
 'month_apr',
 'month_aug',
 'month_dec',
 'month_jul',
 'month_jun',
 'month_mar',
 'month_may',
 'month_nov',
 'month_oct',
 'month_sep',
 'day_of_week_fri',
 'day_of_week_mon',
 'day_of_week_thu',
 'day_of_week_tue',
 'day_of_week_wed',
 'poutcome_failure',
 'poutcome_nonexistent',
 'poutcome_success']
```

```
categorical=list(columns_categorical)
```

```
all_columns=list(df.columns)
```

```
numerical_columns = [col for col in all_columns if col not in categorical]
```

```
numerical_columns
```

```
['age',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
```

```
 'nr.employed',
 'y']
```

Removing duration as it cannot be input column and y coz it is categorical column

```
# numerical_columns.remove('duration')
```

```
numerical_columns
```

```
['age',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
 'nr.employed',
 'y']
```

Label Encoding Y column

```
from sklearn.preprocessing import LabelEncoder

# Assuming 'column_to_encode' is the column you want to encode
column_to_encode = 'y'

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the column
df[column_to_encode + '_encoded'] = label_encoder.fit_transform(df[column_to_encode])

# Now df[column_to_encode + '_encoded'] contains the encoded values
```

```
df.head(1)
```

| | age | job | marital | education | default | housing | loan | contact | month | day_of_w |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 56 | housemaid | married | basic.4y | no | no | no | telephone | may | r |

```
df = pd.concat([df,encoded_df], axis = 1)
```

```
df.head()
```

| | age | job | marital | education | default | housing | loan | contact | month | day_of_w |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 56 | housemaid | married | basic.4y | no | no | no | telephone | may | |
| 1 | 57 | services | married | high.school | unknown | no | no | telephone | may | |
| 2 | 37 | services | married | high.school | no | yes | no | telephone | may | |
| 3 | 40 | admin. | married | basic.6y | no | no | no | telephone | may | |
| 4 | 56 | services | married | high.school | no | no | yes | telephone | may | |

```
numerical_columns.remove('y')
```

```
input_col=numerical_columns+categorical_enc
```

```
df_enc=df[input_col+['y_encoded']]
```

```
df_enc.shape
```

```
(41176, 64)
```

WE WILL DO 2 APPROACHES AS DATA IS IMBALANCED

1)Stratified Train Valid Test Split & Stratified CV

2)UNDER SAMPLING AND THEN PERFORMING CV

```
X=df_enc.drop(columns=['y_encoded'])
y=df_enc['y_encoded'].values
```

```
print(X.shape)
print(y.shape)
```

```
(41176, 63)
(41176,)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
# Then, split the test set into validation and test sets (50% each)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
X_train_scaled=scaler.fit_transform(X_train)
X_valid_scaled=scaler.transform(X_valid)
X_test_scaled=scaler.transform(X_test)
```

```
X_train=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_valid=pd.DataFrame(X_valid_scaled,columns=X_valid.columns)
X_test=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```

## Model Selection

```
from sklearn.metrics import confusion_matrix, roc_curve, auc

def calculate_classification_metrics(y_true, y_pred, threshold):
    """
    Calculate classification metrics including accuracy, precision, recall, F1-score, and AUC.

    Parameters:
    - y_true (array-like): True labels.
    - y_pred (array-like): Predicted probabilities or scores.
    - threshold (float): Threshold value for binary classification (default=0.5).

    Returns:
    - metrics (dict): A dictionary containing classification metrics.
    """
    # Convert predicted probabilities or scores to binary predictions based on the threshold
    y_pred_binary = (y_pred >= threshold).astype(int)

    accuracy = accuracy_score(y_true, y_pred_binary)
    precision = precision_score(y_true, y_pred_binary)
    recall = recall_score(y_true, y_pred_binary)
    f1 = f1_score(y_true, y_pred_binary)
    confusion = confusion_matrix(y_true, y_pred_binary)

    # Calculate AUC
    fpr, tpr, _ = roc_curve(y_true, y_pred)
    auc_value = auc(fpr, tpr)  # Use the auc function from sklearn.metrics

    metrics = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1,
        'Confusion Matrix': confusion,
        'AUC': auc_value
    }

    return metrics
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

## Logistic Regression

```
lr=LogisticRegression(random_state = 42)
lr.fit(X_train, y_train)
```

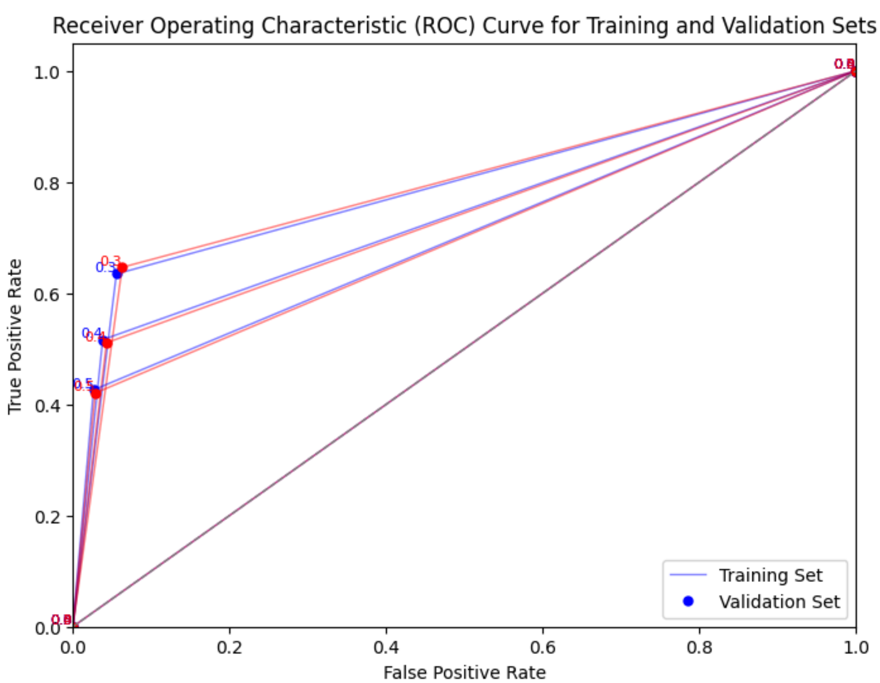
```
▾        LogisticRegression
LogisticRegression(random_state=42)
```

```
y_train_preds = lr.predict_proba(X_train)[:,1]
y_valid_preds = lr.predict_proba(X_valid)[:,1]
```

```
y_train_preds
```

```
array([0.15812684, 0.4709308 , 0.01367171, ..., 0.0957597 , 0.01051456,
       0.01699796])
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

thresholds = [0, 0.3, 0.4, 0.5]
plt.figure(figsize=(8, 6))

for threshold in thresholds:
    # Plot ROC curve for training set
    y_train_pred = (y_train_preds >= threshold).astype(int)
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
    plt.plot(fpr_train, tpr_train, lw=1, color='blue', alpha=0.5)
    plt.plot(fpr_train, tpr_train, 'o', color='blue', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_train, tpr_train):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='blue', fontsize=8, ha='right', va='bottom')

    # Plot ROC curve for validation set
    y_valid_pred = (y_valid_preds >= threshold).astype(int)
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, y_valid_pred)
    plt.plot(fpr_valid, tpr_valid, lw=1, color='red', alpha=0.5)
    plt.plot(fpr_valid, tpr_valid, 'o', color='red', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_valid, tpr_valid):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='red', fontsize=8, ha='right', va='bottom')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Training and Validation Sets')
plt.legend(['Training Set', 'Validation Set'], loc="lower right")
plt.show()
```




```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score
```

```
print('Training:',calculate_classification_metrics(y_train, y_train_preds, threshold=0.3))
print('Validation:',calculate_classification_metrics(y_valid, y_valid_preds, threshold=0.3))
```

```
Training: {'Accuracy': 0.9095324833029751, 'Precision': 0.5918572505654687, 'Recall': 0.6345998383185125, 'F1-score': 0.612483745123537
       [ 1356,  2355]]), 'AUC': 0.9356512676698363}
Validation: {'Accuracy': 0.9043224866440019, 'Precision': 0.5660377358490566, 'Recall': 0.646551724137931, 'F1-score': 0.60362173038229
       [ 164,  300]]), 'AUC': 0.938864234754544}
```

## KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors = 150)
knn.fit(X_train, y_train)
```

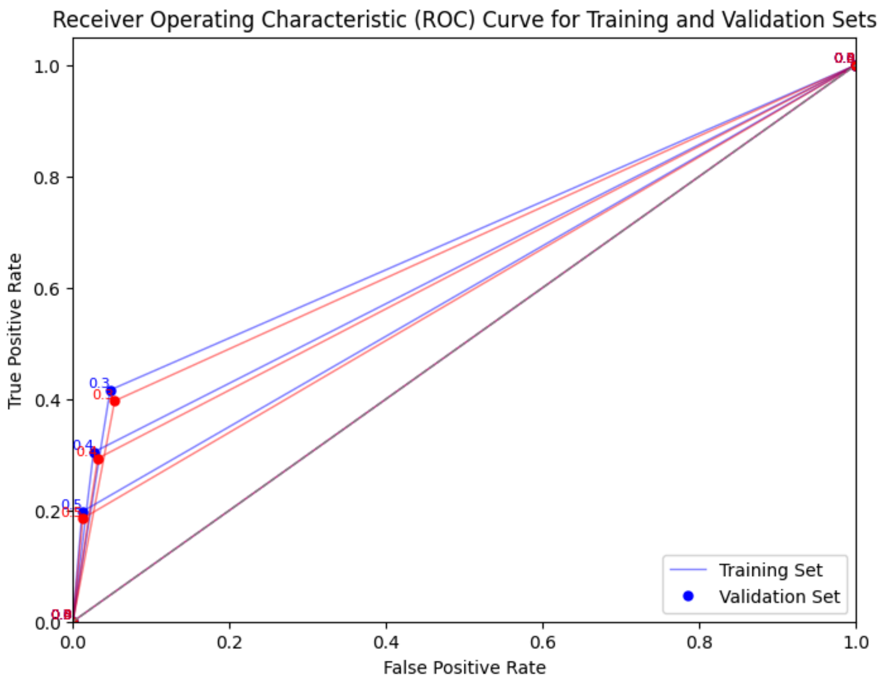
```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=150)
```

```
y_train_predsknn = knn.predict_proba(X_train)[:,1]
y_valid_predsknn = knn.predict_proba(X_valid)[:,1]
```

```
print('Training:',calculate_classification_metrics(y_train, y_train_predsknn, threshold=0.3))
print('Validation:',calculate_classification_metrics(y_valid, y_valid_predsknn, threshold=0.3))
```

```
Training: {'Accuracy': 0.8918943533697632, 'Precision': 0.5280059746079163, 'Recall': 0.3810293721368903, 'F1-score': 0.4426357802473,
       [ 2297,  1414]]), 'AUC': 0.8989852143591607}
Validation: {'Accuracy': 0.8878096163186012, 'Precision': 0.5028571428571429, 'Recall': 0.3793103448275862, 'F1-score': 0.4324324324324
       [ 288,  176]]), 'AUC': 0.9005314794368005}
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

thresholds = [0, 0.3, 0.4, 0.5]
plt.figure(figsize=(8, 6))

for threshold in thresholds:
    # Plot ROC curve for training set
    y_train_pred = (y_train_predsknn >= threshold).astype(int)
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
    plt.plot(fpr_train, tpr_train, lw=1, color='blue', alpha=0.5)
    plt.plot(fpr_train, tpr_train, 'o', color='blue', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_train, tpr_train):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='blue', fontsize=8, ha='right', va='bottom')

    # Plot ROC curve for validation set
    y_valid_pred = (y_valid_predsknn >= threshold).astype(int)
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, y_valid_pred)
    plt.plot(fpr_valid, tpr_valid, lw=1, color='red', alpha=0.5)
    plt.plot(fpr_valid, tpr_valid, 'o', color='red', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_valid, tpr_valid):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='red', fontsize=8, ha='right', va='bottom')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Training and Validation Sets')
plt.legend(['Training Set', 'Validation Set'], loc="lower right")
plt.show()
```



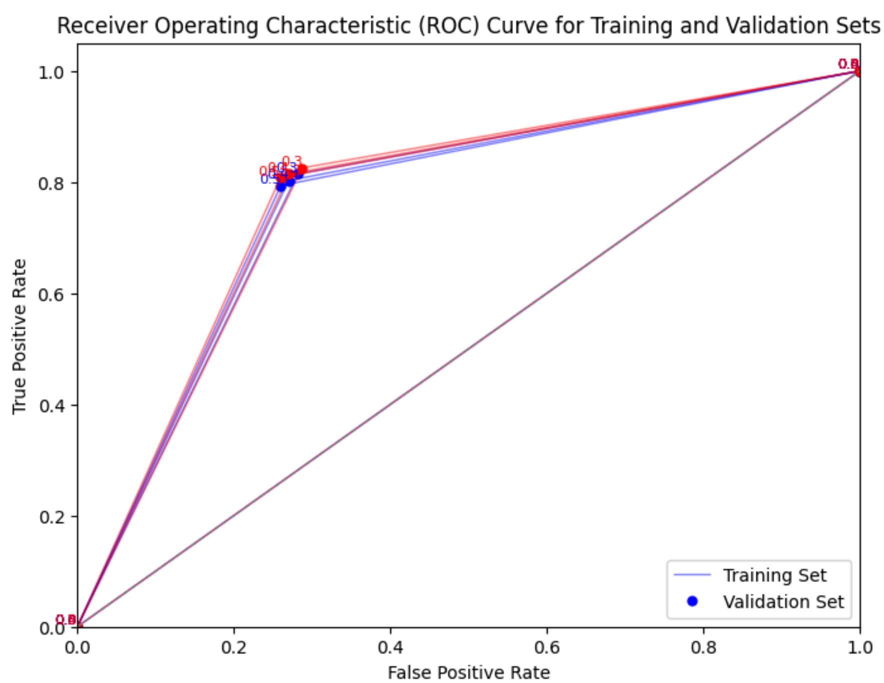

## GaussianNB

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_train_predsNb = nb.predict_proba(X_train)[:,1]
y_valid_predsNb = nb.predict_proba(X_valid)[:,1]
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

thresholds = [0, 0.3, 0.4, 0.5]
plt.figure(figsize=(8, 6))

for threshold in thresholds:
    # Plot ROC curve for training set
    y_train_pred = (y_train_predsNb >= threshold).astype(int)
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
    plt.plot(fpr_train, tpr_train, lw=1, color='blue', alpha=0.5)
    plt.plot(fpr_train, tpr_train, 'o', color='blue', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_train, tpr_train):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='blue', fontsize=8, ha='right', va='bottom')

    # Plot ROC curve for validation set
    y_valid_pred = (y_valid_predsNb >= threshold).astype(int)
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, y_valid_pred)
    plt.plot(fpr_valid, tpr_valid, lw=1, color='red', alpha=0.5)
    plt.plot(fpr_valid, tpr_valid, 'o', color='red', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_valid, tpr_valid):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='red', fontsize=8, ha='right', va='bottom')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Training and Validation Sets')
plt.legend(['Training Set', 'Validation Set'], loc="lower right")
plt.show()
```




```
print('Training:',calculate_classification_metrics(y_train, y_train_predsNb, threshold=0.5))
print('Validation:',calculate_classification_metrics(y_valid, y_valid_predsNb, threshold=0.5))
```

```
Training: {'Accuracy': 0.7460837887067395, 'Precision': 0.2793322583704828, 'Recall': 0.7935866343303691, 'F1-score': 0.413217342500350
       [  766,  2945]]), 'AUC': 0.8301500175824722}
Validation: {'Accuracy': 0.7479358912093249, 'Precision': 0.28323262839879154, 'Recall': 0.8081896551724138, 'F1-score': 0.419463087248
       [   89,   375]]), 'AUC': 0.8331773281052413}
```

```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score
def calc_specificity(y_actual, y_pred, thresh):
    # calculates specificity
    return sum((y_pred < thresh) & (y_actual == 0)) /sum(y_actual ==0)

def print_report(y_actual, y_pred, thresh):

    auc = roc_auc_score(y_actual, y_pred)
    accuracy = accuracy_score(y_actual, (y_pred > thresh))
    recall = recall_score(y_actual, (y_pred > thresh))
    precision = precision_score(y_actual, (y_pred > thresh))
    specificity = calc_specificity(y_actual, y_pred, thresh)
    f1 = 2 * (precision * recall) / (precision + recall)

    print('AUC:%.3f'%auc)
    print('accuracy:%.3f'%accuracy)
    print('recall:%.3f'%recall)
    print('precision:%.3f'%precision)
    print('specificity:%.3f'%specificity)
    print('f1:%.3f'%f1)
    print(' ')
    return auc, accuracy, recall, precision, specificity, f1
```

**As We Have Done Strattified Split Based on The Percentage of target** -As Data Naturally Have 0[negative] instances more therefore auccuracy will enevtually grow as

**1)Accuracy=Total Correct Predicted Label/Total no. of label**

> As data have dominance in 0 labels most of the labels will be classified as correct predicted which enventually increases accuracy

> As we can see it in Logistic Regression 1 class labels have high probability and 0 class label have low probaility

> if we increase the threshold it will predict more 0 than 1 and if we decrease the threshold to ex:0.01 then it will predict more 1's than 0

> Threshold 0.3 is good

## Hence If we do Stratified Split Using Logistic regression will be helpful

Using stratified train-test split on an imbalanced dataset can still be beneficial, especially if you're primarily concerned with evaluating your model's performance accurately. Here's how it can be helpful:

**Unbiased Evaluation:**

Stratified train-test split ensures that the class distribution in both the training and testing sets resembles that of the original dataset. This prevents the testing set from being skewed towards the majority class, which could lead to overestimating the model's performance, especially if the minority class is of particular interest.

**Generalization Performance:**

By maintaining the class distribution in the testing set, you get a more realistic estimate of how well your model will perform on unseen data, which is crucial for assessing its generalization performance. This is important regardless of class imbalance because you want your model to perform well on all classes, not just the majority one.

**Robustness to Changes:**

Ensuring that the class distribution is maintained in the testing set makes your evaluation more robust to changes in the underlying data distribution. If your dataset changes over time (which is common in real-world scenarios), using a stratified approach helps ensure that your evaluation remains consistent and reliable.

## Now Lets Do Undersampling

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)

# Undersampling the majority class
undersampler = RandomUnderSampler(random_state=42)
X_train, y_train = undersampler.fit_resample(X_train, y_train)

# Check the class distribution after undersampling
print("Class distribution after undersampling:", Counter(y_resampled))
```

```
Class distribution after undersampling: Counter({0: 3668, 1: 3668})
```

```
# Then, split the test set into validation and test sets (50% each)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)
```

```
# Undersampling the majority class for valid
undersampler = RandomUnderSampler(random_state=42)
X_valid, y_valid = undersampler.fit_resample(X_valid, y_valid)

# Check the class distribution after undersampling
print("Class distribution after undersampling:", Counter(y_valid))
```

```
Class distribution after undersampling: Counter({0: 486, 1: 486})
```

```
# Undersampling the majority class for test
undersampler = RandomUnderSampler(random_state=42)
X_test, y_test = undersampler.fit_resample(X_test, y_test)

# Check the class distribution after undersampling
print("Class distribution after undersampling:", Counter(y_test))
```

```
Class distribution after undersampling: Counter({0: 485, 1: 485})
```

```
X_train_scaled=scaler.fit_transform(X_train)
X_valid_scaled=scaler.transform(X_valid)
X_test_scaled=scaler.transform(X_test)
X_train=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_valid=pd.DataFrame(X_valid_scaled,columns=X_valid.columns)
X_test=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```

**Each Split Is Now Balanced**

## Model Selection

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

**Logistic Regression**

```
lr=LogisticRegression(random_state = 42)
lr.fit(X_train, y_train)

y_train_predslr = lr.predict_proba(X_train)[:,1]
y_valid_predslr = lr.predict_proba(X_valid)[:,1]
```

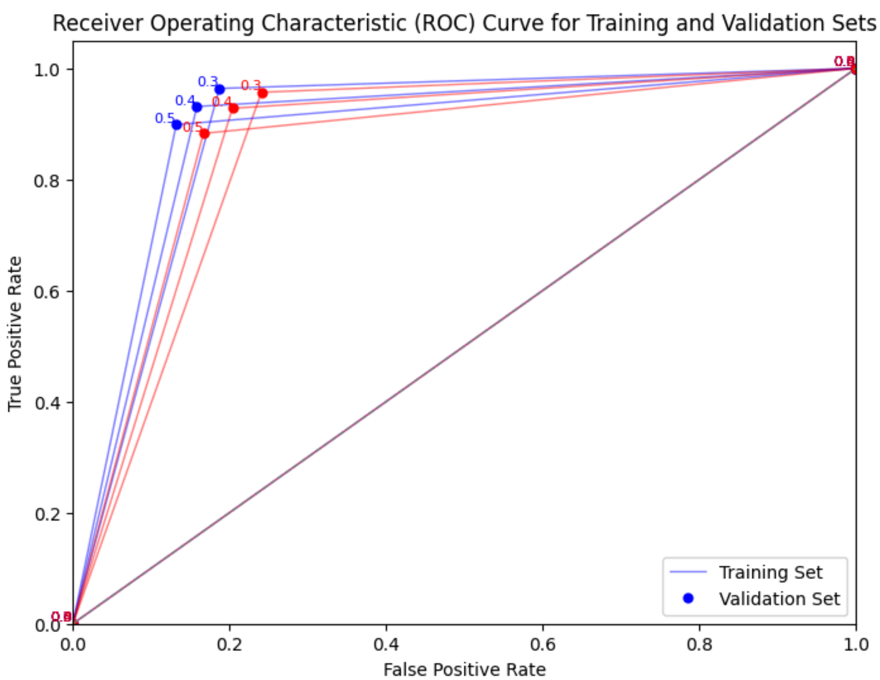
```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score
```

```
print('Training:',calculate_classification_metrics(y_train, y_train_preds, threshold=0.4))
print('Validation:',calculate_classification_metrics(y_valid, y_valid_preds, threshold=0.4))
```

```
Training: {'Accuracy': 0.8653217011995638, 'Precision': 0.8375314861460957, 'Recall': 0.9064885496183206, 'F1-score': 0.870646766169154
       [ 343, 3325]]), 'AUC': 0.9339459488707785}
```

```
    Validation: {'Accuracy': 0.8446502057613169, 'Precision': 0.8107606679035251, 'Recall': 0.8991769547325102, 'F1-score': 0.8526829268292
           [ 49, 437]]), 'AUC': 0.9152314179749022}
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

thresholds = [0, 0.3, 0.4, 0.5]
plt.figure(figsize=(8, 6))

for threshold in thresholds:
    # Plot ROC curve for training set
    y_train_pred = (y_train_predslr >= threshold).astype(int)
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
    plt.plot(fpr_train, tpr_train, lw=1, color='blue', alpha=0.5)
    plt.plot(fpr_train, tpr_train, 'o', color='blue', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_train, tpr_train):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='blue', fontsize=8, ha='right', va='bottom')

    # Plot ROC curve for validation set
    y_valid_pred = (y_valid_predslr >= threshold).astype(int)
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, y_valid_pred)
    plt.plot(fpr_valid, tpr_valid, lw=1, color='red', alpha=0.5)
    plt.plot(fpr_valid, tpr_valid, 'o', color='red', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_valid, tpr_valid):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='red', fontsize=8, ha='right', va='bottom')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Training and Validation Sets')
plt.legend(['Training Set', 'Validation Set'], loc="lower right")
plt.show()
```




## GaussianNb

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)

y_train_predsNb = nb.predict_proba(X_train)[:,1]
```

```
y_valid_predsNb = nb.predict_proba(X_valid)[:,1]


from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

thresholds = [0, 0.3, 0.4, 0.5]
plt.figure(figsize=(8, 6))

for threshold in thresholds:
    # Plot ROC curve for training set
    y_train_pred = (y_train_predsNb >= threshold).astype(int)
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
    plt.plot(fpr_train, tpr_train, lw=1, color='blue', alpha=0.5)
    plt.plot(fpr_train, tpr_train, 'o', color='blue', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_train, tpr_train):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='blue', fontsize=8, ha='right', va='bottom')

    # Plot ROC curve for validation set
    y_valid_pred = (y_valid_predsNb >= threshold).astype(int)
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, y_valid_pred)
    plt.plot(fpr_valid, tpr_valid, lw=1, color='red', alpha=0.5)
    plt.plot(fpr_valid, tpr_valid, 'o', color='red', markersize=5)  # Add filled dots
    for fpr, tpr in zip(fpr_valid, tpr_valid):
        plt.text(fpr, tpr, f'{threshold:.1f}', color='red', fontsize=8, ha='right', va='bottom')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Training and Validation Sets')
plt.legend(['Training Set', 'Validation Set'], loc="lower right")
plt.show()
```