



# Virtual Internship (Data Science)

## Data Intake Report

**Group Name: Project Group 1**

**Members:**

No	Name	Email	Country	College/company	Specialization
1	Preeti Verma	<a href="mailto:vermapreeti.dataanalyst@gmail.com">vermapreeti.dataanalyst@gmail.com</a>	Canada	-	Data Science
2	Thanuja Modiboina	<a href="mailto:thanujayadav953@gmail.com">thanujayadav953@gmail.com</a>	UK	-	Data Science
3	Abishek James	<a href="mailto:abishekjames1998@gmail.com">abishekjames1998@gmail.com</a>	Ireland	-	Data Science

**Name: Bank Marketing (Campaign)**

**Report date: 26-04-2023**

**Internship Batch: LISUM19**

**Data intake by:**

## Data intake reviewer: Data Glacier

### Data storage location:

#### Problem Description :

ABC Bank wants to sell its term deposit product to customers and before launching the product they want to develop a model which helps to understand whether a particular customer will buy their product or not (based on the customer's past interaction with the bank or other Financial Institution). This is an application of the company's marketing data.

#### Business Understanding :

The goal is to build a Machine Learning model that helps in predicting the outcomes of each customer's marketing campaign and analyzing which features have an impact on the outcomes will help the company to understand how to make the campaign more effective. Additionally, categorizing the customer group that subscribed to the term deposit helps to determine who is more likely to purchase the product in the future, thereby developing more targeted marketing campaigns.

This can be accomplished by using an ML model that shortlists the customers whose possibility of purchasing the product is higher. So, marketing such as telemarketing, SMS or email marketing can concentrate only on those customers. It will save time and resources by doing this.

### Project Lifecycle

Deadline ( Date/week)	Plan and Deliverables
19 April 2023(Week 7)	<ul style="list-style-type: none"><li>● Problem statement</li><li>● Business understanding</li><li>● Dataset collection</li></ul>
26 April 2023(Week 8)	<ul style="list-style-type: none"><li>● Data understanding</li><li>● Data analysis - finding null values, and outliers.</li><li>● Data processing</li></ul>
2 May 2023(Week 9)	Data cleaning and transformation
9 May 2023(Week 10)	EDA and Model Recommendation
16 May 2023(Week 11)	EDA Presentation and Proposed Modeling Technique

23 May 2023(Week 12)	Model Selection and Building the Model
30 May 2023(Week 13)	Final project report and code submission

**Tabular data details:**

**File 1: bank\_additional\_full.csv**

<b>Total number of observations</b>	41189
<b>Total number of files</b>	2
<b>Total number of features</b>	21
<b>Base format of the file</b>	.CSV
<b>Size of the data</b>	5.56MB

**File 2: bank\_additional.csv**

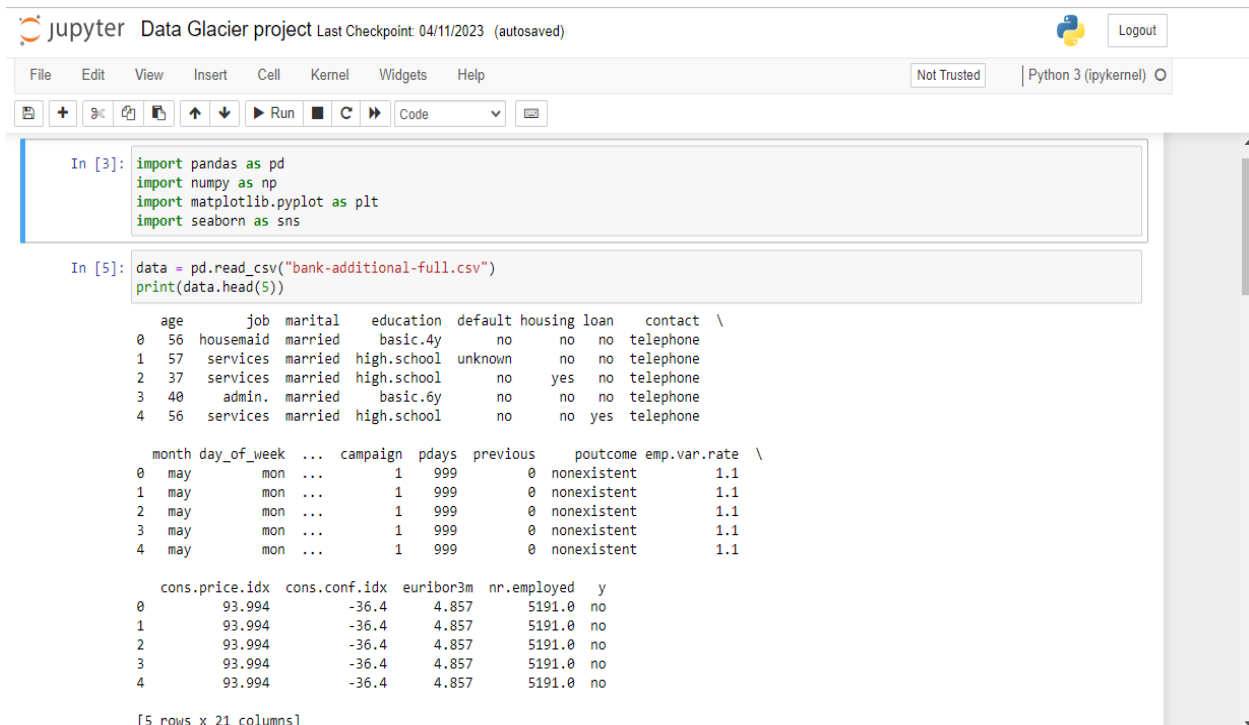
<b>Total number of observations</b>	4120
<b>Total number of files</b>	2
<b>Total number of features</b>	21

**Exploratory Data Analysis**

1. The data covers the period from May 2008 to November 2010.
2. There are 2 datasets, the second dataset is a sample of the first dataset. So, we are not taking the second dataset.
3. There are 10 integers and 11 categorical variables.
4. The missing values in the dataset are presented by an "unknown" string. We changed it to NaN.
5. There are missing values in six variables: job, marital status, education, default, housing, and loan. This will be imputed using various methods.
6. There are 12 duplicates in the first dataset and no duplicates in the sample dataset, this will be dropped since they are minimal and will not affect our analysis

## Assumptions

We assume the data provided is correct and up to date.



The image shows a Jupyter Notebook interface for a project named "Data Glacier project". The interface includes a top bar with the Jupyter logo, project name, last checkpoint date (04/11/2023), and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A "Not Trusted" warning and "Python 3 (ipykernel)" are also visible. The main area contains two code cells. The first cell (In [3]) imports pandas as pd, numpy as np, matplotlib.pyplot as plt, and seaborn as sns. The second cell (In [5]) reads a CSV file named "bank-additional-full.csv" and prints the first five rows. The output shows two data frames. The first data frame has columns: age, job, marital, education, default, housing, loan, and contact. The second data frame has columns: month, day\_of\_week, campaign, pdays, previous, poutcome, emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed, and y. The output is truncated with ellipses in the middle of each data frame.

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

In [5]: data = pd.read_csv("bank-additional-full.csv")
print(data.head(5))
```

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

	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate
0	may	mon	...	1	999	0	nonexistent	1.1
1	may	mon	...	1	999	0	nonexistent	1.1
2	may	mon	...	1	999	0	nonexistent	1.1
3	may	mon	...	1	999	0	nonexistent	1.1
4	may	mon	...	1	999	0	nonexistent	1.1

	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	93.994	-36.4	4.857	5191.0	no
1	93.994	-36.4	4.857	5191.0	no
2	93.994	-36.4	4.857	5191.0	no
3	93.994	-36.4	4.857	5191.0	no
4	93.994	-36.4	4.857	5191.0	no

[5 rows x 21 columns]

Jupyter Data Glacier project Last Checkpoint: 04/11/2023 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted | Python 3 (ipykernel)

4 93.994 -36.4 4.857 5191.0 no

[5 rows x 21 columns]

```
In [6]: print(data.dtypes)
```

age	int64
job	object
marital	object
education	object
default	object
housing	object
loan	object
contact	object
month	object
day_of_week	object
duration	int64
campaign	int64
pdays	int64
previous	int64
poutcome	object
emp.var.rate	float64
cons.price.idx	float64
cons.conf.idx	float64
euribor3m	float64
nr.employed	float64
y	object
dtype:	object

```
In [7]: data.isna().sum()
```

Jupyter Data Glacier project Last Checkpoint: 04/11/2023 (autosaved) Logout

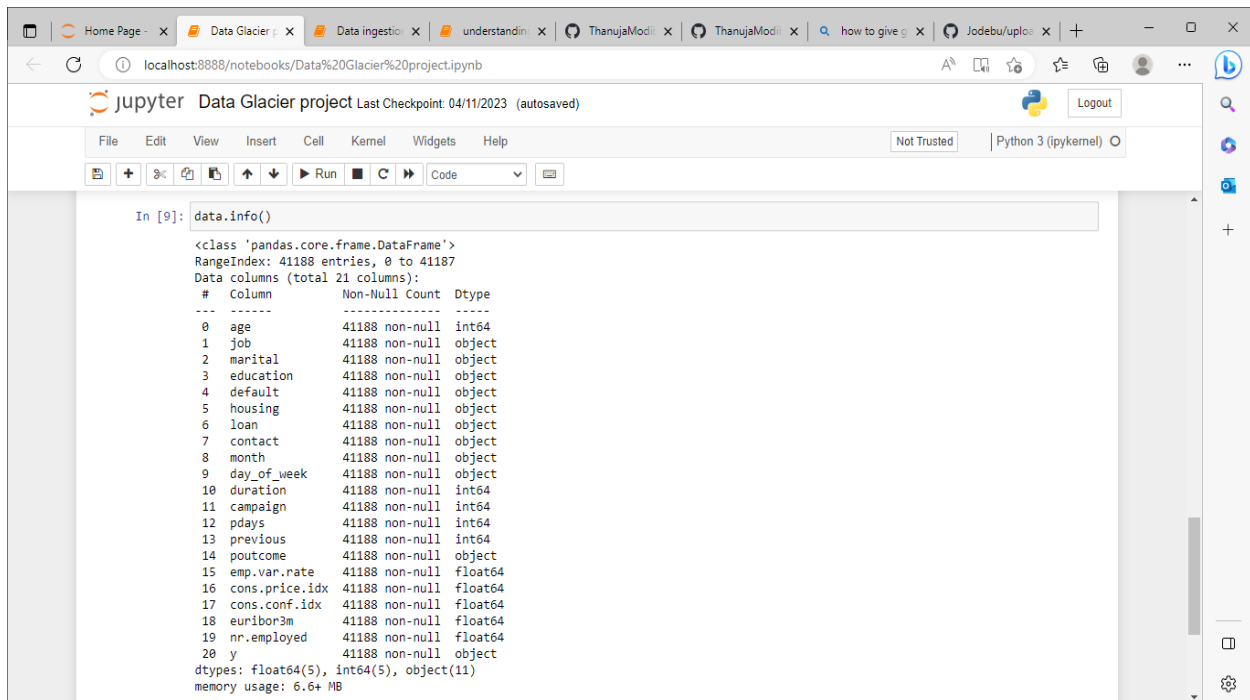
File Edit View Insert Cell Kernel Widgets Help Not Trusted | Python 3 (ipykernel)

```
In [7]: data.isna().sum()
```

```
Out[7]: 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
```

```
In [9]: data.info()
```

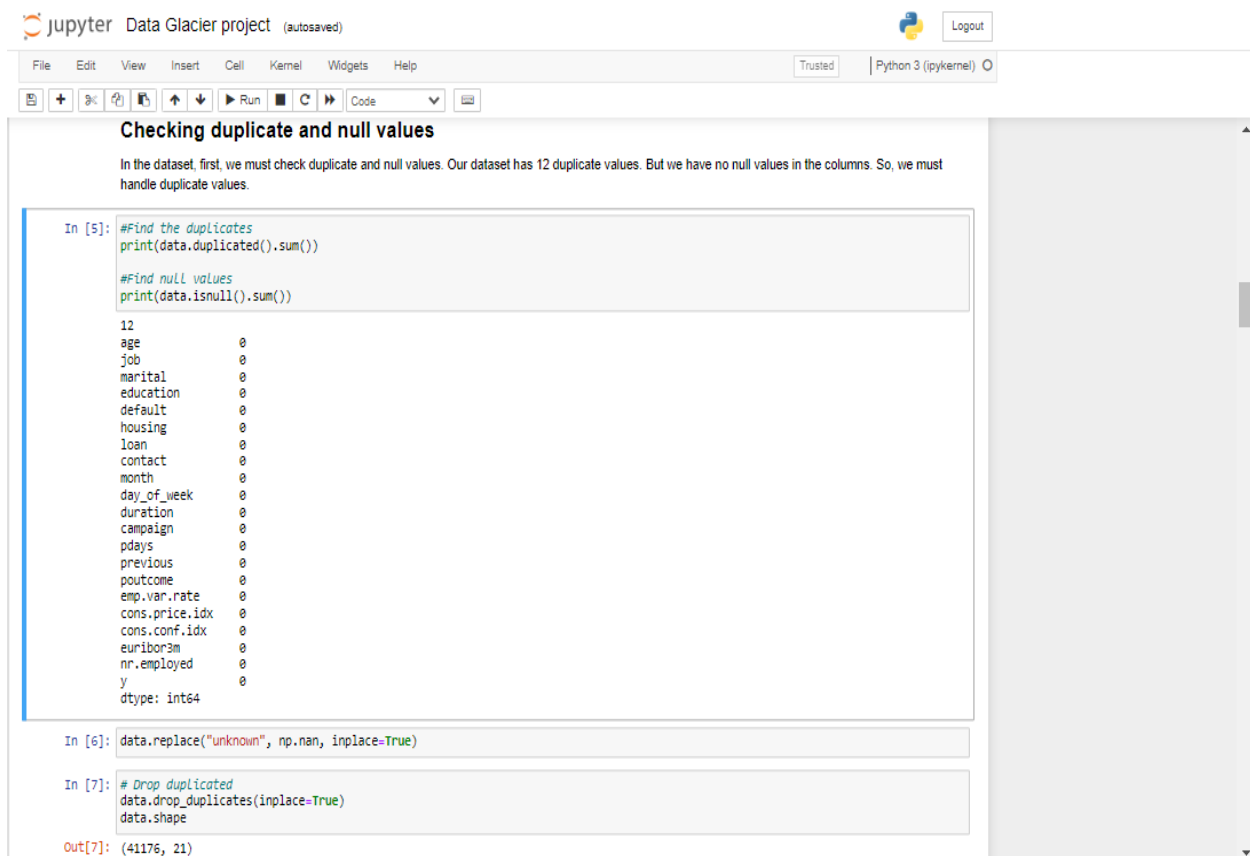
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
#   Column              Non-Null Count  Dtype
# 0  age                 41188 non-null  int64
# 1  job                 41188 non-null  object
# 2  marital             41188 non-null  object
# 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: object(10), int64(10), float64(1)
```



The screenshot shows a Jupyter Notebook titled "Data Glacier project" with a "Last Checkpoint: 04/11/2023 (autosaved)". The notebook is running on a local host. The code cell shows the command `data.info()`, which outputs the following information:

```
<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
```

## Week 8 Assignment:



The screenshot shows a Jupyter Notebook titled "Data Glacier project" with a "Last Checkpoint: 04/11/2023 (autosaved)". The notebook is running on a local host. The code cell shows the following commands:

```
In [5]: #Find the duplicates
print(data.duplicated().sum())

#Find null values
print(data.isnull().sum())

12
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

In [6]: data.replace("unknown", np.nan, inplace=True)

In [7]: # Drop duplicated
data.drop_duplicates(inplace=True)
data.shape

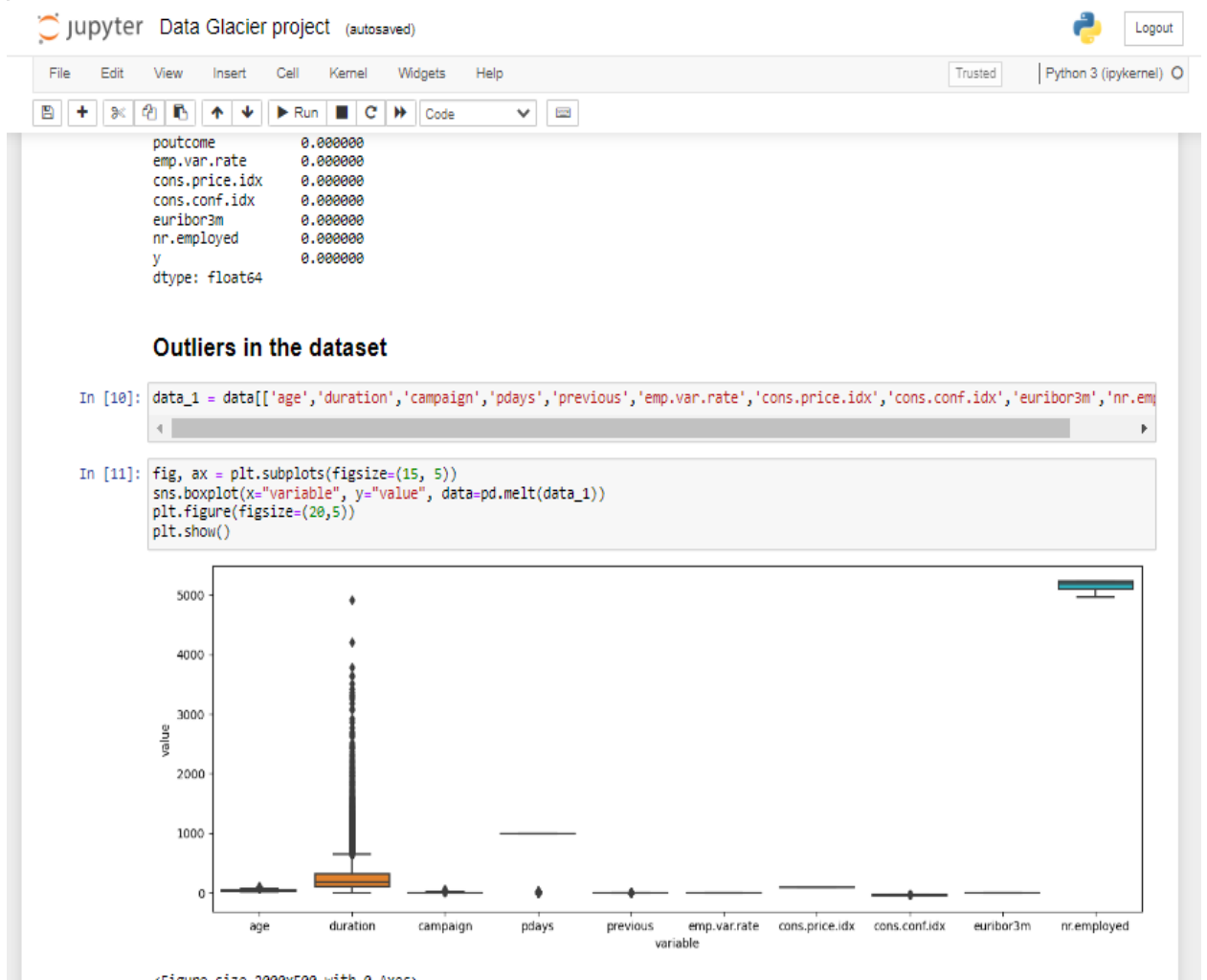
Out[7]: (41176, 21)
```

```
jupyter Data Glacier project (autosaved) Python 3 (ipykernel) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted
In [8]: # data.isna().sum()
print(data.isnull().sum())

age      0
job      330
marital   80
education 1730
default  8596
housing   990
loan      990
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

In [9]: null_percentage = data.isnull().mean()*100
null_percentage

Out[9]: age      0.000000
job      0.001438
marital   0.194288
education 4.201477
default  20.876239
housing   2.404313
loan      2.404313
contact   0.000000
month     0.000000
day_of_week 0.000000
duration  0.000000
```

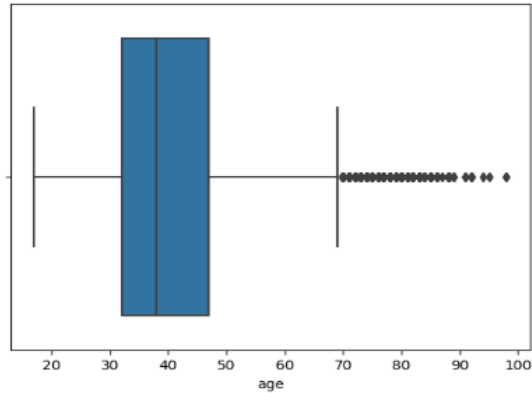


File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

Run Code

```
In [12]: sns.boxplot(x = data_1['age'])
plt.show()
```



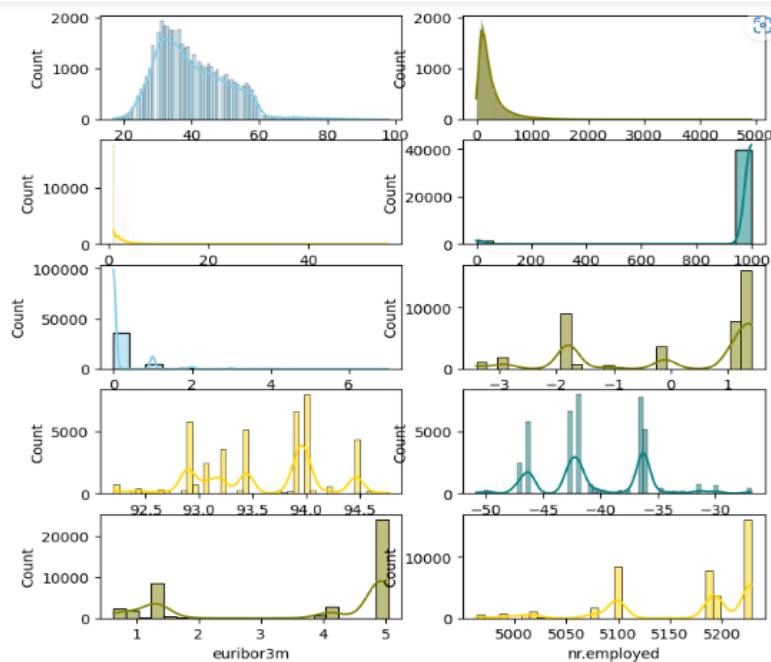
```
In [13]: fig, axes = plt.subplots(5, 2, figsize=(8, 8))
sns.histplot(data=data_1, x="age", kde=True, color="skyblue", ax=axes[0, 0])
sns.histplot(data=data_1, x="duration", kde=True, color="olive", ax=axes[0, 1])
sns.histplot(data=data_1, x="campaign", kde=True, color="gold", ax=axes[1, 0])
sns.histplot(data=data_1, x="pdays", kde=True, color="teal", ax=axes[1, 1])
sns.histplot(data=data_1, x="previous", kde=True, color="skyblue", ax=axes[2, 0])
sns.histplot(data=data_1, x="emp.var.rate", kde=True, color="olive", ax=axes[2, 1])
sns.histplot(data=data_1, x="cons.price.idx", kde=True, color="gold", ax=axes[3, 0])
sns.histplot(data=data_1, x="cons.conf.idx", kde=True, color="teal", ax=axes[3, 1])
sns.histplot(data=data_1, x="euribor3m", kde=True, color="olive", ax=axes[4, 0])
sns.histplot(data=data_1, x="nr.employed", kde=True, color="gold", ax=axes[4, 1])
```

Out[13]: <Axes: xlabel='nr.employed', ylabel='Count'>

File Edit View Insert Cell Kernel Widgets Help

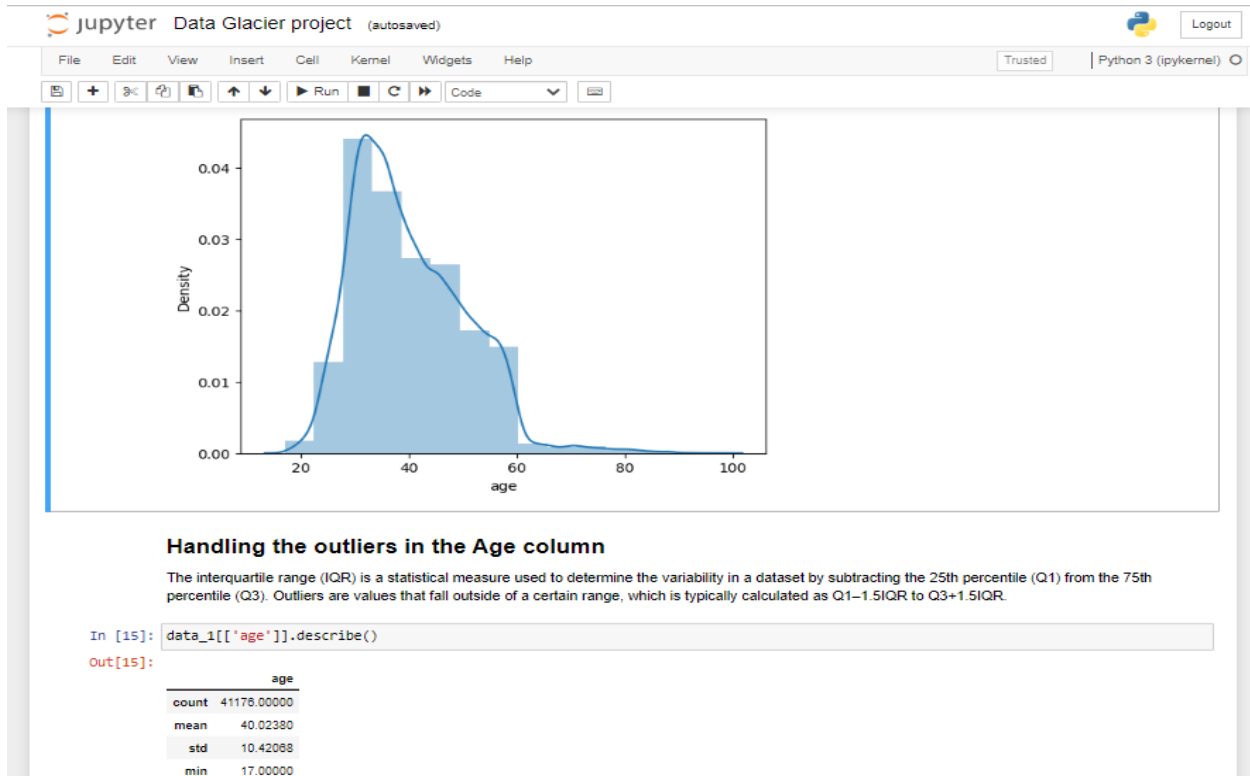
Trusted Python 3 (ipykernel)

Run Code



```
In [14]: sns.distplot(data_1['age'], bins = 15, kde = True)
plt.show()
```





Jupyter Data Glacier project (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [16]: data_1['age'].quantile(0.25)
```

```
Out[16]: 32.0
```

```
In [17]: data_1['age'].quantile(0.75)
```

```
Out[17]: 47.0
```

```
In [18]: Q1 = data_1['age'].quantile(0.25)
Q3 = data_1['age'].quantile(0.75)
IQR = Q3-Q1
IQR
```

```
Out[18]: 15.0
```

```
In [19]: lower_lim = Q1-1.5*IQR
upper_lim = Q1+1.5*IQR
print(lower_lim)
print(upper_lim)
```

```
9.5
54.5
```

```
In [20]: outliers_15_low = (data_1['age']<lower_lim)
# print(outliers_15_low)
outliers_15_up = (data_1['age']>upper_lim)
# print(outliers_15_up)
```

```
In [21]: len(data_1['age']) - (len(data_1['age'])[outliers_15_low]+len(data_1['age'])[outliers_15_up]))
```

```
Out[21]: 36948
```

```
In [22]: data_1['age'][(outliers_15_low|outliers_15_up)]
```

```
Out[22]:
```

0	56
1	57
4	56
6	59
13	57
..	..
41178	62
41179	64
41180	77

Jupyter Data Glacier project (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```

3      40
5      45
7      41
8      24
..
41180 36
41181 37
41182 29
41184 46
41186 44
Name: age, Length: 36948, dtype: int64

```

In [24]: `data_1 = data_1[~(outliers_15_low|outliers_15_up)]`  
`data_1`

Out[24]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
2	37	228	1	999	0	1.1	93.994	-38.4	4.857	5191.0
3	40	151	1	999	0	1.1	93.994	-38.4	4.857	5191.0
5	45	108	1	999	0	1.1	93.994	-38.4	4.857	5191.0
7	41	217	1	999	0	1.1	93.994	-38.4	4.857	5191.0
8	24	380	1	999	0	1.1	93.994	-38.4	4.857	5191.0
...	...	...	...	...	...	...	...	...	...	...
41180	36	254	2	999	0	-1.1	94.787	-50.8	1.028	4993.8
41181	37	281	1	999	0	-1.1	94.787	-50.8	1.028	4993.8
41182	29	112	1	9	1	-1.1	94.787	-50.8	1.028	4993.8
41184	48	383	1	999	0	-1.1	94.787	-50.8	1.028	4993.8
41186	44	442	1	999	0	-1.1	94.787	-50.8	1.028	4993.8

36948 rows x 10 columns

In [25]: `sns.boxplot(x = data_1['age'])`  
`plt.show()`

Jupyter Data Glacier project (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

computing the frequency distribution.

Distribution on the basis of skewness value:

Skewness = 0: Then normally distributed.

Skewness > 0: Then more weight in the left tail of the distribution.

Skewness < 0: Then more weight in the right tail of the distribution.

In [27]:

```

from scipy.stats import skew
# Calculate Pearson's skewness coefficient
skewness1 = skew(data_1)

# Calculate moment coefficient of skewness
skewness2 = np.mean((data_1 - np.mean(data_1)) ** 3) / (np.std(data_1) ** 3)

print("Pearson's skewness coefficient: ", skewness1)
print("Moment coefficient of skewness:\n", skewness2)

```

Pearson's skewness coefficient: [ 0.23795482 3.3107468 4.79008024 -5.28224096 3.90406112 -0.74646063  
-0.19884943 0.27784703 -0.75398964 -1.08125485]

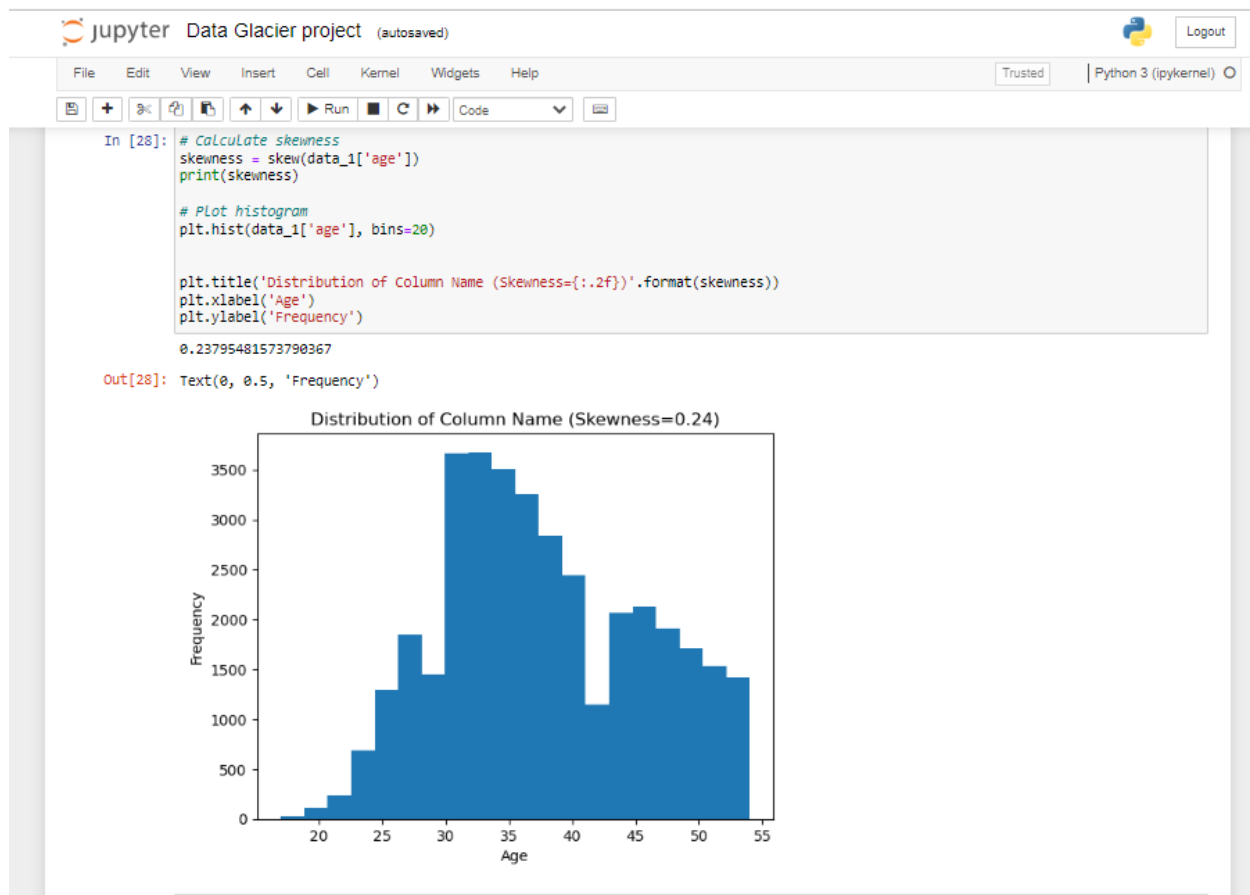
Moment coefficient of skewness:

```

age          0.237955
duration     3.310747
campaign     4.790080
pdays      -5.282241
previous     3.904061
emp.var.rate -0.746461
cons.price.idx -0.198849
cons.conf.idx  0.277847
euribor3m    -0.753990
nr.employed  -1.081255
dtype: float64

```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3430: FutureWarning: In a future version, DataFrame.mean(a  
xis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or just 'fr  
ame.mean()'
 return mean(axis=axis, dtype=dtype, out=out, \*\*kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3430: FutureWarning: In a future version, DataFrame.mean(a  
xis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or just 'fr  
ame.mean()'
 return mean(axis=axis, dtype=dtype, out=out, \*\*kwargs)



## Week 9 Assignment:

Handling missing values is an essential step in data cleaning and preprocessing. There are several methods to handle missing values, and choosing the right method depends on the nature of the data and the missing values.

### Filling missing values using value\_counts method

`value_counts()`: This method is useful when the missing values are represented as NaN. The `value_counts()` method returns a series containing counts of unique values, excluding NaN values. It can be used to determine the most frequent value in a column and impute the missing values with that value.

This column('marital') has 80 missing values. So, we can use the `value_counts` method to fill these values.

### Forward or backward fill:

This method involves filling missing values with the previous (forward fill) or next (backward fill) known value. This method is useful when dealing with time series data where the missing value is expected to be like the previous or next value.

## Mode imputation:

The mode is the most frequent value in a column. Mode imputation involves replacing the missing values with the mode of the column. This method is useful when the missing values are few compared to the total number of observations and the mode is a reasonable representation of the missing values.

## Interpolation:

Interpolation is a statistical technique that involves estimating missing values based on the pattern observed in the data. Linear interpolation is commonly used to estimate missing values. This method is useful when dealing with continuous data and the missing values are not frequent.

```
In [29]: data_1['age'].fillna(data_1['age'].mean(), inplace=True)
```

```
In [30]: data_1['age'].describe()
```

```
Out[30]: count    36948.000000
         mean      37.731731
         std       8.032715
         min       17.000000
         25%       31.000000
         50%       37.000000
         75%       44.000000
         max       54.000000
         Name: age, dtype: float64
```

```
In [31]: data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36948 entries, 2 to 41186
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  -
 0   age                 36948 non-null  int64
 1   duration            36948 non-null  int64
 2   campaign            36948 non-null  int64
 3   pdays               36948 non-null  int64
 4   previous            36948 non-null  int64
 5   emp.var.rate        36948 non-null  float64
 8   euribor3m           36948 non-null  float64
 9   nr.employed         36948 non-null  float64
dtypes: float64(5), int64(5)
memory usage: 3.1 MB
```

```
In [32]: # data[['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
```

## Handling missing values in the dataset

Handling missing values is an essential step in data cleaning and preprocessing. There are several methods to handle missing values, and choosing the right method depends on the nature of the data and the missing values.

### Filling missing values using value\_counts method

value\_counts(): This method is useful when the missing values are represented as NaN. The value\_counts() method returns a series containing counts of unique values, excluding NaN values. It can be used to determine the most frequent value in a column and impute the missing values with that value.

This column('marital') has 80 missing values. So, we can use the value\_counts method to fill these values.

```
In [33]: data['marital'].fillna(data['marital'].value_counts().index[0], inplace = True)
```

### Fill missing values with forward or backward fill:

Forward or backward fill: This method involves filling missing values with the previous (forward fill) or next (backward fill) known value. This method is useful when dealing with time series data where the missing value is expected to be similar to the previous or next value.

```
In [34]: data['housing'] = data['housing'].fillna(method='ffill')
data['default'] = data['default'].fillna(method='bfill')
```

### Fill missing values with mode:

Mode imputation: The mode is the most frequent value in a column. Mode imputation involves replacing the missing values with the mode of the column. This method is useful when the missing values are few compared to the total number of observations and the mode is a reasonable representation of the missing values.

```
In [35]: data['loan'] = data['loan'].fillna(data['loan'].mode().iloc[0])
```

### Interpolation:

Interpolation: Interpolation is a statistical technique that involves estimating missing values based on the pattern observed in the data. Linear interpolation is commonly used to estimate missing values. This method is useful when dealing with continuous data and the missing values are not frequent.

```
In [36]: data['job'] = data['job'].interpolate()
```

```
In [37]: data.isna().sum()
```

```
Out[37]: age                0
job                330
marital           0
education        1730
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
```

## Week 10 Assignment:

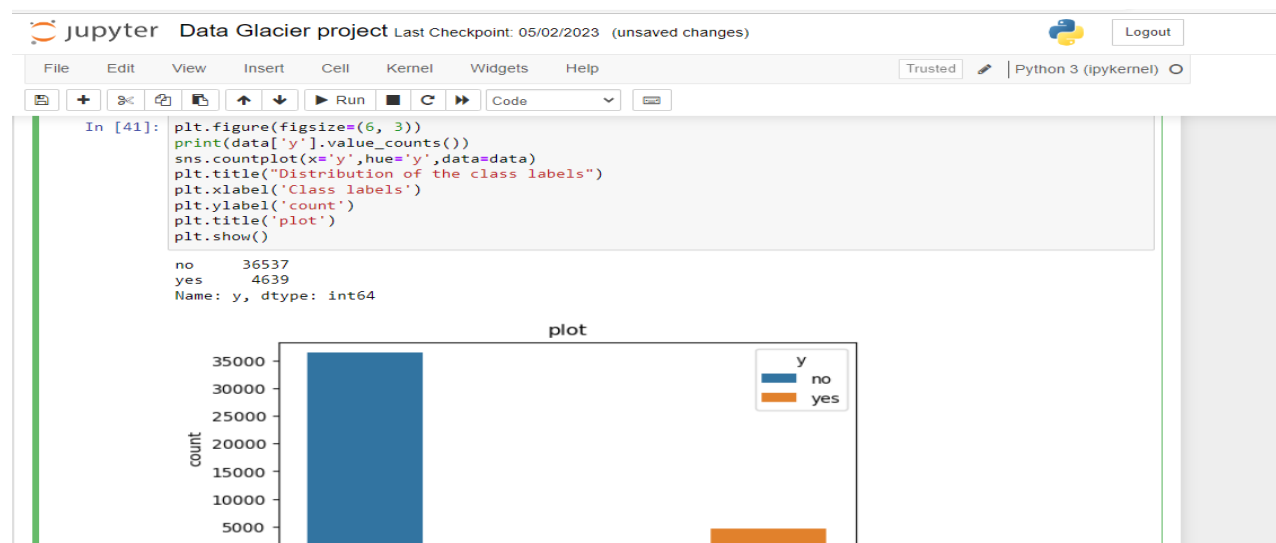
### Exploratory Data Analysis (EDA) process:

Exploratory Data Analysis (EDA) is an important step in data science that involves analyzing and understanding the structure and characteristics of the data before applying any modeling techniques. The main objective of EDA is to identify patterns and relationships in the data that can inform the modeling process and ultimately lead to better insights and more accurate predictions.

Plotting features is an important part of Exploratory Data Analysis (EDA) in data science because it allows us to visually explore the distribution of the data and identify patterns and relationships between variables. By plotting the data, we can gain insights that may not be apparent from simple summary statistics or tabular data.

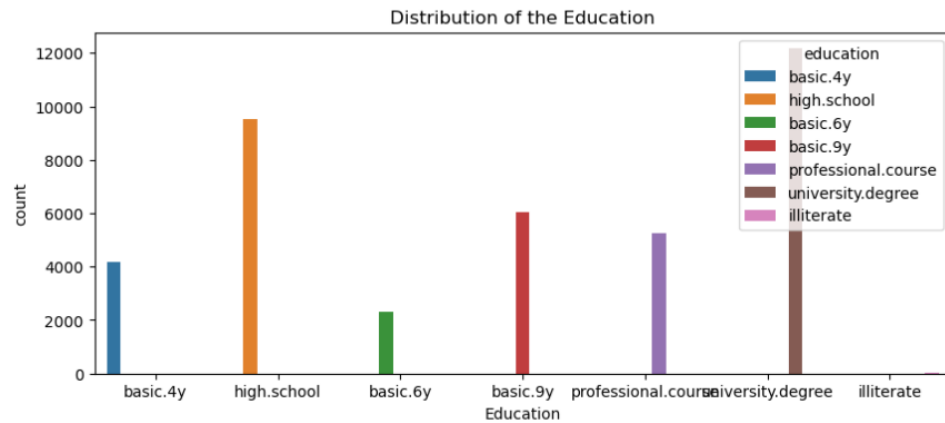
Plotting features can help to communicate findings to stakeholders clearly and concisely, using visualizations that are easy to understand. This can help to build trust in the analysis and improve the overall effectiveness of data-driven decision-making.

Below is plotted the distribution of a few features such as job, education, marital, housing, loan, contact, default, and month.

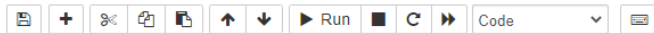




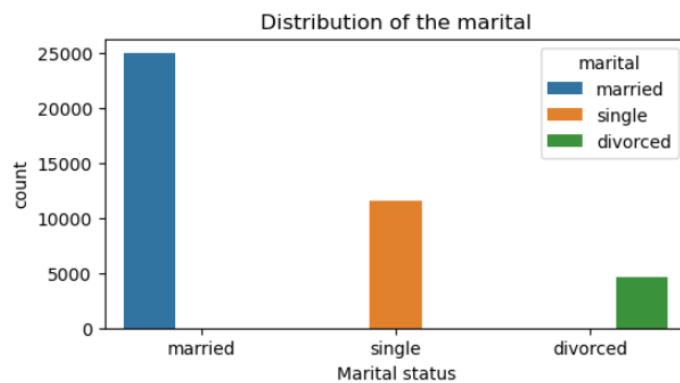
```
plt.ylabel('count')
plt.show()
```



```
In [44]: plt.figure(figsize=(6, 3))
sns.countplot(x='marital', hue='marital', data=data)
plt.title("Distribution of the marital")
plt.show()
```



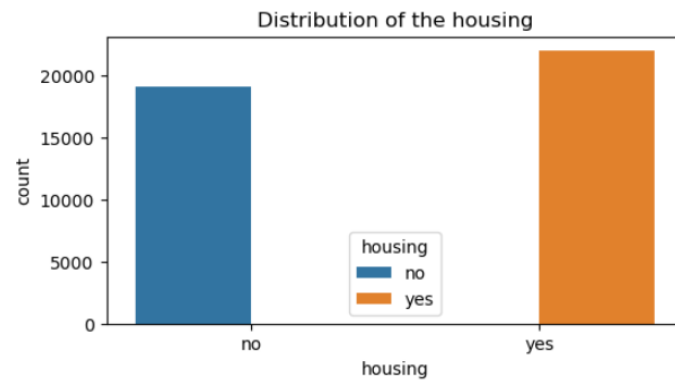
```
plt.title("Distribution of the marital")
plt.xlabel('Marital status')
plt.ylabel('count')
plt.show()
```



```
In [45]: plt.figure(figsize=(6, 3))
sns.countplot(x='housing', hue='housing', data=data)
plt.title("Distribution of the housing")
plt.show()
```



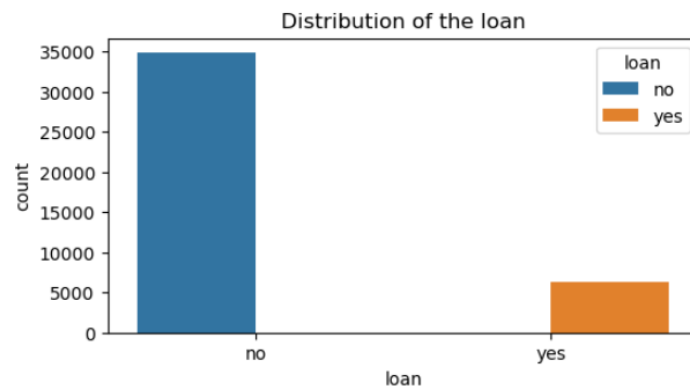
```
plt.xlabel('housing')
plt.ylabel('count')
plt.show()
```



```
In [46]: plt.figure(figsize=(6, 3))
sns.countplot(x='loan',hue='loan',data=data)
plt.title("Distribution of the loan")
plt.xlabel('loan')
```



```
plt.xlabel('loan')
plt.ylabel('count')
plt.show()
```



```
In [47]: plt.figure(figsize=(6, 3))
sns.countplot(x='contact',hue='contact',data=data)
plt.title("Distribution of contact type")
plt.xlabel('cotact type')
plt.ylabel('count')
```



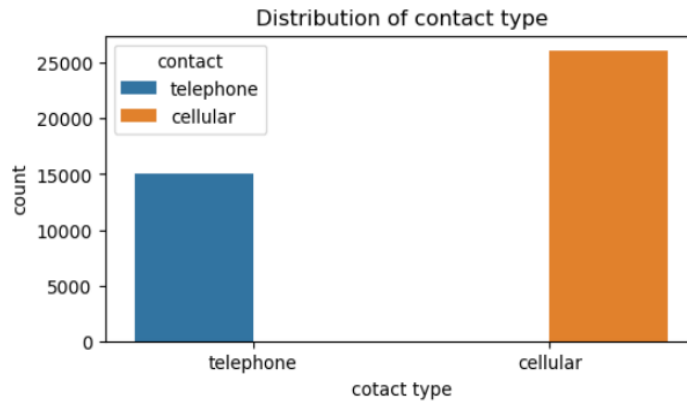
File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

Run Code

```
plt.xlabel('contact type')
plt.ylabel('count')
plt.show()
```



```
In [48]: plt.figure(figsize=(6, 3))
sns.countplot(x='default', hue='default', data=data)
plt.title("Distribution of the default column")
plt.xlabel('default')
plt.ylabel('count')
```

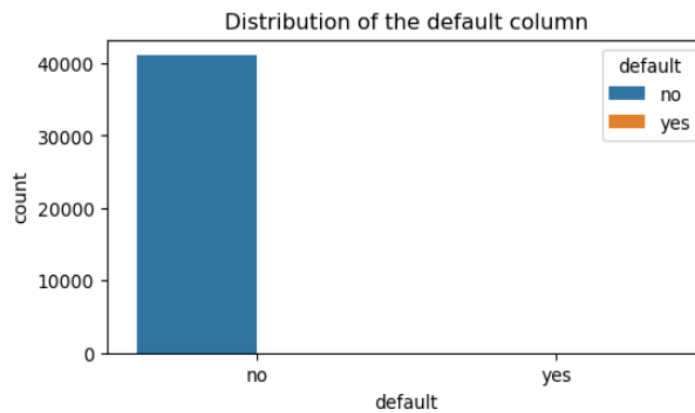
File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

Run Code

```
plt.ylabel('count')
plt.show()
```



```
In [51]: plt.figure(figsize=(10, 4))
sns.countplot(x='month', hue='month', data=data)
plt.title("Distribution of the Months")
plt.xlabel('Month')
plt.ylabel('count')
```



```
In [51]: plt.figure(figsize=(10, 4))
sns.countplot(x='month', hue='month', data=data)
plt.title("Distribution of the Months")
plt.xlabel('Month')
plt.ylabel('count')
plt.show()
```

