Experiment No. 4
To explore the data visualization techniques.
Date of Performance: 13-02-2024
Date of Submission: 13-02-2024

Academic Year: 2023-24 **Semester: VIII**
Class / Branch: BE Computer **Subject: Applied Data Science Lab**
Name : Riya Khot **Roll No. 21**

Experiment No. 4

1. **Aim:** To explore the data visualization techniques.

Dataset: In this experiment, data visualization techniques are explored on seaborn tips dataset.

2. **Software used:** Google Colaboratory / Jupyter Notebook

3. **Theory :-** Visualizations make it easier to explore and extract relevant information from the data by identifying patterns, relationships, outliers, and much more. Seaborn is a statistical plotting library in Python and is an extended version of Matplotlib. It supports complex visualizations and makes the plotting of graphs simple and intuitive. It can be used in Python scripts, Jupyter notebook, and web application servers.Seaborn uses less syntax as compared to Matplotlib. Hence, it is easier to use.It is easier to customize themes and high-level interfaces in Seaborn to make the plots more attractive and readable.Seaborn is much more functional and organized than Matplotlib and is better integrated to work with Pandas DataFrames.

Seaborn provides different plots for different types of variables as follows:

a. Frequency Distribution - Categorical Variables

* countplot
* catplot

b. Distribution of the Numerical Variable

* distplot(histogram)
* kdeplot
* boxplot
* violinplot

c. Relationship between 2 Numerical Variables

* lineplot
* scatterplot
* relplot
* lmplot
* heatmap
* pairplot
* facetgrid

d. Relationship between Numerical and Categorical Variables
* pointplot
* barplot
* boxplot

* violinplot
* swarmplot
* catplot
* facetgrid

4. Program :

adse4

February 13, 2024

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from scipy import stats
import warnings
warnings.simplefilter(action='ignore',category=FutureWarning)
```

Read Csv File

[2]:
```
df=pd.read_csv("/content/tips-expt4 - tips-expt4.csv")
```

[3]:
```
df.head()
```

[3]:
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

Preprocessing

[4]:
```
df.isnull().sum()
```

[4]:
```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

[5]:
```
df.describe()
```

[5]:
```
       total_bill         tip        size
count  244.000000  244.000000  244.000000
```

```
mean    19.785943    2.998279    2.569672
std      8.902412    1.383638    0.951100
min      3.070000    1.000000    1.000000
25%     13.347500    2.000000    2.000000
50%     17.795000    2.900000    2.000000
75%     24.127500    3.562500    3.000000
max     50.810000   10.000000    6.000000
```

[6]:
```
df.tip.describe()
```

[6]:
```
count    244.000000
mean       2.998279
std        1.383638
min        1.000000
25%        2.000000
50%        2.900000
75%        3.562500
max       10.000000
Name: tip, dtype: float64
```

Five Number Summary For Bill and Tip

[7]:
```
bill = df.total_bill
print("Maximum Bill : ",np.max(bill))
print("Minimum Bill : ",np.min(bill))
print("Standard Deviation : ",np.std(bill))
print("Median : ",np.median(bill))
print("Mean : ",np.mean(bill))
```

```
Maximum Bill :  50.81
Minimum Bill :  3.07
Standard Deviation :  8.884150577771132
Median :  17.795
Mean :  19.78594262295082
```

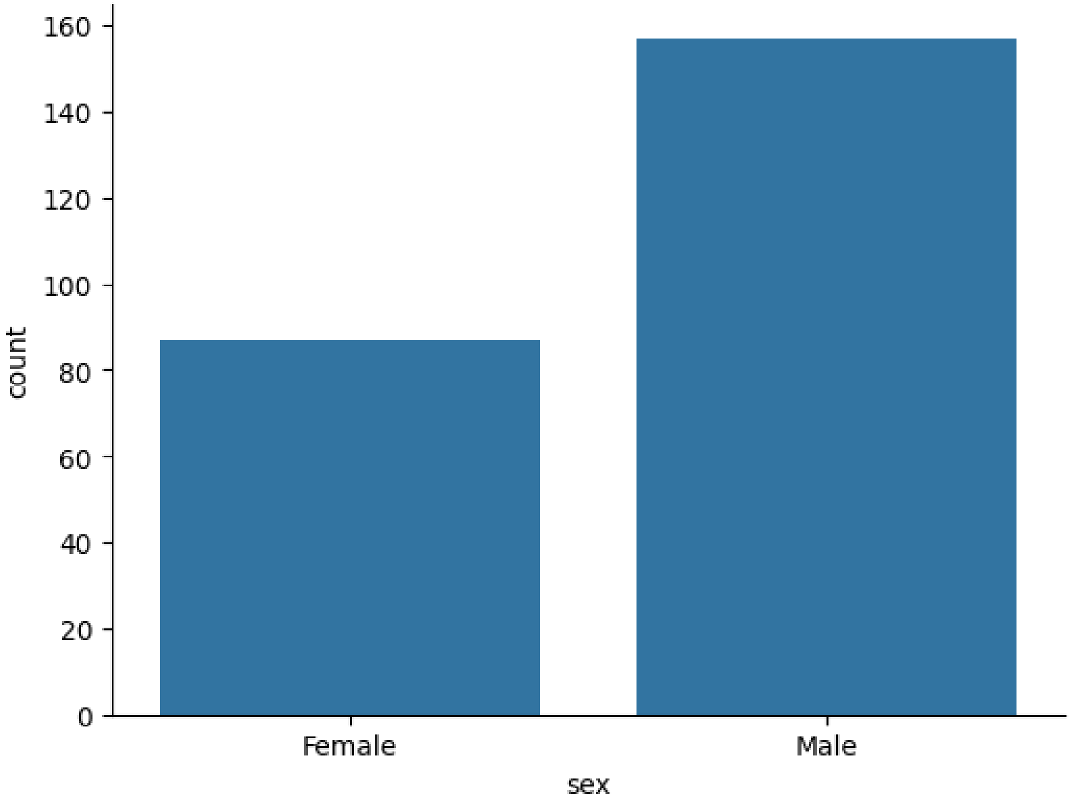
[8]:
```
tip = df.tip
print("Maximum Bill : ",np.max(tip))
print("Minimum Bill : ",np.min(tip))
print("Standard Deviation : ",np.std(tip))
print("Median : ",np.median(tip))
print("Mean : ",np.mean(tip))
```

```
Maximum Bill :  10.0
Minimum Bill :  1.0
Standard Deviation :  1.3807999538298954
Median :  2.9
Mean :  2.99827868852459
```

Exploratory Data Analysis

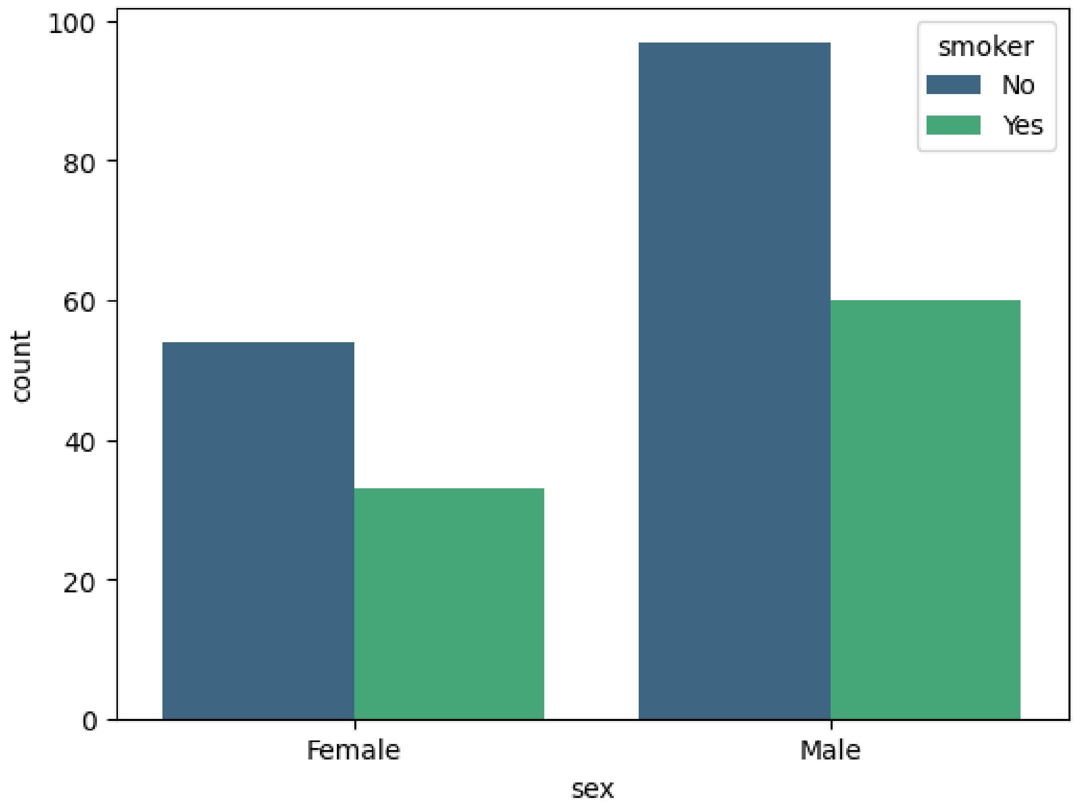
[10]:
```
sb.countplot(x='sex',data=df)
sb.despine()

print(df.sex.value_counts())
```

```
Male      157
Female     87
Name: sex, dtype: int64
```



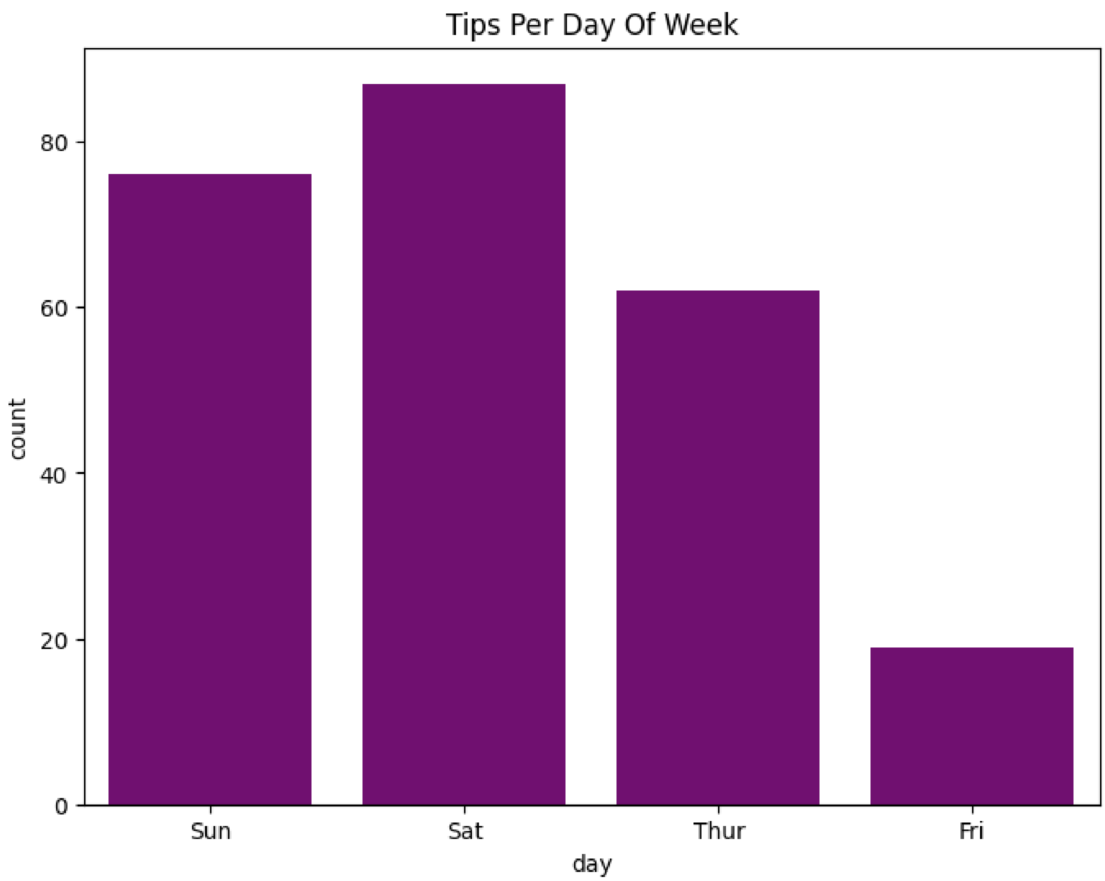
[11]:
```
sb.countplot(x='sex',data=df,hue='smoker',palette='viridis')
```

[11]: <Axes: xlabel='sex', ylabel='count'>



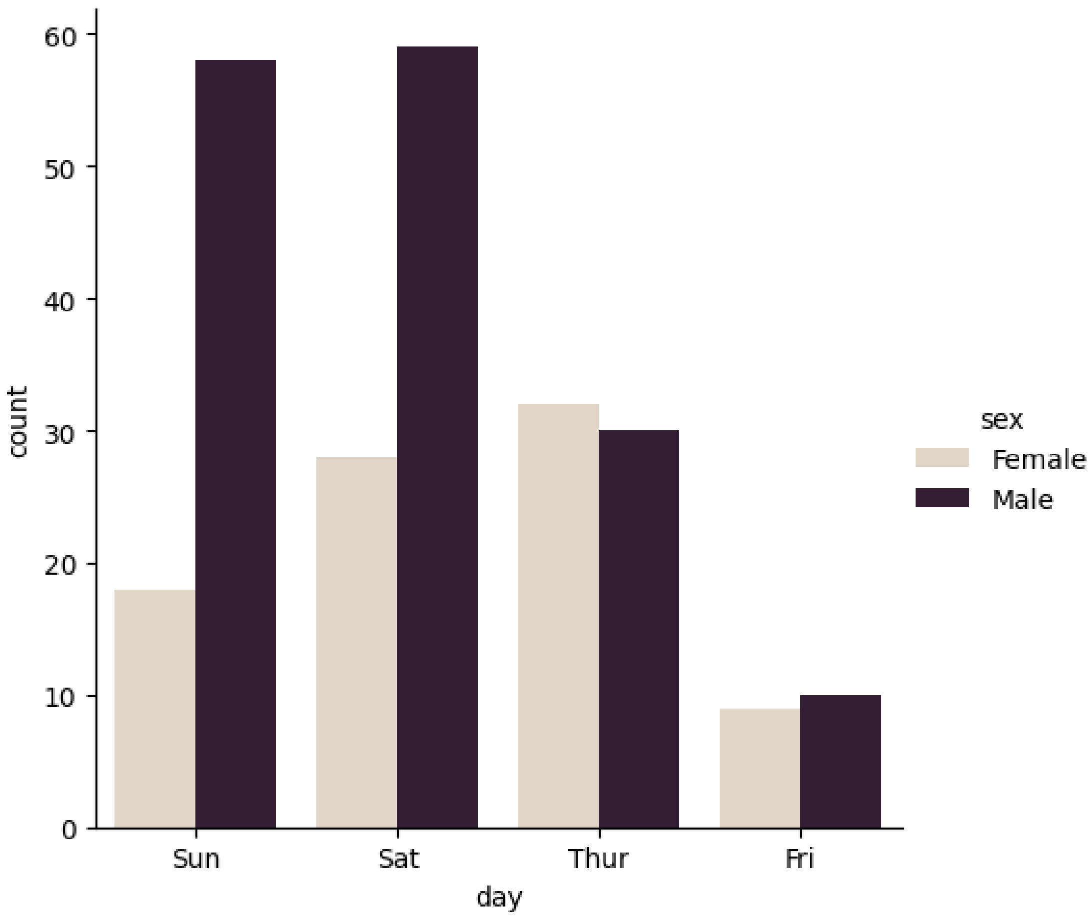
[13]:
```
plt.figure(figsize=(8,6))
plt.title('Tips Per Day Of Week')
sb.countplot(x=df['day'],color='purple')
```

[13]: <Axes: title={'center': 'Tips Per Day Of Week'}, xlabel='day', ylabel='count'>



[14]:
```
sb.catplot(x='day',data=df,hue='sex',palette='ch:.25',kind='count')
```

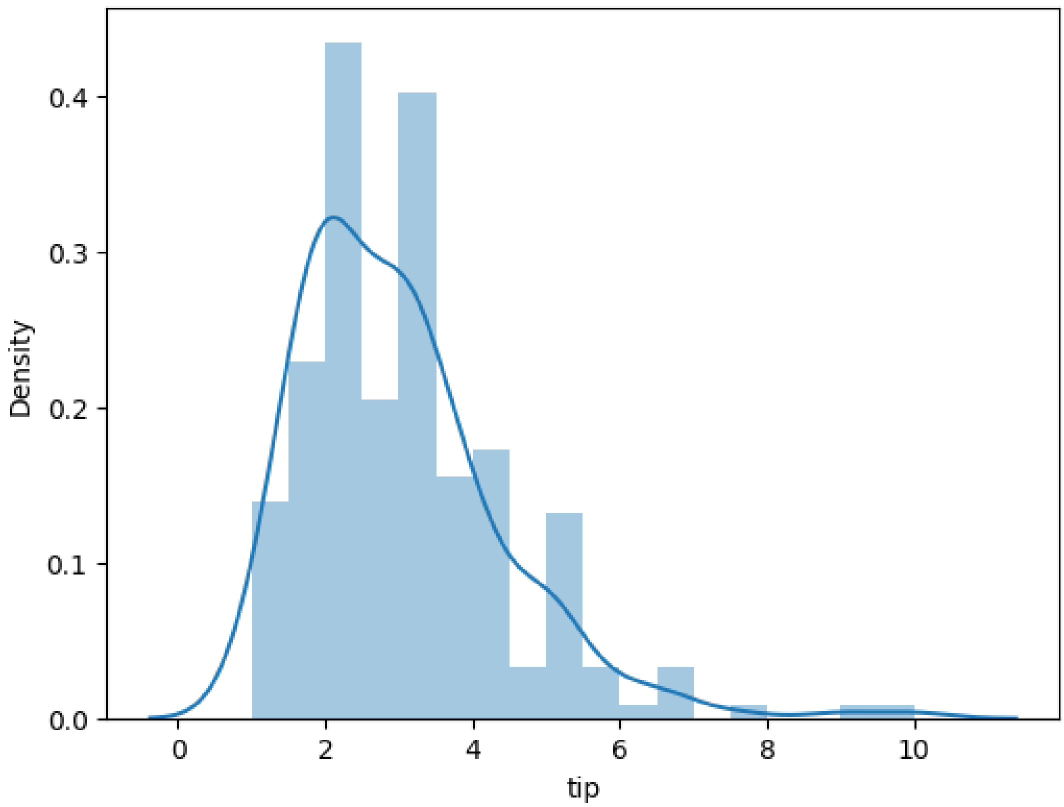
[14]: <seaborn.axisgrid.FacetGrid at 0x7cafd2aa9300>



[15]:
```
sb.distplot(df['tip'])
```

```
<ipython-input-15-8ce3da0f29bb>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sb.distplot(df['tip'])
```

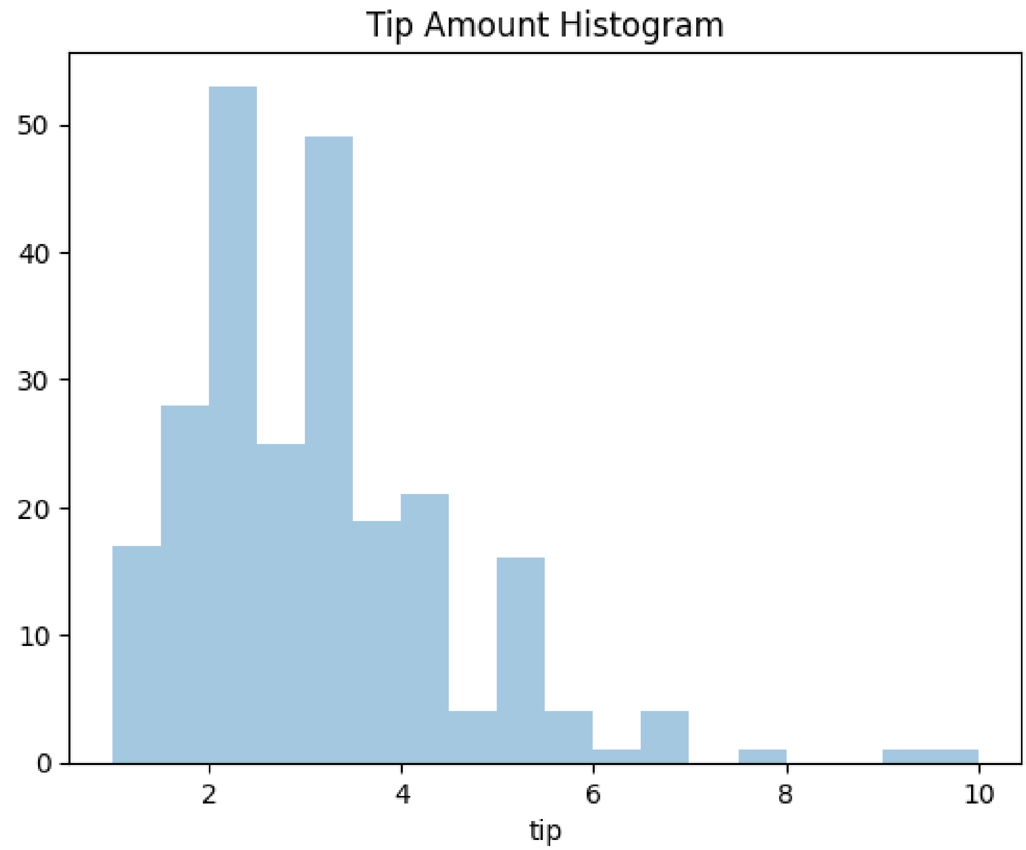
[15]:
```
<Axes: xlabel='tip', ylabel='Density'>
```




```
[16]: g=sb.distplot(df.tip,kde=False)
      g.set_title('Tip Amount Histogram')
```

```
<ipython-input-16-4137c74c6c3b>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  g=sb.distplot(df.tip,kde=False)
```

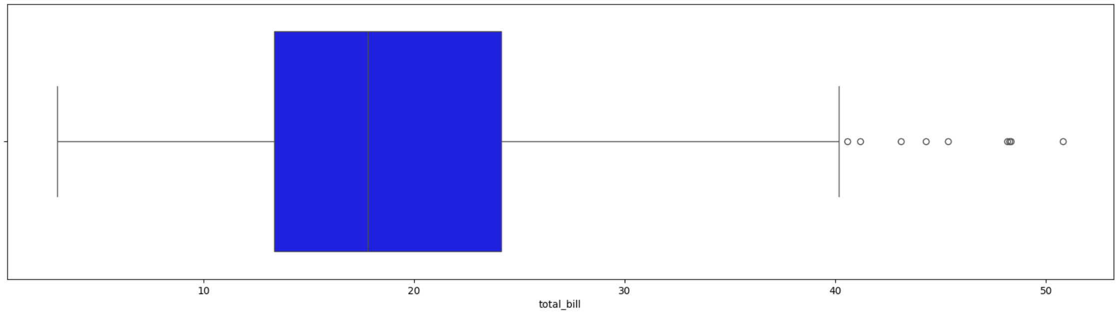
```
[16]: Text(0.5, 1.0, 'Tip Amount Histogram')
```



Outliers In bill column

```
[17]: plt.figure(figsize=(20,5))
      sb.boxplot(x=bill,color='b')
```

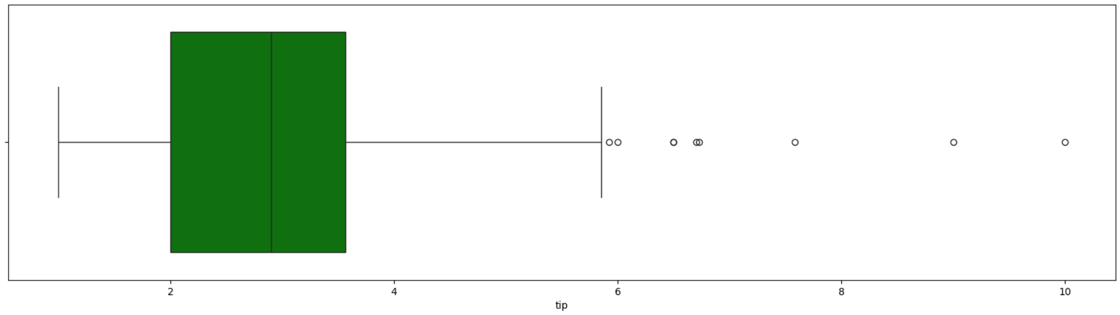
```
[17]: <Axes: xlabel='total_bill'>
```



Outliers In tip column

[18]:
```
plt.figure(figsize=(20,5))
sb.boxplot(x=tip,color='g')
```

[18]: <Axes: xlabel='tip'>



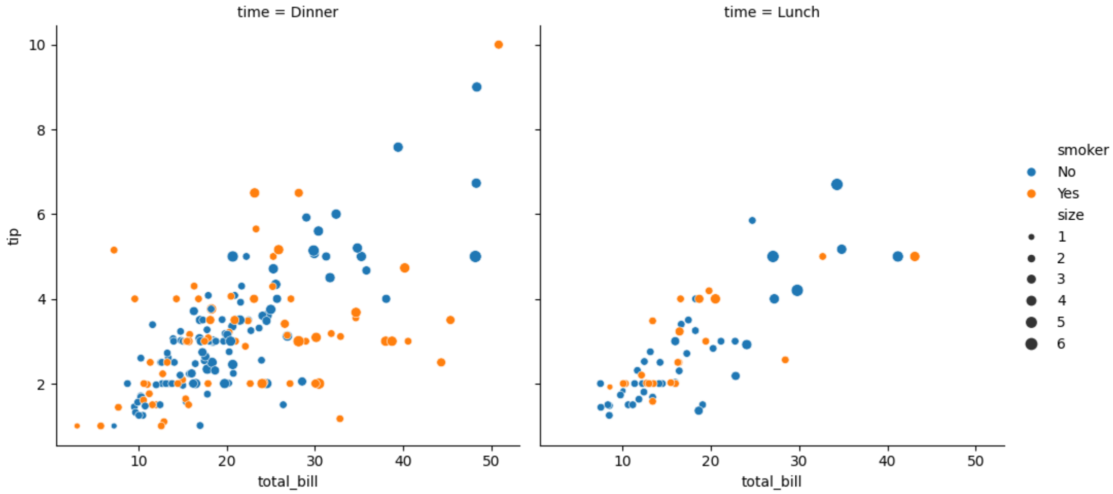
IQR Value

[20]:
```
bill_tip=pd.DataFrame(df,columns=['total_bill','tips','size'])

print(bill_tip)

print("IQR For Total Bill : ",stats.iqr(bill))
print("IQR For Tip : ",stats.iqr(tip))
```

```
     total_bill  tips  size
0         16.99   NaN     2
1         10.34   NaN     3
2         21.01   NaN     3
3         23.68   NaN     2
4         24.59   NaN     4
..          ...   ...   ...
239       29.03   NaN     3
240       27.18   NaN     2
241       22.67   NaN     2
242       17.82   NaN     2
243       18.78   NaN     2

[244 rows x 3 columns]
IQR For Total Bill :  10.779999999999998
IQR For Tip :  1.5625
```

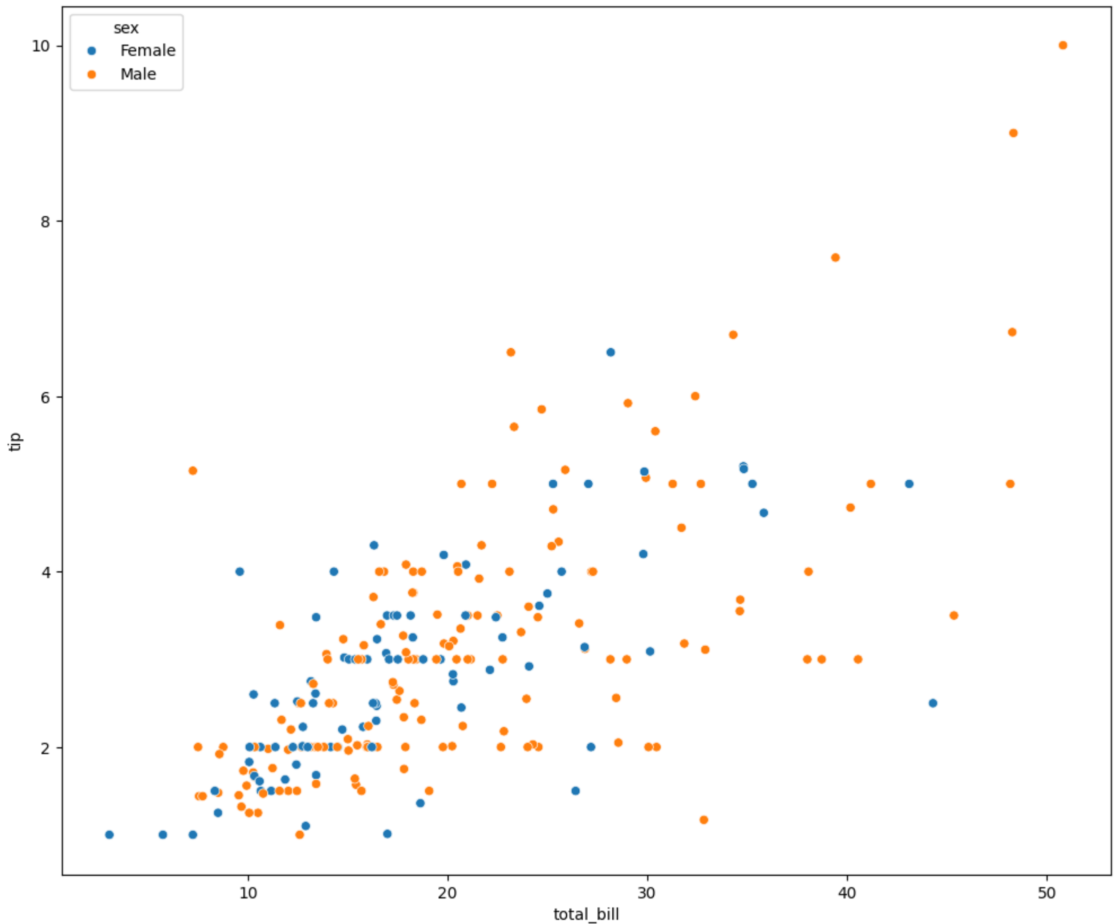
[21]:
```
sb.relplot(x='total_bill',y='tip',data=df,col='time',hue='smoker',size='size')
```

[21]: <seaborn.axisgrid.FacetGrid at 0x7cafd26373a0>



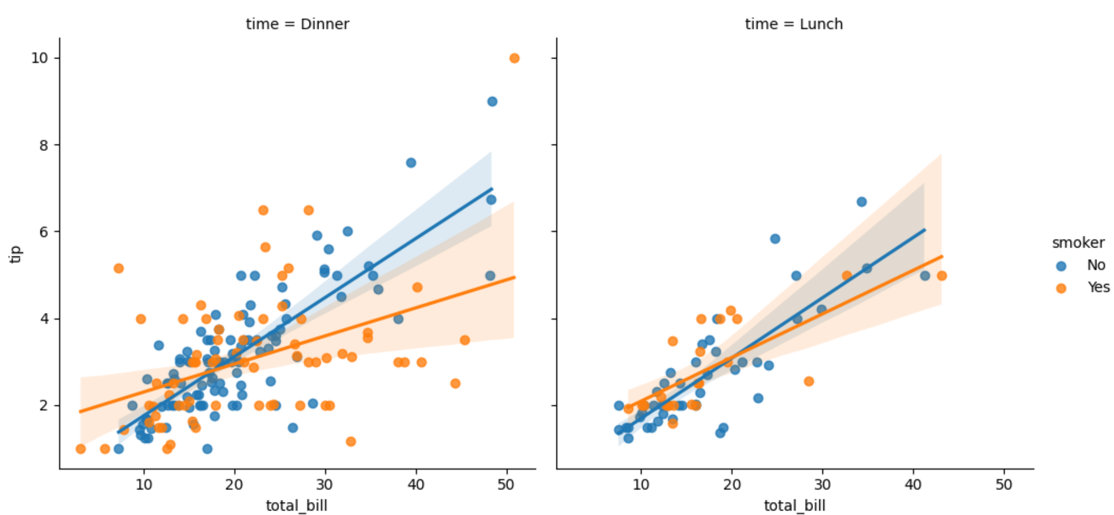
[22]:
```
plt.figure(figsize=(12,10))
sb.scatterplot(data=df,x='total_bill',y="tip",hue="sex")
```

[22]: `<Axes: xlabel='total_bill', ylabel='tip'>`



[27]:
```
sb.lmplot(x='total_bill',y='tip',data=df,col='time',hue='smoker')
```

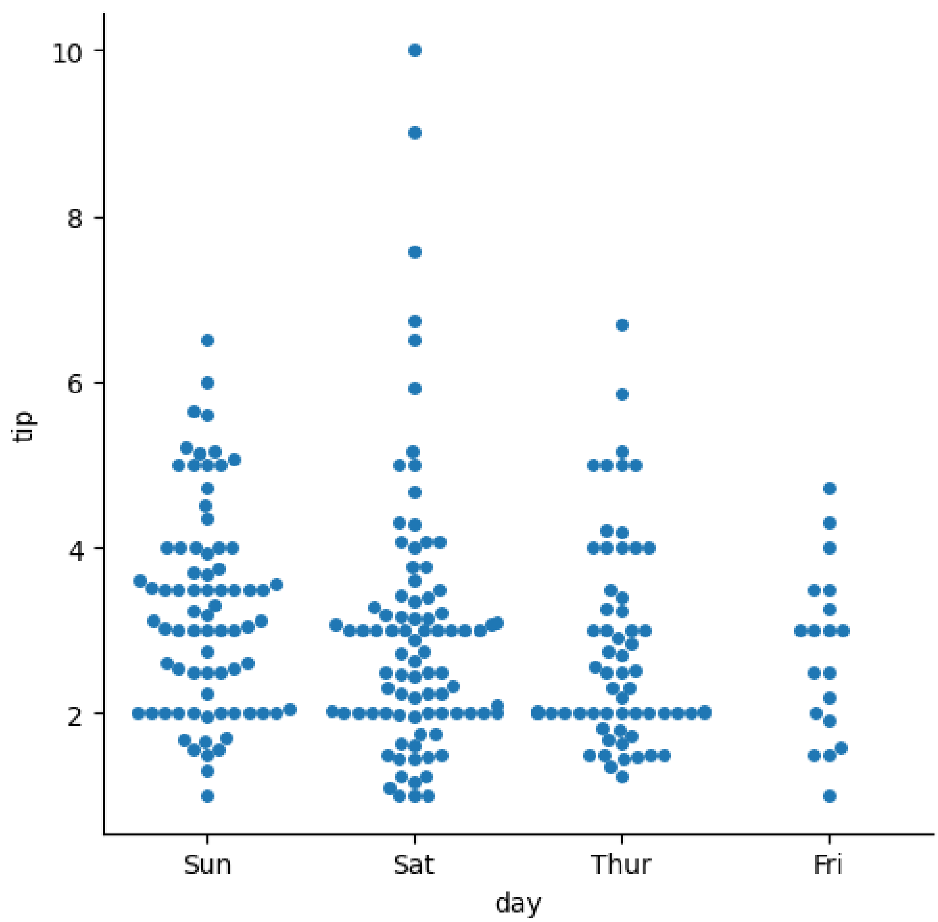
[27]: <seaborn.axisgrid.FacetGrid at 0x7cafd254e1d0>



[30]:
```
sb.catplot(x='day',y='tip',data=df,kind='swarm')
```

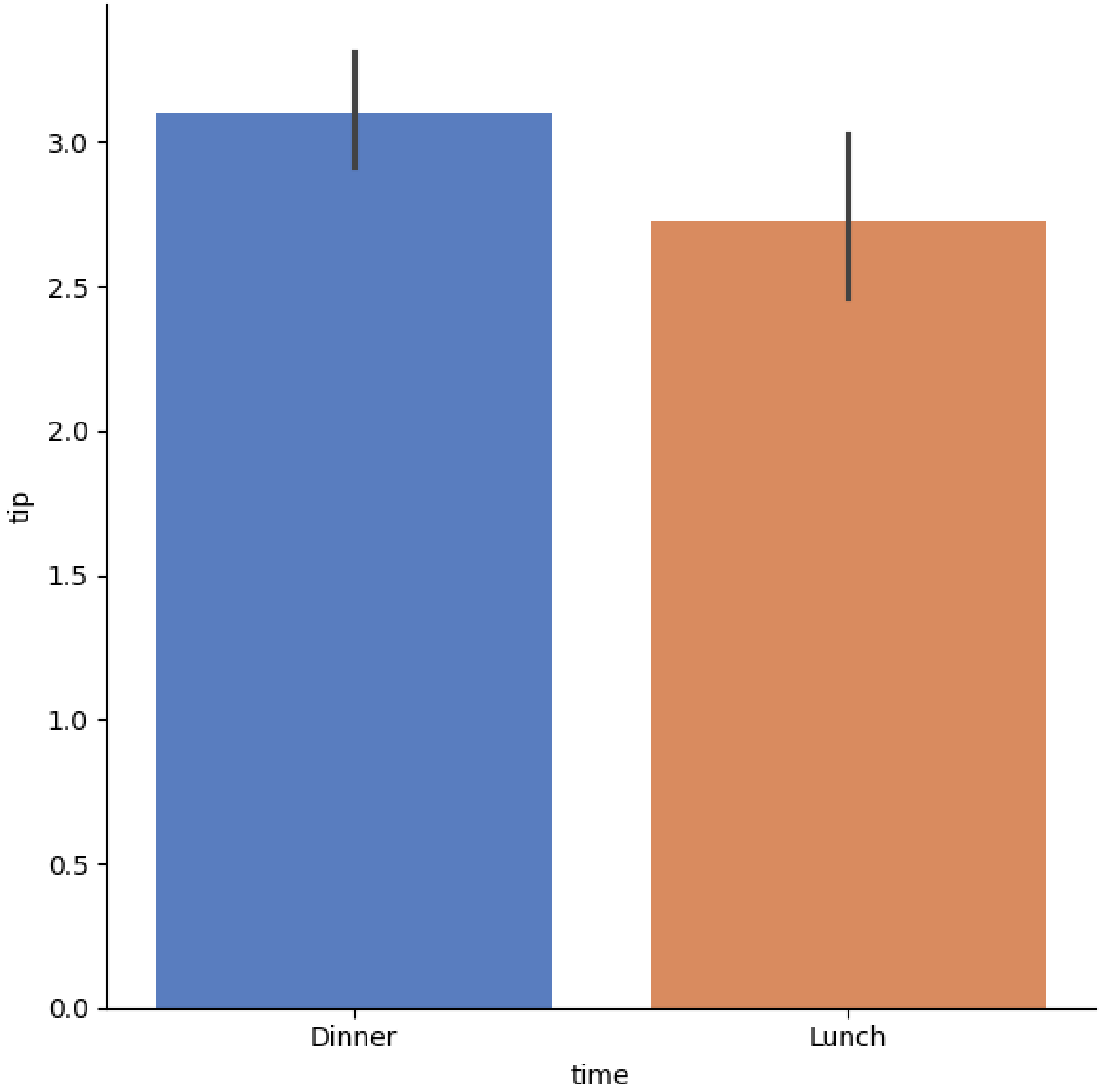
```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3398:
UserWarning: 8.1% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

[30]: <seaborn.axisgrid.FacetGrid at 0x7cafd29ad4b0>



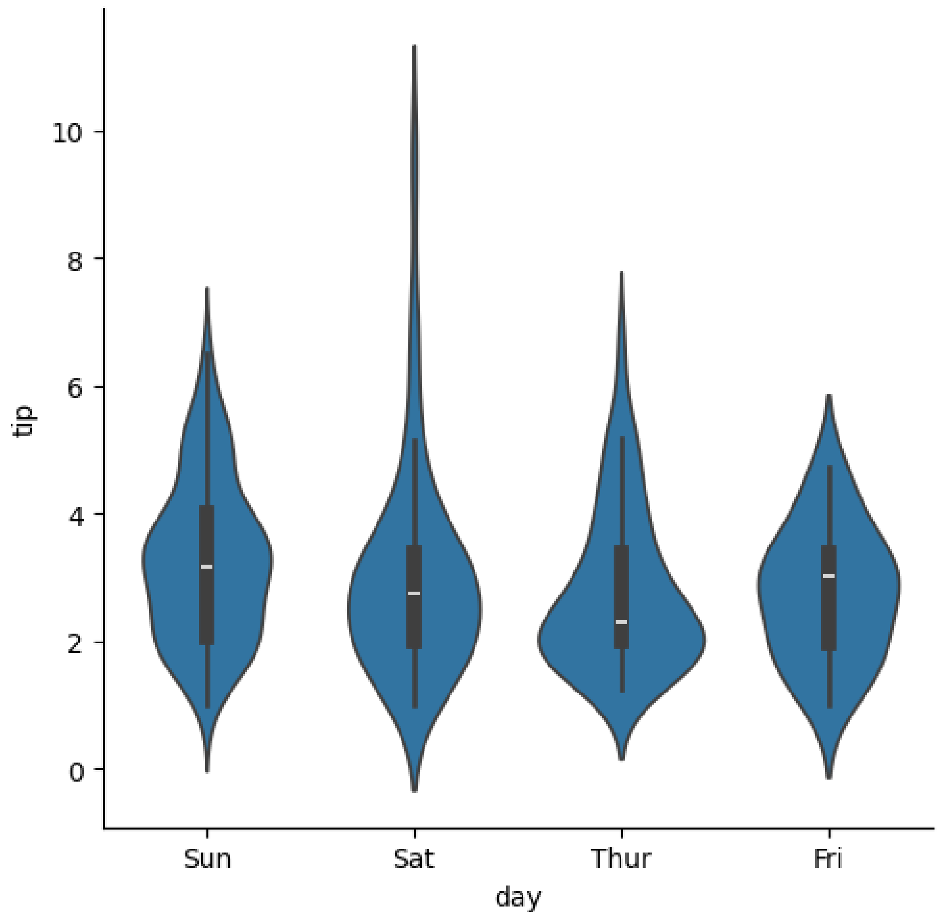
[31]:
```
sb.catplot(x='time',y='tip',data=df,height=6,kind='bar',palette='muted')
```

[31]: <seaborn.axisgrid.FacetGrid at 0x7cafcdfe17b0>



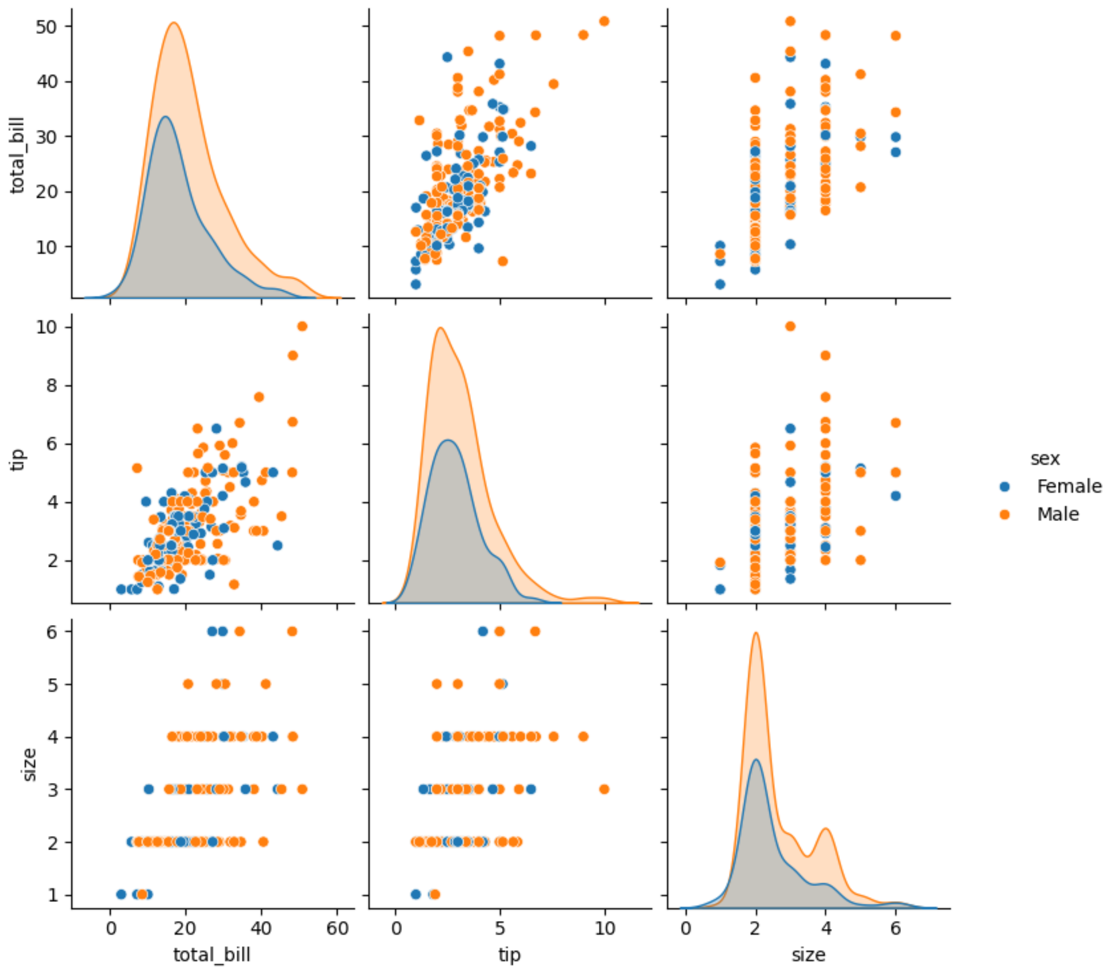
[32]:
```
sb.catplot(x='day',y='tip',data=df,kind='violin')
```

[32]: <seaborn.axisgrid.FacetGrid at 0x7cafd264e6b0>



[33]:
```
sb.pairplot(df,hue='sex')
```

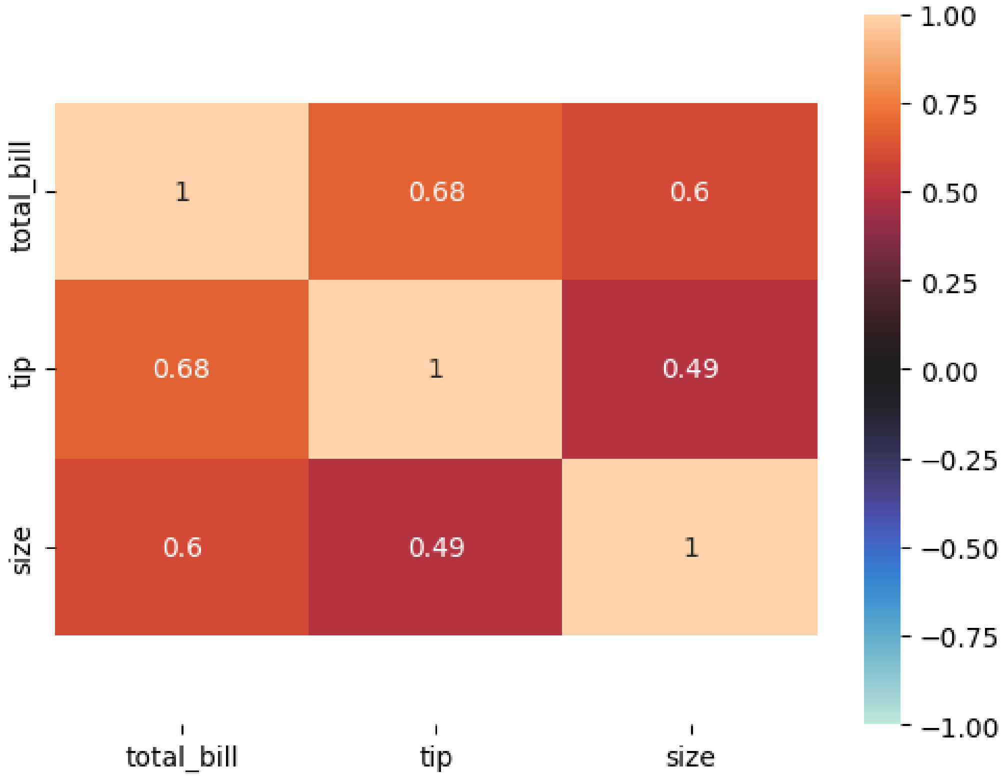
[33]: <seaborn.axisgrid.PairGrid at 0x7cafcde796f0>



Correlation Matrix

```
[39]: corr_matrix=df.corr()
      ax=sb.heatmap(data=corr_matrix,annot=True,vmax=1,vmin=-1,center=0)
      bottom,top=ax.get_ylim()
      ax.set_ylim(bottom + 0.5,top - 0.5)
```

```
[39]: (3.5, -0.5)
```



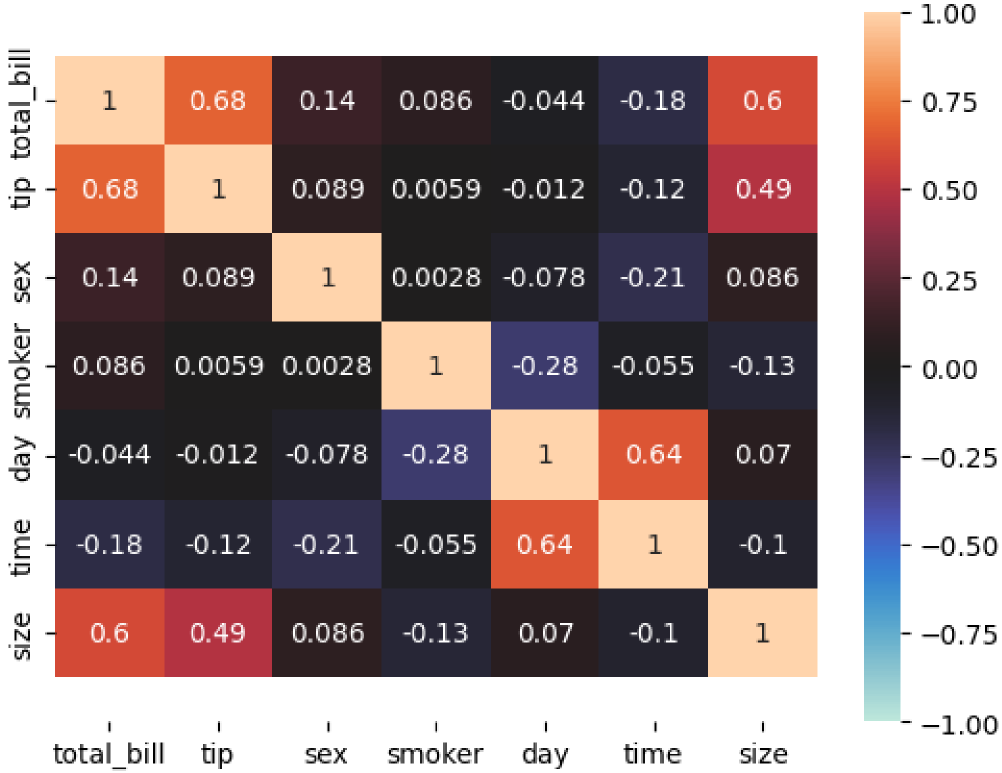
[40]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder_df = LabelEncoder()
df['sex']=labelencoder_df.fit_transform(df['sex'])
df['smoker']=labelencoder_df.fit_transform(df['smoker'])
df['day']=labelencoder_df.fit_transform(df['day'])
df['time']=labelencoder_df.fit_transform(df['time'])
df.head()
```

[40]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	0	0	2	0	2
1	10.34	1.66	1	0	2	0	3
2	21.01	3.50	1	0	2	0	3
3	23.68	3.31	1	0	2	0	2
4	24.59	3.61	0	0	2	0	4

[41]:
```
corr_matrix=df.corr()
ax=sb.heatmap(data=corr_matrix,annot=True,vmax=1,vmin=-1,center=0)
bottom,top=ax.get_ylim()
ax.set_ylim(bottom + 0.5,top - 0.5)
```

[41]: (7.5, -0.5)


total_bill
tip
sex
smoker
day
time
size
1 0.68 0.14 0.086 -0.044 -0.18 0.6
0.68 1 0.089 0.0059 -0.012 -0.12 0.49
0.14 0.089 1 0.0028 -0.078 -0.21 0.086
0.086 0.0059 0.0028 1 -0.28 -0.055 -0.13
-0.044 -0.012 -0.078 -0.28 1 0.64 0.07
-0.18 -0.12 -0.21 -0.055 0.64 1 -0.1
0.6 0.49 0.086 -0.13 0.07 -0.1 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

5.Conclusion :- Data visualization is done on the tips dataset of Seaborn using plots for different types of variables and inferences are made about the relationship between total bill, tip, day, time, gender, smoker or non-smoker etc.

Name : Riya Khot	
Roll No. : 21	
Experiment No. 2	
Apply data cleaning techniques on the given dataset	
Date of Performance:23-01-2024	
Date of Submission:23-01-2024	

Department of Computer Engineering

Academic Year: 2023-24 **Semester: VIII**
Class / Branch: BE Computer **Subject: Applied Data Science Lab**
Name:Riya Khot

Experiment No. 2

1. **Aim:** To apply data cleaning techniques.

Dataset: In this experiment, a fictitious data containing 10 observations and 4 variables is used. The dataset contains Country, Age, Salary, and purchased columns. The dataset has categorical variables and missing values in these columns.

2. **Software used:** Google Colaboratory / Jupyter Notebook

3. **Theory :-**

Data cleaning is just the collective name to a series of actions we perform on our data in the process of getting it ready for analysis.

Some of the steps in data cleaning are:

- Handling missing values
- Encoding categorical features
- Outliers detection
- Transformations etc.

Handling missing values is a key part of data preprocessing and hence, it is of utmost importance for data scientists/machine learning engineers to learn different techniques in relation imputing / replacing numerical or categorical missing values with appropriate value based on appropriate strategies. That's primarily the reason we need to convert categorical columns to numerical columns so that a machine learning algorithm understands it. This process is called categorical encoding.

SimpleImputer is a class found in package sklearn.impute. It is used to impute / replace the numerical or categorical missing data related to one or more features with appropriate values.

Typically, any structured dataset includes multiple columns – a combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too. There are multiple ways of handling Categorical variables.
The two most widely used techniques:

- Label Encoding

- One-Hot Encoding

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

One-Hot Encoding is the process of creating dummy variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

4. Program:

EXPERIMENT 2

Imports

[1]:
```
import numpy as np
import pandas as pd
```

Read CSV file

[2]:
```
df=pd.read_csv("/content/CountryAgeSalary - CountryAgeSalary.csv")
```

[3]:
```
df
```

[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

[4]:
```
df.isnull().sum()
```

[4]:
```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

[5]:
```
x = df.iloc[:,0:2].values
y = df.iloc[:,2].values
type(x)
```

[5]:
```
numpy.ndarray
```

[6]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.NaN,strategy='median')
x[:,1:2]=imputer.fit_transform(x[:,1:2])
print(x)
```

```
[['France' 44.0]
 ['Spain' 27.0]
 ['Germany' 30.0]
 ['Spain' 38.0]
 ['Germany' 40.0]
 ['France' 35.0]
 ['Spain' 38.0]
 ['France' 48.0]
 ['Germany' 50.0]
 ['France' 37.0]]
```

[8]:
```
imputer=SimpleImputer(missing_values=np.NaN,strategy='median')
df.Age=imputer.fit_transform(df["Age"].values.reshape(-1,1))[:,0]
```

[9]:
```
df
```

[9]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	38.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

[10]:
```
imputer=SimpleImputer(missing_values=np.NaN,strategy='mean')
```

[11]:
```
imputer=SimpleImputer(missing_values=np.NaN,strategy='median')
```

[12]:
```
imputer=SimpleImputer(missing_values=np.NaN,strategy='most_frequent')
```

[15]:
```
imputer=SimpleImputer(missing_values=np.NaN,strategy='constant',fill_value=80)
```

[16]:
```
df
```

[16]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No

```
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain  38.0  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

[17]:

```
imputer=SimpleImputer(missing_values=np.NaN,strategy='mean')
df.Salary=imputer.fit_transform(df["Salary"].values.reshape(-1,1))[:,0]
```

[18]:

```
df
```

[18]:

```
   Country   Age        Salary Purchased
0   France  44.0  72000.000000        No
1    Spain  27.0  48000.000000       Yes
2  Germany  30.0  54000.000000        No
3    Spain  38.0  61000.000000        No
4  Germany  40.0  63777.777778       Yes
5   France  35.0  58000.000000       Yes
6    Spain  38.0  52000.000000        No
7   France  48.0  79000.000000       Yes
8  Germany  50.0  83000.000000        No
9   France  37.0  67000.000000       Yes
```

5.Conclusion :-

Sklearn.impute class SimpleImputer can be used to impute/replace missing values for both numerical and categorical features. For numerical missing values, a strategy such as mean, median, most frequent, and constant can be used. For categorical features, a strategy such as the most frequent and constant can be used. Categorical variables can be converted into numerical using label encoding or one-hot encoding.

Name : Riya Khot
Roll No. : 21
Experiment No. 3
Explore Inferential Statistic on the given dataset
Date of Performance:16-02-2024
Date of Submission:16-02-2024

Aim: Explore Inferential Statistic on the given dataset

Objective: Able to perform various inferential statistics on the given dataset.

Theory:

Z-Test & T-Tests are Parametric Tests, where the Null Hypothesis is less than, greater than or equal to some value. • A z-test is used if the population variance is known, or if the sample size is larger than 30, for an unknown population variance. • If the sample size is less than 30 and the population variance is unknown, we must use a t-test. T test is a type of inferential statistic used to study if there is a statistical difference between two groups. Mathematically, it establishes the problem by assuming that the means of the two distributions are equal (H_0: $\mu_1=\mu_2$). If the t-test rejects the null hypothesis (H_0: $\mu_1=\mu_2$), it indicates that the groups are highly probably different. The statistical test can be one-tailed or two-tailed. The one-tailed test is appropriate when there is a difference between groups in a specific direction. It is less common than the two-tailed test. When choosing a t test, you will need to consider two things: whether the groups being compared come from a single population or two different populations, and whether you want to test the difference in a specific direction.

There are three main types of t-test :

• One Sample t-test : Compares mean of a single group against a known/hypothesized/ population mean.

• Two Sample: Paired Sample T Test: Compares means from the same group at different times.

• Two Sample: Independent Sample T Test: Compares means for two different groups.

One Sample t-test:

$$t=\frac{\text{(Sample Mean} - \text{Population Mean)}}{\text{Standard Error}}$$

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

$\bar{x}$ Sample mean
μ Population mean
s Sample standard deviation
n Sample size

Two-sample - Paired Sample t-test

$$t = \frac{\bar{d}}{s/\sqrt{n}}$$

$\bar{d}$ =Mean of the difference
s=Standard deviation of the difference
n =is the sample size (i.e., size of d)

If the calculated t value is less than critical t value or greater that the critical value (obtained from a critical value table called the T-distribution table) then reject the null hypothesis.

P-value <significance level (α) => Reject your null hypothesis in favor of your alternative hypothesis. Your result is statistically significant.

P-value >= significance level (α) => Fail to reject your null hypothesis. Your result is not statistically significant.

expt3

February 16, 2024

RelianceDataMart Dataset

[100]:
```
import numpy as np
import pandas as pd
from scipy import stats
```

[101]:
```
RDM=pd.read_excel('/content/RelianceDataMart.xlsx')
RDM
```

[101]:

	Rice_Bag_Weight
0	24.50
1	24.70
2	25.60
3	25.00
4	24.70
5	23.30
6	23.30
7	24.00
8	25.10
9	24.30
10	23.30
11	24.10
12	24.10
13	24.20
14	25.20
15	24.90
16	24.70
17	24.10
18	25.00
19	24.70
20	24.90
21	25.00
22	24.00
23	23.98
24	24.30
25	24.20
26	24.56
27	24.50

```
28             24.70
```

[102]:
```
print(RDM.mean())
```

```
Rice_Bag_Weight    24.446207
dtype: float64
```

[103]:
```
RDM.describe()
```

[103]:

	Rice_Bag_Weight
count	29.000000
mean	24.446207
std	0.569463
min	23.300000
25%	24.100000
50%	24.500000
75%	24.900000
max	25.600000

[104]:
```
one_sample_result=stats.ttest_1samp(RDM,24.446)
print(one_sample_result)
```

```
TtestResult(statistic=array([0.00195653]), pvalue=array([0.99845279]),
df=array([28]))
```

Crocin_data Dataset

[105]:
```
crocin_data=pd.read_excel('/content/Crocin_Data_ST.xlsx')
crocin_data
```

[105]:

	Before_Crocin	After_Crocin	diff	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	101.0	99	2.000000	NaN	NaN	NaN
1	99.0	98	1.000000	NaN	NaN	NaN
2	101.0	97	4.000000	NaN	NaN	NaN
3	99.9	99	0.900000	NaN	NaN	NaN
4	99.8	98	1.800000	NaN	NaN	NaN
5	98.0	97	1.000000	NaN	NaN	NaN
6	97.0	99	-2.000000	NaN	NaN	NaN
7	101.0	98	3.000000	NaN	NaN	NaN
8	102.0	96	6.000000	NaN	NaN	NaN
9	103.0	98	5.000000	NaN	NaN	NaN
10	99.0	94	5.000000	NaN	NaN	NaN
11	99.9	96	3.900000	NaN	NaN	NaN
12	99.8	97	2.800000	NaN	NaN	NaN
13	99.7	99	0.700000	NaN	NaN	NaN
14	101.1	98	3.100000	NaN	NaN	NaN
15	102.3	97	5.300000	NaN	NaN	NaN
16	101.0	99	2.000000	NaN	NaN	NaN

17	99.0	98	1.000000	NaN	NaN	NaN
18	101.0	97	4.000000	NaN	NaN	NaN
19	99.9	99	0.900000	NaN	NaN	NaN
20	99.8	98	1.800000	NaN	NaN	NaN
21	98.0	96	2.000000	NaN	NaN	NaN
22	97.0	97	0.000000	NaN	NaN	NaN
23	101.0	99	2.000000	NaN	NaN	NaN
24	102.0	97	5.000000	NaN	NaN	NaN
25	103.0	99	4.000000	NaN	NaN	NaN
26	99.0	98	1.000000	NaN	NaN	NaN
27	99.9	97	2.900000	NaN	NaN	NaN
28	99.8	99	0.800000	NaN	NaN	NaN
29	NaN	mean	2.444828	NaN	t val	7.071713
30	NaN	std dev	1.861755	NaN	NaN	NaN
31	NaN	sq root n	5.385165	NaN	NaN	NaN

[106]:
```
crocin_data=crocin_data.drop("Unnamed: 3",axis=1)
crocin_data=crocin_data.drop("Unnamed: 4",axis=1)
crocin_data=crocin_data.drop("Unnamed: 5",axis=1)
```

[107]:
```
crocin_data
```

[107]:

	Before_Crocin	After_Crocin	diff
0	101.0	99	2.000000
1	99.0	98	1.000000
2	101.0	97	4.000000
3	99.9	99	0.900000
4	99.8	98	1.800000
5	98.0	97	1.000000
6	97.0	99	-2.000000
7	101.0	98	3.000000
8	102.0	96	6.000000
9	103.0	98	5.000000
10	99.0	94	5.000000
11	99.9	96	3.900000
12	99.8	97	2.800000
13	99.7	99	0.700000
14	101.1	98	3.100000
15	102.3	97	5.300000
16	101.0	99	2.000000
17	99.0	98	1.000000
18	101.0	97	4.000000
19	99.9	99	0.900000
20	99.8	98	1.800000
21	98.0	96	2.000000
22	97.0	97	0.000000
23	101.0	99	2.000000

	Before_Crocin	After_Crocin	diff
24	102.0	97	5.000000
25	103.0	99	4.000000
26	99.0	98	1.000000
27	99.9	97	2.900000
28	99.8	99	0.800000
29	NaN	mean	2.444828
30	NaN	std dev	1.861755
31	NaN	sq root n	5.385165

[108]:
```
crocin_data=crocin_data.iloc[:29]
```

[109]:
```
crocin_data
```

[109]:

	Before_Crocin	After_Crocin	diff
0	101.0	99	2.0
1	99.0	98	1.0
2	101.0	97	4.0
3	99.9	99	0.9
4	99.8	98	1.8
5	98.0	97	1.0
6	97.0	99	-2.0
7	101.0	98	3.0
8	102.0	96	6.0
9	103.0	98	5.0
10	99.0	94	5.0
11	99.9	96	3.9
12	99.8	97	2.8
13	99.7	99	0.7
14	101.1	98	3.1
15	102.3	97	5.3
16	101.0	99	2.0
17	99.0	98	1.0
18	101.0	97	4.0
19	99.9	99	0.9
20	99.8	98	1.8
21	98.0	96	2.0
22	97.0	97	0.0
23	101.0	99	2.0
24	102.0	97	5.0
25	103.0	99	4.0
26	99.0	98	1.0
27	99.9	97	2.9
28	99.8	99	0.8

[110]:
```
two_sample_result=stats.ttest_rel(crocin_data ["Before_Crocin"], crocin_data
    ["After_Crocin"])
```

[111]:
```
two_sample_result
```

[111]:
```
TtestResult(statistic=7.071712959273876, pvalue=1.0800112658101922e-07, df=28)
```

Pre_post_score Dataset

[112]:
```
pre_post_score=pd.read_excel('/content/Pre_Post_Score.xlsx')
```

[113]:
```
pre_post_score
```

[113]:
```
    Pre_Score    Post_Score      Diff  Unnamed: 3  Unnamed: 4 Unnamed: 5  \
0        18.0            22 -4.000000         NaN         NaN        NaN
1        21.0            25 -4.000000         NaN         NaN        NaN
2        16.0            17 -1.000000         NaN         NaN        NaN
3        22.0            24 -2.000000         NaN         NaN        NaN
4        19.0            16  3.000000         NaN         NaN        NaN
5        24.0            29 -5.000000         NaN         NaN        NaN
6        17.0            20 -3.000000         NaN         NaN        NaN
7        21.0            23 -2.000000         NaN         NaN        NaN
8        23.0            19  4.000000         NaN         NaN        NaN
9        18.0            20 -2.000000         NaN         NaN        NaN
10       14.0            15 -1.000000         NaN         NaN        NaN
11       16.0            15  1.000000         NaN         NaN        NaN
12       16.0            18 -2.000000         NaN         NaN        NaN
13       19.0            26 -7.000000         NaN         NaN        NaN
14       18.0            18  0.000000         NaN         NaN        NaN
15       20.0            24 -4.000000         NaN         NaN        NaN
16       12.0            18 -6.000000         NaN         NaN        NaN
17       22.0            25 -3.000000         NaN         NaN     t val=
18       15.0            19 -4.000000         NaN         NaN        NaN
19       17.0            16  1.000000         NaN         NaN        NaN
20        NaN          mean -2.050000         NaN         NaN        NaN
21        NaN       std dev  2.837252         NaN         NaN        NaN
22        NaN  sq root  of n  4.472136         NaN         NaN        NaN

    Unnamed: 6
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
5          NaN
6          NaN
7          NaN
8          NaN
9          NaN
10         NaN
```

```
11          NaN
12          NaN
13          NaN
14          NaN
15          NaN
16          NaN
17    -3.231253
18          NaN
19          NaN
20          NaN
21          NaN
22          NaN
```

[114]:
```
pre_post_score=pre_post_score.drop("Unnamed: 3",axis=1)
pre_post_score=pre_post_score.drop("Unnamed: 4",axis=1)
pre_post_score=pre_post_score.drop("Unnamed: 5",axis=1)
pre_post_score=pre_post_score.drop("Unnamed: 6",axis=1)
```

[115]:
```
pre_post_score
```

[115]:

	Pre_Score	Post_Score	Diff
0	18.0	22	-4.000000
1	21.0	25	-4.000000
2	16.0	17	-1.000000
3	22.0	24	-2.000000
4	19.0	16	3.000000
5	24.0	29	-5.000000
6	17.0	20	-3.000000
7	21.0	23	-2.000000
8	23.0	19	4.000000
9	18.0	20	-2.000000
10	14.0	15	-1.000000
11	16.0	15	1.000000
12	16.0	18	-2.000000
13	19.0	26	-7.000000
14	18.0	18	0.000000
15	20.0	24	-4.000000
16	12.0	18	-6.000000
17	22.0	25	-3.000000
18	15.0	19	-4.000000
19	17.0	16	1.000000
20	NaN	mean	-2.050000
21	NaN	std dev	2.837252
22	NaN	sq root of n	4.472136

[116]:
```
pre_post_score=pre_post_score.iloc[:20]
```

[117]:
```
two_sample_result=stats.ttest_rel (pre_post_score ["Pre_Score"], pre_post_score␣
↪["Post_Score"])
```

[118]:
```
two_sample_result
```

[118]:
```
TtestResult(statistic=-3.231252665580312, pvalue=0.004394965993185664, df=19)
```

Conclusion:

One sample t-test has been done on the reliance data mart dataset and it has been found that a difference exists between the rice bag population mean and rice bag sample mean. Two sample paired t-test have been done on the prescore-post score dataset and Crocin dataset. In the prescore-post score dataset difference exists between the mean pre-score before studying the module and mean prescore after studying the module. In the crocin dataset it is found that temperature difference exists before and after having the crocin tablet.

Name : Riya Khot	
Roll No. : 21	
Experiment No. 1	
Explore the descriptive statistics on the given dataset	
Date of Performance:08-01-2024	
Date of Submission:08-01-2024	

Academic Year: 2023-24 **Semester: VIII**
Class / Branch: BE Computer **Subject: Applied Data Science Lab**
Name : Riya Khot

Experiment No. 1

1. **Aim: Explore the descriptive statistics on the given dataset.**
   Dataset: In this experiment, fictitious data of Body Mass Index(BMI) containing 10 observations and 5 variables is used. The dataset contains Height, Weight, Age, BMI, and Gender columns.

2. **Software used:** Google Colaboratory/ Jupyter Notebook

3. **Theory :-**

Descriptive Statistics:
Descriptive statistics can be defined as the measures that summarize a given data, and these measures can be broken down further
1. Measure of central tendency
2. Measure of spread/dispersion
3. Measure of symmetry/shape

Measure of Central Tendency
Measure of central tendency is used to describe the middle/centre value of the data. Mean, Median, Mode are measures of central tendency.

1. Mean
- Mean is the average value of the dataset.
- Mean is calculated by adding all values in the dataset divided by the number of values in the dataset.
- We can calculate the mean for only numerical variables.

2. Median
- The Median is the middle number in the dataset.
- Median is the best measure when we have outliers.

3. Mode
The mode is used to find the common number in the dataset.

Measure of spread
- The measure of spread/dispersion is used to describe how data is spread. It also describes the **variability** of the dataset.
- **Standard Deviation, Variance, Range, IQR,** are used to describe the measure of spread/dispersion
- The measure of spread can be shown in graphs like **boxplot**.

1. Variance

- Variance is used to describe how far each number in the dataset is from the mean.
- Formula to calculate population variance

$$\sigma^2 = \frac{\sum (x-\mu)^2}{N}$$

2. Standard Deviation

- Standard Deviation is the measure of the spread of data from the mean.
- Standard deviation is the square root of variance.
- More the standard deviation, more the spread.

3. Range

- The range is the difference between the largest number and the smallest number.
- Larger the range, the more the dispersion.

4. Interquartile range (IQR)

- Quartiles describe the spread of data by breaking into quarters. The median exactly divides the data into two parts.
- **Q1(Lower quartile)** is the middle value in the first half of the sorted dataset.
- **Q2**– is the median value
- **Q3 (Upper quartile)** is the middle value in the second half of the sorted dataset
- The interquartile range is the difference between the 75th percentile(Q3) and the 25th percentile(Q1).
- 50% of data fall within this range.

Boxplot is used to describe how the data is distributed in the dataset. This graph represents five-point summary (minimum, maximum, median, lower quartile, and upper quartile) and is used to identify **outliers**.

- whiskers — denote the spread of data
- box— represents the IQR- 50% of data lies within this range.

Measure of shape

1. Skewness

Skewness, which is the measure of the symmetry, or lack of it, for a real-valued random variable about its mean. The skewness value can be positive, negative, or undefined. In a perfectly symmetrical distribution, the mean, the median, and the mode will all have the same value.

2.*Kurtosis*

Kurtosis provides a measurement about the extremities (i.e. tails) of the distribution of data, and therefore provides an indication of the presence of outliers. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers.

4.Program :

adse1

January 8, 2024

Imports

[107]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from scipy import stats
```

Import CSV File

[108]:
```
df=pd.read_csv("/content/bmi - bmi.csv")
df
```

[108]:

	Gender	Height	Weight	bmi	Age
0	Male	174	80	26.4	25
1	Male	189	87	24.4	27
2	Female	185	80	23.4	30
3	Female	165	70	25.7	26
4	Male	149	61	27.5	28
5	Male	177	70	22.3	29
6	Female	147	65	30.1	31
7	Male	154	62	26.1	32
8	Male	174	90	29.7	27

[109]:
```
df.mean()
```

```
<ipython-input-109-c61f0c8f89b5>:1: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  df.mean()
```

[109]:
```
Height    168.222222
Weight     73.888889
bmi        26.177778
Age        28.333333
dtype: float64
```

[110]:
```
df.median()
```

```
<ipython-input-110-6d467abf240d>:1: FutureWarning: The default value of
numeric_only in DataFrame.median is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  df.median()
```

[110]:
```
Height    174.0
Weight     70.0
bmi        26.1
Age        28.0
dtype: float64
```

[111]:
```
df.mode()
```

[111]:

	Gender	Height	Weight	bmi	Age
0	Male	174.0	70.0	22.3	27.0
1	NaN	NaN	80.0	23.4	NaN
2	NaN	NaN	NaN	24.4	NaN
3	NaN	NaN	NaN	25.7	NaN
4	NaN	NaN	NaN	26.1	NaN
5	NaN	NaN	NaN	26.4	NaN
6	NaN	NaN	NaN	27.5	NaN
7	NaN	NaN	NaN	29.7	NaN
8	NaN	NaN	NaN	30.1	NaN

[112]:
```
df.describe()
```

[112]:

	Height	Weight	bmi	Age
count	9.000000	9.000000	9.000000	9.000000
mean	168.222222	73.888889	26.177778	28.333333
std	15.368619	10.740629	2.639497	2.345208
min	147.000000	61.000000	22.300000	25.000000
25%	154.000000	65.000000	24.400000	27.000000
50%	174.000000	70.000000	26.100000	28.000000
75%	177.000000	80.000000	27.500000	30.000000
max	189.000000	90.000000	30.100000	32.000000

[113]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  9 non-null      object
```

```
 1   Height  9 non-null      int64
 2   Weight  9 non-null      int64
 3   bmi     9 non-null      float64
 4   Age     9 non-null      int64
dtypes: float64(1), int64(3), object(1)
memory usage: 488.0+ bytes
```

[114]:
```
print("Mean of Age : ",df["Age"].mean())
print("Median of Age : ",df["Age"].median())
print("Mode of Age : ",df["Age"].mode())
```

```
Mean of Age :  28.333333333333332
Median of Age :  28.0
Mode of Age :  0    27
Name: Age, dtype: int64
```

[115]:
```
print("Mean of Weight : ",df["Weight"].mean())
print("Median of Weight : ",df["Weight"].median())
print("Mode of Weight : ",df["Weight"].mode())
```

```
Mean of Weight :  73.88888888888889
Median of Weight :  70.0
Mode of Weight :  0    70
1    80
Name: Weight, dtype: int64
```

[116]:
```
print("Mean of Height : ",df["Height"].mean())
print("Median of Height : ",df["Height"].median())
print("Mode of Height : ",df["Height"].mode())
```

```
Mean of Height :  168.22222222222223
Median of Height :  174.0
Mode of Height :  0    174
Name: Height, dtype: int64
```

[117]:
```
print("Mean of bmi : ",df["bmi"].mean())
print("Median of bmi : ",df["bmi"].median())
print("Mode of bmi : ",df["bmi"].mode())
```

```
Mean of bmi :  26.177777777777774
Median of bmi :  26.1
Mode of bmi :  0    22.3
1    23.4
2    24.4
3    25.7
4    26.1
5    26.4
6    27.5
```

```
7    29.7
8    30.1
Name: bmi, dtype: float64
```

[118]:
```
print("Describe Age: \n",df["Age"].describe())
print("\nStandard Deviation : ",df["Age"].std())
print("\nVariance : ",df["Age"].var())
print("\nMinimum : ",df["Age"].min())
print("\nMaximum : ",df["Age"].max())
```

```
Describe Age:
 count     9.000000
mean     28.333333
std       2.345208
min      25.000000
25%      27.000000
50%      28.000000
75%      30.000000
max      32.000000
Name: Age, dtype: float64

Standard Deviation :  2.345207879911715

Variance :  5.5

Minimum :  25

Maximum :  32
```

[119]:
```
print("Describe Weight: \n",df["Weight"].describe())
print("\nStandard Deviation : ",df["Weight"].std())
print("\nVariance : ",df["Weight"].var())
print("\nMinimum : ",df["Weight"].min())
print("\nMaximum : ",df["Weight"].max())
```

```
Describe Weight:
 count     9.000000
mean     73.888889
std      10.740629
min      61.000000
25%      65.000000
50%      70.000000
75%      80.000000
max      90.000000
Name: Weight, dtype: float64

Standard Deviation :  10.740628990478683
```

```
Variance :  115.36111111111113

Minimum :  61

Maximum :  90
```

[120]:
```
print("Describe Height: \n",df["Height"].describe())
print("\nStandard Deviation : ",df["Height"].std())
print("\nVariance : ",df["Height"].var())
print("\nMinimum : ",df["Height"].min())
print("\nMaximum : ",df["Height"].max())
```

```
Describe Height:
 count      9.000000
mean     168.222222
std       15.368619
min      147.000000
25%      154.000000
50%      174.000000
75%      177.000000
max      189.000000
Name: Height, dtype: float64

Standard Deviation :  15.36861882032489

Variance :  236.19444444444443

Minimum :  147

Maximum :  189
```

[121]:
```
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
```

```
<ipython-input-121-6355bfef3137>:1: FutureWarning: The default value of
numeric_only in DataFrame.quantile is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  Q1=df.quantile(0.25)
<ipython-input-121-6355bfef3137>:2: FutureWarning: The default value of
numeric_only in DataFrame.quantile is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  Q3=df.quantile(0.75)
```

[122]:
```
print("AGE")
q1=df["Age"].quantile(0.25)
print("q1 = ",q1)
```

```
q3=df["Age"].quantile(0.75)
print("q3 = ",q3)
IQR=q3-q1
print("IQR = ",IQR)
```

```
AGE
q1 =  27.0
q3 =  30.0
IQR =  3.0
```

[123]:
```
print("WEIGHT")
q1=df["Weight"].quantile(0.25)
print("q1 = ",q1)
q3=df["Weight"].quantile(0.75)
print("q3 = ",q3)
IQR=q3-q1
print("IQR = ",IQR)
```

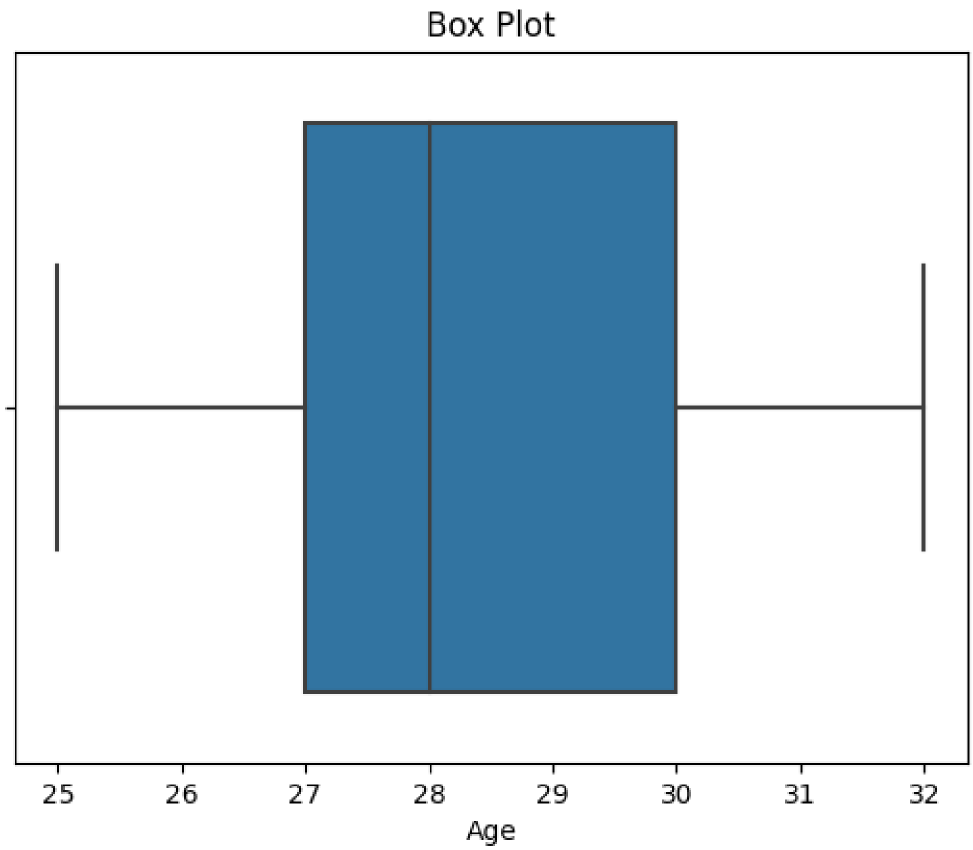
```
WEIGHT
q1 =  65.0
q3 =  80.0
IQR =  15.0
```

[124]:
```
print("HEIGHT")
q1=df["Height"].quantile(0.25)
print("q1 = ",q1)
q3=df["Height"].quantile(0.75)
print("q3 = ",q3)
IQR=q3-q1
print("IQR = ",IQR)
```

```
HEIGHT
q1 =  154.0
q3 =  177.0
IQR =  23.0
```

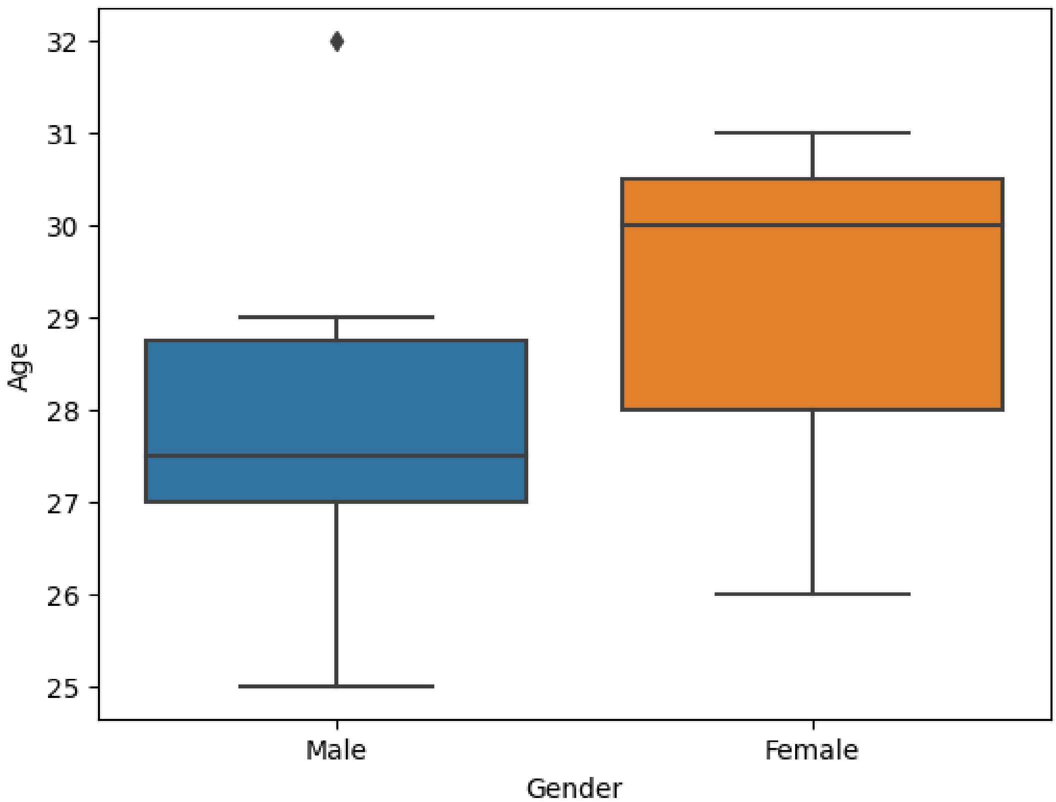
[125]:
```
sb.boxplot(x="Age", data=df)
plt.title("Box Plot")
```

[125]:
```
Text(0.5, 1.0, 'Box Plot')
```



[126]:
```
sb.boxplot(x='Gender', y='Age', data=df)
```

[126]: <Axes: xlabel='Gender', ylabel='Age'>



[127]:
```
sd=df["Age"].std()
Kurtosis=df["Age"].kurtosis()
Skew=df["Age"].skew()
mean=df["Age"].mean()
DQ=sd/mean
Harmonic_Mean=stats.hmean(df["Age"])
Risk=Harmonic_Mean/mean
print("SD : ",sd,"\nKurtosis : ",Kurtosis,"\nSkew : ",Skew,"\nDQ : ",DQ,"\nRisk␣
↪: ",Risk)
zscore=stats.zscore(df["Age"])
print("Z-Score :")
print(zscore)
```

```
SD :  2.345207879911715
Kurtosis :  -1.041322314049585
Skew :  0.232582599660668
DQ :  0.08277204282041346
Risk :  0.993970111565506
Z-Score :
0   -1.507557
1   -0.603023
```

```
2    0.753778
3   -1.055290
4   -0.150756
5    0.301511
6    1.206045
7    1.658312
8   -0.603023
Name: Age, dtype: float64
```

5. Conclusion :- Measures of central tendency help us to understand the center or average of a dataset.The mean is the sum of all values divided by the number of values, giving an overall average. The median is the middle value when the numbers are arranged, and the mode is the value that appears most frequently.Measures of dispersion quantify the spread or variability of a set of data points. They provide insights into how much the values in a dataset deviate from the central tendency measures.Measures of shape, also known as measures of skewness and kurtosis, provide information about the asymmetry and peakedness of a probability distribution. These measures help describe the shape of the distribution beyond what central tendency and dispersion measures offer.