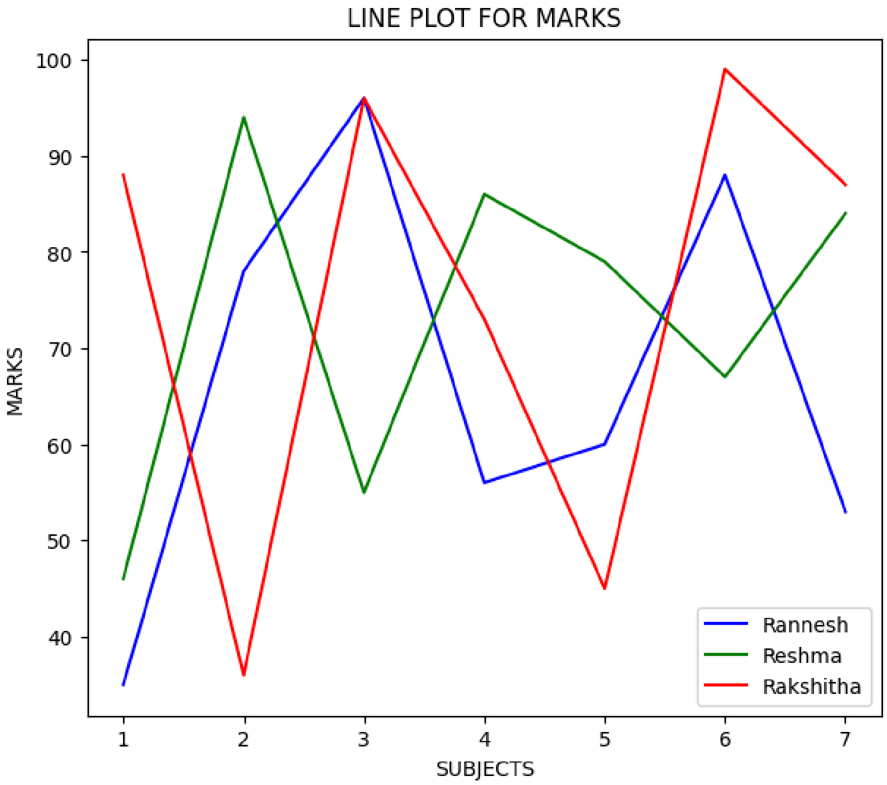
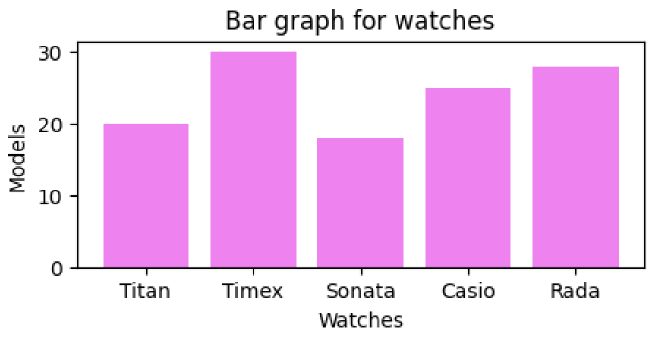
```
#RAKSHITHA R
#240701418
#Fundamentals of Data Science
#17.07.2025
#LinePlot,Bargraph,Piechart,Histogram and Scatter plot using matplotlib
```

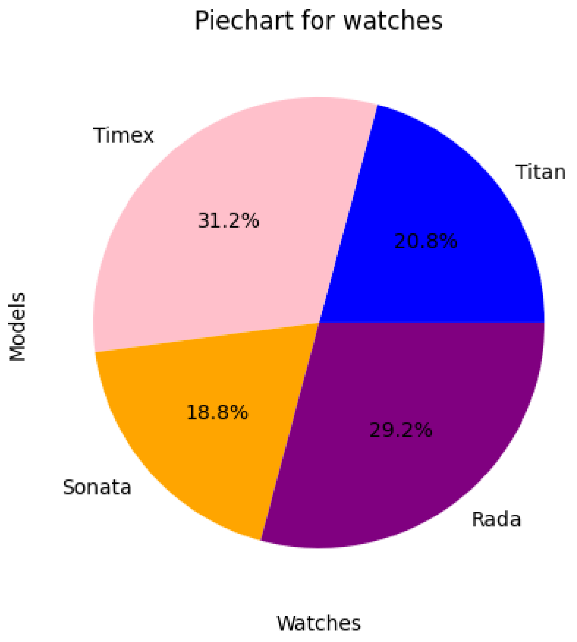
```
import matplotlib.pyplot as plt
sub=list(range(1,8,1))
p1=[35,78,96,56,60,88,53]
p2=[46,94,55,86,79,67,84]
p3=[88,36,96,73,45,99,87]
plt.figure(figsize=(7,6))
plt.plot(sub,p1,color='blue',label='Rannesh')
plt.plot(sub,p2,color='green',label='Reshma')
plt.plot(sub,p3,color='red',label='Rakshitha')
plt.title('LINE PLOT FOR MARKS')
plt.xlabel('SUBJECTS')
plt.ylabel('MARKS')
plt.legend(loc='lower right')
plt.show()
```



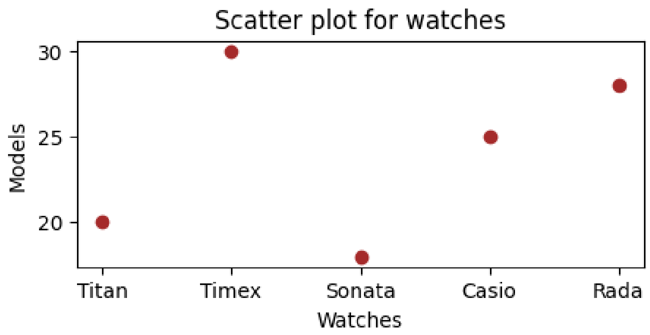
```
import matplotlib.pyplot as plt
watches=['Titan','Timex','Sonata','Casio','Rada']
models=[20,30,18,25,28]
plt.figure(figsize=(5,2))
plt.bar(watches,models,color='violet')
plt.title("Bar graph for watches")
plt.xlabel("Watches")
plt.ylabel("Models")
plt.show()
```



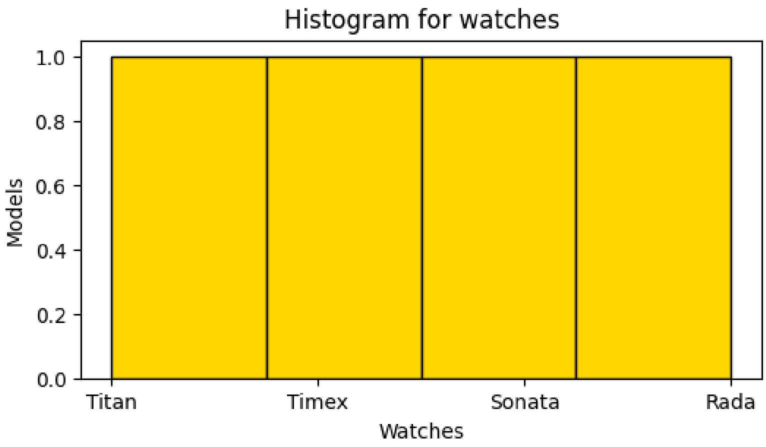
```
import matplotlib.pyplot as plt
watches=['Titan','Timex','Sonata','Rada']
models=[20,30,18,28]
color=['blue','pink','orange','purple']
plt.figure(figsize=(5,5))
plt.pie(models,labels=watches,colors=color,autopct='%1.1f%%')
plt.title("Piechart for watches")
plt.xlabel("Watches")
plt.ylabel("Models")
plt.show()
```



```
import matplotlib.pyplot as plt
watches=['Titan','Timex','Sonata','Casio','Rada']
models=[20,30,18,25,28]
plt.figure(figsize=(5,2))
plt.scatter(watches,models,color='brown')
plt.title("Scatter plot for watches")
plt.xlabel("Watches")
plt.ylabel("Models")
plt.show()
```



```
import matplotlib.pyplot as plt
watches=['Titan','Timex','Sonata','Rada']
models=[20,15,18,12]
color=['blue','pink','orange','purple']
plt.figure(figsize=(6,3))
plt.hist(watches,bins=4,color='gold',edgecolor='black')
plt.title("Histogram for watches")
plt.xlabel("Watches")
plt.ylabel("Models")
plt.show()
```



```
# RAKSHITHA R
# 240701418
# 24.7.25
# Data preprocessing
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
file_path='sales_data(2).csv'
df = pd.read_csv(file_path)
```

```
df
```

	Date	Product	Sales	Quantity	Region
0	01-01-2023	Product A	200	4	North
1	02-01-2023	Product B	150	3	South
2	03-01-2023	Product A	220	5	North
3	04-01-2023	Product C	300	6	East
4	05-01-2023	Product B	180	4	West
5	06-01-2023	Product A	210	5	North
6	07-01-2023	Product C	320	7	East
7	08-01-2023	Product B	160	3	South
8	09-01-2023	Product A	230	6	North
9	10-01-2023	Product C	310	7	East
10	11-01-2023	Product B	190	4	West
11	12-01-2023	Product A	240	6	North
12	13-01-2023	Product C	330	8	East
13	14-01-2023	Product B	170	3	South
14	15-01-2023	Product A	250	7	North
15	16-01-2023	Product C	340	8	East

```
df['Sales'].fillna(df['Sales'].mean())
df.dropna(subset=['Product', 'Quantity', 'Region'])
```

```
          Date    Product  Sales  Quantity  Region
0   01-01-2023  Product A    200         4   North
1   02-01-2023  Product B    150         3   South
2   03-01-2023  Product A    220         5   North
3   04-01-2023  Product C    300         6    East
4   05-01-2023  Product B    180         4    West
5   06-01-2023  Product A    210         5   North
6   07-01-2023  Product C    320         7    East
7   08-01-2023  Product B    160         3   South
8   09-01-2023  Product A    230         6   North
9   10-01-2023  Product C    310         7    East
10  11-01-2023  Product B    190         4    West
11  12-01-2023  Product A    240         6   North
12  13-01-2023  Product C    330         8    East
13  14-01-2023  Product B    170         3   South
14  15-01-2023  Product A    250         7   North
15  16-01-2023  Product C    340         8    East
```

```
df.describe()
```

```
            Sales   Quantity
count   16.000000  16.000000
mean   237.500000   5.375000
std     64.031242   1.746425
min    150.000000   3.000000
25%    187.500000   4.000000
50%    225.000000   5.500000
75%    302.500000   7.000000
max    340.000000   8.000000
```

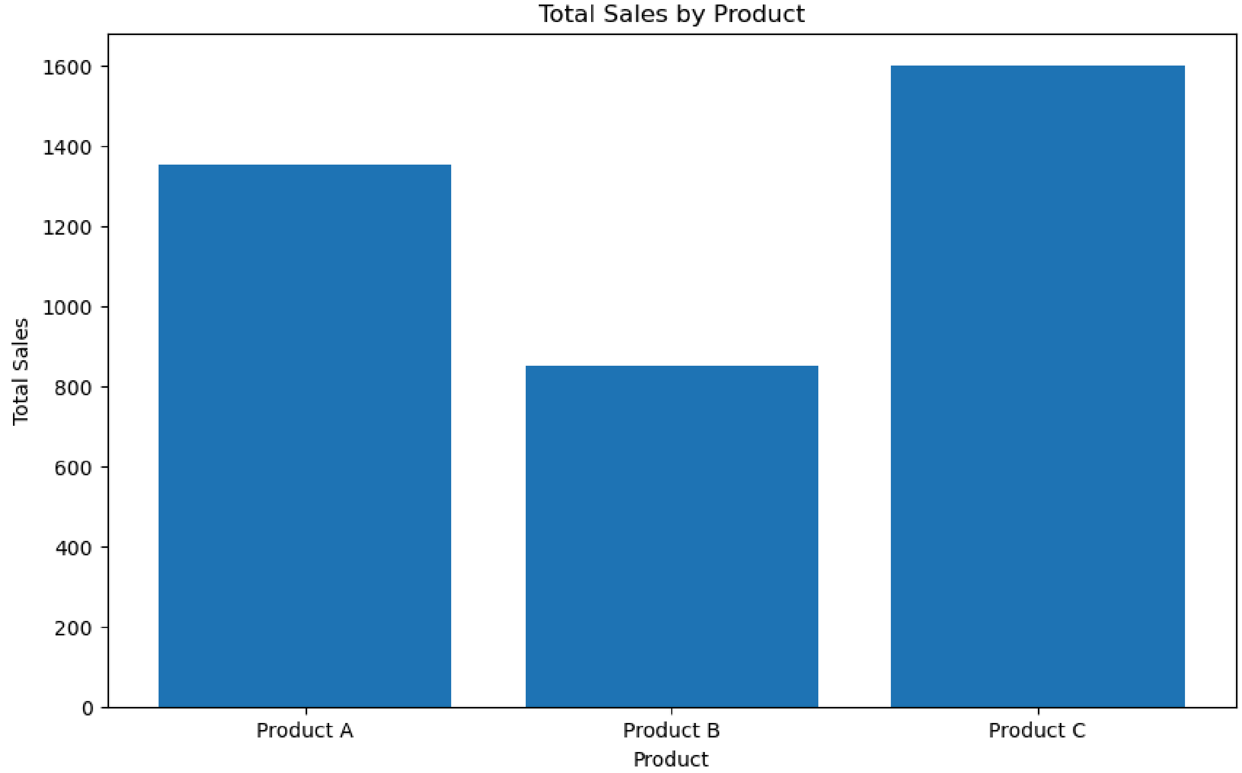
```
product_summary = df.groupby('Product').agg({
'Sales': 'sum',
'Quantity': 'sum'
}).reset_index()
```

```
product_summary
```

```
     Product  Sales  Quantity
0  Product A   1350        33
1  Product B    850        17
2  Product C   1600        36
```

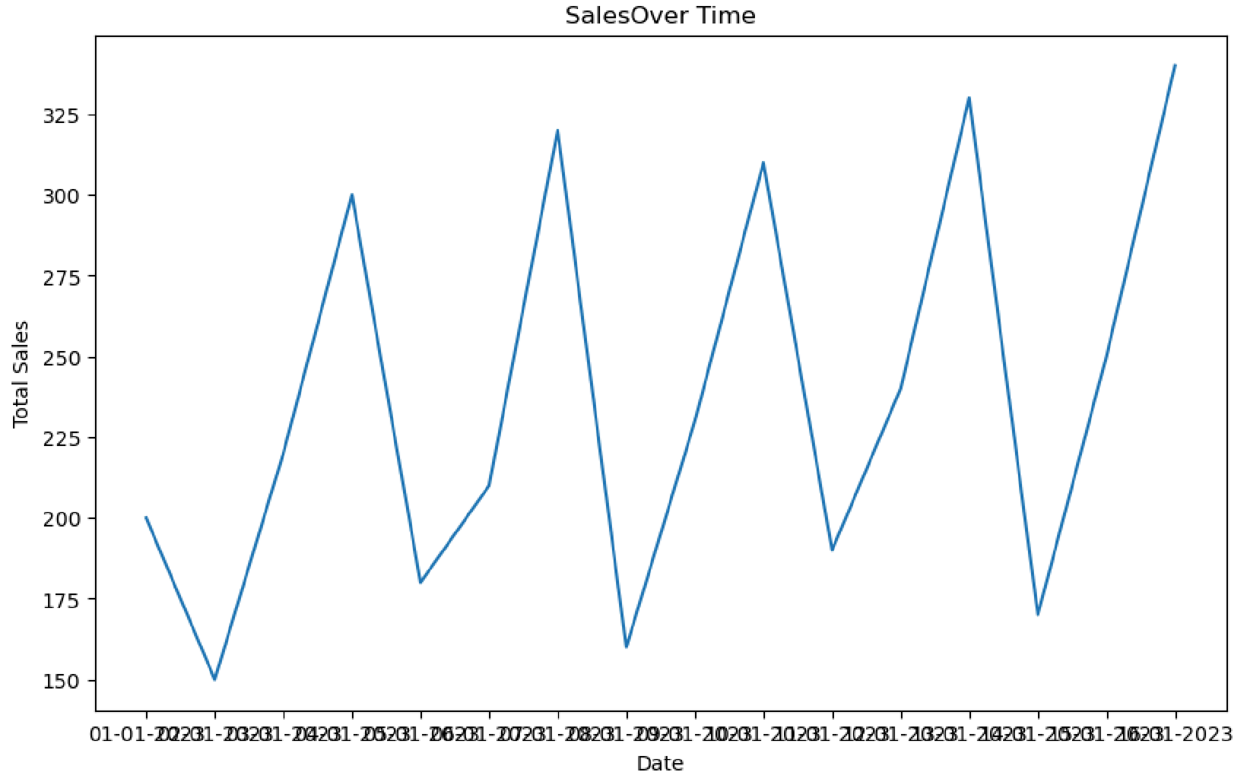
```
plt.figure(figsize=(10, 6))
plt.bar(product_summary['Product'], product_summary['Sales'])
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.title('Total Sales by Product')
```

```
Text(0.5, 1.0, 'Total Sales by Product')
```



```
sales_over_time = df.groupby('Date').agg({'Sales': 'sum'}).reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'],sales_over_time['Sales'])
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('SalesOver Time')
```

```
Text(0.5, 1.0, 'SalesOver Time')
```



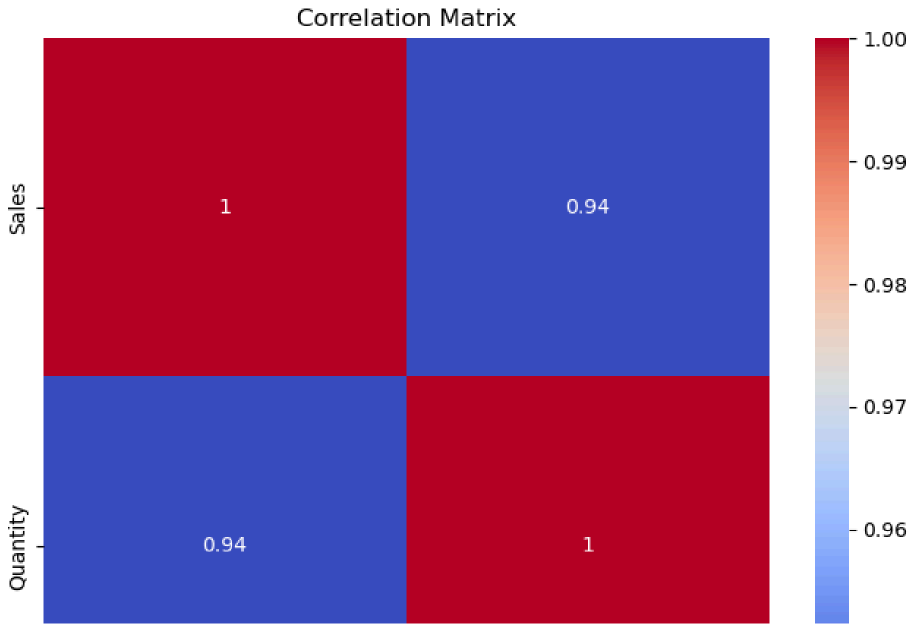
```
pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product',aggfunc="sum", fill_value=0)
```

```
pivot_table
```

Product	Product A	Product B	Product C
Region			
East	0	0	1600
North	1350	0	0
South	0	480	0
West	0	370	0

```
correlation_matrix = df.corr()
```

```
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



```
# RAKSHITHA R
# 240701418
# 7.8.25
# Data preprocessing
```

```
import numpy as np
import pandas as pd
df=pd.read_csv("pre_process_datasample(1).csv")
df
```

	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

```
df.info()
df.Country.mode()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    10 non-null     object 
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 452.0+ bytes
0    France
Name: Country, dtype: object
```

```
df.Country.mode()[0]
```

```
'France'
```

```
type(df.Country.mode())
```

```
pandas.core.series.Series
```

```
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Salary'] = df['Salary'].fillna(round(df['Salary'].mean()))
```

```
pd.get_dummies(df.Country)
```

	France	Germany	Spain
0	True	False	False
1	False	False	True
2	False	True	False
3	False	False	True
4	False	True	False
5	True	False	False
6	False	False	True
7	True	False	False
8	False	True	False
9	True	False	False

```
updated_dataset=pd.concat([pd.get_dummies(df.Country),df.iloc[:,[1,2,3]]],axis=1)
updated_dataset
```

	France	Germany	Spain	Age	Salary	Purchased
0	True	False	False	44.0	72000.0	No
1	False	False	True	27.0	48000.0	Yes
2	False	True	False	30.0	54000.0	No
3	False	False	True	38.0	61000.0	No
4	False	True	False	40.0	63778.0	Yes
5	True	False	False	35.0	58000.0	Yes
6	False	False	True	38.0	52000.0	No
7	True	False	False	48.0	79000.0	Yes
8	False	True	False	50.0	83000.0	No
9	True	False	False	37.0	67000.0	Yes

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    10 non-null     object 
 1   Age        10 non-null     float64
 2   Salary     10 non-null     float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 452.0+ bytes
```

```
updated_dataset.Purchased.replace(['No','Yes'],[0,1])
```

```
0    0
1    1
2    0
3    0
4    1
5    1
6    0
7    1
8    0
9    1
Name: Purchased, dtype: int64
```

```
updated_dataset
```

```
   France  Germany  Spain   Age   Salary  Purchased
0    True    False  False  44.0  72000.0          0
1   False    False   True  27.0  48000.0          1
2   False     True  False  30.0  54000.0          0
3   False    False   True  38.0  61000.0          0
4   False     True  False  40.0  63778.0          1
5    True    False  False  35.0  58000.0          1
6   False    False   True  38.0  52000.0          0
7    True    False  False  48.0  79000.0          1
8   False     True  False  50.0  83000.0          0
9    True    False  False  37.0  67000.0          1
```

Start coding or generate with AI.

```
# RAKSHITHA
# 240701418
# 7.8.25
# Handling Missing Values
```

```
import numpy as np
import pandas as pd
df=pd.read_csv("Hotel_Dataset.csv")
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
9	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
10	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

```
df.duplicated()
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CustomerID       11 non-null     int64 
 1   Age_Group        11 non-null     object
 2   Rating(1-5)      11 non-null     int64 
 3   Hotel            11 non-null     object
 4   FoodPreference   11 non-null     object
 5   Bill             11 non-null     int64 
 6   NoOfPax          11 non-null     int64 
 7   EstimatedSalary  11 non-null     int64 
 8   Age_Group.1      11 non-null     object
dtypes: int64(5), object(4)
memory usage: 924.0+ bytes
```

```
df.drop_duplicates(inplace=True)
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
10	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

```
len(df)

index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group.1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	35+	3	Ibys	Non-Veg	1909	2	122220	35+
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
9	10	30-35	5	RedFox	non-Veg	-6755	4	87777	30-35

```
df.drop(['Age_Group.1'],axis=1,inplace=True)
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1	20-25	4	Ibis	veg	1300	2	40000
1	2	30-35	5	LemonTree	Non-Veg	2000	3	59000
2	3	25-30	6	RedFox	Veg	1322	2	30000
3	4	20-25	-1	LemonTree	Veg	1234	2	120000
4	5	35+	3	Ibis	Vegetarian	989	2	45000
5	6	35+	3	Ibys	Non-Veg	1909	2	122220
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122
7	8	20-25	7	LemonTree	Veg	2999	-10	345673
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999
9	10	30-35	5	RedFox	non-Veg	-6755	4	87777

```
df.CustomerID.loc[df.CustomerID<0]=np.nan
df.Bill.loc[df.Bill<0]=np.nan
df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
df
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:1: FutureWarning: ChainedAssignmentError: behavic
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re

  df.CustomerID.loc[df.CustomerID<0]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
  df.CustomerID.loc[df.CustomerID<0]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:2: FutureWarning: ChainedAssignmentError: behavic
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re

  df.Bill.loc[df.Bill<0]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
  df.Bill.loc[df.Bill<0]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:3: FutureWarning: ChainedAssignmentError: behavic
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re

  df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2080958306.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
  df.EstimatedSalary.loc[df.EstimatedSalary<0]=np.nan
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2	45000.0
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	-1	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	-10	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3	NaN
9	10.0	30-35	5	RedFox	non-Veg	NaN	4	87777.0

```
df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
df
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2129877948.py:1: FutureWarning: ChainedAssignmentError: behaviou
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-o
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this k

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret

  df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
C:\Users\HP\AppData\Local\Temp\ipykernel_13572\2129877948.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
  df['NoOfPax'].loc[(df['NoOfPax']<1) | (df['NoOfPax']>20)]=np.nan
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	veg	1300.0	2.0	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3	Ibis	Vegetarian	989.0	2.0	45000.0
5	6.0	35+	3	Ibys	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4	RedFox	Vegetarian	1000.0	NaN	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	NaN	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	NaN
9	10.0	30-35	5	RedFox	non-Veg	NaN	4.0	87777.0

```
df.Age_Group.unique()
```

```
array(['20-25', '30-35', '25-30', '35+'], dtype=object)
```

```
df.Hotel.unique()
```

```
array(['Ibis', 'LemonTree', 'RedFox', 'Ibys'], dtype=object)
```

```
df.Hotel.replace(['Ibys'],'Ibis')
df.FoodPreference.unique
```

```
<bound method Series.unique of 0        Veg
1    Non-Veg
2        Veg
3        Veg
4        Veg
5    Non-Veg
6        Veg
7        Veg
8    Non-Veg
9    Non-Veg
Name: FoodPreference, dtype: object>
```

```
df.FoodPreference.replace(['Vegetarian','veg'],'Veg',inplace=True)
df.FoodPreference.replace(['non-Veg'],'Non-Veg',inplace=True)
```

```
df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()))
df.NoOfPax.fillna(round(df.NoOfPax.median()),inplace=True)
df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()))
df.Bill.fillna(round(df.Bill.mean()))
df
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4	Ibis	Veg	1300.0	2.0	40000.0
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	3.0	59000.0
2	3.0	25-30	6	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	-1	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3	Ibis	Veg	989.0	2.0	45000.0
5	6.0	35+	3	Ibis	Non-Veg	1909.0	2.0	122220.0
6	7.0	35+	4	RedFox	Veg	1000.0	2.0	21122.0
7	8.0	20-25	7	LemonTree	Veg	2999.0	2.0	345673.0
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	3.0	96755.0
9	10.0	30-35	5	RedFox	Non-Veg	1801.0	4.0	87777.0

```
Start coding or generate with AI.
```

```
# RAKSHITHA R
# 240701418
# 14.8.25
# Outliers
```

```
import numpy as np
array=np.random.randint(1,100,16)
def outDetection(array):
    sorted(array)
```

```
def outDetection(array):
    sorted(array)
    Q1,Q3=np.percentile(array,[25,75])
    IQR=Q3-Q1
    lr=Q1-(1.5*IQR)
    ur=Q3+(1.5*IQR)
    return lr,ur
lr,ur=outDetection(array)
```

```
lr,ur
```

```
(np.float64(-15.875), np.float64(121.125))
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.distplot(array)
plt.show()
```

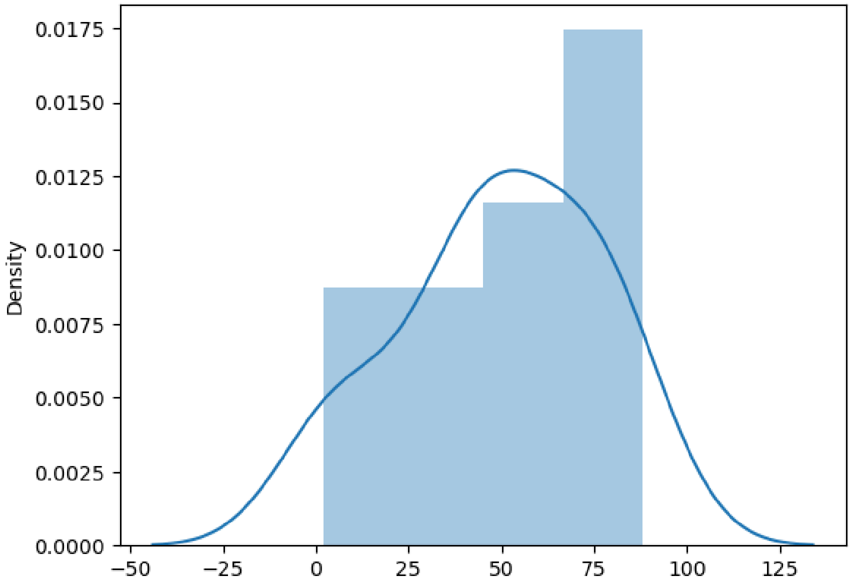
```
C:\Users\HP\AppData\Local\Temp\ipykernel_6860\579271414.py:3: UserWarning:

`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

  sns.distplot(array)
```



Start coding or generate with AI.

```
Start coding or generate with AI.
```

```
# RAKSHITHA
# 240701418
# 21.8.25
# Feature scaling
```

```
import numpy as np
import pandas as pd
df=pd.read_csv('pre_process_datasample(1).csv')
```

```
df
```

	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

```
df.Country.fillna(df.Country.mode()[0])
features=df.iloc[:,:-1].values
```

```
label=df.iloc[:,-1].values
```

```
from sklearn.impute import SimpleImputer
age=SimpleImputer(strategy="mean",missing_values=np.nan)
Salary=SimpleImputer(strategy="mean",missing_values=np.nan)
```

```
age.fit(features[:,[1]])
```

```
▾ SimpleImputer ⓘ ?
SimpleImputer()
```

```
Salary.fit(features[:,[2]])
```

```
▾ SimpleImputer ⓘ ?
SimpleImputer()
```

```
SimpleImputer()
```

```
▾ SimpleImputer ⓘ ?
SimpleImputer()
```

```
features[:,[1]]=age.transform(features[:,[1]])
features[:,[2]]=Salary.transform(features[:,[2]])
features
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
from sklearn.preprocessing import OneHotEncoder
oh = OneHotEncoder(sparse_output=False)
Country=oh.fit_transform(features[:,[0]])
Country
```

```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

```
final_set=np.concatenate((Country,features[:,[1,2]]),axis=1)
final_set
```

```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(final_set)
feat_standard_scaler=sc.transform(final_set)
feat_standard_scaler
```

```
array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler=mms.transform(final_set)
feat_minmax_scaler
```

```
array([[1.        , 0.        , 0.        , 0.73913043, 0.68571429],
       [0.        , 0.        , 1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        , 0.13043478, 0.17142857],
       [0.        , 0.        , 1.        , 0.47826087, 0.37142857],
       [0.        , 1.        , 0.        , 0.56521739, 0.45079365],
       [1.        , 0.        , 0.        , 0.34782609, 0.28571429],
       [0.        , 0.        , 1.        , 0.51207729, 0.11428571],
       [1.        , 0.        , 0.        , 0.91304348, 0.88571429],
       [0.        , 1.        , 0.        , 1.        , 1.        ],
       [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

```
Start coding or generate with AI.
```

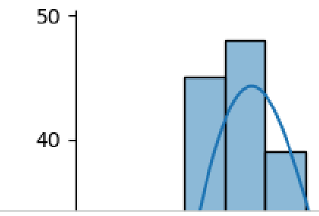
```
# RAKSHITHA R
# 240701418
# 28.8.25
# EDA
```

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
tips=sns.load_dataset('tips')
```

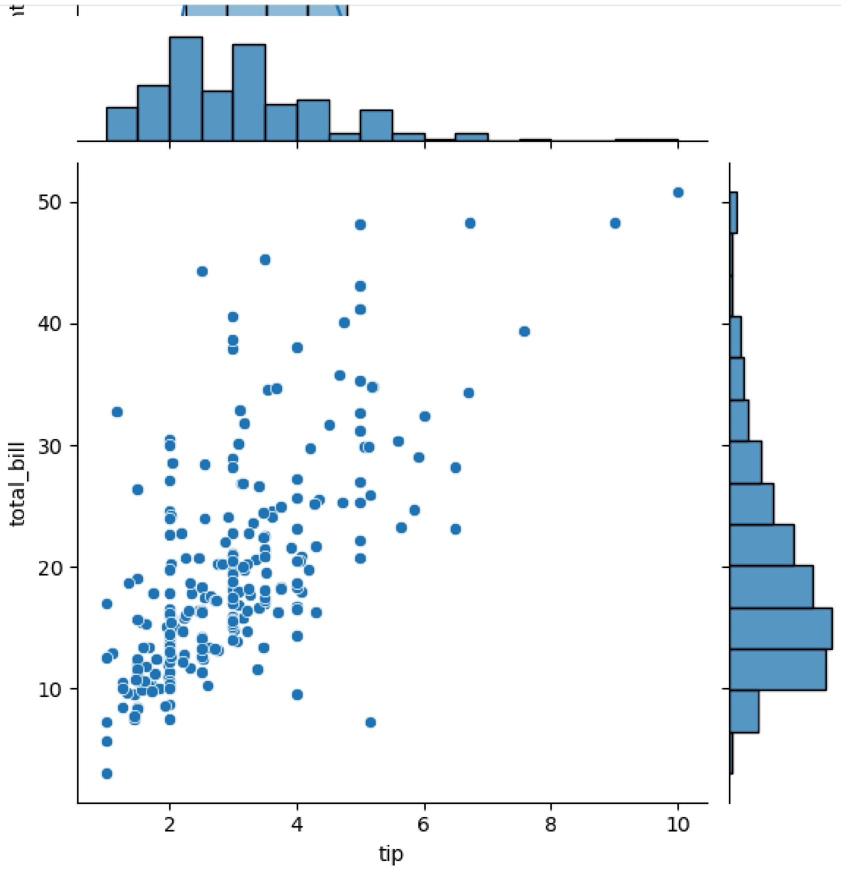
```
tips.head()
```

	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

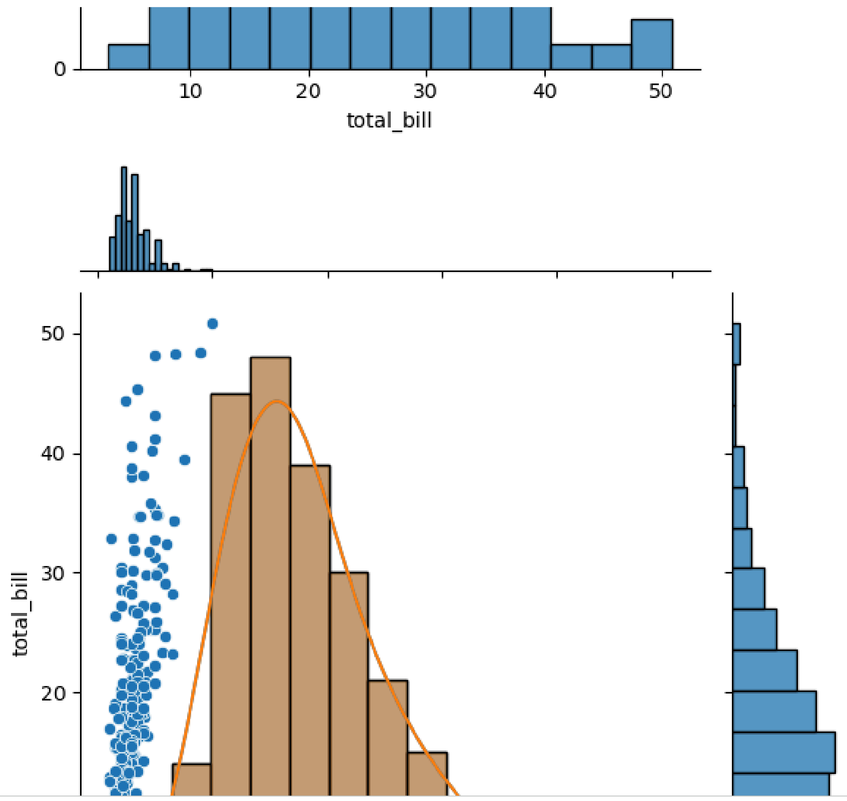
```
sns.histplot(tips.total_bill,kde=True)
plt.show()
```

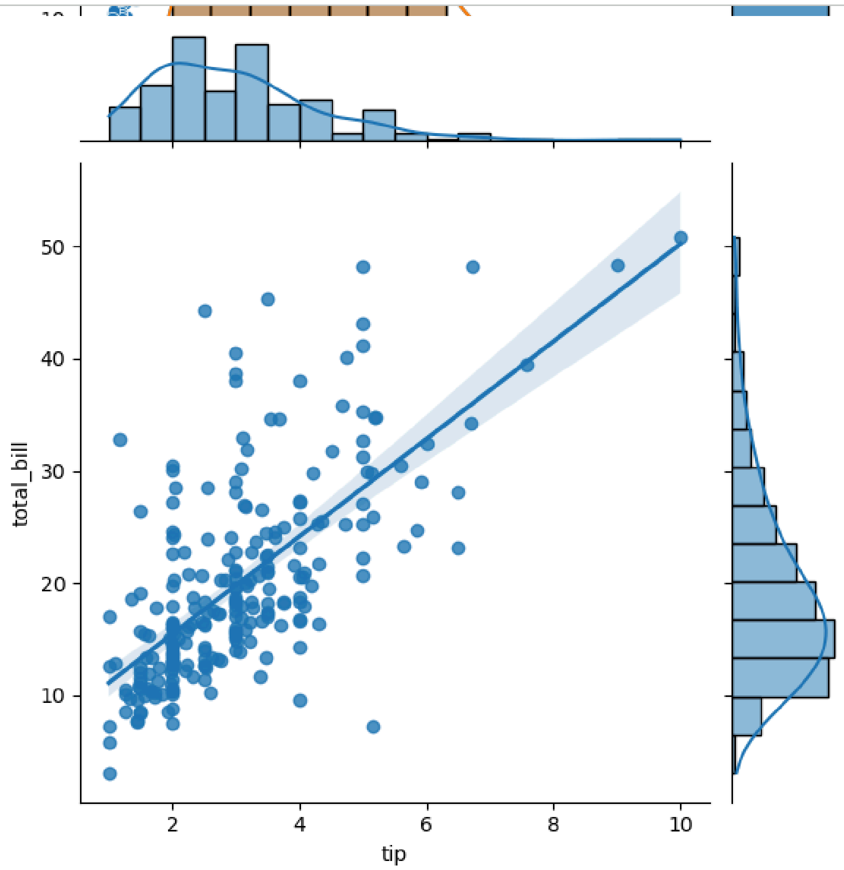


```
sns.jointplot(x=tips.tip,y=tips.total_bill)
plt.show()
```

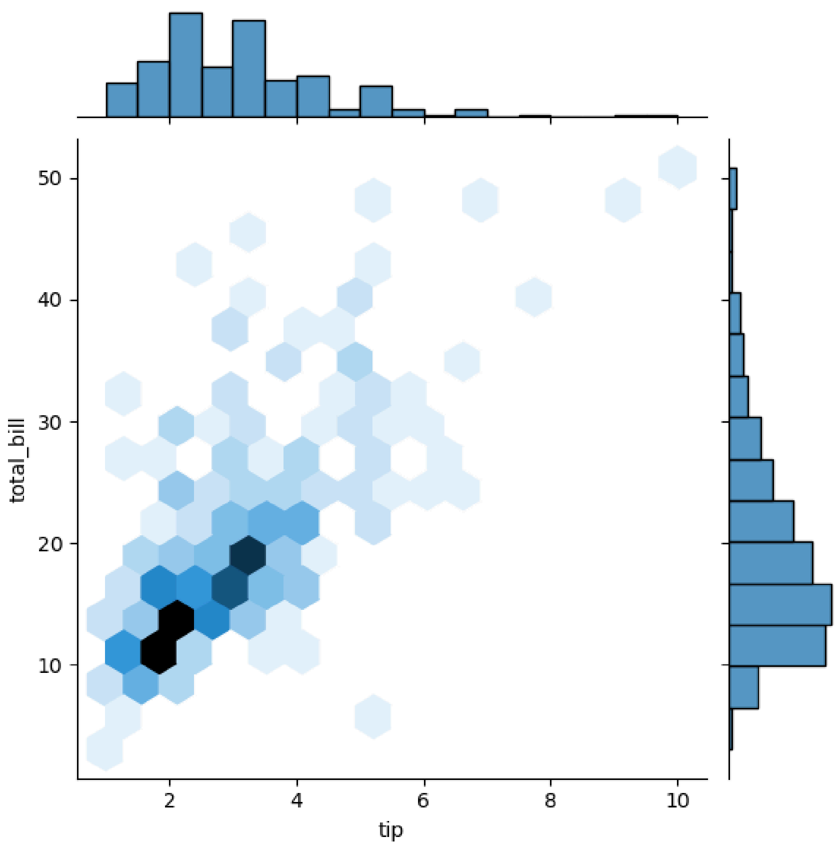


```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
plt.show()
```

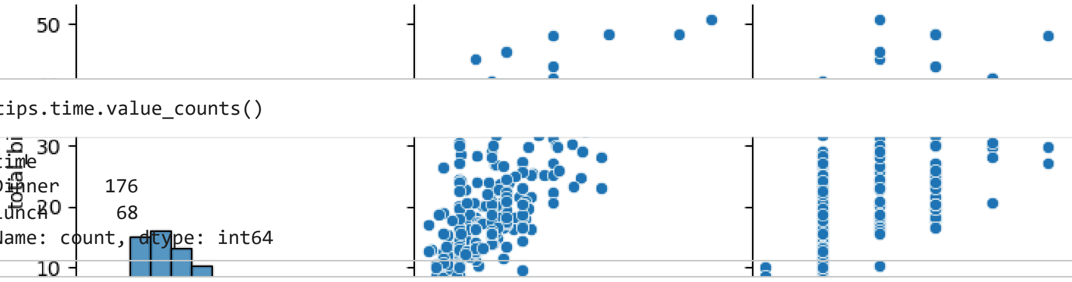




```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
plt.show()
```



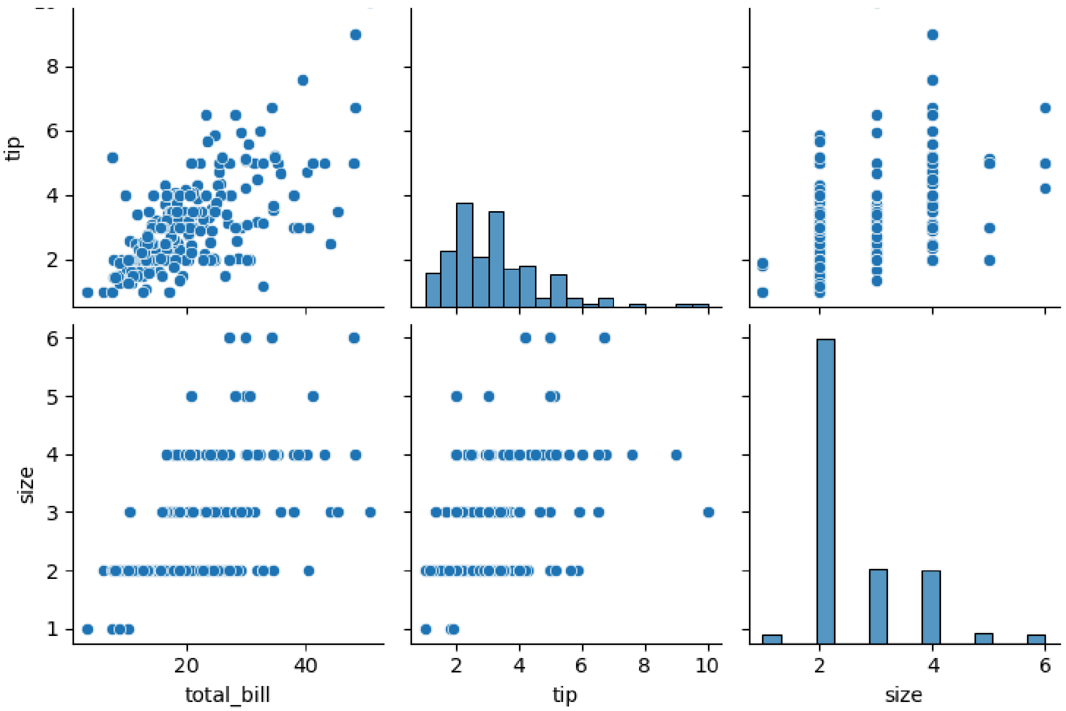
```
sns.pairplot(tips)
plt.show()
```

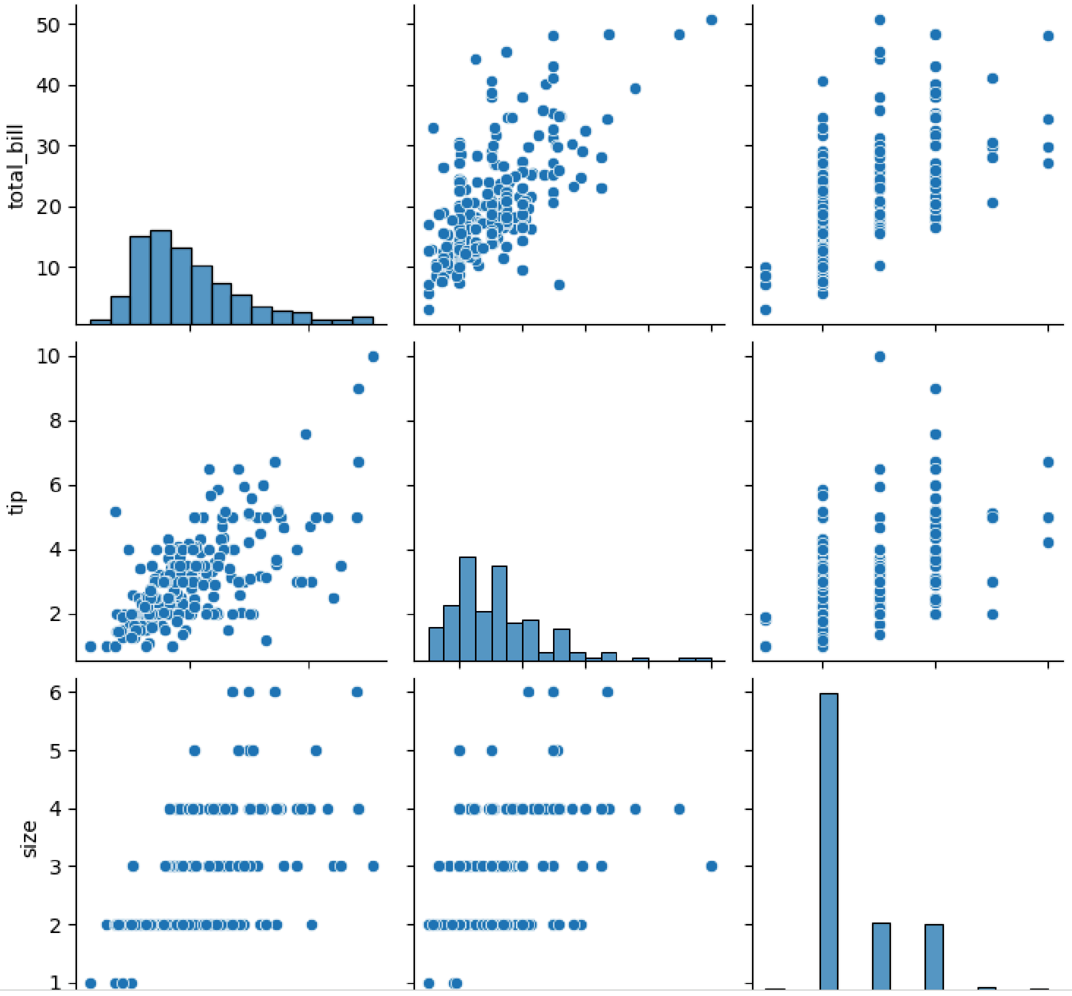


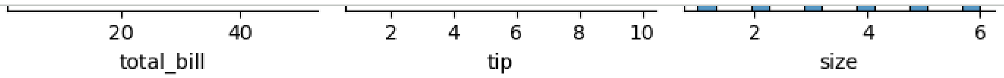
```
tips.time.value_counts()
```

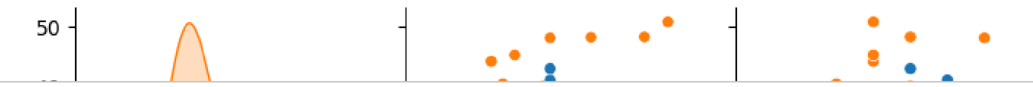
```
time
Dinner    176
Lunch      68
Name: count, dtype: int64
```

```
sns.pairplot(tips,hue='time')
plt.show()
```

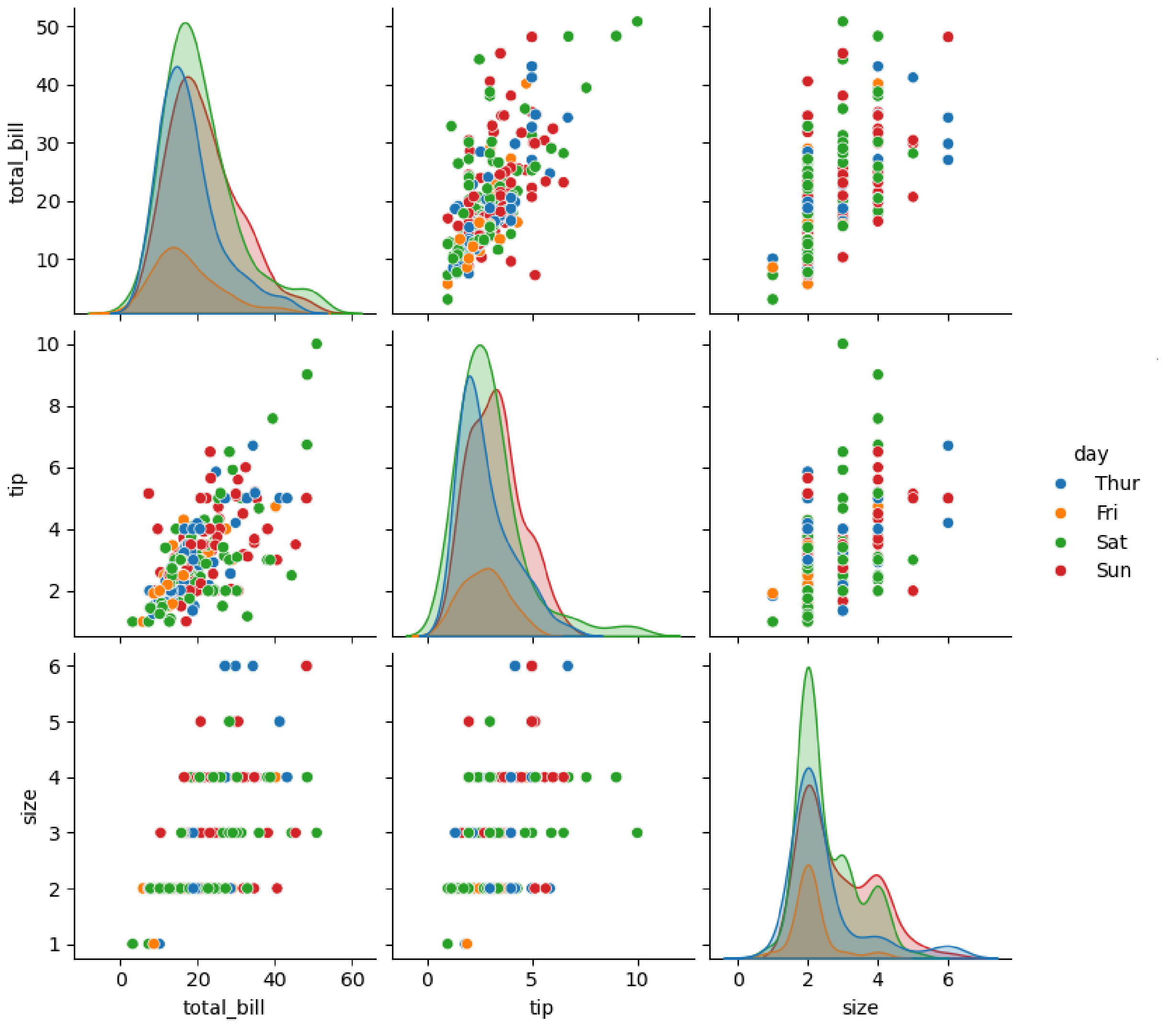




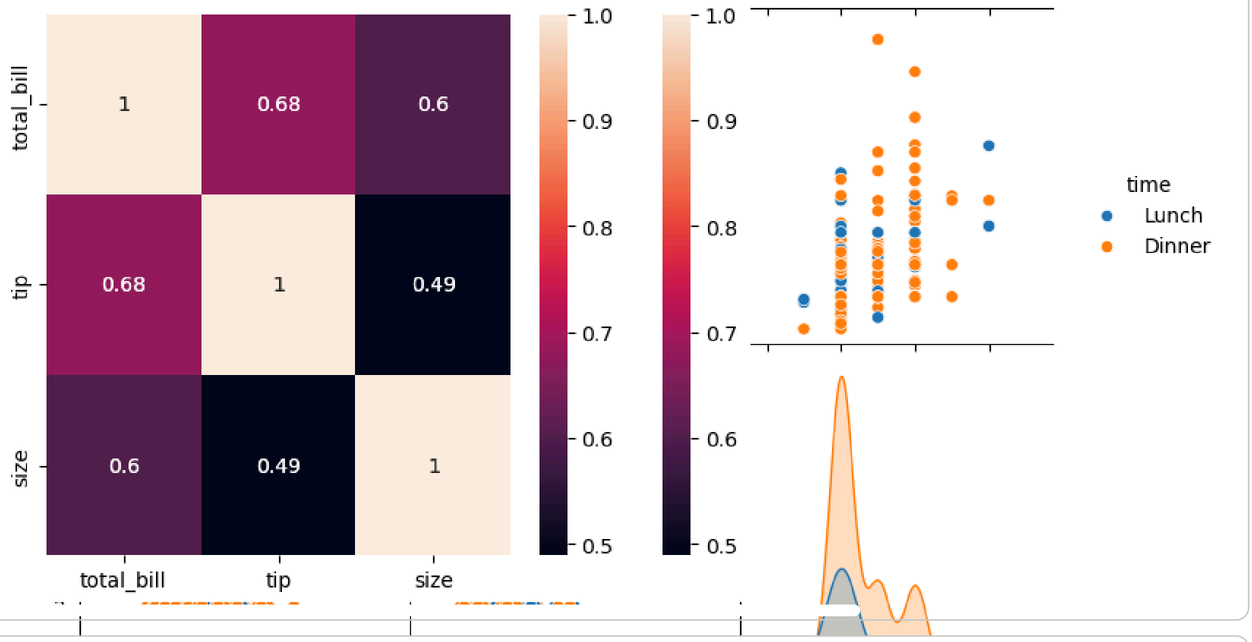

20
40
total_bill
2
4
6
8
10
tip
2
4
6
size



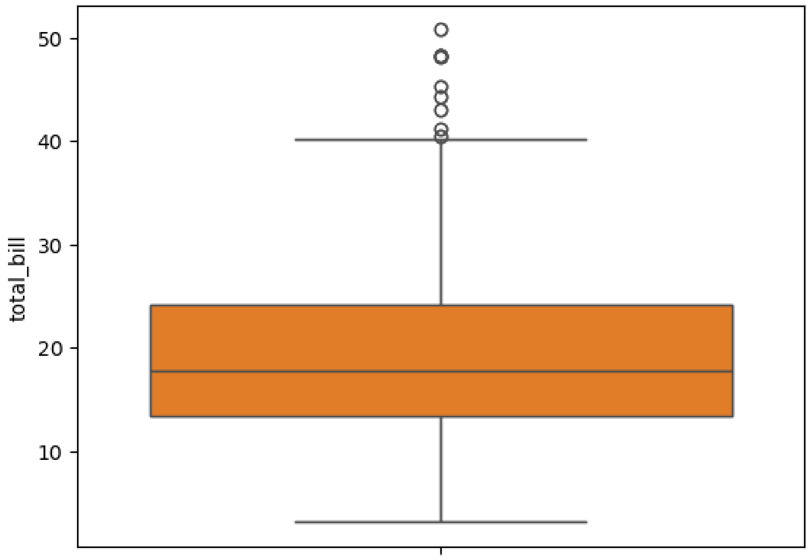
```
sns.pairplot(tips,hue='day')
plt.show()
```



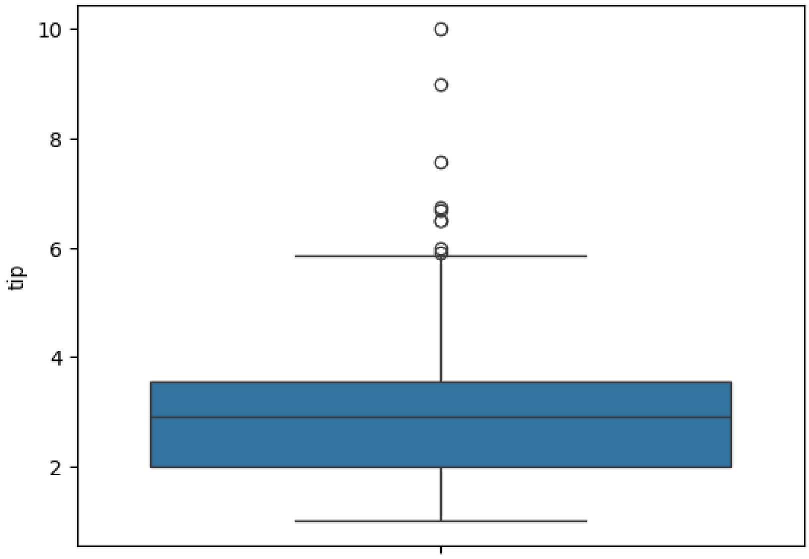
```
sns.heatmap(tips.corr(numeric_only=True),annot=True)
plt.show()
```



```
sns.boxplot(tips.total_bill)
plt.show()
```



```
sns.boxplot(tips.tip)
plt.show()
```



```
sns.countplot(tips.day)
plt.show()
```

```
# RAKSHITHA R
# 240701418
# 17.9.25
# Linear Regression
```

```
import numpy as np
import pandas as pd
df=pd.read_csv('Salary_data.csv')
df
```

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
```

```
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
```

```
df.dropna(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
```

```
df.describe()
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```
features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=25)
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
▾ LinearRegression ⓘ ?
LinearRegression()
```

```
model.score(x_train,y_train)
```

```
0.9577907749872991
```

```
model.score(x_test,y_test)
```

```
0.9531732818427658
```

```
model.coef_
```

```
array([[9339.90339715]])
```

```
model.intercept_
```

```
array([26561.50676243])
```

```
import pickle
pickle.dump(model,open('SalaryPred.model','wb'))
```

```
model=pickle.load(open('SalaryPred.model','rb'))
```

```
yr_of_exp=float(input("Enter Years of Experience: "))
y=yr_of_exp
yr_of_exp_NP=np.array([[yr_of_exp]])
Salary=model.predict(yr_of_exp_NP)

print("Estimated salary for {} year of exp is{} ".format(yr_of_exp,Salary))
```

```
Enter Years of Experience:  33
estimated salary for 33.0 year of exp is[[334778.31886853]]
```

```
array([[ 39343],
       [ 46205],
       [ 37731],
       [ 43525],
       [ 39891],
       [ 56642],
       [ 60150],
       [ 54445],
       [ 64445],
       [ 57189],
       [ 63218],
       [ 55794],
       [ 56957],
       [ 57081],
       [ 61111],
       [ 67938],
       [ 66029],
       [ 83088],
       [ 81363],
       [ 93940],
       [ 91738],
       [ 98273],
       [101302],
       [113812],
       [109431],
       [105582],
       [116969],
       [112635],
       [122391],
       [121872]])
```

```
Start coding or generate with AI.
```

```
# RAKSHITHA R
# 240701418
# 17.9.25
# Logistic Regression
```

```
import numpy as np
import pandas as pd
df=pd.read_csv('Social_Network_Ads.csv')
df
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
features=df.iloc[:,[2,3]].values
label=df.iloc[:,4].values
features
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
       [    29,  80000],
       [    47,  25000],
       [    45,  26000],
       [    46,  28000],
       [    48,  29000],
       [    45,  22000],
       [    47,  49000],
       [    48,  41000],
       [    45,  22000],
       [    46,  23000],
       [    47,  20000],
       [    49,  28000],
       [    47,  30000],
       [    29,  43000],
       [    31,  18000],
       [    31,  74000],
       [    27, 137000],
       [    21,  16000],
       [    28,  44000],
       [    27,  90000],
       [    35,  27000],
       [    33,  28000],
       [    30,  49000],
       [    26,  72000],
       [    27,  31000],
       [    27,  17000],
       [    33,  51000],
       [    35, 108000],
       [    30,  15000],
       [    28,  84000],
       [    23,  20000],
       [    25,  79000],
       [    27,  54000],
       [    30, 135000],
       [    31,  89000],
       [    24,  32000],
       [    18,  44000],
       [    29,  83000],
       [    35,  23000],
       [    27,  58000],
       [    24,  55000],
       [    23,  48000],
```

```
label
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
for i in range(1,401):
    x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=i)
    model=LogisticRegression()
    model.fit(x_train,y_train)
    train_score=model.score(x_train,y_train)
    test_score=model.score(x_test,y_test)
```

```
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.2,random_state=25)
finalModel=LogisticRegression()
finalModel.fit(x_train,y_train)
```

```
▾ LogisticRegression  ⓘ ?
LogisticRegression()
```

```
print(finalModel.score(x_train,y_train))
print(finalModel.score(x_test,y_test))
```

```
0.846875
0.8
```

```
from sklearn.metrics import classification_report
print(classification_report(label,finalModel.predict(features)))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       257
           1       0.82      0.69      0.75       143
```

```
    accuracy                           0.84       400
   macro avg       0.83      0.81      0.82       400
weighted avg       0.84      0.84      0.83       400
```

```
Start coding or generate with AI.
```

```
# RAKSHITHA R
# 240701418
# 24.10.25
# K means Clustering
```

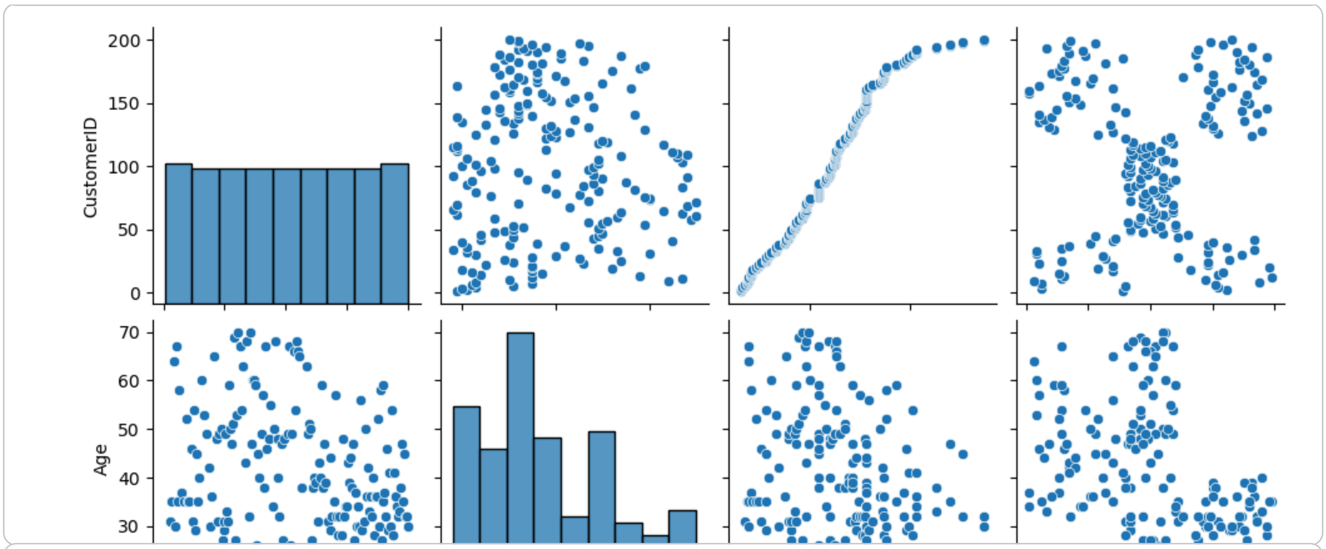
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df=pd.read_csv('Mall_Customers.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
sns.pairplot(df)
plt.show()
```



```
features=df.iloc[:,[3,4]].values
```

```
from sklearn.cluster import KMeans
model=KMeans(n_clusters=5)
model.fit(features)
KMeans(n_clusters=5)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
```

▾ KMeans ⓘ ?

KMeans(n_clusters=5)

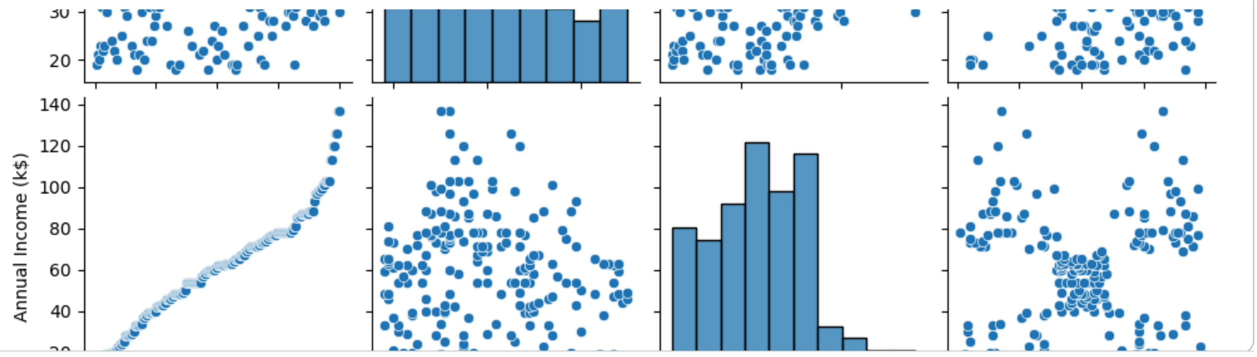
```
Final=df.iloc[:,[3,4]]
Final['label']=model.predict(features)
Final.head()
```

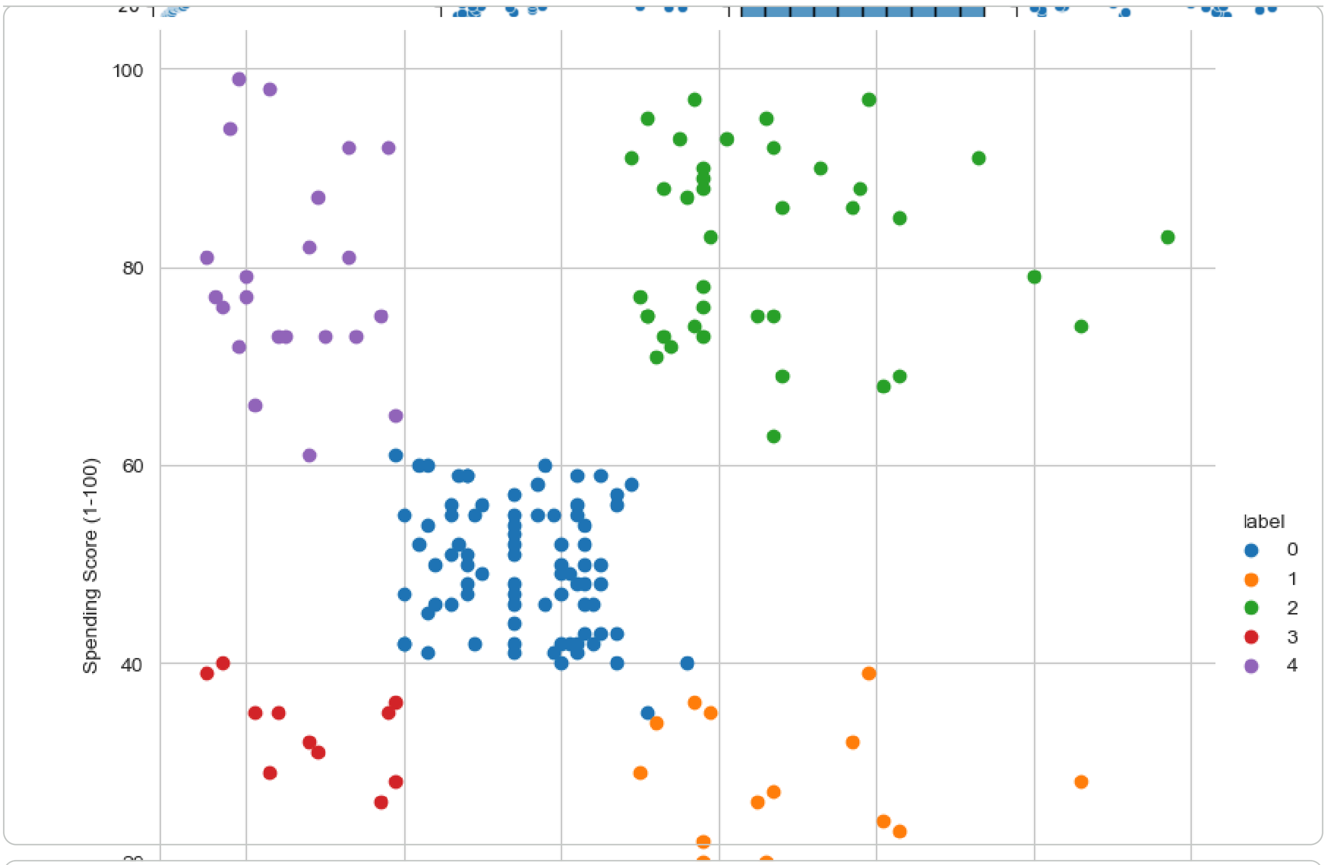
```
C:\Users\HP\AppData\Local\Temp\ipykernel_15280\470183701.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
  Final['label']=model.predict(features)
```

	Annual Income (k$)	Spending Score (1-100)	label
0	15	39	3
1	15	81	4
2	16	6	3
3	16	77	4
4	17	40	3

```
sns.set_style("whitegrid")
sns.FacetGrid(Final,hue="label",height=8) \
.map(plt.scatter,"Annual Income (k$)", "Spending Score (1-100)") \
.add_legend();
plt.show()
```

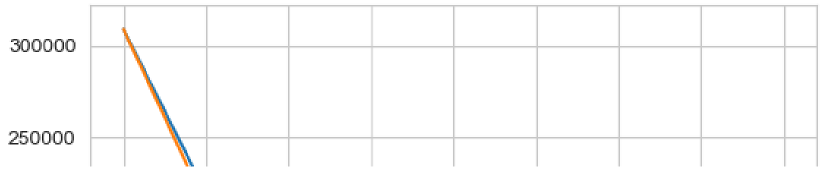




```
features_el=df.iloc[:,[2,3,4]].values
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,10):
    model=KMeans(n_clusters=i)
    model.fit(features_el)
    wcss.append(model.inertia_)
plt.plot(range(1,10),wcss)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to hav
  warnings.warn(
[<matplotlib.lines.Line2D at 0x246b722c410>]
```

```
plt.show()
```



```
# RAKSHITHA R
# 240701418
# 1.10.25
# KNN
```

```
import numpy as np
import pandas as pd
```

```
df=pd.read_csv('Iris.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
df.Species.value_counts()
df
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
features=df.iloc[:,:-1].values
label=df.iloc[:,5].values
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=.2,random_state=20)
model_KNN=KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(xtrain,ytrain)
```

```
▾ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()
```

```
print(model_KNN.score(xtrain,ytrain))
print(model_KNN.score(xtest,ytest))
```

```
1.0
1.0
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(label,model_KNN.predict(features))
```

```
array([[50,  0,  0],
       [ 0, 50,  0],
       [ 0,  0, 50]])
```

```
from sklearn.metrics import classification_report
print(classification_report(label,model_KNN.predict(features)))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       1.00      1.00      1.00        50
 Iris-virginica       1.00      1.00      1.00        50

       accuracy                           1.00       150
      macro avg       1.00      1.00      1.00       150
   weighted avg       1.00      1.00      1.00       150
```

```
#RAKSHITHA R
#240701418
# 8.10.25
#T-test
```

```
import numpy as np
from scipy import stats
```

```
# RAKSHITHA R
# 240701418
# 8.10.25
# Z-test
```

```
import numpy as np
from math import sqrt
```

```
# RAKSHITHA R
# 240701418
# 8.10.25
# Anova test
```

```
import numpy as np
from scipy import stats
```