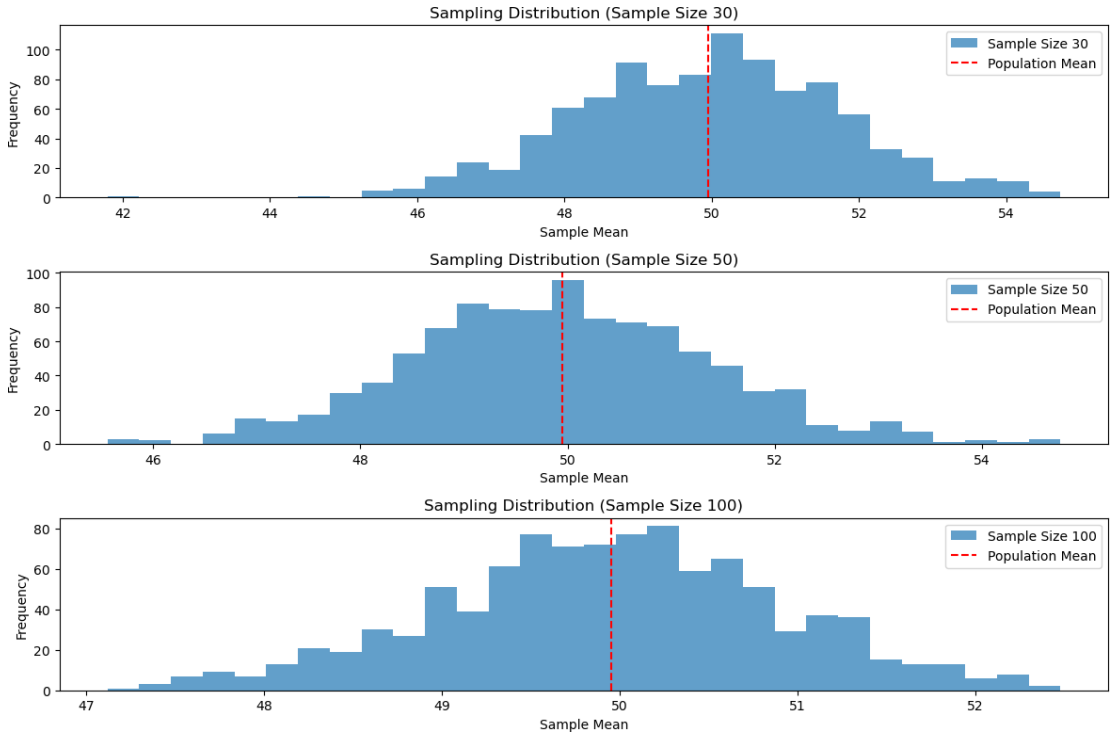
Dharshini R S
230701076
CSE -B

Central-limit-theorem-visualization

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Generate a population (e.g., normal distribution)
population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)

# Step 2: Random sampling
sample_sizes = [30, 50, 100]  # different sample sizes to consider
num_samples = 1000  # number of samples for each sample size

sample_means = {}

for size in sample_sizes:
    sample_means[size] = []
    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))

# Step 3: Plotting sampling distributions
plt.figure(figsize=(12, 8))

for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i+1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f"Sample Size
    ↪{size}")
    plt.axvline(np.mean(population), color="red", linestyle="dashed",
    ↪linewidth=1.5, label="Population Mean")
    plt.title(f"Sampling Distribution (Sample Size {size})")
    plt.xlabel("Sample Mean")
    plt.ylabel("Frequency")
    plt.legend()

plt.tight_layout()
```


Sampling Distribution (Sample Size 30)
Sample Size 30
Population Mean
Frequency
Sample Mean
Sampling Distribution (Sample Size 50)
Sample Size 50
Population Mean
Frequency
Sample Mean
Sampling Distribution (Sample Size 100)
Sample Size 100
Population Mean
Frequency
Sample Mean

[ ]:

hypothesis-testing

[2]:
```
# Import the necessary libraries:
import seaborn as sns
import numpy as np
from scipy import stats

# Load the Iris dataset:
iris = sns.load_dataset('iris')

# Filter the dataset for the two species we want to compare:
setosa = iris[iris['species'] == 'setosa']
versicolor = iris[iris['species'] == 'versicolor']

# Extract the petal lengths for each species:
setosa_petal_lengths = setosa['petal_length']
versicolor_petal_lengths = versicolor['petal_length']

# Perform the t-test:
t_stat, p_value = stats.ttest_ind(setosa_petal_lengths,␣
↪versicolor_petal_lengths)

# Interpret the results:
alpha = 0.05
if p_value < alpha:
        print("Reject the null hypothesis; there is a significant difference␣
↪between the petal lengths of Iris setosa and Iris versicolor.")
else:
    print("Fail to reject the null hypothesis; there is no significant␣
↪difference between the petal lengths of Iris setosa and Iris versicolor.")
```

Reject the null hypothesis; there is a significant difference between the petal lengths of Iris setosa and Iris versicolor.

[ ]:

mpy-array-operations-and-reshaping

[1]:
```
import numpy as np
array=np.random.randint(1,100,9)
array
```

[1]: array([69, 82, 26, 46, 64, 44, 96, 48, 81])

[2]:
```
np.sqrt(array)
```

[2]: array([8.30662386, 9.05538514, 5.09901951, 6.78232998, 8. ,
6.63324958, 9.79795897, 6.92820323, 9.])

[3]:
```
array.ndim
```

[3]: 1

[5]:
```
new_array=array.reshape(3,3)
new_array
```

[5]: array([[69, 82, 26],
[46, 64, 44],
[96, 48, 81]])

[6]:
```
new_array.ndim
```

[6]: 2

[7]:
```
new_array.ravel()
```

[7]: array([69, 82, 26, 46, 64, 44, 96, 48, 81])

[9]:
```
newm=new_array.reshape(3,3)
newm
```

[9]: array([[69, 82, 26],
[46, 64, 44],
[96, 48, 81]])

[10]:
```
newm[2,1:3]
```

[10]: array([48, 81])

[11]:
```
newm[1:2,1:3]
```

[11]: array([[64, 44]])

[12]:
```
new_array[0:3,0:0]
```

[12]: array([], shape=(3, 0), dtype=int32)

[13]:
```
new_array[0:2,0:1]
```

[13]: array([[69],
[46]])

[14]:
```
new_array[0:3,0:1]
```

[14]: array([[69],
[46],
[96]])

[15]:
```
new_array[1:3]
```

[15]: array([[46, 64, 44],
[96, 48, 81]])

[ ]:

one-sample-t-test

[1]:
```
# Import necessary libraries
import numpy as np
from scipy import stats

# Given student scores
student_scores = np.array([72, 89, 65, 73, 79, 84, 63, 76, 85, 75])


# Hypothesized population mean
mu = 70

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(student_scores, mu)
print("T statistic:", t_stat)
print("P-value:", p_value)

# Setting significance level
alpha = 0.05

# Interpret the results
if p_value < alpha:
    print("Reject the null hypothesis; there is a significant difference between the sample mean and the hypothesized population mean.")
else:
    print("Fail to reject the null hypothesis; there is no significant difference between the sample mean and the hypothesized population mean.")
```

```
T statistic: 2.2894683580127317
P-value: 0.047816221110566944
Reject the null hypothesis; there is a significant difference between the sample mean and the hypothesized population mean.
```

[ ]:

data-cleaning-and-indexing-py

[1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("Hotel_Dataset.csv")
df
```

[1]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1	20-25	4	Ibis	veg	1300	
1	2	30-35	5	LemonTree	Non-Veg	2000	
2	3	25-30	6	RedFox	Veg	1322	
3	4	20-25	-1	LemonTree	Veg	1234	
4	5	35+	3	Ibis	Vegetarian	989	
5	6	35+	3	Ibys	Non-Veg	1909	
6	7	35+	4	RedFox	Vegetarian	1000	
7	8	20-25	7	LemonTree	Veg	2999	
8	9	25-30	2	Ibis	Non-Veg	3456	
9	9	25-30	2	Ibis	Non-Veg	3456	
10	10	30-35	5	RedFox	non-Veg	-6755	

	NoOfPax	EstimatedSalary	Age_Group.1
0	2	40000	20-25
1	3	59000	30-35
2	2	30000	25-30
3	2	120000	20-25
4	2	45000	35+
5	2	122220	35+
6	-1	21122	35+
7	-10	345673	20-25
8	3	-99999	25-30
9	3	-99999	25-30
10	4	87777	30-35

[2]:
```
df.duplicated()
```

[2]:
```
0     False
1     False
2     False
3     False
```

```
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool
```

[3]:
```
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: 920.0+ bytes
```

[4]:
```
df.drop_duplicates(inplace=True)
df
```

[4]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1	20-25	4	Ibis	veg	1300	
1	2	30-35	5	LemonTree	Non-Veg	2000	
2	3	25-30	6	RedFox	Veg	1322	
3	4	20-25	-1	LemonTree	Veg	1234	
4	5	35+	3	Ibis	Vegetarian	989	
5	6	35+	3	Ibys	Non-Veg	1909	
6	7	35+	4	RedFox	Vegetarian	1000	
7	8	20-25	7	LemonTree	Veg	2999	
8	9	25-30	2	Ibis	Non-Veg	3456	
10	10	30-35	5	RedFox	non-Veg	-6755	

	NoOfPax	EstimatedSalary	Age_Group.1
0	2	40000	20-25
1	3	59000	30-35
2	2	30000	25-30

```
3          2          120000       20-25
4          2           45000         35+
5          2          122220         35+
6         -1           21122         35+
7        -10          345673       20-25
8          3          -99999       25-30
10         4           87777       30-35
```

[5]:
```
len(df)
```

[5]: 10

[6]:
```
index=np.array(list(range(0,len(df))))
df.set_index(index,inplace=True)
index

df
```

[6]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	\
0	1	20-25	4	Ibis	veg	1300	2	
1	2	30-35	5	LemonTree	Non-Veg	2000	3	
2	3	25-30	6	RedFox	Veg	1322	2	
3	4	20-25	-1	LemonTree	Veg	1234	2	
4	5	35+	3	Ibis	Vegetarian	989	2	
5	6	35+	3	Ibys	Non-Veg	1909	2	
6	7	35+	4	RedFox	Vegetarian	1000	-1	
7	8	20-25	7	LemonTree	Veg	2999	-10	
8	9	25-30	2	Ibis	Non-Veg	3456	3	
9	10	30-35	5	RedFox	non-Veg	-6755	4	

	EstimatedSalary	Age_Group.1
0	40000	20-25
1	59000	30-35
2	30000	25-30
3	120000	20-25
4	45000	35+
5	122220	35+
6	21122	35+
7	345673	20-25
8	-99999	25-30
9	87777	30-35

[7]:
```
df.drop(["Age_Group.1"],axis=1,inplace=True)
df
```

[7]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	\
0	1	20-25	4	Ibis	veg	1300	2	

1	2	30-35	5	LemonTree	Non-Veg	2000	3
2	3	25-30	6	RedFox	Veg	1322	2
3	4	20-25	-1	LemonTree	Veg	1234	2
4	5	35+	3	Ibis	Vegetarian	989	2
5	6	35+	3	Ibys	Non-Veg	1909	2
6	7	35+	4	RedFox	Vegetarian	1000	-1
7	8	20-25	7	LemonTree	Veg	2999	-10
8	9	25-30	2	Ibis	Non-Veg	3456	3
9	10	30-35	5	RedFox	non-Veg	-6755	4

	EstimatedSalary
0	40000
1	59000
2	30000
3	120000
4	45000
5	122220
6	21122
7	345673
8	-99999
9	87777

[8]:
```
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\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\core\indexing.py:1732: 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#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

[8]:

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

	NoOfPax	EstimatedSalary
0	2	40000.0
1	3	59000.0
2	2	30000.0
3	2	120000.0
4	2	45000.0
5	2	122220.0
6	-1	21122.0
7	-10	345673.0
8	3	NaN
9	4	87777.0

[9]:
```
df["NoOfPax"].loc[(df["NoOfPax"]<1) | (df["NoOfPax"]>20)]=np.nan
df
```

c:\users\asus\appdata\local\programs\python\python37\lib\site-packages\pandas\core\indexing.py:1732: 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#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)

[9]:

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

	NoOfPax	EstimatedSalary
0	2.0	40000.0
1	3.0	59000.0
2	2.0	30000.0
3	2.0	120000.0
4	2.0	45000.0
5	2.0	122220.0
6	NaN	21122.0
7	NaN	345673.0
8	3.0	NaN
9	4.0	87777.0

[11]:
```
df.Age_Group.unique()
```

[11]:
```
array(['20-25', '30-35', '25-30', '35+'], dtype=object)
```

[12]:
```
df.Hotel.unique()
```

[12]:
```
array(['Ibis', 'LemonTree', 'RedFox', 'Ibys'], dtype=object)
```

[13]:
```

```

[13]:
```
<bound method Series.unique of 0           veg
1        Non-Veg
2            Veg
3            Veg
4     Vegetarian
5        Non-Veg
6     Vegetarian
7            Veg
8        Non-Veg
9        non-Veg
Name: FoodPreference, dtype: object>
```

[14]:
```
df.FoodPreference.replace(["Vegetarian","veg"],"Veg",inplace=True)
df.FoodPreference.replace(["non-Veg"],"Non-Veg",inplace=True)
```

[15]:
```
df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()),inplace=True)
df.NoOfPax.fillna(round(df.NoOfPax.median()),inplace=True)
df["Rating(1-5)"].fillna(round(df["Rating(1-5)"].median()), inplace=True)
df.Bill.fillna(round(df.Bill.mean()),inplace=True)
df
```

[15]:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	\
0	1.0	20-25	4	Ibis	Veg	1300.0	
1	2.0	30-35	5	LemonTree	Non-Veg	2000.0	
2	3.0	25-30	6	RedFox	Veg	1322.0	
3	4.0	20-25	-1	LemonTree	Veg	1234.0	
4	5.0	35+	3	Ibis	Veg	989.0	
5	6.0	35+	3	Ibis	Non-Veg	1909.0	
6	7.0	35+	4	RedFox	Veg	1000.0	
7	8.0	20-25	7	LemonTree	Veg	2999.0	
8	9.0	25-30	2	Ibis	Non-Veg	3456.0	
9	10.0	30-35	5	RedFox	Non-Veg	1801.0	

	NoOfPax	EstimatedSalary
0	2.0	40000.0
1	3.0	59000.0

2	2.0	30000.0
3	2.0	120000.0
4	2.0	45000.0
5	2.0	122220.0
6	2.0	21122.0
7	2.0	345673.0
8	3.0	96755.0
9	4.0	87777.0

[ ]:

Iris-data-analysis-and-preprocessing

[3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
data=pd.read_csv('Iris.csv')
data
```

[3]:

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

	Species
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa
..	...
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica
149	Iris-virginica

[150 rows x 6 columns]

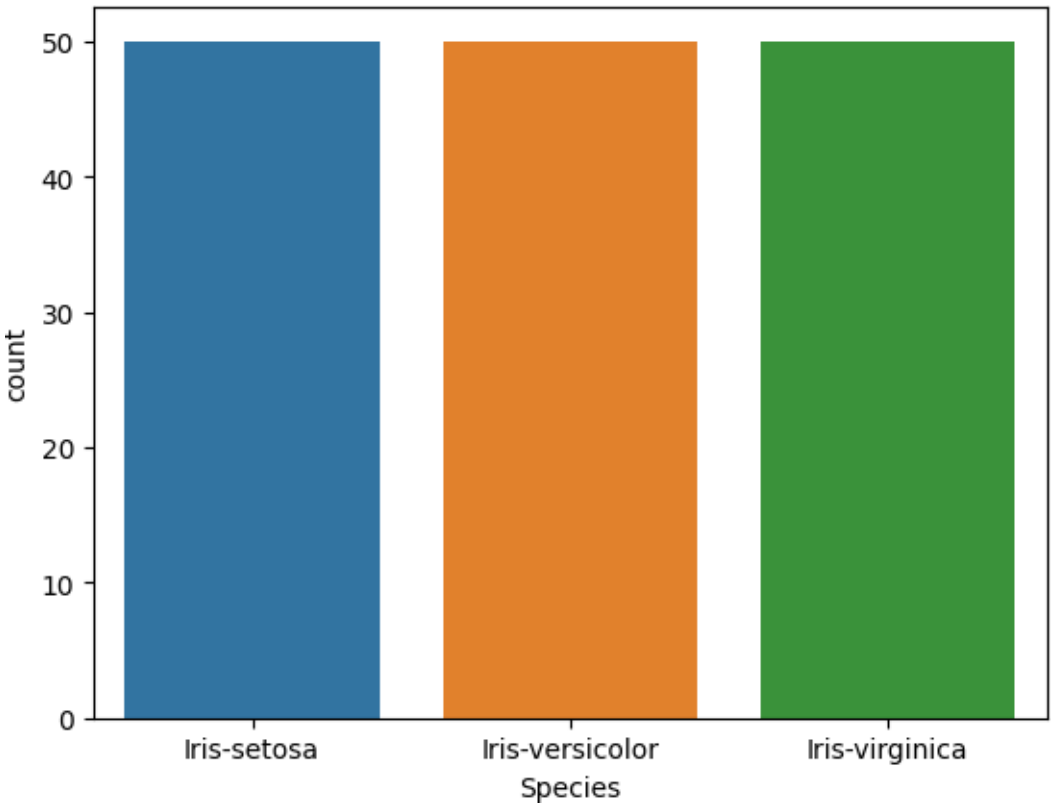
[4]:
```
data.info()
```

```
<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
```

[7]:
```
data.value_counts("Species")
```

[7]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

[11]:
```
sns.countplot(x="Species",data=data,)
plt.show()

dummies=pd.get_dummies(data.Species)
FinalDataset=pd.concat([pd.get_dummies(data.Species),data.iloc[:
  ↪,[0,1,2,3]]],axis=1)
FinalDataset.head()
```



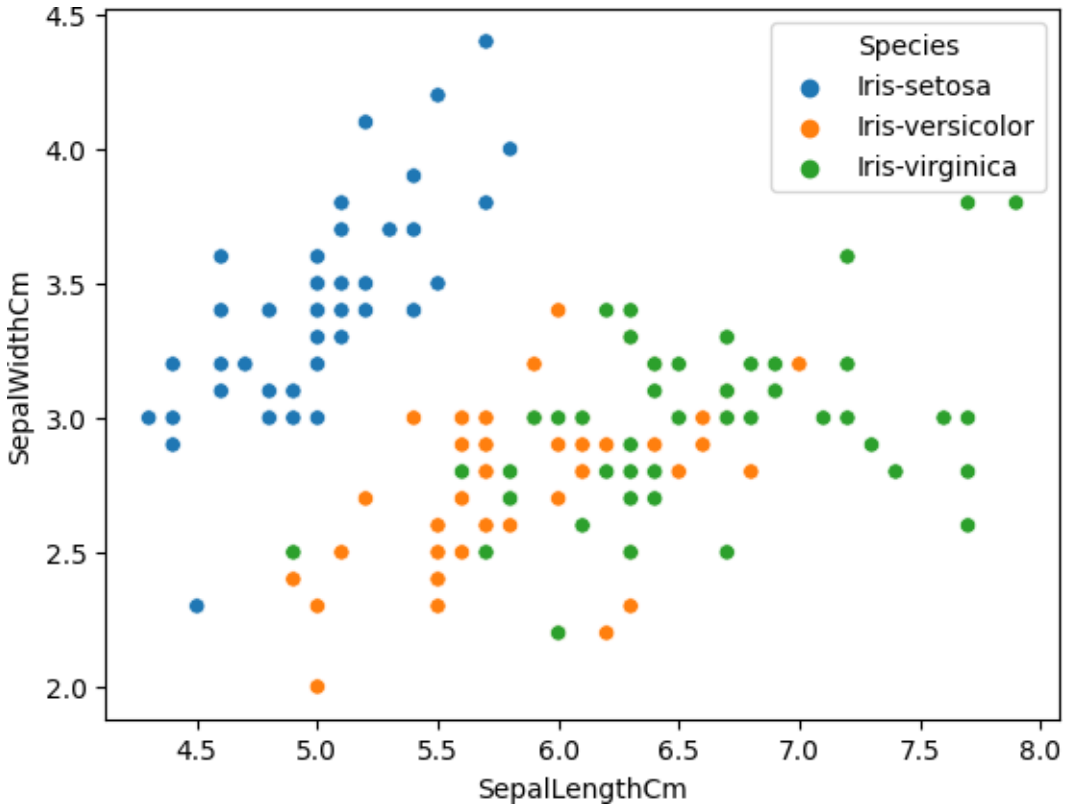
[11]:

	Iris-setosa	Iris-versicolor	Iris-virginica	Id	SepalLengthCm	\
0	1	0	0	1	5.1	
1	1	0	0	2	4.9	
2	1	0	0	3	4.7	
3	1	0	0	4	4.6	
4	1	0	0	5	5.0	

	SepalWidthCm	PetalLengthCm
0	3.5	1.4
1	3.0	1.4
2	3.2	1.3
3	3.1	1.5
4	3.6	1.4

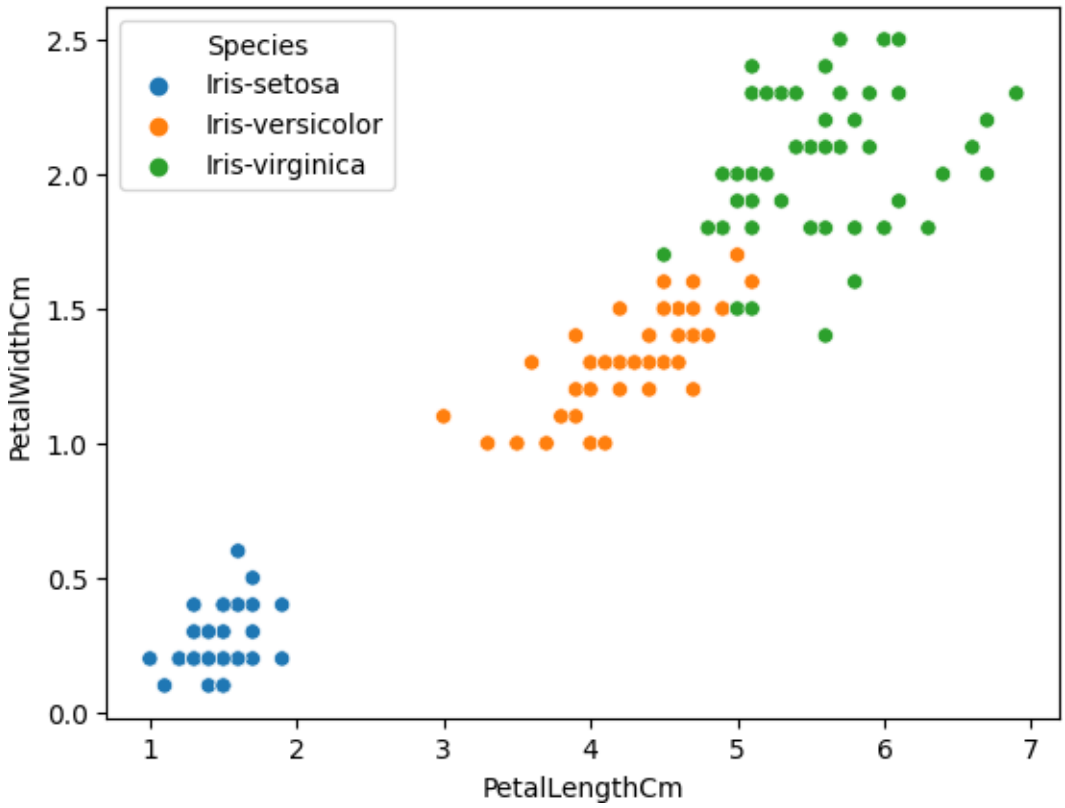
[12]:
```
sns.scatterplot(x="SepalLengthCm",y="SepalWidthCm",hue="Species",data=data)
```

[12]: <AxesSubplot:xlabel='SepalLengthCm', ylabel='SepalWidthCm'>

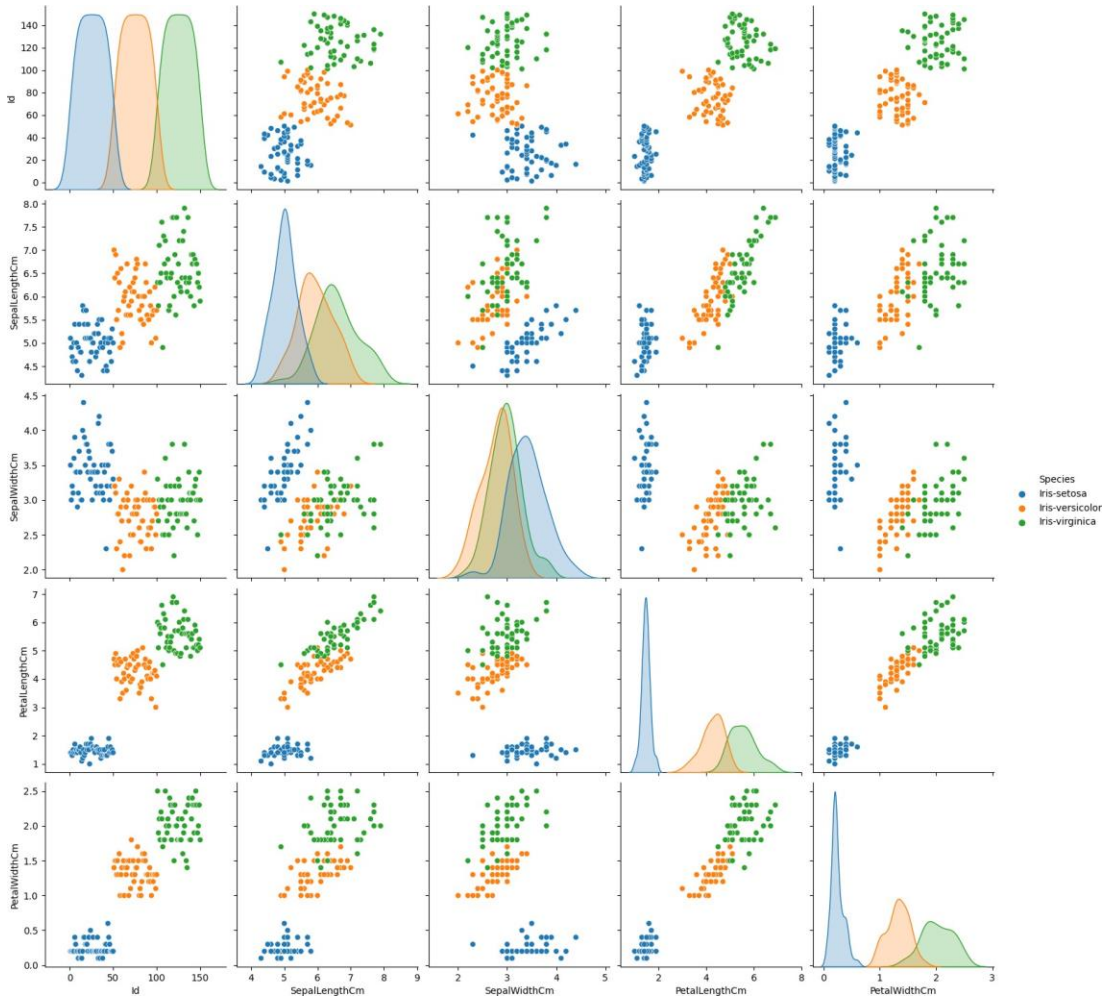


[13]:
```
sns.scatterplot(x="PetalLengthCm",y="PetalWidthCm",hue="Species",data=data)
```

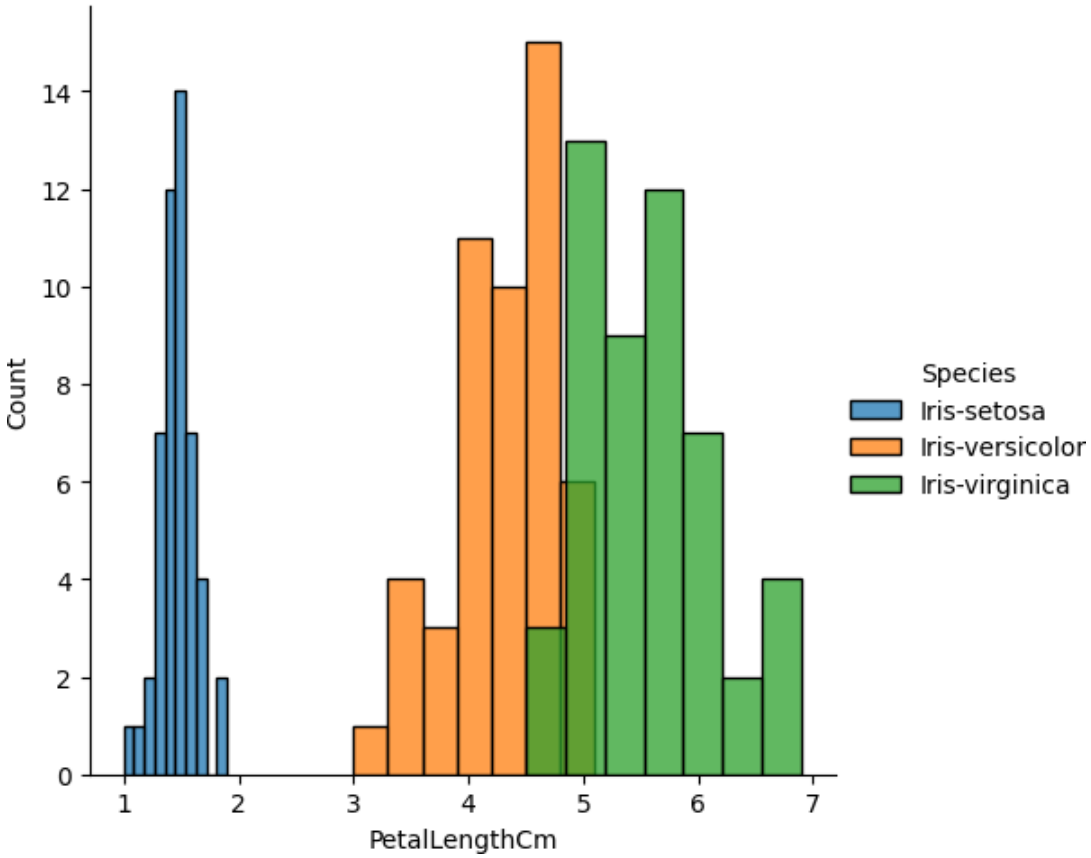
[13]: <AxesSubplot:xlabel='PetalLengthCm', ylabel='PetalWidthCm'>



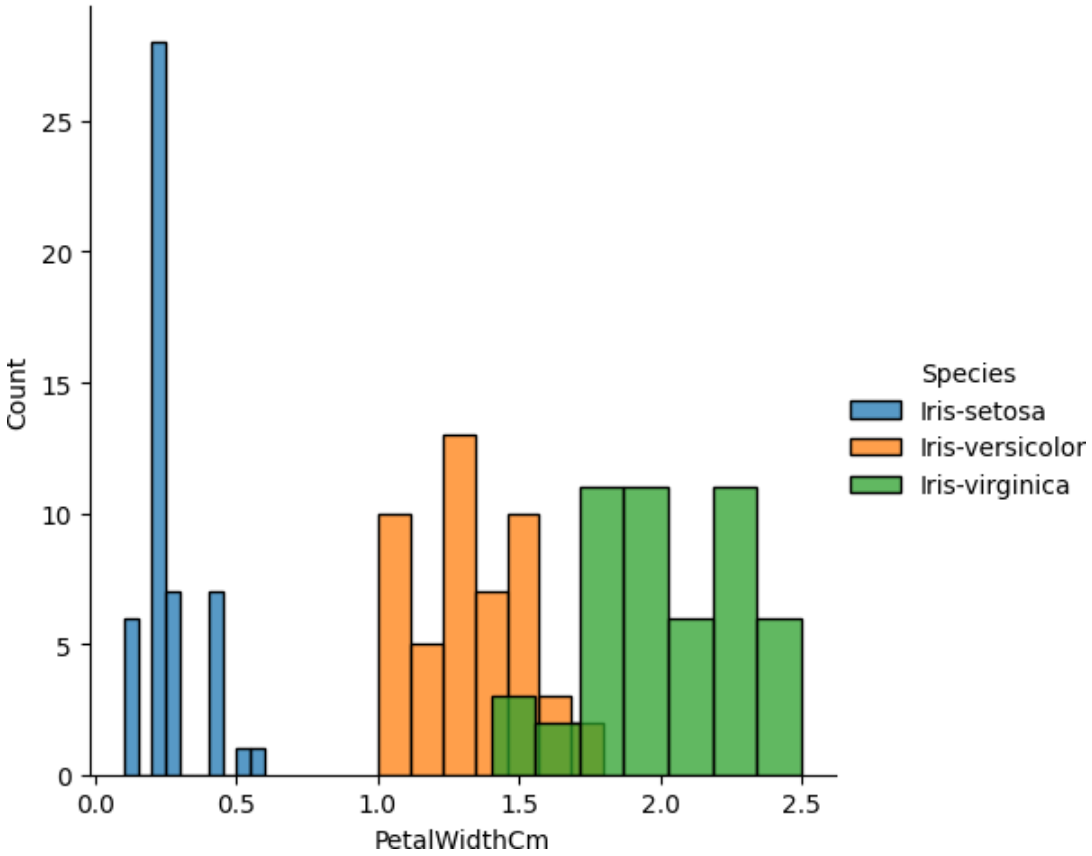
[14]:
```
sns.pairplot(data,hue="Species",height=3);
```



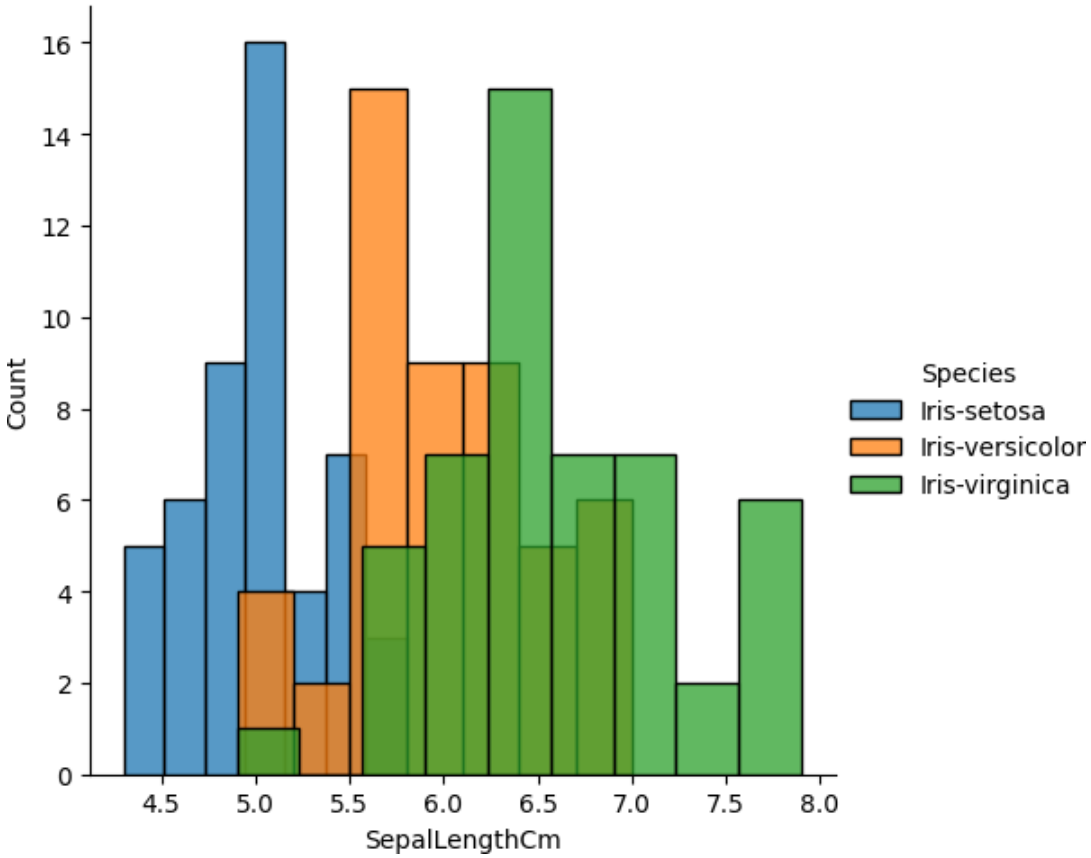
[15]:
```
sns.FacetGrid(data,hue="Species",height=5).map(sns.histplot,"PetalLengthCm").
↪add_legend(); plt.show();
```



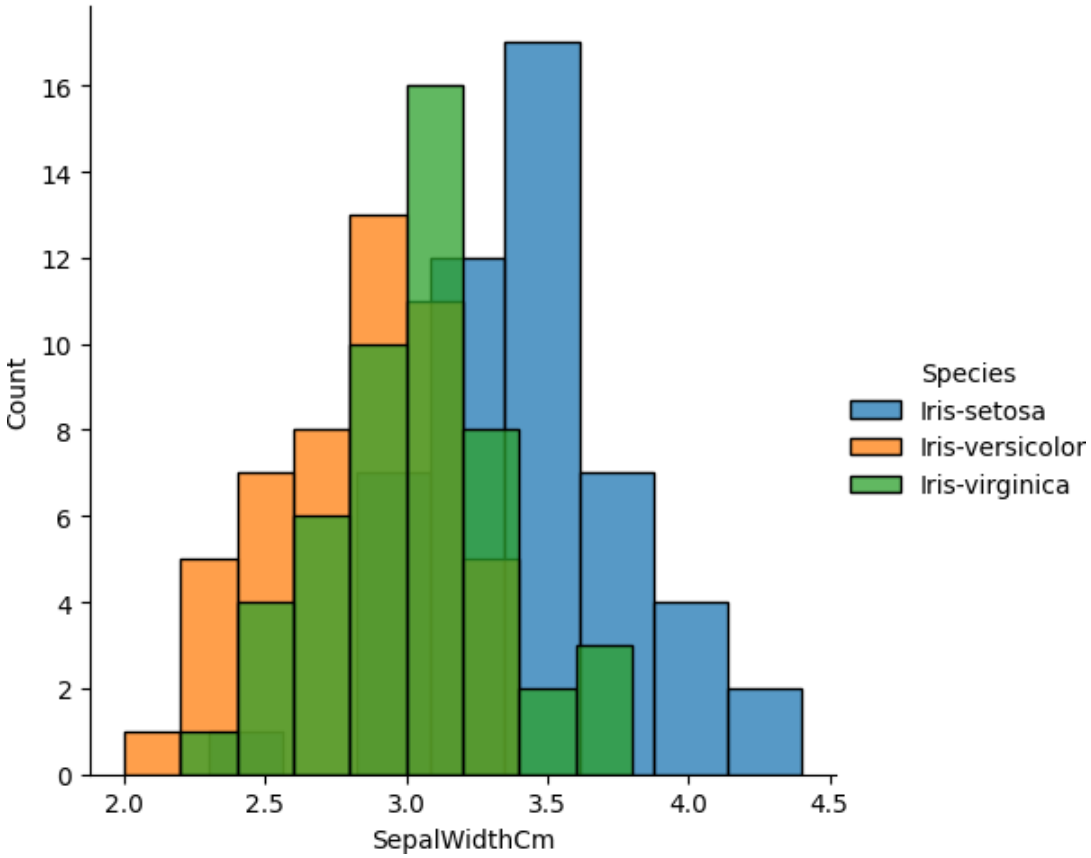
[16]:
```
sns.FacetGrid(data,hue="Species",height=5).map(sns.histplot,"PetalWidthCm").
↪add_legend(); plt.show();
```



[17]:
```
sns.FacetGrid(data,hue="Species",height=5).map(sns.histplot,"SepalLengthCm").
↪add_legend(); plt.show();
```



[18]:
```
sns.FacetGrid(data,hue="Species",height=5).map(sns.histplot,"SepalWidthCm").
↪add_legend();
plt.show();
```


Count
SepalWidthCm
Species
Iris-setosa
Iris-versicolor
Iris-virginica

[ ]:

kmeans-clustering

[1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("Social_Network_Ads.csv")
df
```

[1]:

	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 x 5 columns]

[2]:
```
df.head()
```

[2]:

	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

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

[3]:
```
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],
[     28,  79000],
[     22,  18000],
[     32, 117000],
[     27,  20000],
[     25,  87000],
[     23,  66000],
[     32, 120000],
[     59,  83000],
[     24,  58000],
[     24,  19000],
[     23,  82000],
[     22,  63000],
[     31,  68000],
[     25,  80000],
[     24,  27000],
[     20,  23000],
[     33, 113000],
[     32,  18000],
[     34, 112000],
[     18,  52000],
[     22,  27000],
[     28,  87000],
[     26,  17000],
[     30,  80000],
[     39,  42000],
[     20,  49000],
[     35,  88000],
[     30,  62000],
[     31, 118000],
[     24,  55000],
[     28,  85000],
[     26,  81000],
[     35,  50000],
[     22,  81000],
[     30, 116000],
[     26,  15000],
[     29,  28000],
[     29,  83000],
[     35,  44000],
[     35,  25000],
[     28, 123000],
```

```
[     35,  73000],
[     28,  37000],
[     27,  88000],
[     28,  59000],
[     32,  86000],
[     33, 149000],
[     19,  21000],
[     21,  72000],
[     26,  35000],
[     27,  89000],
[     26,  86000],
[     38,  80000],
[     39,  71000],
[     37,  71000],
[     38,  61000],
[     37,  55000],
[     42,  80000],
[     40,  57000],
[     35,  75000],
[     36,  52000],
[     40,  59000],
[     41,  59000],
[     36,  75000],
[     37,  72000],
[     40,  75000],
[     35,  53000],
[     41,  51000],
[     39,  61000],
[     42,  65000],
[     26,  32000],
[     30,  17000],
[     26,  84000],
[     31,  58000],
[     33,  31000],
[     30,  87000],
[     21,  68000],
[     28,  55000],
[     23,  63000],
[     20,  82000],
[     30, 107000],
[     28,  59000],
[     19,  25000],
[     19,  85000],
[     18,  68000],
[     35,  59000],
[     30,  89000],
[     34,  25000],
```

```
[    24,  89000],
[    27,  96000],
[    41,  30000],
[    29,  61000],
[    20,  74000],
[    26,  15000],
[    41,  45000],
[    31,  76000],
[    36,  50000],
[    40,  47000],
[    31,  15000],
[    46,  59000],
[    29,  75000],
[    26,  30000],
[    32, 135000],
[    32, 100000],
[    25,  90000],
[    37,  33000],
[    35,  38000],
[    33,  69000],
[    18,  86000],
[    22,  55000],
[    35,  71000],
[    29, 148000],
[    29,  47000],
[    21,  88000],
[    34, 115000],
[    26, 118000],
[    34,  43000],
[    34,  72000],
[    23,  28000],
[    35,  47000],
[    25,  22000],
[    24,  23000],
[    31,  34000],
[    26,  16000],
[    31,  71000],
[    32, 117000],
[    33,  43000],
[    33,  60000],
[    31,  66000],
[    20,  82000],
[    33,  41000],
[    35,  72000],
[    28,  32000],
[    24,  84000],
[    19,  26000],
```

```
[    29,  43000],
[    19,  70000],
[    28,  89000],
[    34,  43000],
[    30,  79000],
[    20,  36000],
[    26,  80000],
[    35,  22000],
[    35,  39000],
[    49,  74000],
[    39, 134000],
[    41,  71000],
[    58, 101000],
[    47,  47000],
[    55, 130000],
[    52, 114000],
[    40, 142000],
[    46,  22000],
[    48,  96000],
[    52, 150000],
[    59,  42000],
[    35,  58000],
[    47,  43000],
[    60, 108000],
[    49,  65000],
[    40,  78000],
[    46,  96000],
[    59, 143000],
[    41,  80000],
[    35,  91000],
[    37, 144000],
[    60, 102000],
[    35,  60000],
[    37,  53000],
[    36, 126000],
[    56, 133000],
[    40,  72000],
[    42,  80000],
[    35, 147000],
[    39,  42000],
[    40, 107000],
[    49,  86000],
[    38, 112000],
[    46,  79000],
[    40,  57000],
[    37,  80000],
[    46,  82000],
```

```
[     53, 143000],
[     42, 149000],
[     38,  59000],
[     50,  88000],
[     56, 104000],
[     41,  72000],
[     51, 146000],
[     35,  50000],
[     57, 122000],
[     41,  52000],
[     35,  97000],
[     44,  39000],
[     37,  52000],
[     48, 134000],
[     37, 146000],
[     50,  44000],
[     52,  90000],
[     41,  72000],
[     40,  57000],
[     58,  95000],
[     45, 131000],
[     35,  77000],
[     36, 144000],
[     55, 125000],
[     35,  72000],
[     48,  90000],
[     42, 108000],
[     40,  75000],
[     37,  74000],
[     47, 144000],
[     40,  61000],
[     43, 133000],
[     59,  76000],
[     60,  42000],
[     39, 106000],
[     57,  26000],
[     57,  74000],
[     38,  71000],
[     49,  88000],
[     52,  38000],
[     50,  36000],
[     59,  88000],
[     35,  61000],
[     37,  70000],
[     52,  21000],
[     48, 141000],
[     37,  93000],
```

```
[     37,  62000],
[     48, 138000],
[     41,  79000],
[     37,  78000],
[     39, 134000],
[     49,  89000],
[     55,  39000],
[     37,  77000],
[     35,  57000],
[     36,  63000],
[     42,  73000],
[     43, 112000],
[     45,  79000],
[     46, 117000],
[     58,  38000],
[     48,  74000],
[     37, 137000],
[     37,  79000],
[     40,  60000],
[     42,  54000],
[     51, 134000],
[     47, 113000],
[     36, 125000],
[     38,  50000],
[     42,  70000],
[     39,  96000],
[     38,  50000],
[     49, 141000],
[     39,  79000],
[     39,  75000],
[     54, 104000],
[     35,  55000],
[     45,  32000],
[     36,  60000],
[     52, 138000],
[     53,  82000],
[     41,  52000],
[     48,  30000],
[     48, 131000],
[     41,  60000],
[     41,  72000],
[     42,  75000],
[     36, 118000],
[     47, 107000],
[     38,  51000],
[     48, 119000],
[     42,  65000],
```

```
[     40,  65000],
[     57,  60000],
[     36,  54000],
[     58, 144000],
[     35,  79000],
[     38,  55000],
[     39, 122000],
[     53, 104000],
[     35,  75000],
[     38,  65000],
[     47,  51000],
[     47, 105000],
[     41,  63000],
[     53,  72000],
[     54, 108000],
[     39,  77000],
[     38,  61000],
[     38, 113000],
[     37,  75000],
[     42,  90000],
[     37,  57000],
[     36,  99000],
[     60,  34000],
[     54,  70000],
[     41,  72000],
[     40,  71000],
[     42,  54000],
[     43, 129000],
[     53,  34000],
[     47,  50000],
[     42,  79000],
[     42, 104000],
[     59,  29000],
[     58,  47000],
[     46,  88000],
[     38,  71000],
[     54,  26000],
[     60,  46000],
[     60,  83000],
[     39,  73000],
[     59, 130000],
[     37,  80000],
[     46,  32000],
[     46,  74000],
[     42,  53000],
[     41,  87000],
[     58,  23000],
```

```
       [    42,  64000],
       [    48,  33000],
       [    44, 139000],
       [    49,  28000],
       [    57,  33000],
       [    56,  60000],
       [    49,  39000],
       [    39,  71000],
       [    47,  34000],
       [    48,  35000],
       [    48,  33000],
       [    47,  23000],
       [    45,  45000],
       [    60,  42000],
       [    39,  59000],
       [    46,  41000],
       [    51,  23000],
       [    50,  20000],
       [    36,  33000],
       [    49,  36000]], dtype=int64)
```

[4]:
```
label
```

[4]:
```
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], dtype=int64)
```

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

[7]:
```
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)
    if test_score>train_score:
        print("Test {} Train{} Random State {}".
↪format(test_score,train_score,i))
```

```
Test 0.6875 Train0.63125 Random State 3
Test 0.7375 Train0.61875 Random State 4
Test 0.6625 Train0.6375 Random State 5
Test 0.65 Train0.640625 Random State 6
Test 0.675 Train0.634375 Random State 7
Test 0.675 Train0.634375 Random State 8
Test 0.65 Train0.640625 Random State 10
Test 0.6625 Train0.6375 Random State 11
Test 0.7125 Train0.625 Random State 13
Test 0.675 Train0.634375 Random State 16
Test 0.7 Train0.628125 Random State 17
Test 0.7 Train0.628125 Random State 21
Test 0.65 Train0.640625 Random State 24
Test 0.6625 Train0.6375 Random State 25
Test 0.75 Train0.615625 Random State 26
Test 0.675 Train0.634375 Random State 27
Test 0.7 Train0.628125 Random State 28
Test 0.6875 Train0.63125 Random State 29
Test 0.6875 Train0.63125 Random State 31
Test 0.6625 Train0.6375 Random State 37
Test 0.7 Train0.628125 Random State 39
Test 0.7 Train0.628125 Random State 40
Test 0.65 Train0.640625 Random State 42
Test 0.725 Train0.621875 Random State 46
Test 0.65 Train0.640625 Random State 48
Test 0.675 Train0.634375 Random State 50
Test 0.65 Train0.640625 Random State 51
Test 0.65 Train0.640625 Random State 54
Test 0.7 Train0.634375 Random State 55
Test 0.65 Train0.640625 Random State 56
Test 0.6625 Train0.6375 Random State 58
Test 0.6875 Train0.63125 Random State 59
Test 0.7 Train0.628125 Random State 60
Test 0.6625 Train0.6375 Random State 62
Test 0.6875 Train0.63125 Random State 63
Test 0.65 Train0.640625 Random State 66
```

Test 0.7 Train0.628125 Random State 70
Test 0.65 Train0.640625 Random State 74
Test 0.65 Train0.640625 Random State 75
Test 0.6875 Train0.63125 Random State 76
Test 0.6875 Train0.63125 Random State 80
Test 0.675 Train0.634375 Random State 81
Test 0.875 Train0.8375 Random State 82
Test 0.7 Train0.628125 Random State 83
Test 0.675 Train0.634375 Random State 84
Test 0.675 Train0.634375 Random State 86
Test 0.65 Train0.640625 Random State 87
Test 0.675 Train0.634375 Random State 90
Test 0.65 Train0.640625 Random State 91
Test 0.7 Train0.628125 Random State 93
Test 0.7375 Train0.61875 Random State 94
Test 0.65 Train0.640625 Random State 97
Test 0.7 Train0.628125 Random State 99
Test 0.675 Train0.634375 Random State 101
Test 0.6625 Train0.6375 Random State 102
Test 0.725 Train0.621875 Random State 103
Test 0.65 Train0.640625 Random State 106
Test 0.65 Train0.640625 Random State 109
Test 0.75 Train0.615625 Random State 114
Test 0.675 Train0.634375 Random State 116
Test 0.65 Train0.640625 Random State 117
Test 0.675 Train0.634375 Random State 119
Test 0.65 Train0.640625 Random State 120
Test 0.6625 Train0.6375 Random State 121
Test 0.725 Train0.621875 Random State 125
Test 0.65 Train0.640625 Random State 127
Test 0.65 Train0.640625 Random State 128
Test 0.6875 Train0.63125 Random State 129
Test 0.6875 Train0.63125 Random State 130
Test 0.6625 Train0.6375 Random State 132
Test 0.6875 Train0.63125 Random State 133
Test 0.675 Train0.634375 Random State 134
Test 0.675 Train0.634375 Random State 138
Test 0.7 Train0.628125 Random State 139
Test 0.7125 Train0.63125 Random State 141
Test 0.725 Train0.621875 Random State 142
Test 0.6625 Train0.6375 Random State 143
Test 0.6625 Train0.6375 Random State 145
Test 0.7125 Train0.625 Random State 150
Test 0.65 Train0.640625 Random State 152
Test 0.6625 Train0.6375 Random State 154
Test 0.675 Train0.634375 Random State 155
Test 0.8875 Train0.834375 Random State 158
Test 0.6625 Train0.6375 Random State 159

Test 0.7125 Train0.625 Random State 161
Test 0.675 Train0.634375 Random State 162
Test 0.6625 Train0.6375 Random State 163
Test 0.65 Train0.640625 Random State 165
Test 0.6625 Train0.6375 Random State 169
Test 0.675 Train0.634375 Random State 170
Test 0.7125 Train0.625 Random State 173
Test 0.65 Train0.640625 Random State 176
Test 0.6625 Train0.6375 Random State 178
Test 0.6625 Train0.6375 Random State 179
Test 0.6625 Train0.6375 Random State 180
Test 0.6625 Train0.6375 Random State 181
Test 0.65 Train0.640625 Random State 184
Test 0.6625 Train0.6375 Random State 185
Test 0.675 Train0.634375 Random State 188
Test 0.7375 Train0.61875 Random State 189
Test 0.7 Train0.628125 Random State 192
Test 0.65 Train0.640625 Random State 193
Test 0.7 Train0.628125 Random State 194
Test 0.65 Train0.640625 Random State 195
Test 0.6625 Train0.6375 Random State 196
Test 0.675 Train0.634375 Random State 198
Test 0.8875 Train0.8375 Random State 199
Test 0.6875 Train0.63125 Random State 204
Test 0.6625 Train0.6375 Random State 209
Test 0.7 Train0.628125 Random State 211
Test 0.65 Train0.640625 Random State 212
Test 0.6625 Train0.6375 Random State 215
Test 0.6625 Train0.6375 Random State 217
Test 0.6875 Train0.63125 Random State 220
Test 0.6625 Train0.6375 Random State 223
Test 0.6625 Train0.6375 Random State 225
Test 0.6625 Train0.6375 Random State 226
Test 0.6875 Train0.63125 Random State 229
Test 0.65 Train0.640625 Random State 232
Test 0.7125 Train0.625 Random State 233
Test 0.6625 Train0.6375 Random State 234
Test 0.6625 Train0.6375 Random State 235
Test 0.6875 Train0.63125 Random State 238
Test 0.725 Train0.621875 Random State 239
Test 0.65 Train0.640625 Random State 241
Test 0.725 Train0.621875 Random State 242
Test 0.6625 Train0.6375 Random State 244
Test 0.675 Train0.634375 Random State 245
Test 0.6875 Train0.63125 Random State 246
Test 0.7 Train0.628125 Random State 247
Test 0.6875 Train0.63125 Random State 248
Test 0.65 Train0.640625 Random State 251

Test 0.7 Train0.628125 Random State 252
Test 0.65 Train0.640625 Random State 253
Test 0.675 Train0.634375 Random State 255
Test 0.75 Train0.615625 Random State 257
Test 0.7 Train0.628125 Random State 260
Test 0.6625 Train0.6375 Random State 261
Test 0.65 Train0.640625 Random State 263
Test 0.6625 Train0.6375 Random State 265
Test 0.8625 Train0.840625 Random State 266
Test 0.6875 Train0.63125 Random State 269
Test 0.6625 Train0.6375 Random State 275
Test 0.7 Train0.628125 Random State 276
Test 0.6625 Train0.6375 Random State 277
Test 0.7 Train0.628125 Random State 278
Test 0.7125 Train0.625 Random State 279
Test 0.6875 Train0.63125 Random State 282
Test 0.6875 Train0.63125 Random State 283
Test 0.7125 Train0.625 Random State 287
Test 0.6625 Train0.6375 Random State 292
Test 0.65 Train0.640625 Random State 293
Test 0.6625 Train0.6375 Random State 294
Test 0.675 Train0.634375 Random State 296
Test 0.675 Train0.634375 Random State 300
Test 0.675 Train0.634375 Random State 302
Test 0.6625 Train0.6375 Random State 303
Test 0.8625 Train0.834375 Random State 305
Test 0.6875 Train0.63125 Random State 306
Test 0.7 Train0.628125 Random State 310
Test 0.7125 Train0.625 Random State 311
Test 0.8625 Train0.834375 Random State 313
Test 0.9125 Train0.834375 Random State 314
Test 0.7 Train0.628125 Random State 315
Test 0.6625 Train0.6375 Random State 317
Test 0.7625 Train0.6125 Random State 318
Test 0.6625 Train0.6375 Random State 319
Test 0.65 Train0.640625 Random State 321
Test 0.7125 Train0.625 Random State 322
Test 0.675 Train0.634375 Random State 323
Test 0.6625 Train0.6375 Random State 325
Test 0.7125 Train0.625 Random State 327
Test 0.6625 Train0.6375 Random State 328
Test 0.7 Train0.628125 Random State 329
Test 0.65 Train0.640625 Random State 330
Test 0.65 Train0.640625 Random State 332
Test 0.675 Train0.634375 Random State 336
Test 0.6875 Train0.63125 Random State 340
Test 0.65 Train0.640625 Random State 344
Test 0.6625 Train0.6375 Random State 345

```
Test 0.7 Train0.628125 Random State 346
Test 0.65 Train0.640625 Random State 348
Test 0.725 Train0.621875 Random State 349
Test 0.6875 Train0.63125 Random State 350
Test 0.675 Train0.634375 Random State 352
Test 0.725 Train0.621875 Random State 353
Test 0.675 Train0.634375 Random State 354
Test 0.6875 Train0.63125 Random State 355
Test 0.6625 Train0.6375 Random State 356
Test 0.7375 Train0.61875 Random State 357
Test 0.6625 Train0.6375 Random State 358
Test 0.6625 Train0.6375 Random State 359
Test 0.7 Train0.628125 Random State 360
Test 0.65 Train0.640625 Random State 361
Test 0.6625 Train0.6375 Random State 362
Test 0.65 Train0.640625 Random State 363
Test 0.6625 Train0.6375 Random State 364
Test 0.6875 Train0.63125 Random State 365
Test 0.6625 Train0.6375 Random State 366
Test 0.6625 Train0.6375 Random State 368
Test 0.65 Train0.640625 Random State 370
Test 0.725 Train0.621875 Random State 371
Test 0.65 Train0.640625 Random State 373
Test 0.7 Train0.628125 Random State 376
Test 0.6875 Train0.63125 Random State 378
Test 0.675 Train0.634375 Random State 379
Test 0.65 Train0.640625 Random State 387
Test 0.6625 Train0.6375 Random State 393
Test 0.675 Train0.634375 Random State 396
Test 0.7 Train0.628125 Random State 397
Test 0.7125 Train0.625 Random State 400
```

[9]:
```
x_train,x_test,y_train,y_test=train_test_split(features,label,test_size=0.
↪2,random_state=i)
finalModel=LogisticRegression()
finalModel.fit(x_train,y_train)
```

[9]:
```
LogisticRegression()
```

[10]:
```
print(finalModel.score(x_train,y_train))
print(finalModel.score(x_test,y_test))
```

```
0.625
0.7125
```

[12]:
```
from sklearn.metrics import classification_report
```

```
print(classification_report(label,  finalModel.predict(features),␣
↪zero_division=1))
```

	precision	recall	f1-score	support
0	0.64	1.00	0.78	257
1	1.00	0.00	0.00	143
accuracy			0.64	400
macro avg	0.82	0.50	0.39	400
weighted avg	0.77	0.64	0.50	400

[ ]:

Predicting-salary-based-on-experience

[1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("Salary_data.csv")
df
```

[1]:

	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

[2]:
```
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: 608.0 bytes
```

[4]:
```
df.dropna(inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 720.0 bytes
```

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

[5]:

	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

[8]:
```
features=df.iloc[:,[0]].values
label=df.iloc[:,[1]].values
```

[21]:
```
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=42)
```

[22]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

[22]: LinearRegression()

[23]:
```
model.score(x_train,y_train)
```

[23]: 0.9645401573418146

[24]:
```
model.score(x_test,y_test)
```

[24]: 0.9024461774180497

[25]:
```
model.coef_
```

[25]: array([[9423.81532303]])

[26]:
```
model.intercept_
```

[26]: array([25321.58301178])

[27]:
```
import pickle
pickle.dump(model,open("SalaryPred.model","wb"))
```

[28]:
```
model=pickle.load(open("SalaryPred.model","rb"))
```

[29]:
```
yr_of_exp=float(input("Enter Years of Experience: "))
yr_of_exp_NP=np.array([[yr_of_exp]])
Salary=model.predict(yr_of_exp_NP)
```

```
Enter Years of Experience: 44
```

[31]:
```
print("Estimated Salary for {} years of experience is {}: ".format(yr_of_exp,
↪Salary))
```

```
Estimated Salary for 44.0 years of experience is [[439969.45722514]]:
```

[ ]:

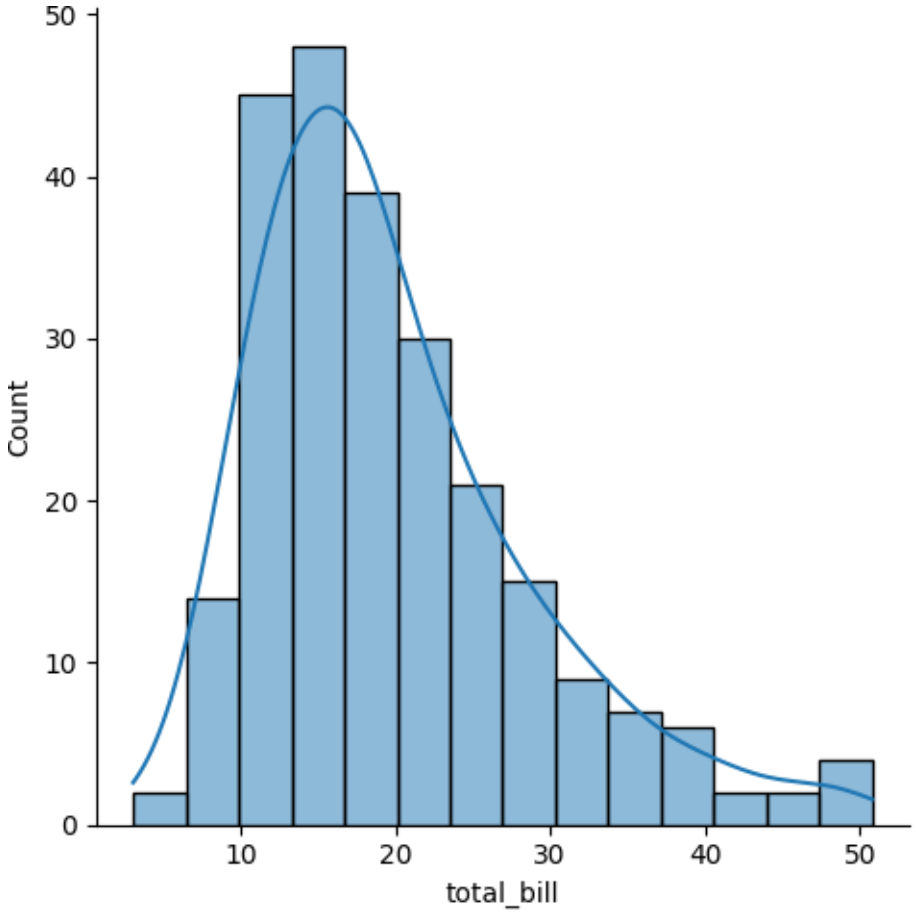
Laboratory-data-analysis-with-seaborn

November 21, 2024

[ ]:
```
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()
```

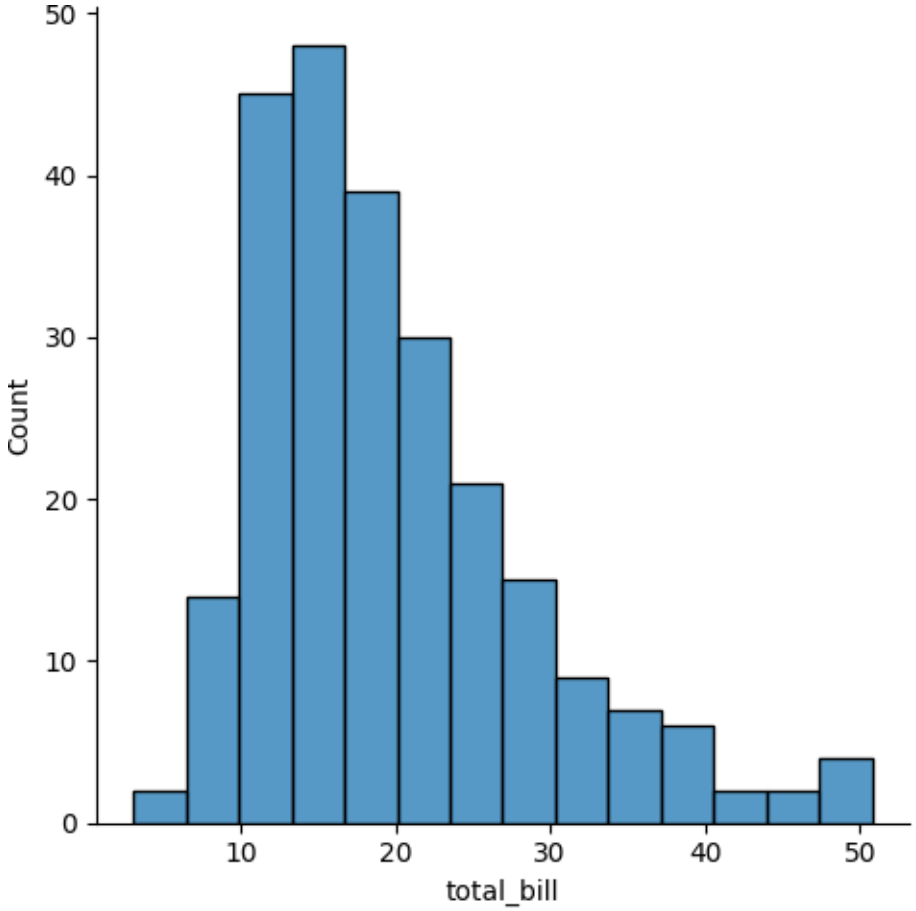
[2]:
```
sns.displot(tips.total_bill,kde=True)
```

[2] :
```
<seaborn.axisgrid.FacetGrid at 0x132efab8348>
```


50
40
30
20
10
0
Count
10
20
30
40
50
total_bill

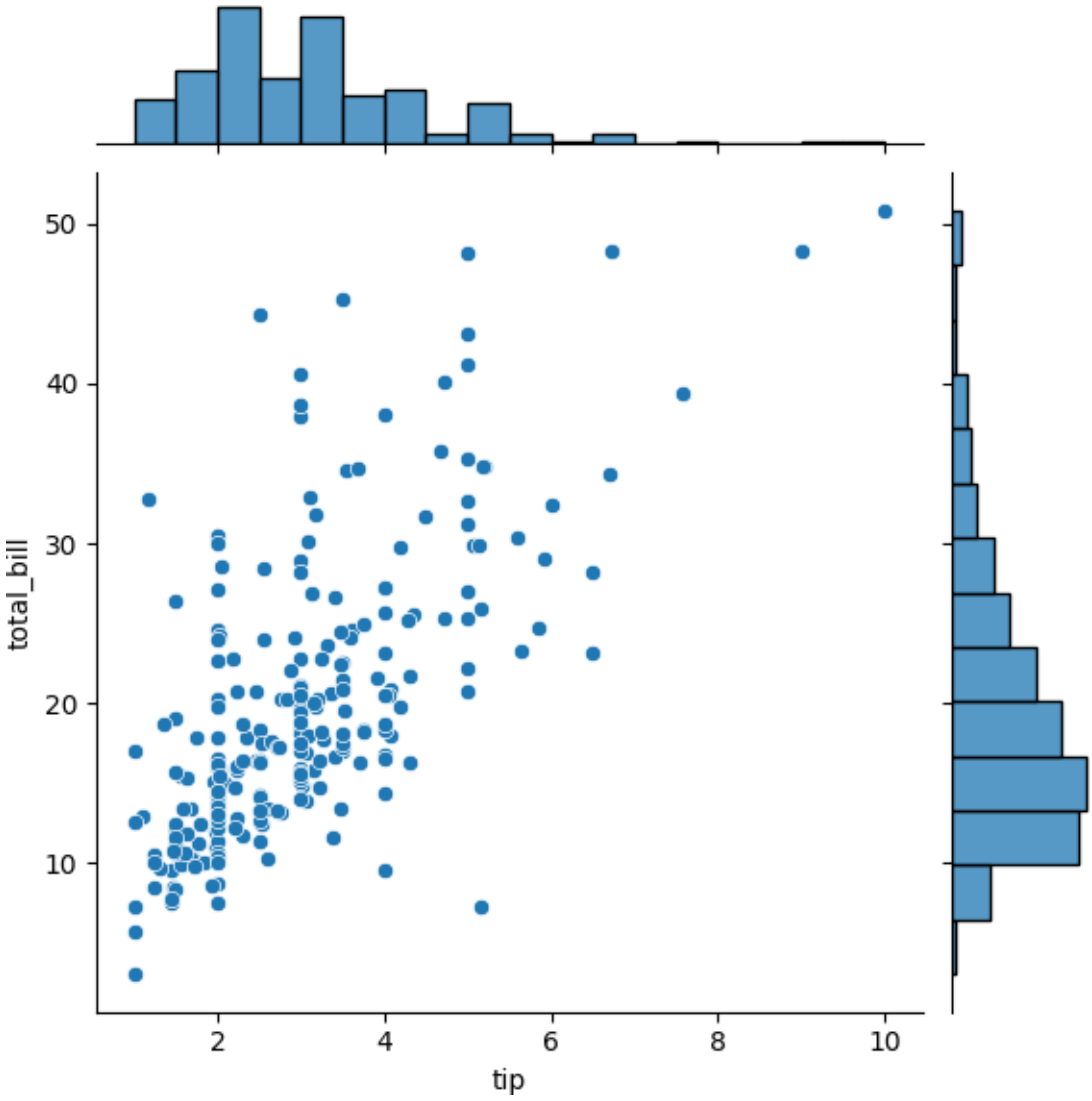
[3]:
```
sns.displot(tips.total_bill,kde=False)
```

[3] : <seaborn.axisgrid.FacetGrid at 0x132f1e88148>



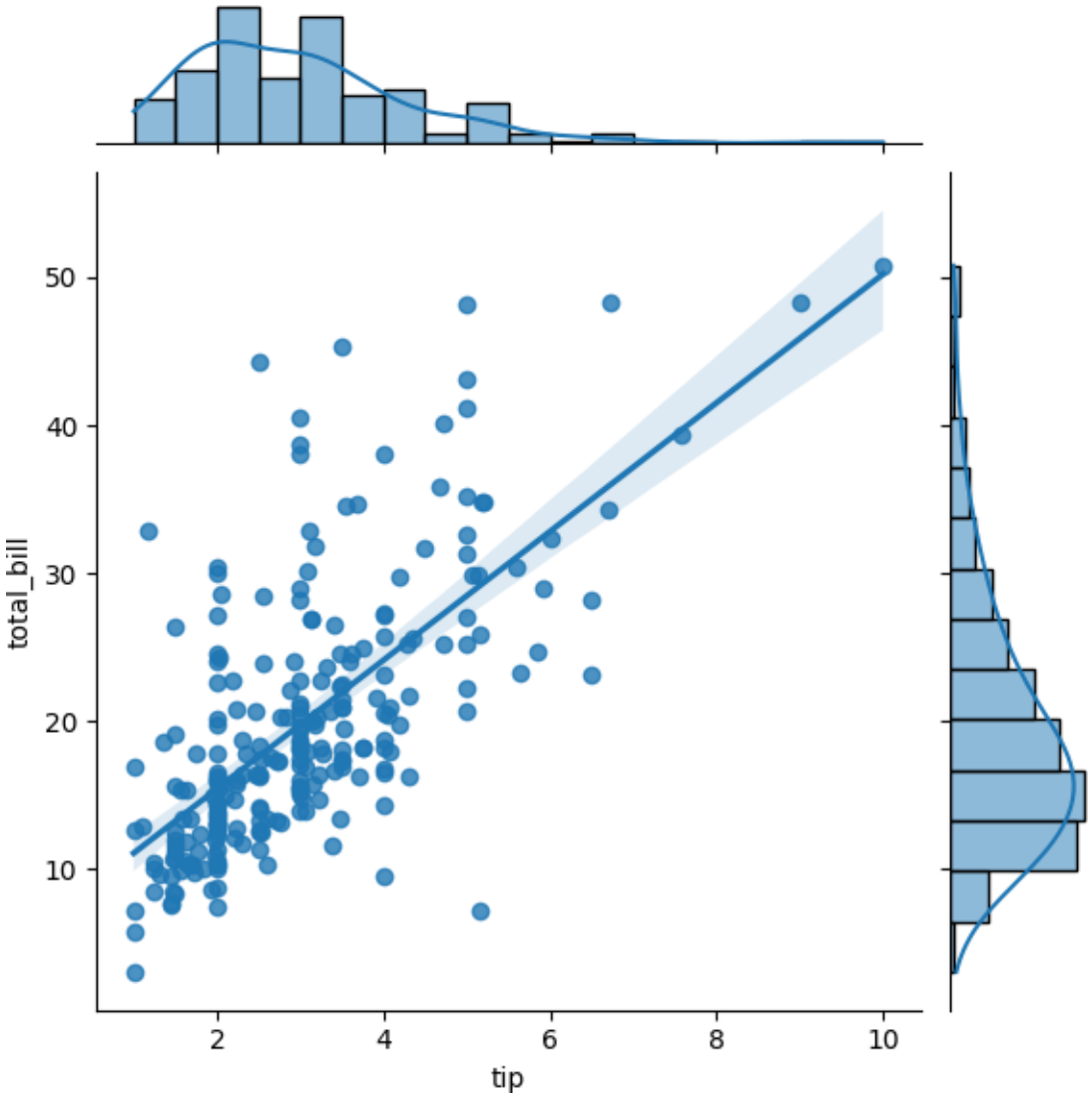
[4]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

[4] : <seaborn.axisgrid.JointGrid at 0x132f1f636c8>



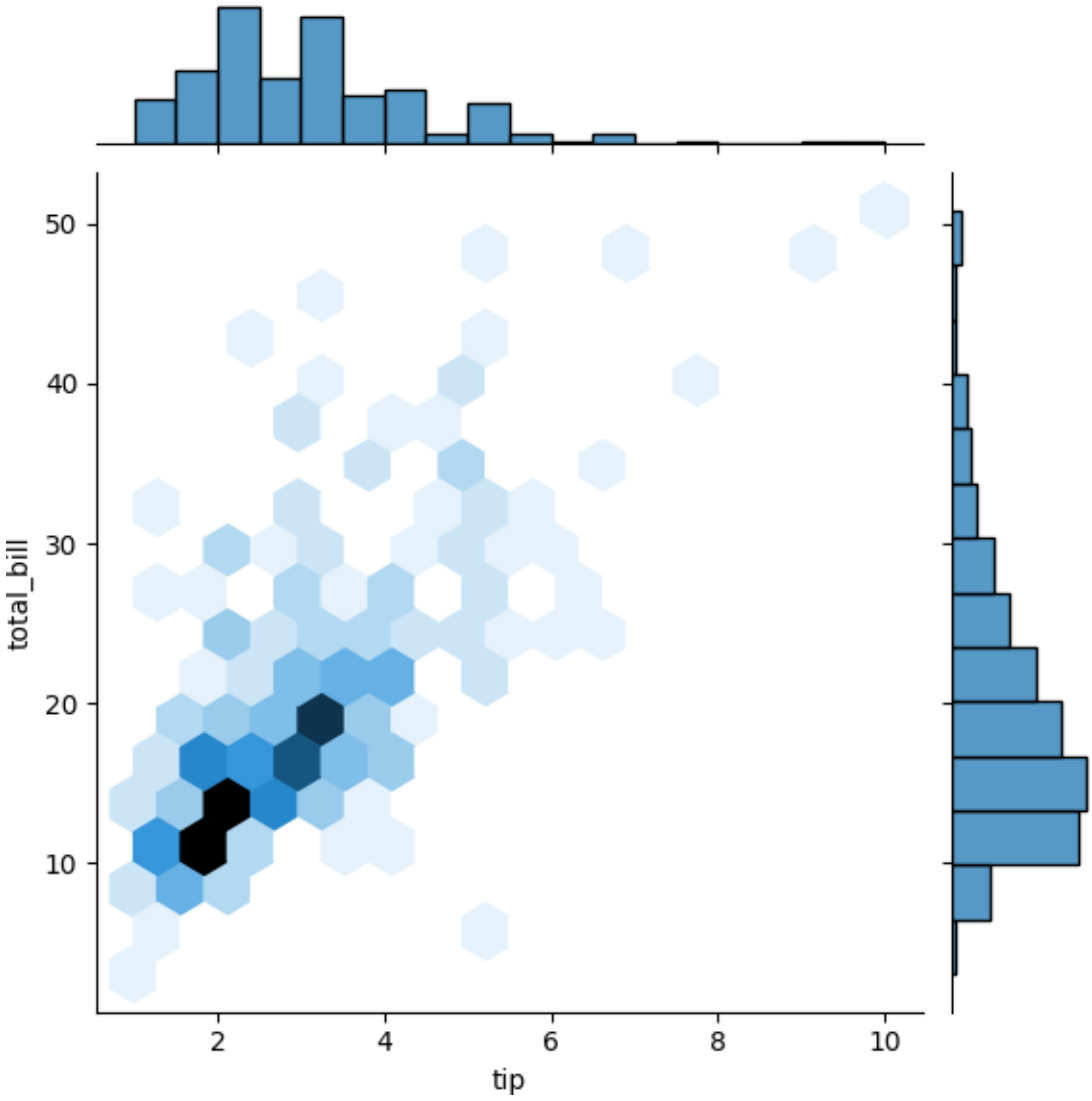
[5]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

[5] : <seaborn.axisgrid.JointGrid at 0x132f2224e88>



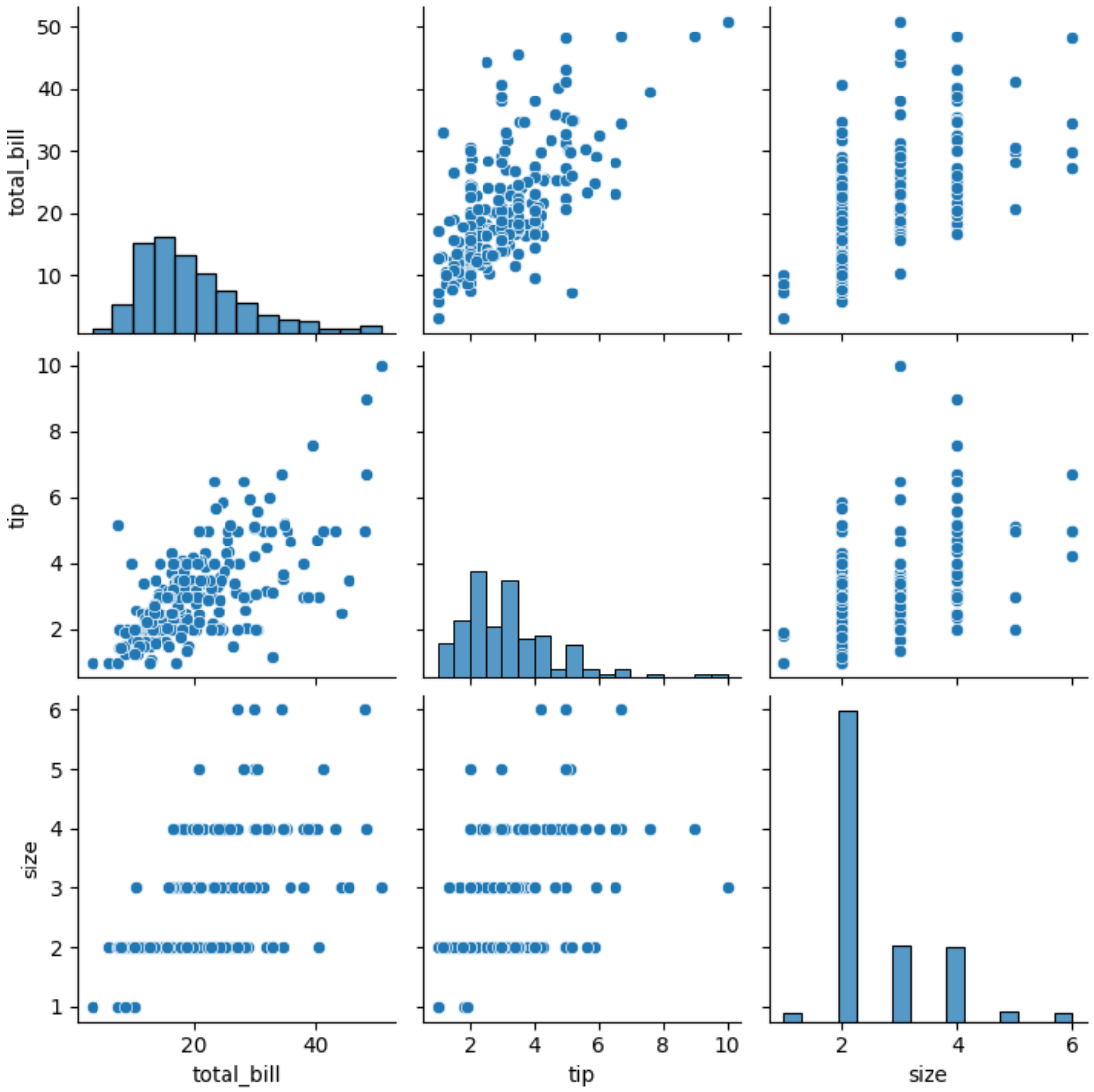
[6]:
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

[6] : <seaborn.axisgrid.JointGrid at 0x132f26f5d08>



[7]:
```
sns.pairplot(tips)
```

[7] : <seaborn.axisgrid.PairGrid at 0x132f26f7708>

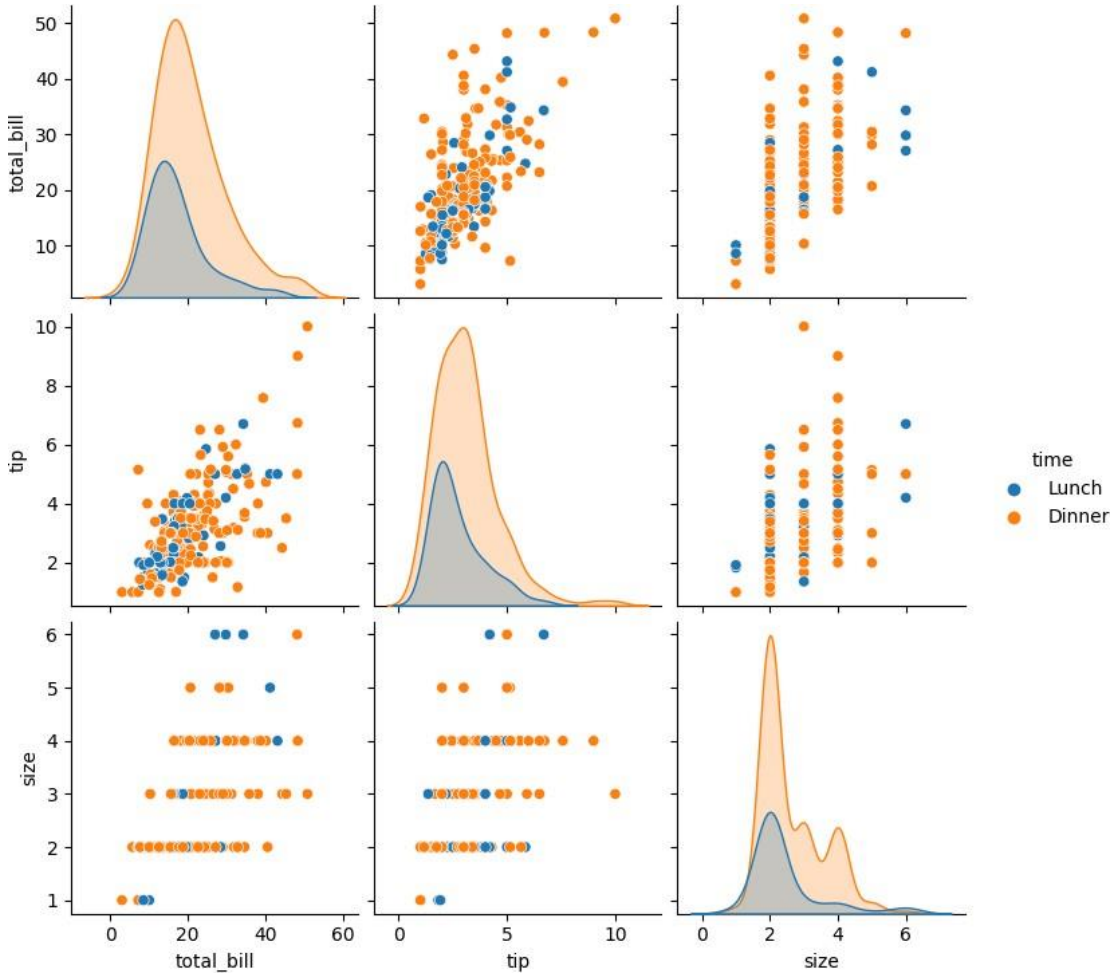


[8]:
```
tips.time.value_counts()
```

[8] :
```
Dinner    176
Lunch      68
Name: time, dtype: int64
```

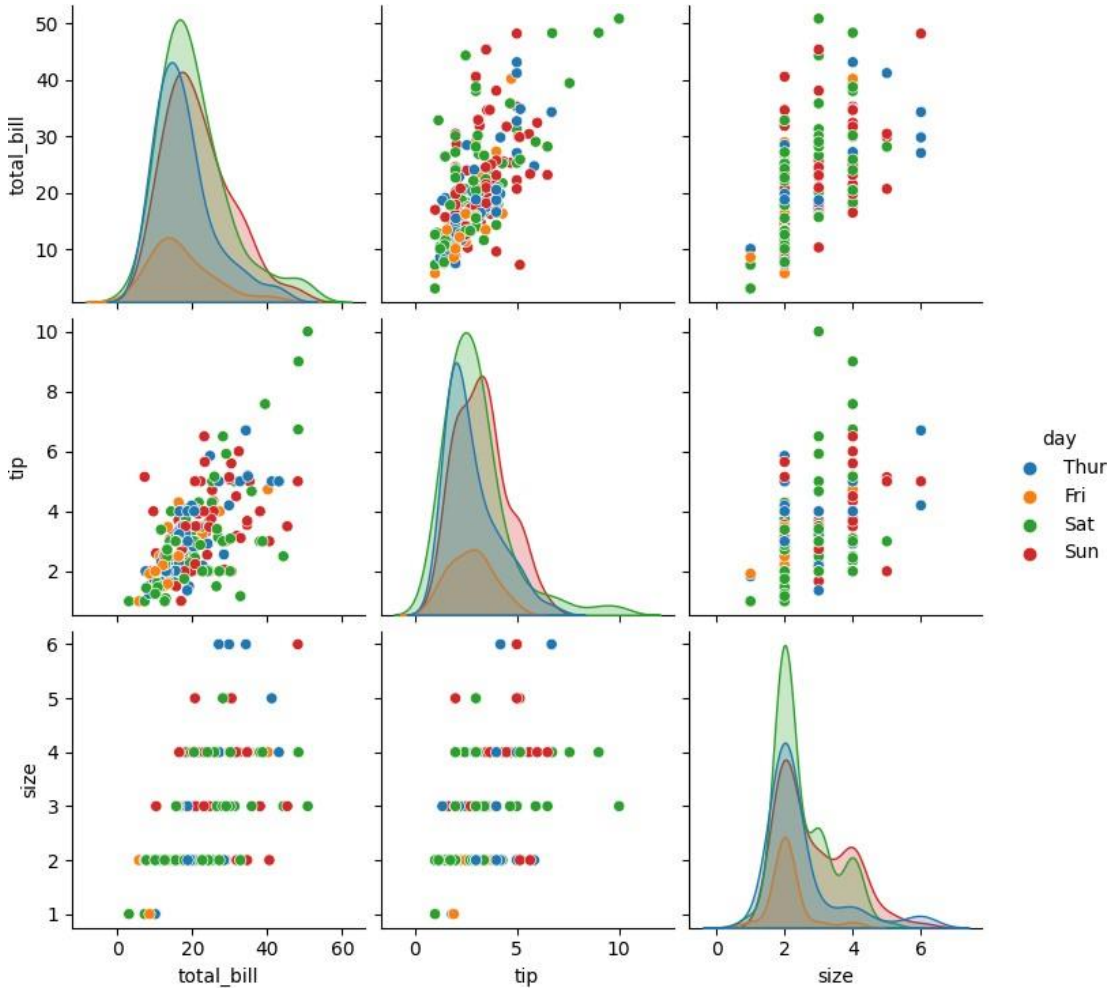
[9]:
```
sns.pairplot(tips,hue='time')
```

[9] : <seaborn.axisgrid.PairGrid at 0x132f3420d88>



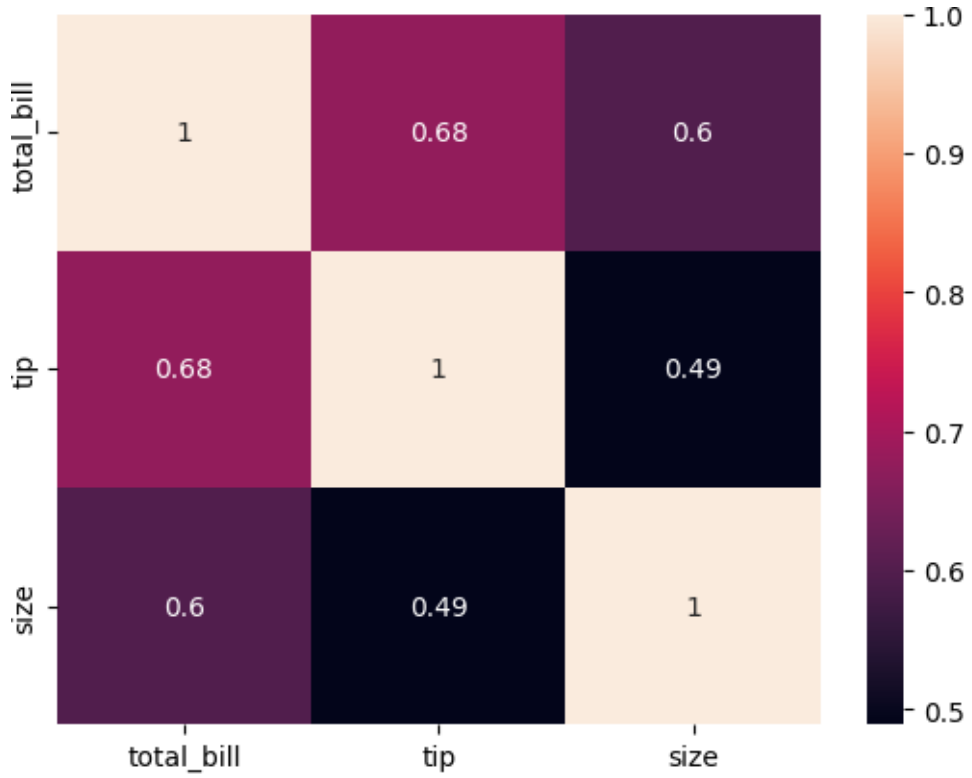
[10]:
```
sns.pairplot(tips,hue="day")
```

[10] : <seaborn.axisgrid.PairGrid at 0x132f4c14088>



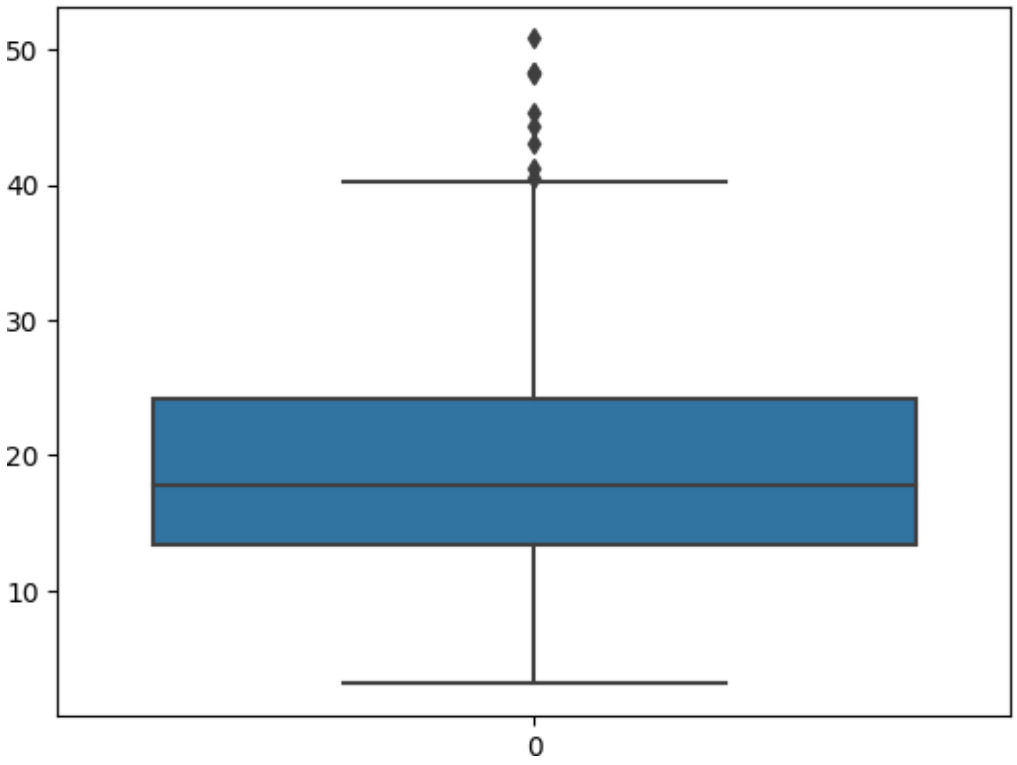
[12]:
```
sns.heatmap(tips.select_dtypes(include=['number']).corr(), annot=True)
```

[12]: <AxesSubplot:>



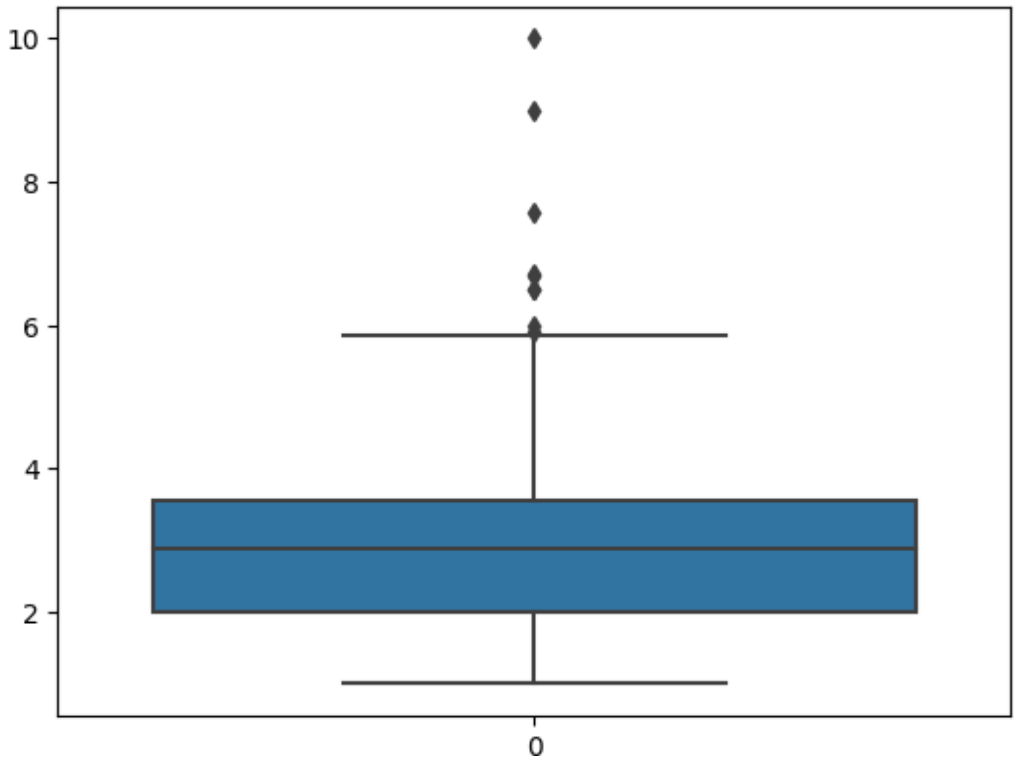
[13]:
```
sns.boxplot(tips.total_bill)
```

[13]: <AxesSubplot:>



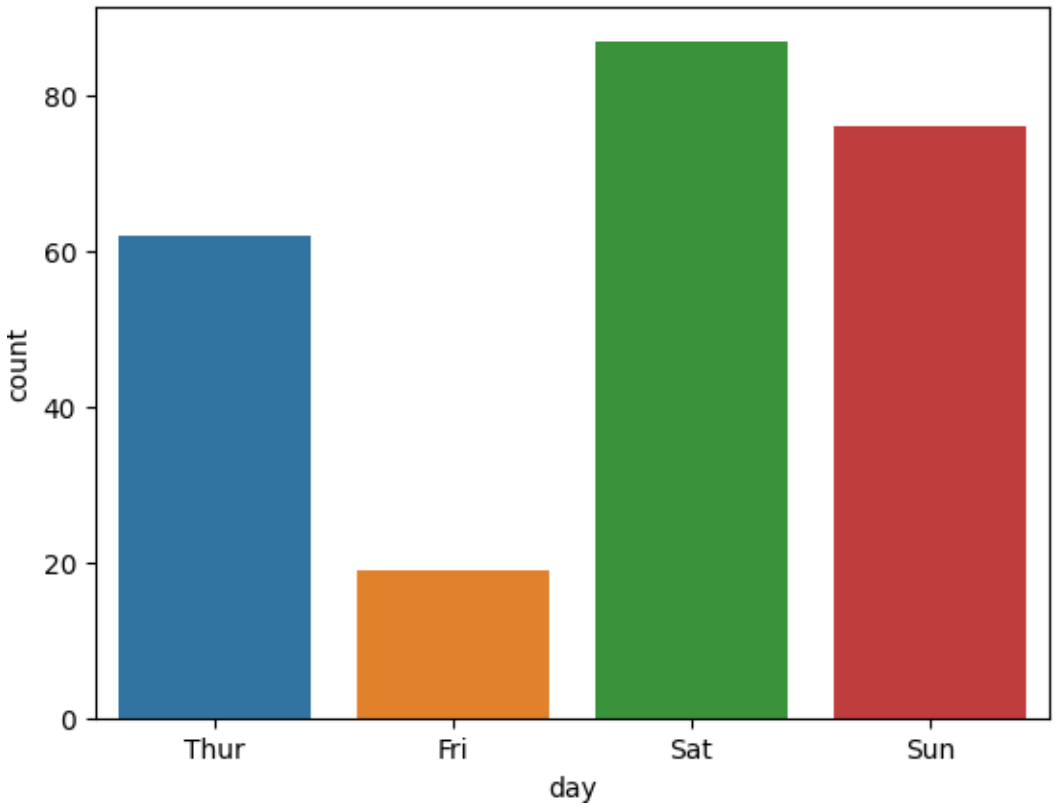
[14]:
```
sns.boxplot(tips.tip)
```

[14] : <AxesSubplot:>



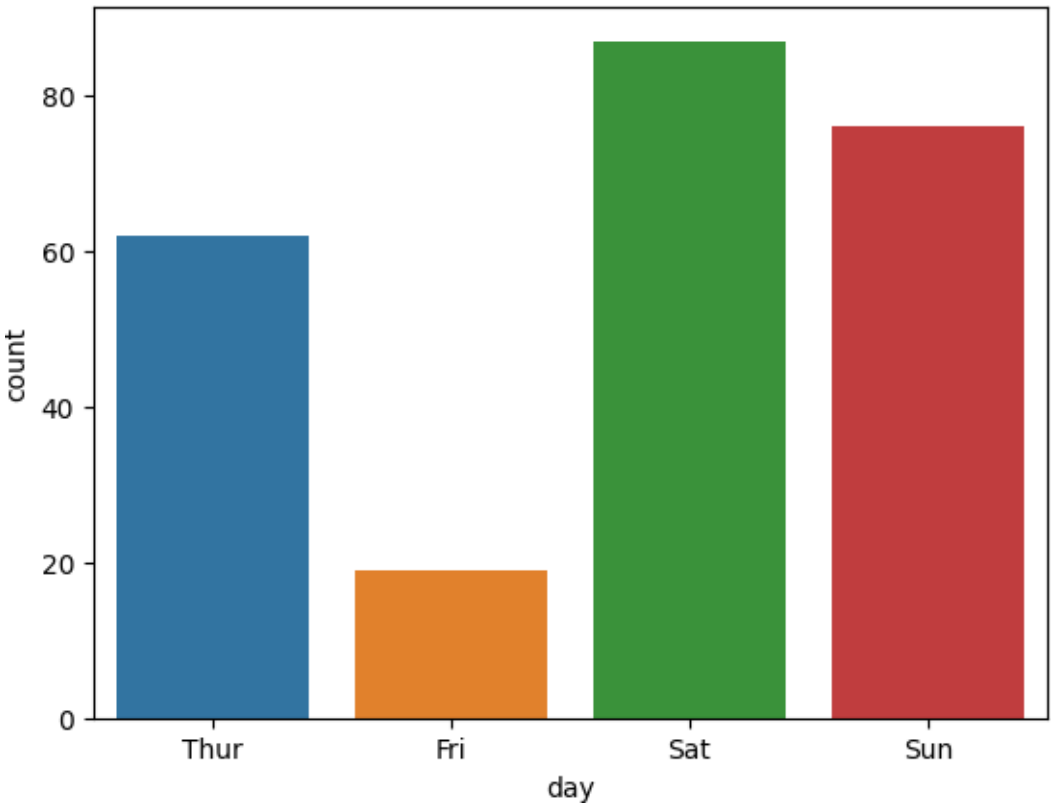
[16]:
```
sns.countplot(x="day", data=tips)
```

[16]: <AxesSubplot:xlabel='day', ylabel='count'>



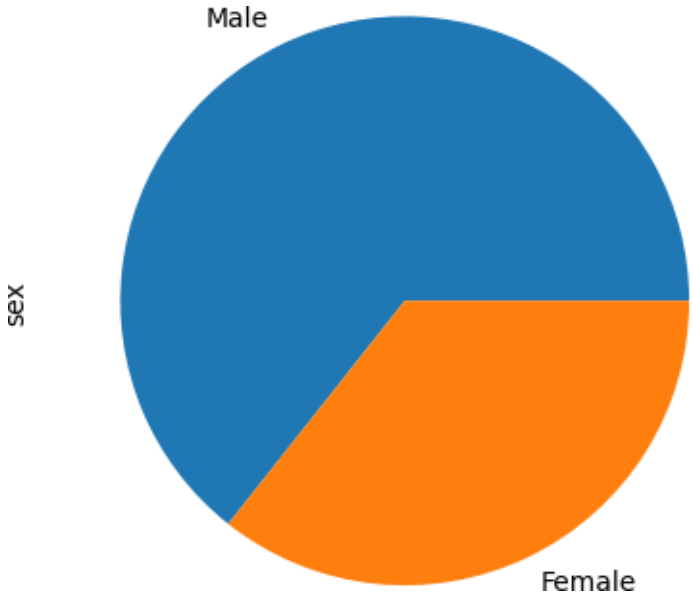
[18]:
```
sns.countplot(x="day", data=tips)
```

[18]: <AxesSubplot:xlabel='day', ylabel='count'>



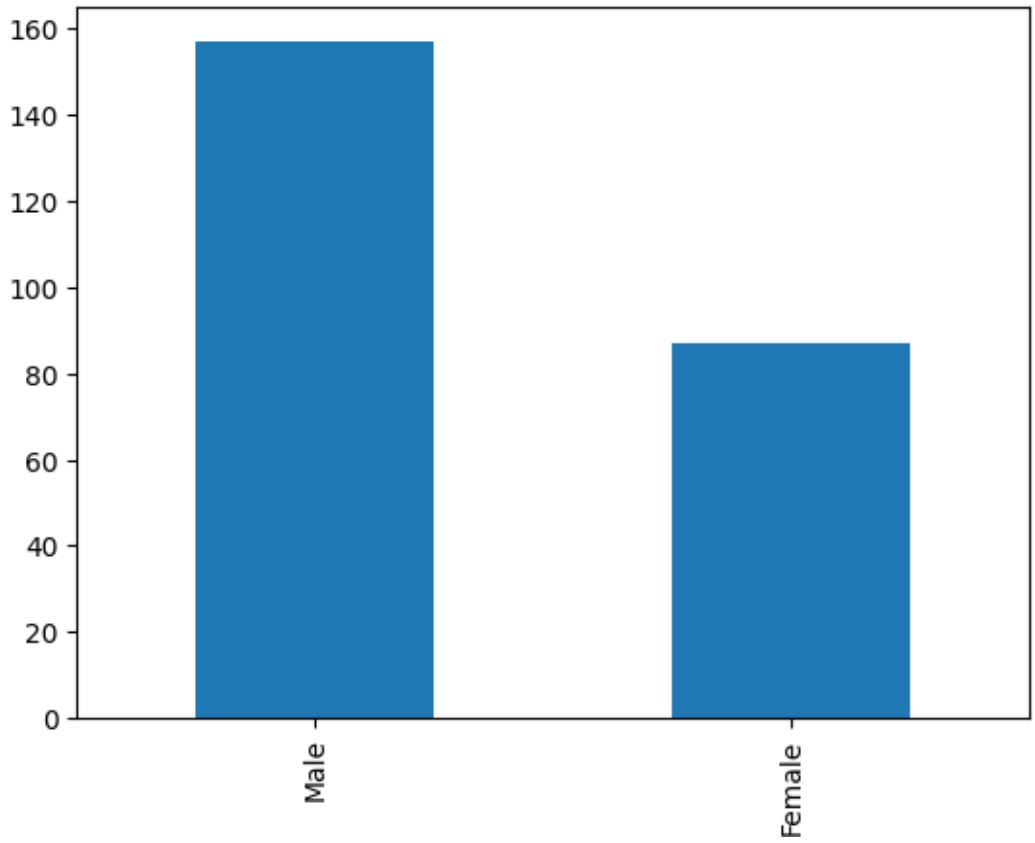
[19]:
```
tips.sex.value_counts().plot(kind="pie")
```

[19] : <AxesSubplot:ylabel='sex'>



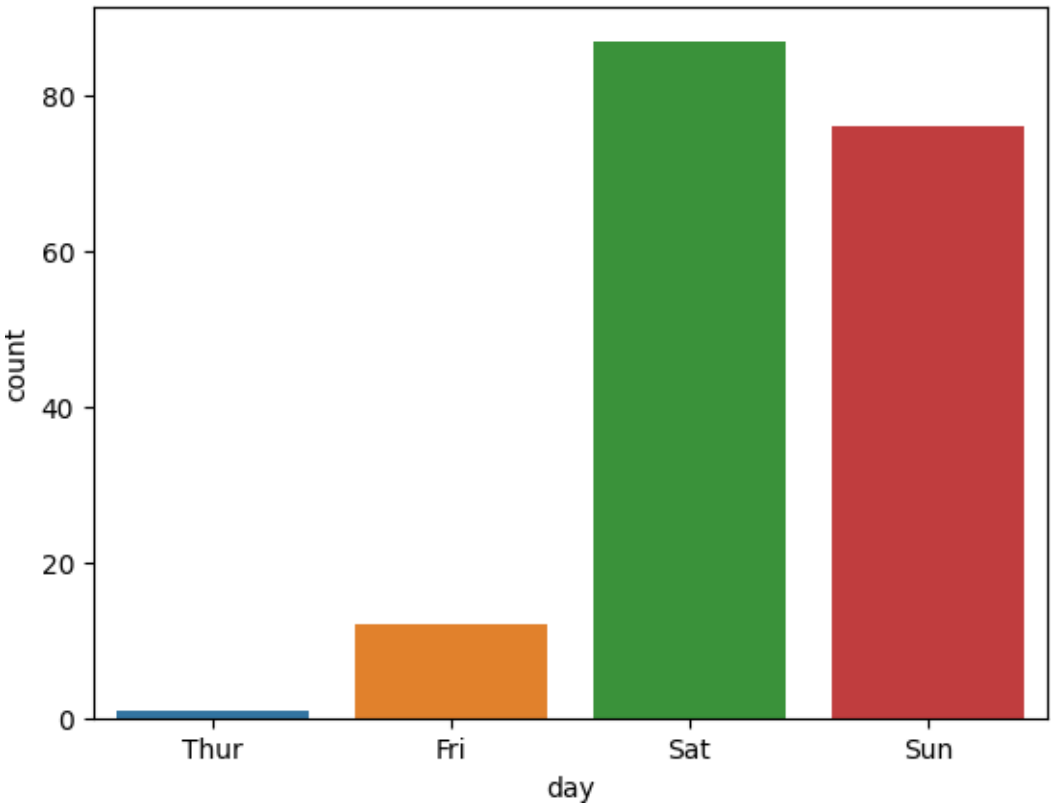
[20]:
```
tips.sex.value_counts().plot(kind="bar")
```

[20] : <AxesSubplot:>



[22]:
```
sns.countplot(x="day", data=tips[tips["time"] == "Dinner"])
```

[22]: <AxesSubplot:xlabel='day', ylabel='count'>


80
60
40
20
0
count
Thur
Fri
Sat
Sun
day

[ ]:

feature scaling

Generate | 10 random numbers using numpy | Close

```
import pandas as pd
from sklearn.preprocessing import StandardScaler


dataset = pd.read_csv('Social_Network_Ads.csv')
print(dataset.head())
```

```
    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
```

```
X = dataset[['Age', 'EstimatedSalary']]
y = dataset['Purchased']


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


X_scaled_df = pd.DataFrame(X_scaled, columns=['Age_scaled', 'EstimatedSalary_scaled'])
print(X_scaled_df.head())
```

```
   Age_scaled  EstimatedSalary_scaled
0   -1.781797               -1.490046
1   -0.253587               -1.460681
2   -1.113206               -0.785290
3   -1.017692               -0.374182
4   -1.781797                0.183751
```

Start coding or generate with AI.

pandas and numpy lib

```
import pandas as pd
import numpy as np

# Load the data
data = pd.read_csv('50_Startups.csv')


print("First few rows of the dataset:\n", data.head())
```

```
First few rows of the dataset:
    R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
3  144372.41       118671.85        383199.62    New York  182901.99
4  142107.34        91391.77        366168.42     Florida  166187.94
```

```
print("\nData Information:\n")
print(data.info())
```

```
Data Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
None
```

```
print("\nSummary Statistics:\n", data.describe())
```

```
Summary Statistics:
            R&D Spend  Administration  Marketing Spend         Profit
count      50.000000       50.000000        50.000000      50.000000
mean    73721.615600   121344.639600    211025.097800  112012.639200
std     45902.256482    28017.802755    122290.310726   40306.180338
min         0.000000    51283.140000         0.000000   14681.400000
25%     39936.370000   103730.875000    129300.132500   90138.902500
50%     73051.080000   122699.795000    212716.240000  107978.190000
75%    101602.800000   144842.180000    299469.085000  139765.977500
max    165349.200000   182645.560000    471784.100000  192261.830000
```

```
rd_mean = np.mean(data['R&D Spend'])
rd_median = np.median(data['R&D Spend'])
rd_mode = data['R&D Spend'].mode()[0]


print("\nR&D Spend - Mean:", rd_mean)
print("R&D Spend - Median:", rd_median)
print("R&D Spend - Mode:", rd_mode)
```

```
R&D Spend - Mean: 73721.6156
R&D Spend - Median: 73051.08
R&D Spend - Mode: 0.0
```

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

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

```
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: 608.0 bytes
```

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

Generate randomly select 5 items from a list Close

```
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: 608.0 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

Generate a slider using jupyter widgets Close

```
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_s
```

Generate print hello world using rot13 Close

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

```
LinearRegression()
```

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

```
0.9645401573418146
```

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

```
0.9024461774180497
```

```
model.coef_
```

```
array([[9423.81532303]])
```

Generate a slider using jupyter widgets Close

```
model.intercept_
```

```
array([25321.58301178])
```

Generate a slider using jupyter widgets Close

```
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: "))
yr_of_exp_NP=np.array([[yr_of_exp]])
Salary=model.predict(yr_of_exp_NP)
```

```
Enter Years of Experience: 25
```

```
print("Estimated Salary for {} years of experience is: {}".format(yr_of_exp, Salary))
```

```
Estimated Salary for 25.0 years of experience is: [[260916.96608755]]
```

Start coding or generate with AI.

Generate | 10 random numbers using numpy | Close

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

Generate | a slider using jupyter widgets | Close

```
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

Generate | print hello world using rot13 | Close

```
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],
            [    28,  79000]])
```

Generate print hello world using rot13 Close

```
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
```

```
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=0.3, 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)

    if test_score > train_score:
        print("Test: {:.2f}, Train: {:.2f}, Random State: {}".format(test_score, train_score, i))
```

```
Test: 0.86, Train: 0.85, Random State: 319
Test: 0.87, Train: 0.84, Random State: 321
Test: 0.91, Train: 0.83, Random State: 322
Test: 0.86, Train: 0.84, Random State: 331
Test: 0.85, Train: 0.84, Random State: 333
Test: 0.90, Train: 0.84, Random State: 336
Test: 0.87, Train: 0.84, Random State: 337
Test: 0.88, Train: 0.83, Random State: 343
Test: 0.90, Train: 0.84, Random State: 351
Test: 0.87, Train: 0.83, Random State: 352
Test: 0.90, Train: 0.82, Random State: 354
Test: 0.87, Train: 0.84, Random State: 357
Test: 0.87, Train: 0.84, Random State: 358
Test: 0.85, Train: 0.85, Random State: 361
Test: 0.88, Train: 0.83, Random State: 362
Test: 0.93, Train: 0.82, Random State: 363
Test: 0.88, Train: 0.83, Random State: 366
Test: 0.89, Train: 0.85, Random State: 369
Test: 0.89, Train: 0.84, Random State: 371
Test: 0.90, Train: 0.83, Random State: 376
Test: 0.91, Train: 0.81, Random State: 377
Test: 0.88, Train: 0.84, Random State: 378
Test: 0.88, Train: 0.84, Random State: 379
Test: 0.86, Train: 0.84, Random State: 381
Test: 0.86, Train: 0.83, Random State: 382
Test: 0.88, Train: 0.85, Random State: 386
Test: 0.84, Train: 0.84, Random State: 388
Test: 0.86, Train: 0.85, Random State: 390
Test: 0.88, Train: 0.82, Random State: 394
Test: 0.88, Train: 0.85, Random State: 399
Test: 0.88, Train: 0.84, Random State: 400
```

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

# Splitting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)

# Initializing and fitting the logistic regression model
finalModel = LogisticRegression()
finalModel.fit(x_train, y_train)
```

```
LogisticRegression()
```

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

```
0.8375
0.8875
```

Generate | 10 random numbers using numpy | Close

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

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       257
           1       0.85      0.70      0.77       143

    accuracy                           0.85       400
   macro avg       0.85      0.81      0.83       400
weighted avg       0.85      0.85      0.84       400
```

Start coding or generate with AI.