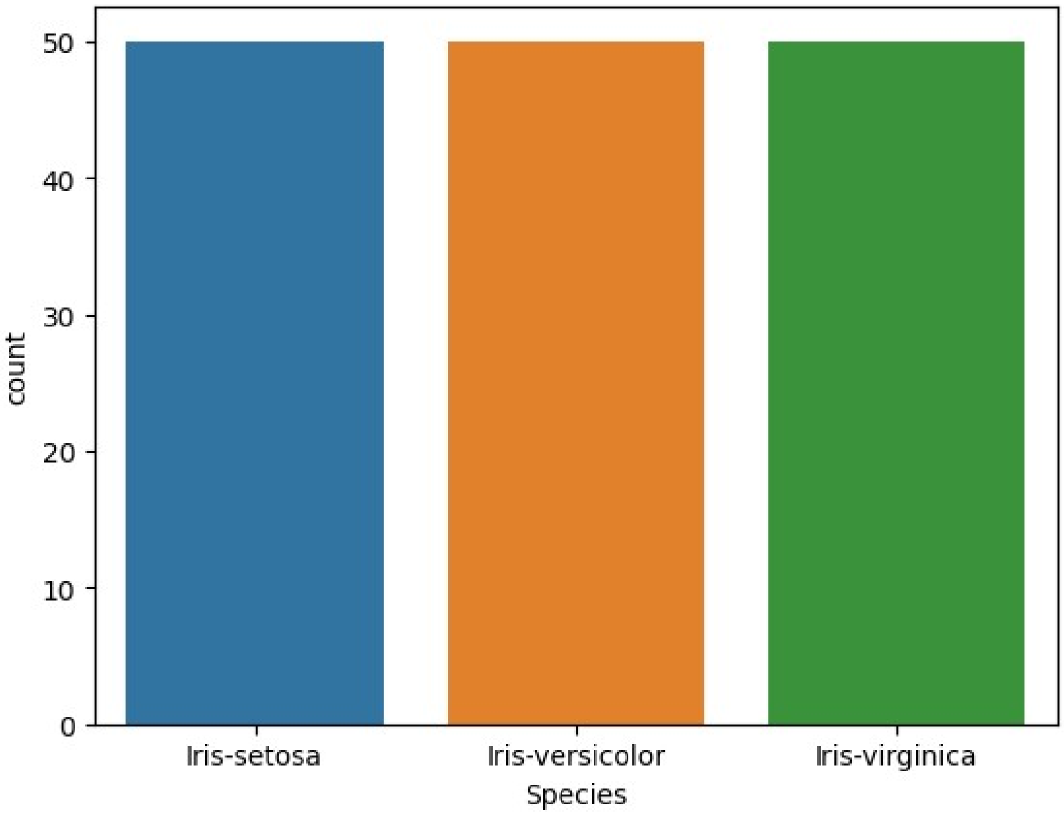
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
data=pd.read_csv('Iris.csv')
data
```

```
      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]
```

```
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

data.value_counts('Species')

Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64

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()
```



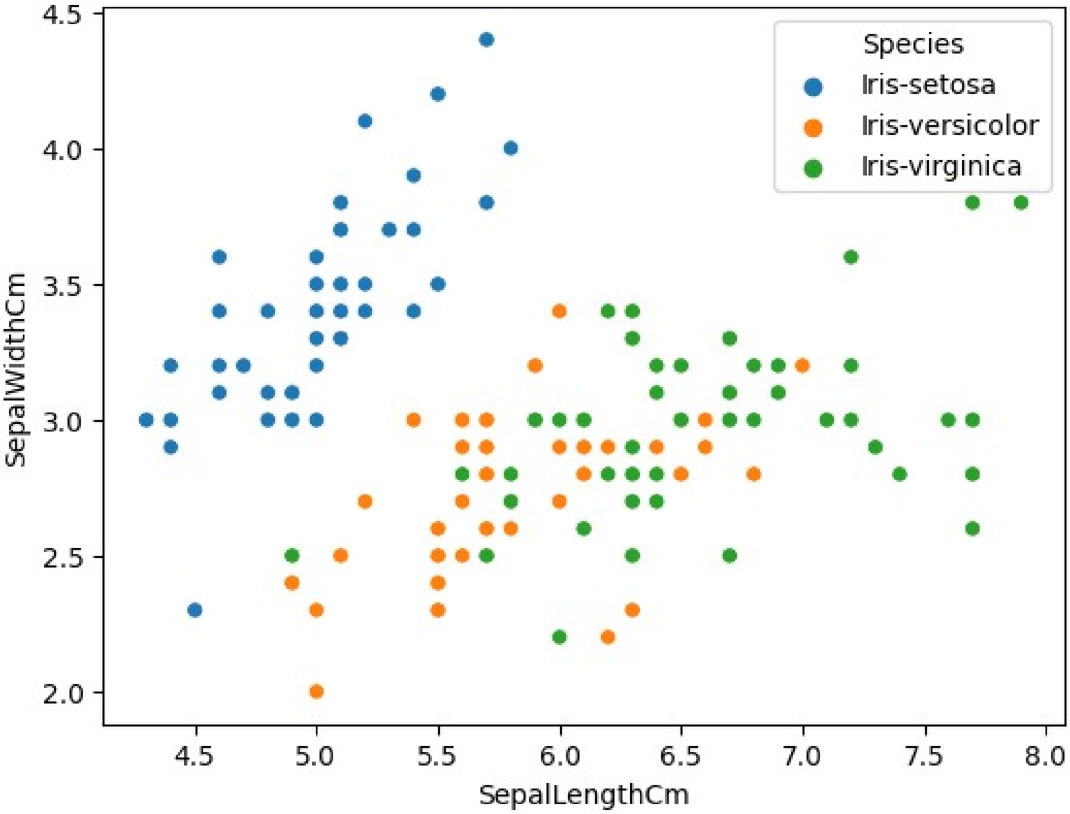
```
   Iris-setosa  Iris-versicolor  Iris-virginica  Id  SepalLengthCm  \
0            1                0               0   1            5.1
1            1                0               0   2            4.9
2            1                0               0   3            4.7
3            1                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
```

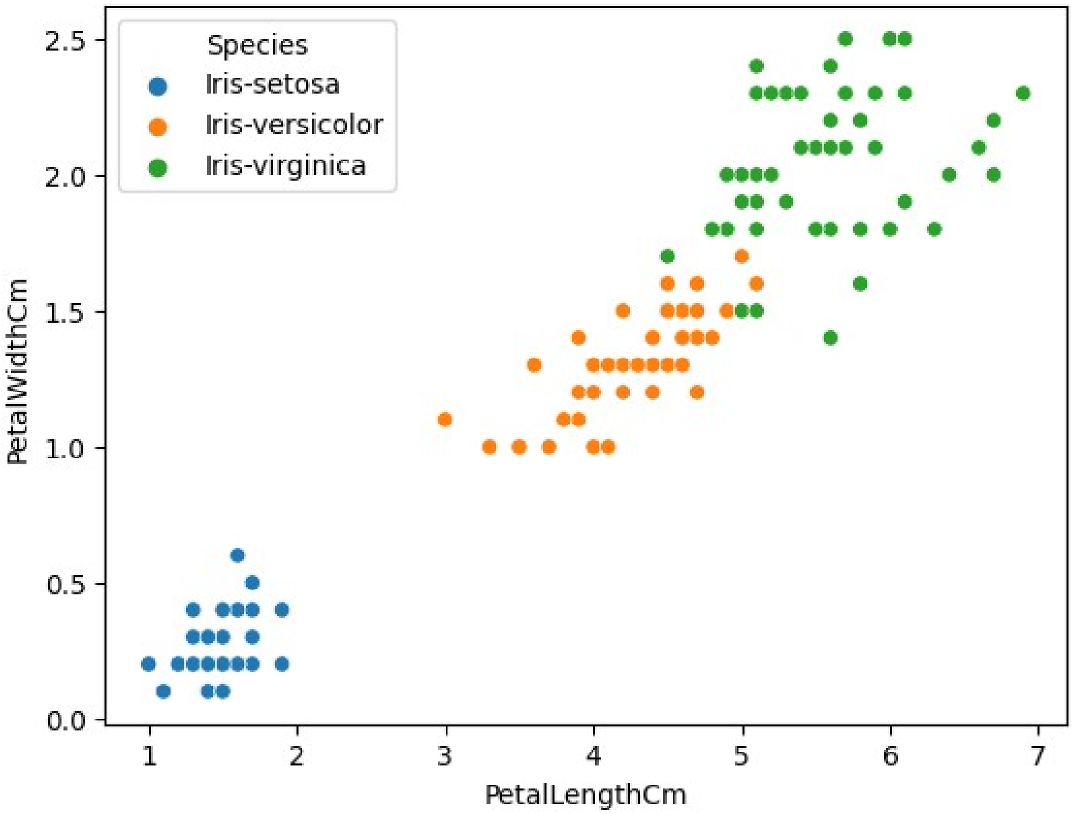
```
sns.scatterplot(x='SepalLengthCm',y='SepalWidthCm',hue='Species',data=
data)
```

```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='SepalWidthCm'>
```

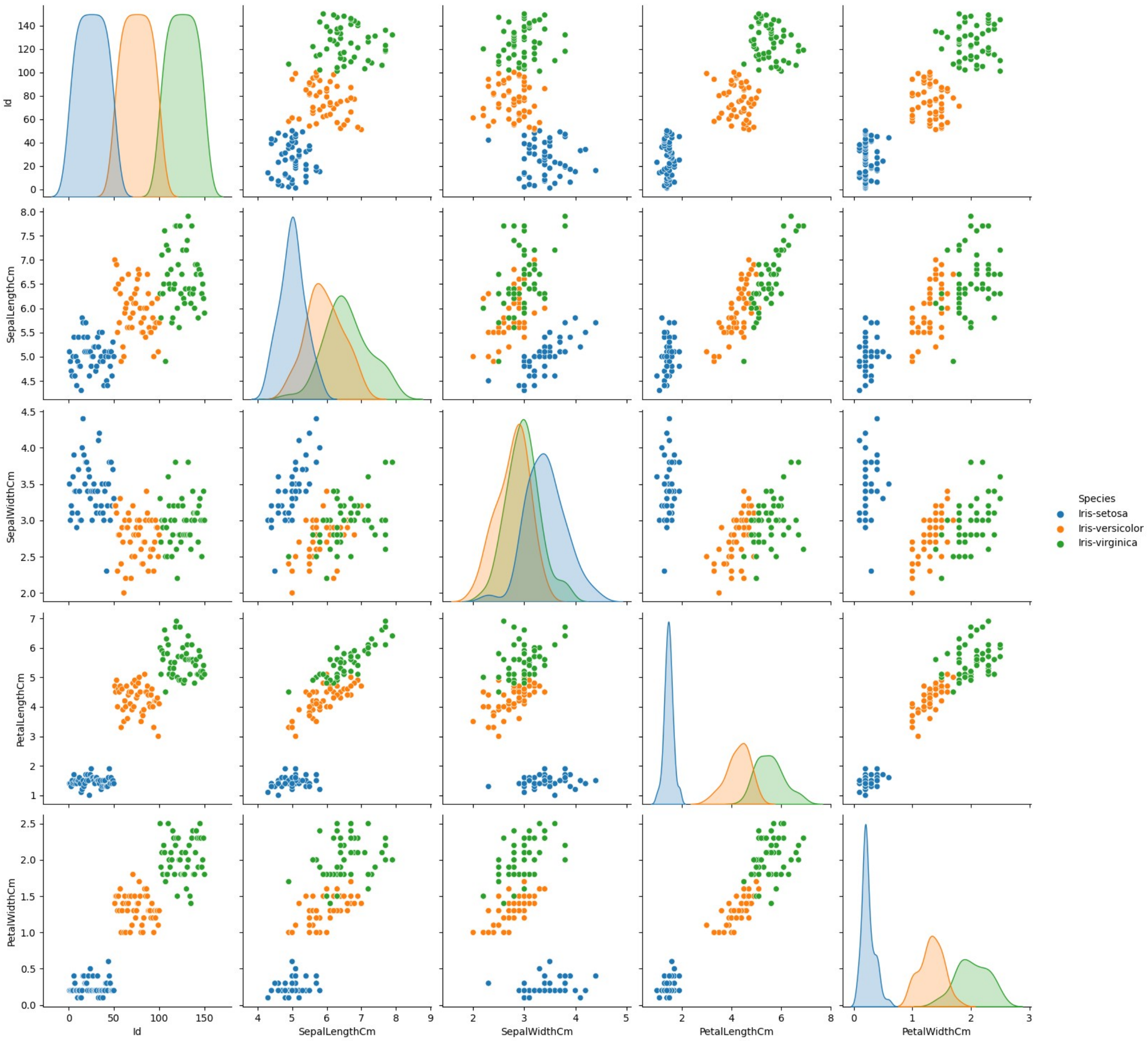


```
sns.scatterplot(x='PetalLengthCm',y='PetalWidthCm',hue='Species',data=
data)
```

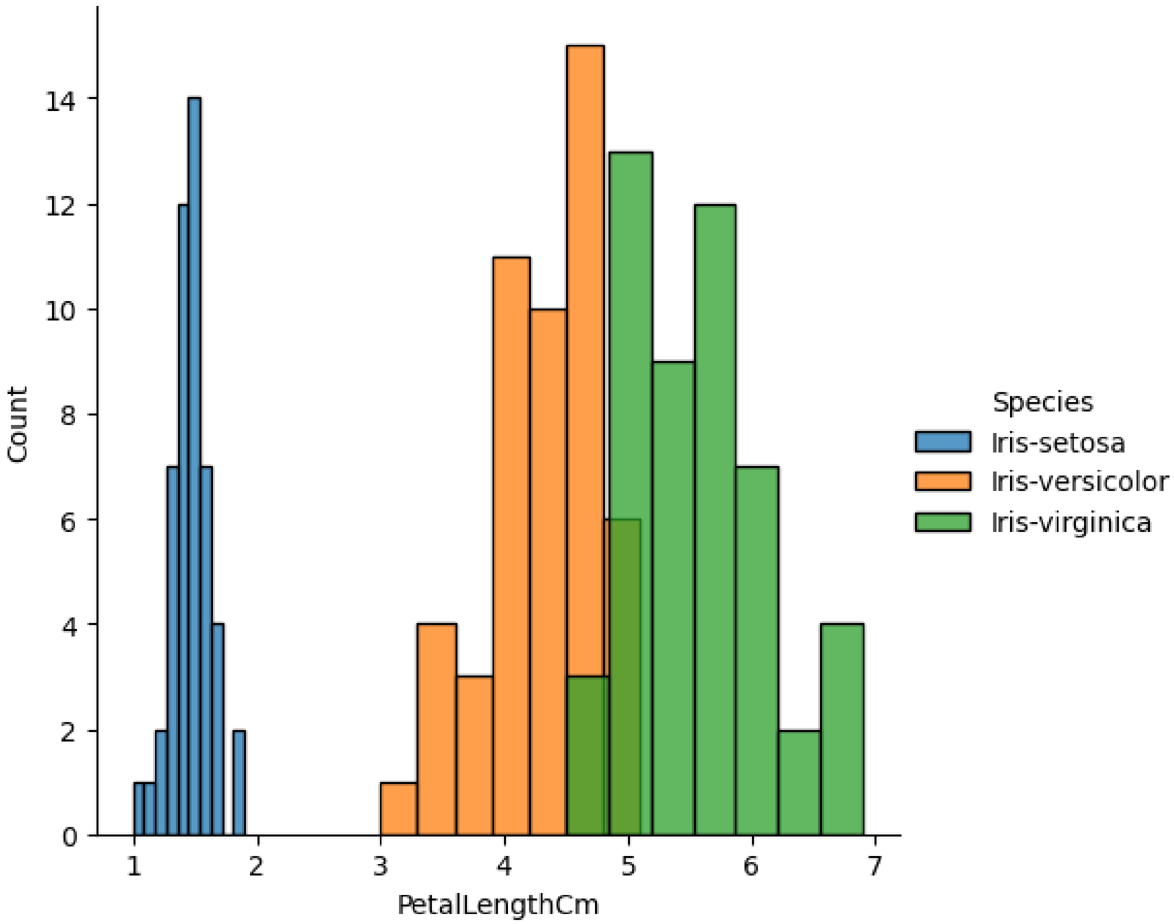
```
<AxesSubplot:xlabel='PetalLengthCm', ylabel='PetalWidthCm'>
```



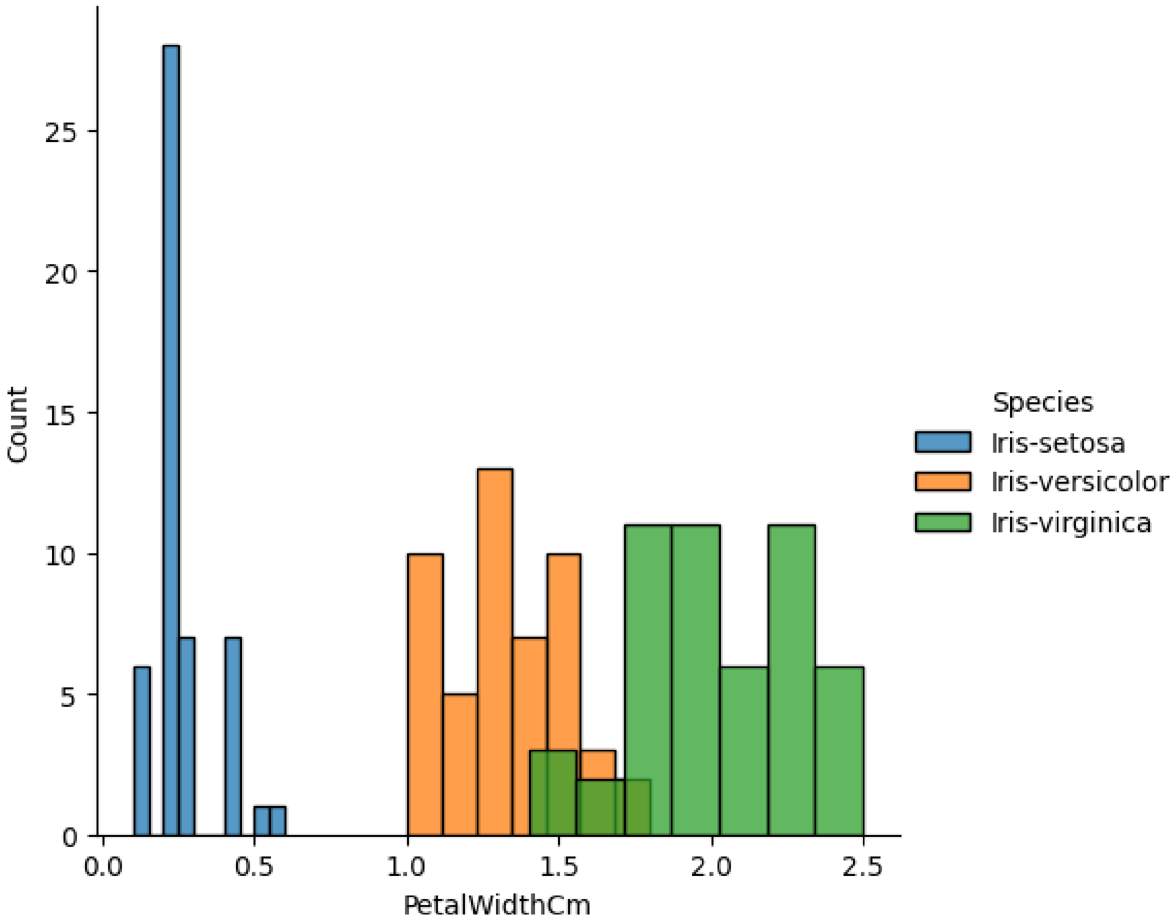
```
sns.pairplot(data,hue='Species',height=3);
```



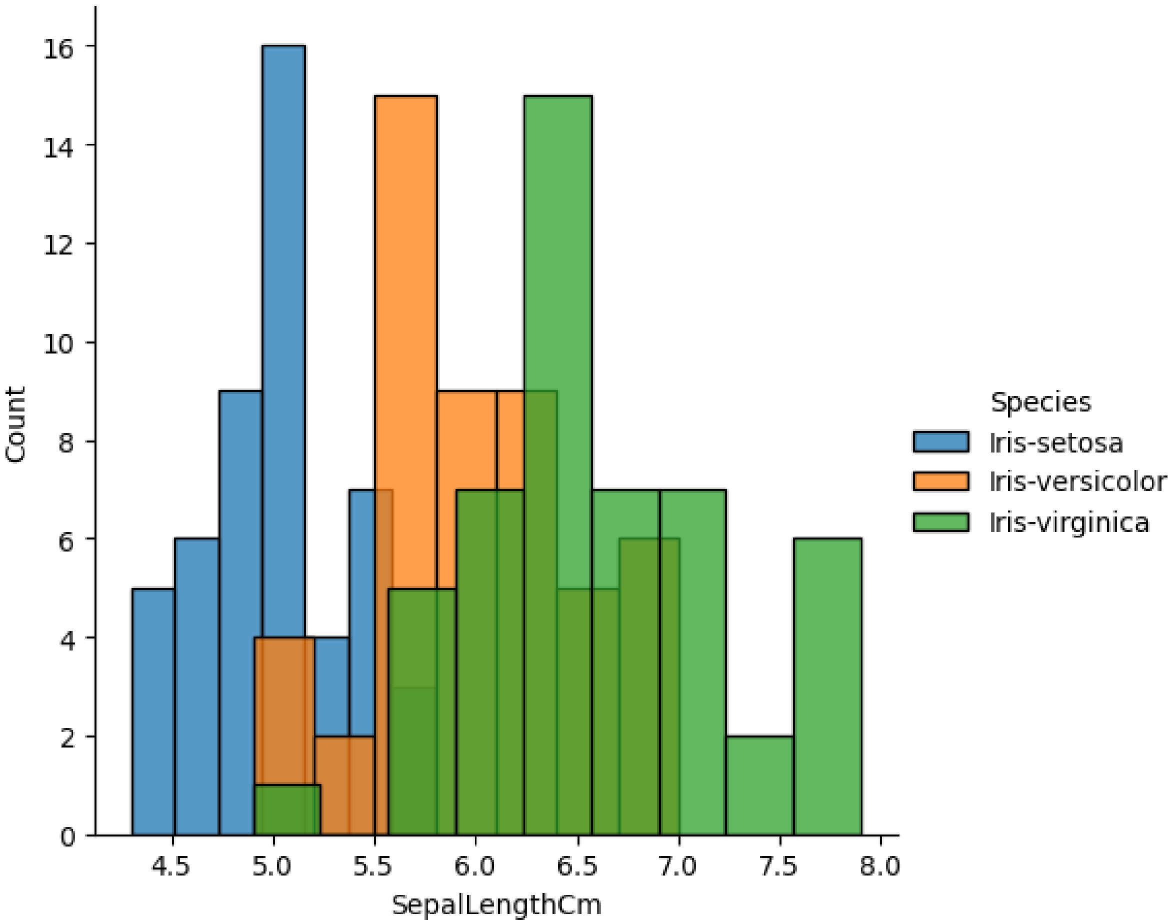
```
sns.FacetGrid(data,hue='Species',height=5).map(sns.histplot,'PetalLengthCm').add_legend(); plt.show();
```



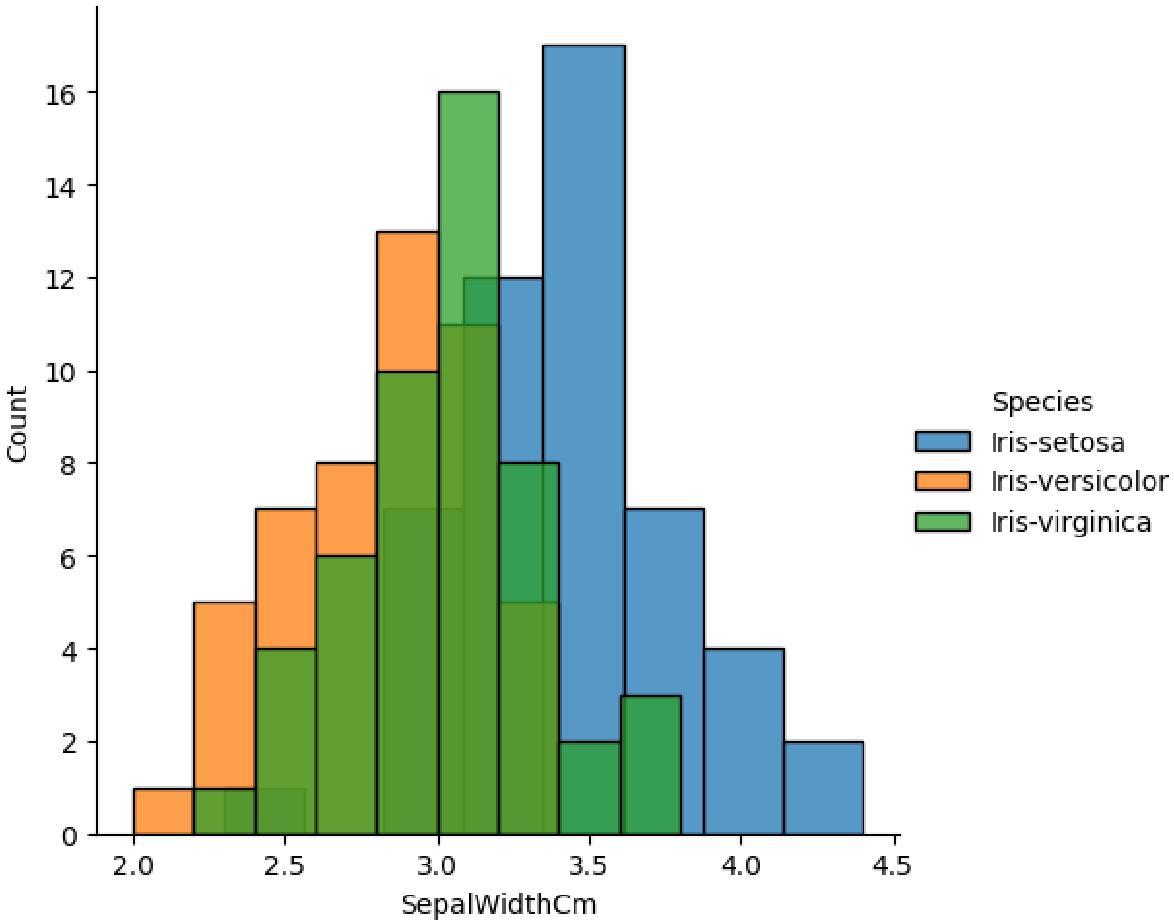
```
sns.FacetGrid(data,hue='Species',height=5).map(sns.histplot,'PetalWidt
hCm').add_legend(); plt.show();
```



```
sns.FacetGrid(data,hue='Species',height=5).map(sns.histplot,'SepalLeng
thCm').add_legend(); plt.show();
```



```
sns.FacetGrid(data,hue='Species',height=5).map(sns.histplot,'SepalWidthCm').add_legend();
plt.show();
```


Count
0
2
4
6
8
10
12
14
16
2.0
2.5
3.0
3.5
4.0
4.5
SepalWidthCm
Species
Iris-setosa
Iris-versicolor
Iris-virginica

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

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

```
np.sqrt(array)
```

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

```
array.ndim
```

```
1
```

```
new_array=array.reshape(3,3)
new_array
```

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

```
new_array.ndim
```

```
2
```

```
new_array.ravel()
```

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

```
newm=new_array.reshape(3,3)
newm
```

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

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

```
array([48, 81])
```

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

```
array([[64, 44]])
```

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

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

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

```
array([[69],
       [46]])
```

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

```
array([[69],
       [46],
       [96]])
```

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

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

```
import numpy as np
import pandas as pd
list=[[1,'Smith',50000],[2,'Jones',60000]]

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

```
   0      1      2
0  1  Smith  50000
1  2  Jones  60000
```

```
df.columns=['Empd','Name','Salary']
df
```

```
   Empd   Name  Salary
0     1  Smith   50000
1     2  Jones   60000
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Empd    2 non-null      int64 
 1   Name    2 non-null      object
 2   Salary  2 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 176.0+ bytes
```

```
df=pd.read_csv("50_Startups.csv")
df.info()
```

```
<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
```

```
df.head()
```

```
   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
```

```
df.tail()
```

```
    R&D Spend  Administration  Marketing Spend       State    Profit
45    1000.23       124153.04          1903.93    New York  64926.08
46    1315.46       115816.21        297114.46     Florida  49490.75
47       0.00       135426.92             0.00  California  42559.73
48     542.05        51743.15             0.00    New York  35673.41
49       0.00       116983.80         45173.06  California  14681.40
```

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

```
----------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
~\AppData\Local\Temp\ipykernel_42792\3506309008.py in <module>
      1 import numpy as np
      2 import pandas as pd
----> 3 df=pd.read_csv("employee.csv")

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\util\_decorators.py in wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312 
    313         return wrapper

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\readers.py in read_csv(filepath_or_buffer,
sep, delimiter, header, names, index_col, usecols, squeeze, prefix,
mangle_dupe_cols, dtype, engine, converters, true_values,
false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
parse_dates, infer_datetime_format, keep_date_col, date_parser,
dayfirst, cache_dates, iterator, chunksize, compression, thousands,
decimal, lineterminator, quotechar, quoting, doublequote, escapechar,
comment, encoding, encoding_errors, dialect, error_bad_lines,
warn_bad_lines, on_bad_lines, delim_whitespace, low_memory,
memory_map, float_precision, storage_options)
    584     kwds.update(kwds_defaults)
    585 
--> 586     return _read(filepath_or_buffer, kwds)
```

```
    587
    588

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\readers.py in _read(filepath_or_buffer,
kwds)
    480
    481     # Create the parser.
--> 482     parser = TextFileReader(filepath_or_buffer, **kwds)
    483
    484     if chunksize or iterator:

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\readers.py in __init__(self, f, engine,
**kwds)
    809             self.options["has_index_names"] =
kwds["has_index_names"]
    810
--> 811         self._engine = self._make_engine(self.engine)
    812
    813     def close(self):

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\readers.py in _make_engine(self, engine)
   1038             )
   1039         # error: Too many arguments for "ParserBase"
-> 1040         return mapping[engine](self.f, **self.options)  #
type: ignore[call-arg]
   1041
   1042     def _failover_to_python(self):

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\c_parser_wrapper.py in __init__(self, src,
**kwds)
     49
     50         # open handles
---> 51         self._open_handles(src, kwds)
     52         assert self.handles is not None
     53

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\pandas\io\parsers\base_parser.py in _open_handles(self, src,
kwds)
    227             memory_map=kwds.get("memory_map", False),
    228             storage_options=kwds.get("storage_options", None),
--> 229             errors=kwds.get("encoding_errors", "strict"),
    230         )
    231

c:\users\asus\appdata\local\programs\python\python37\lib\site-
```

```
packages\pandas\io\common.py in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
    705                 encoding=ioargs.encoding,
    706                 errors=errors,
--> 707                 newline="",
    708             )
    709         else:

FileNotFoundError: [Errno 2] No such file or directory: 'employee.csv'
```

```
#sample calculation for low range(lr) , upper range (ur),percentile
import numpy as np
array=np.random.randint(1,100,16) # randomly generate 16 numbers
between 1 to 100
array
```

```
array([ 6, 66,  9, 37, 29, 40, 12, 23, 20, 77, 26, 95, 56, 14,  9,
67])
```

```
array.mean()
```

```
36.625
```

```
np.percentile(array,25)
```

```
13.5
```

```
np.percentile(array,50)
```

```
27.5
```

```
np.percentile(array,75)
```

```
58.5
```

```
np.percentile(array,100)
```

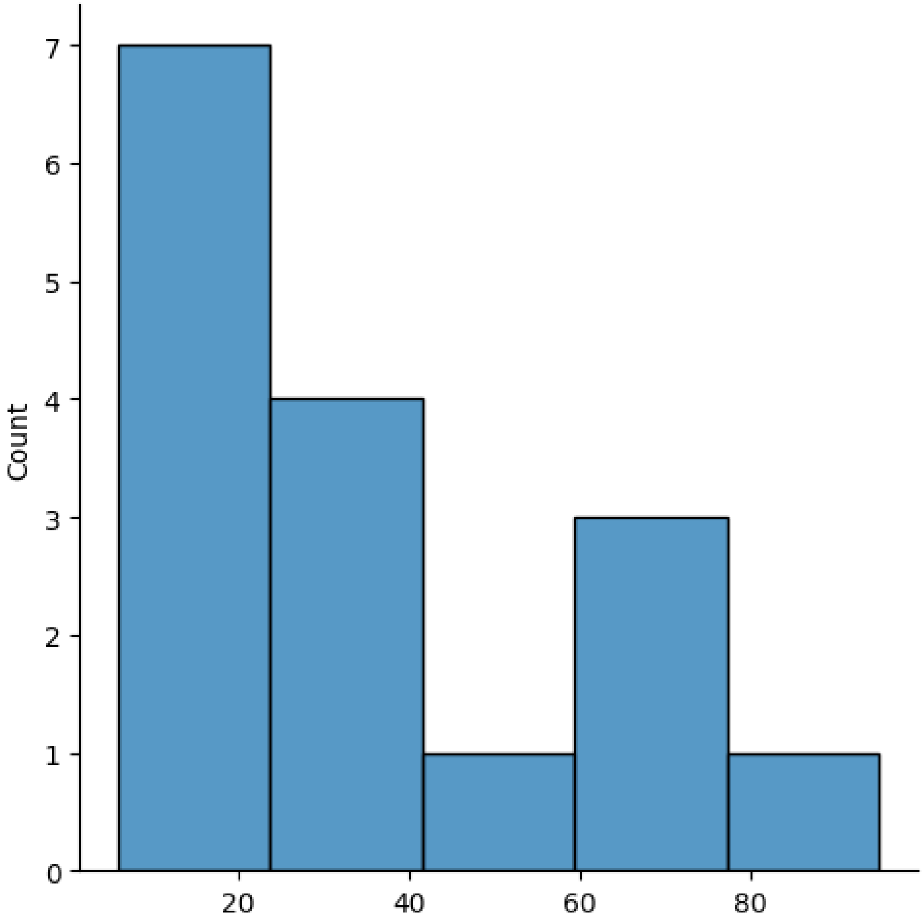
```
95.0
```

```
#outliers detection
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
```

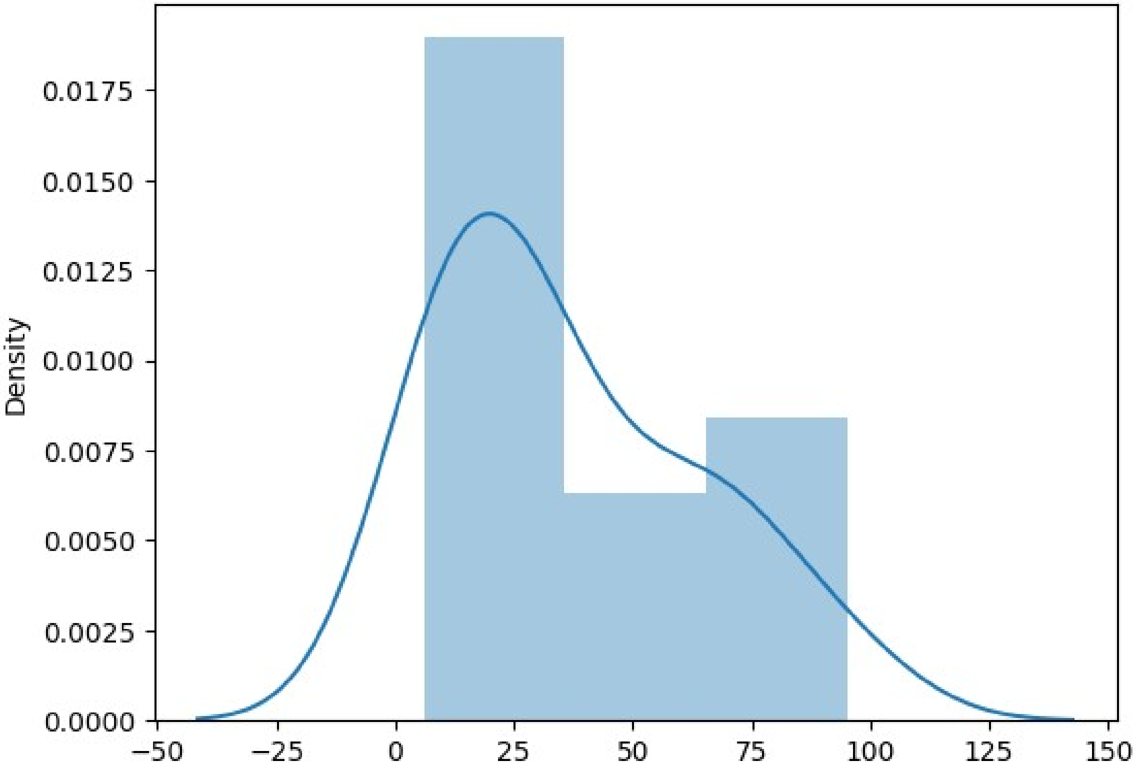
```
(-54.0, 126.0)
```

```
import seaborn as sns
%matplotlib inline
sns.displot(array)
```

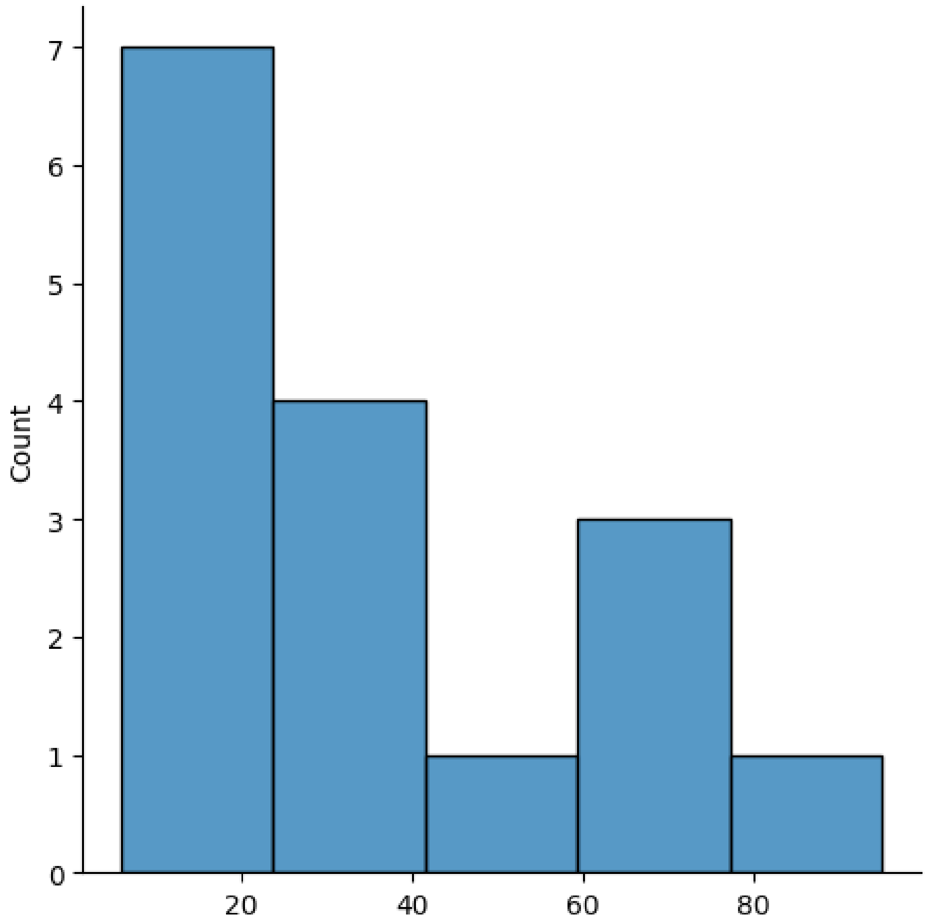
```
<seaborn.axisgrid.FacetGrid at 0x1dbcd642bc8>
```



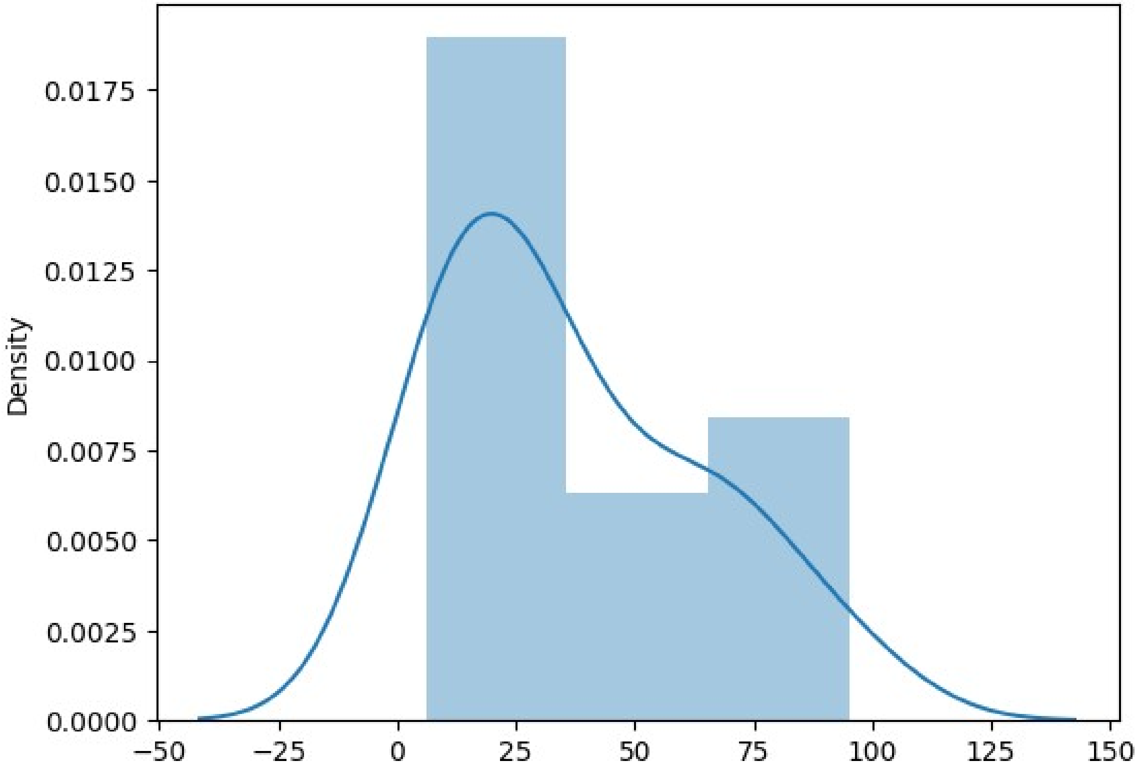
```
sns.distplot(array)

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\ipykernel_launcher.py:1: UserWarning:

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

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

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

  """Entry point for launching an IPython kernel.

<AxesSubplot:ylabel='Density'>
```



```
new_array=array[(array>lr) & (array<ur)]
new_array

array([ 6, 66,  9, 37, 29, 40, 12, 23, 20, 77, 26, 95, 56, 14,  9,
67])

sns.displot(new_array)

<seaborn.axisgrid.FacetGrid at 0x1dbcfbd1f08>
```



```
lr1,ur1=outDetection(new_array)
lr1,ur1

(-54.0, 126.0)

final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array

array([ 6, 66,  9, 37, 29, 40, 12, 23, 20, 77, 26, 95, 56, 14,  9,
67])

sns.distplot(final_array)

c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\ipykernel_launcher.py:1: UserWarning:

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

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

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

  """Entry point for launching an IPython kernel.

<AxesSubplot:ylabel='Density'>
```



```
```

```
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
\
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
```

```
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: 920.0+ bytes
```

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

```
    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
```

```
len(df)
```

```
10
```

```
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  \
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
```

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

```
   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
```

```
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)
```

```
   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
```

```
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)
```

```
   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
```

```
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',inplace=True)
df.FoodPreference.unique
```

```
<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>
```

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

df.EstimatedSalary.fillna(round(df.EstimatedSalary.mean()),inplace=Tru
e)
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
```

```
   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
```

```
import numpy as np
import pandas as pd
df=pd.read_csv('pre-process_datasample.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.head()
```

```
   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
```

```
df.Country.fillna(df.Country.mode()[0],inplace=True)
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()
```

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

```
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=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]])
```

```
import numpy as np
import pandas as pd
df=pd.read_csv("pre-process_datasample.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()
```

```
<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: 448.0+ bytes
```

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

```
0    France
dtype: object
```

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

```
'France'
```

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

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

```
df.Country.fillna(df.Country.mode()[0],inplace=True)
df.Age.fillna(df.Age.median(),inplace=True)
df.Salary.fillna(round(df.Salary.mean()),inplace=True)
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  63778.0       Yes
5   France  35.0  58000.0       Yes
6    Spain  38.0  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

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

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

```
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       1        0      0  44.0  72000.0        No
1       0        0      1  27.0  48000.0       Yes
2       0        1      0  30.0  54000.0        No
3       0        0      1  38.0  61000.0        No
4       0        1      0  40.0  63778.0       Yes
5       1        0      0  35.0  58000.0       Yes
6       0        0      1  38.0  52000.0        No
7       1        0      0  48.0  79000.0       Yes
8       0        1      0  50.0  83000.0        No
9       1        0      0  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: 448.0+ bytes
```

```
updated_dataset
```

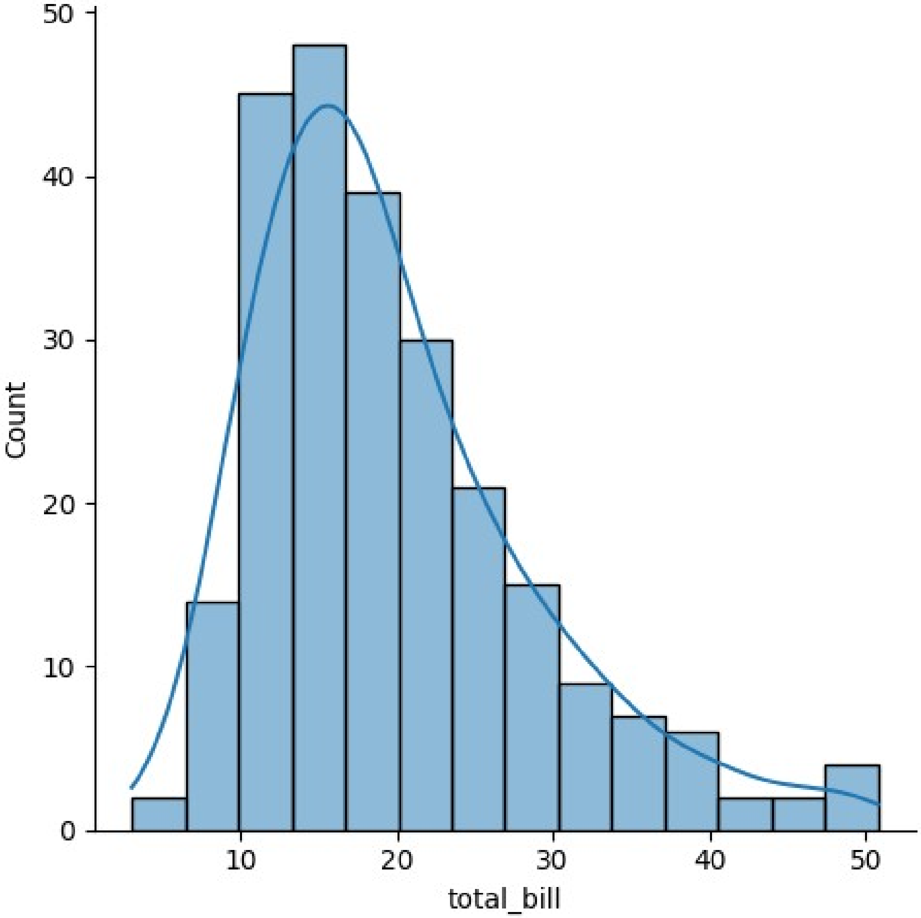
```
   France  Germany  Spain   Age   Salary Purchased
0       1        0      0  44.0  72000.0        No
1       0        0      1  27.0  48000.0       Yes
2       0        1      0  30.0  54000.0        No
3       0        0      1  38.0  61000.0        No
4       0        1      0  40.0  63778.0 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

```
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
```

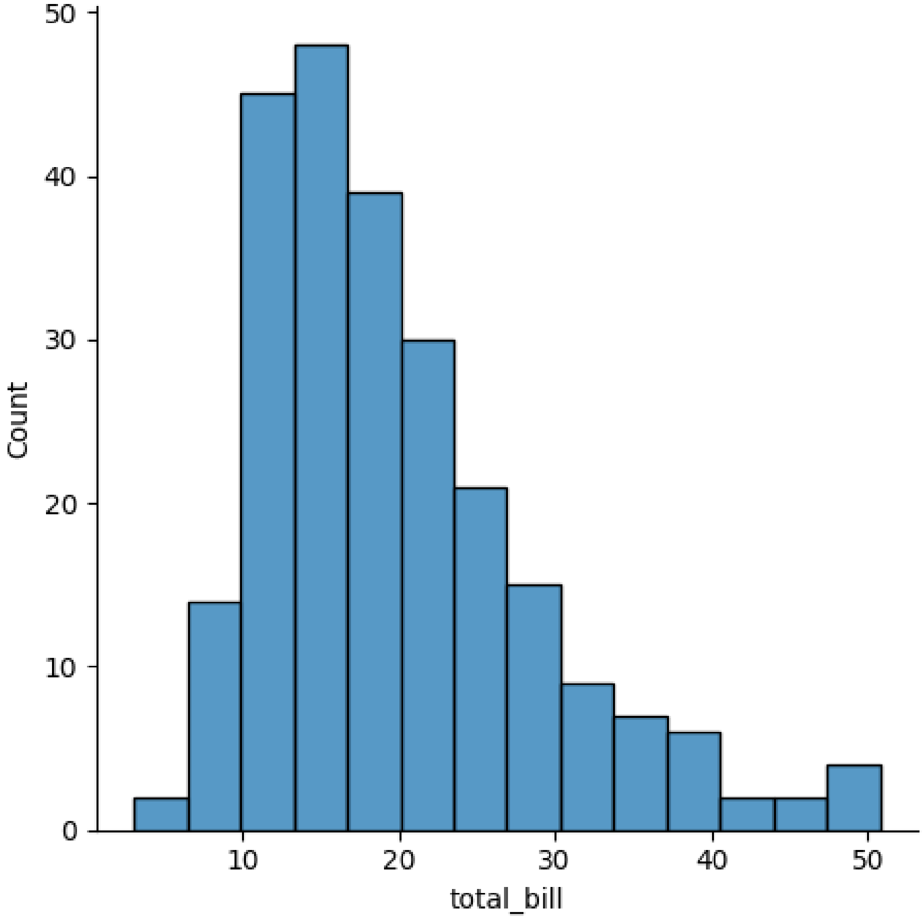
```
sns.displot(tips.total_bill,kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x132efab8348>
```



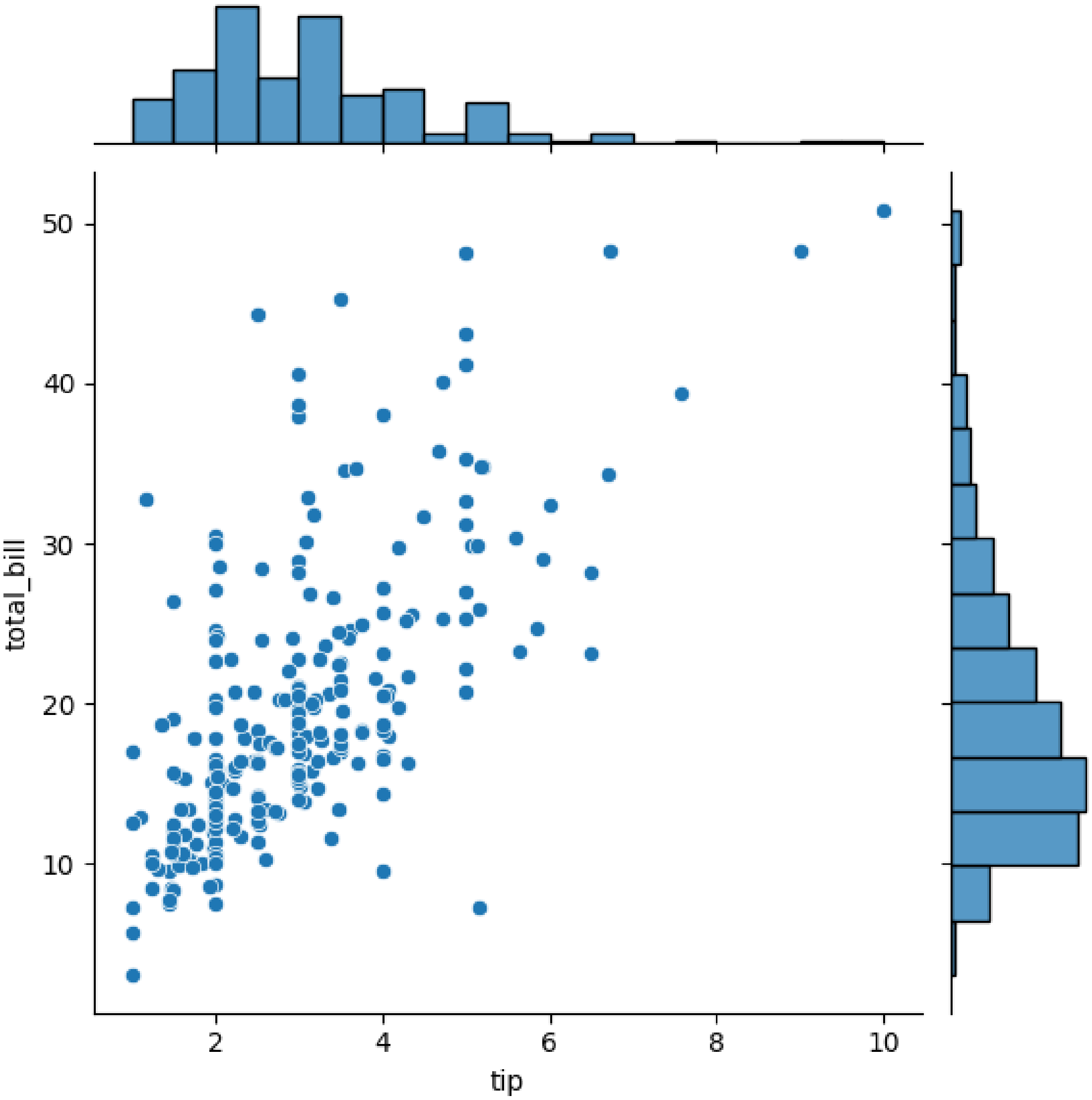
```
sns.displot(tips.total_bill,kde=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x132f1e88148>
```



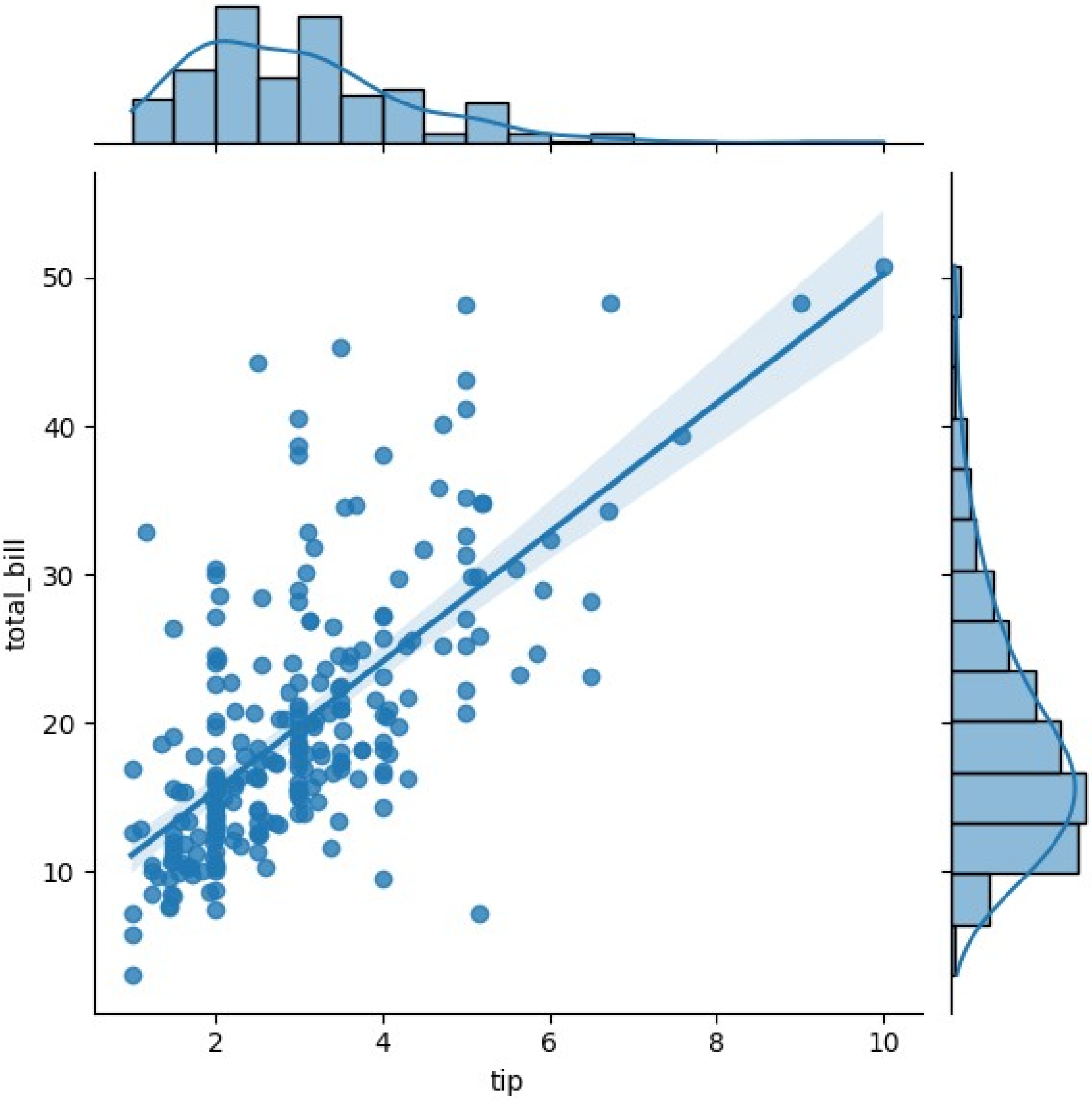
```
sns.jointplot(x=tips.tip,y=tips.total_bill)
```

```
<seaborn.axisgrid.JointGrid at 0x132f1f636c8>
```



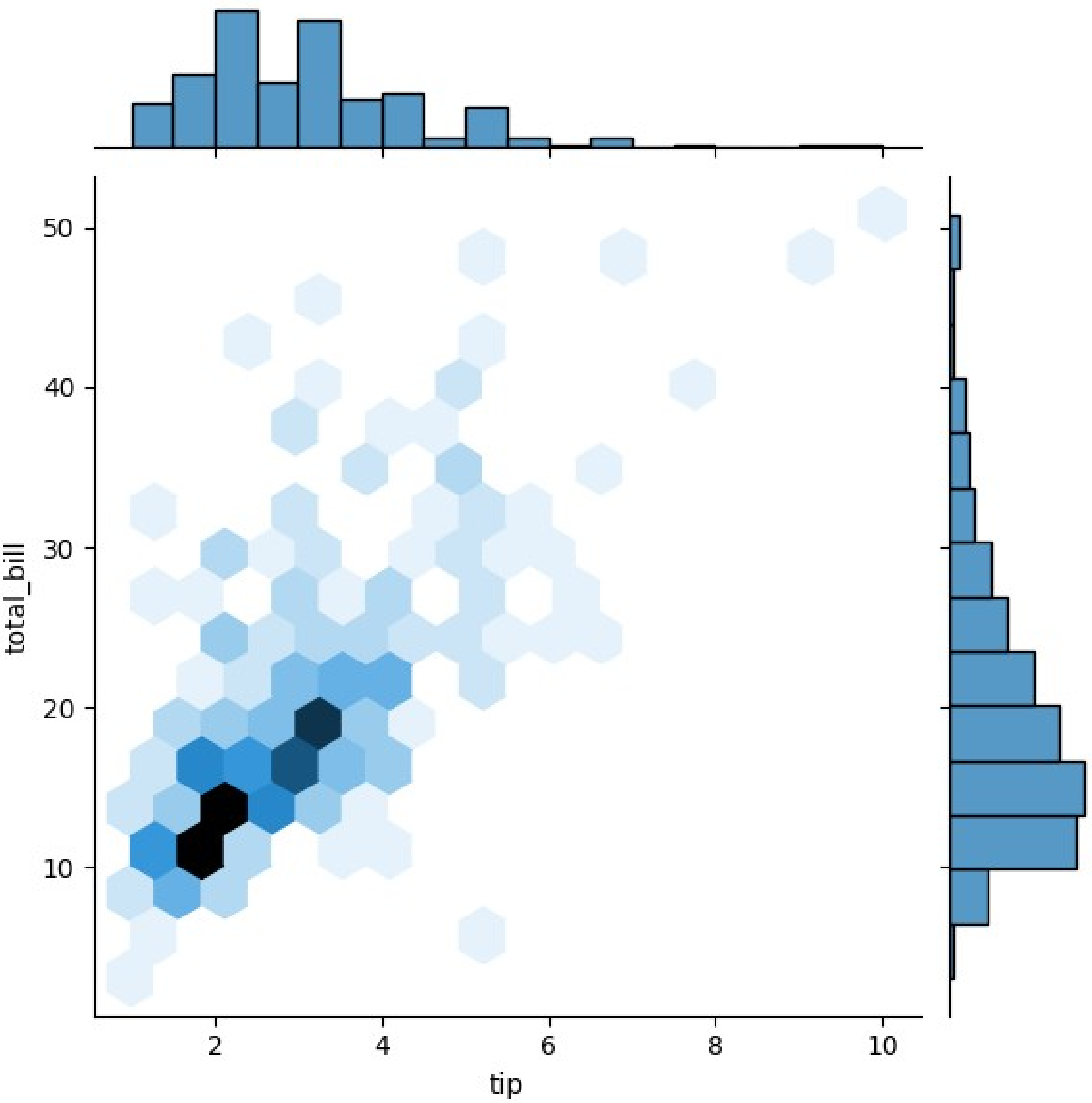
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x132f2224e88>
```



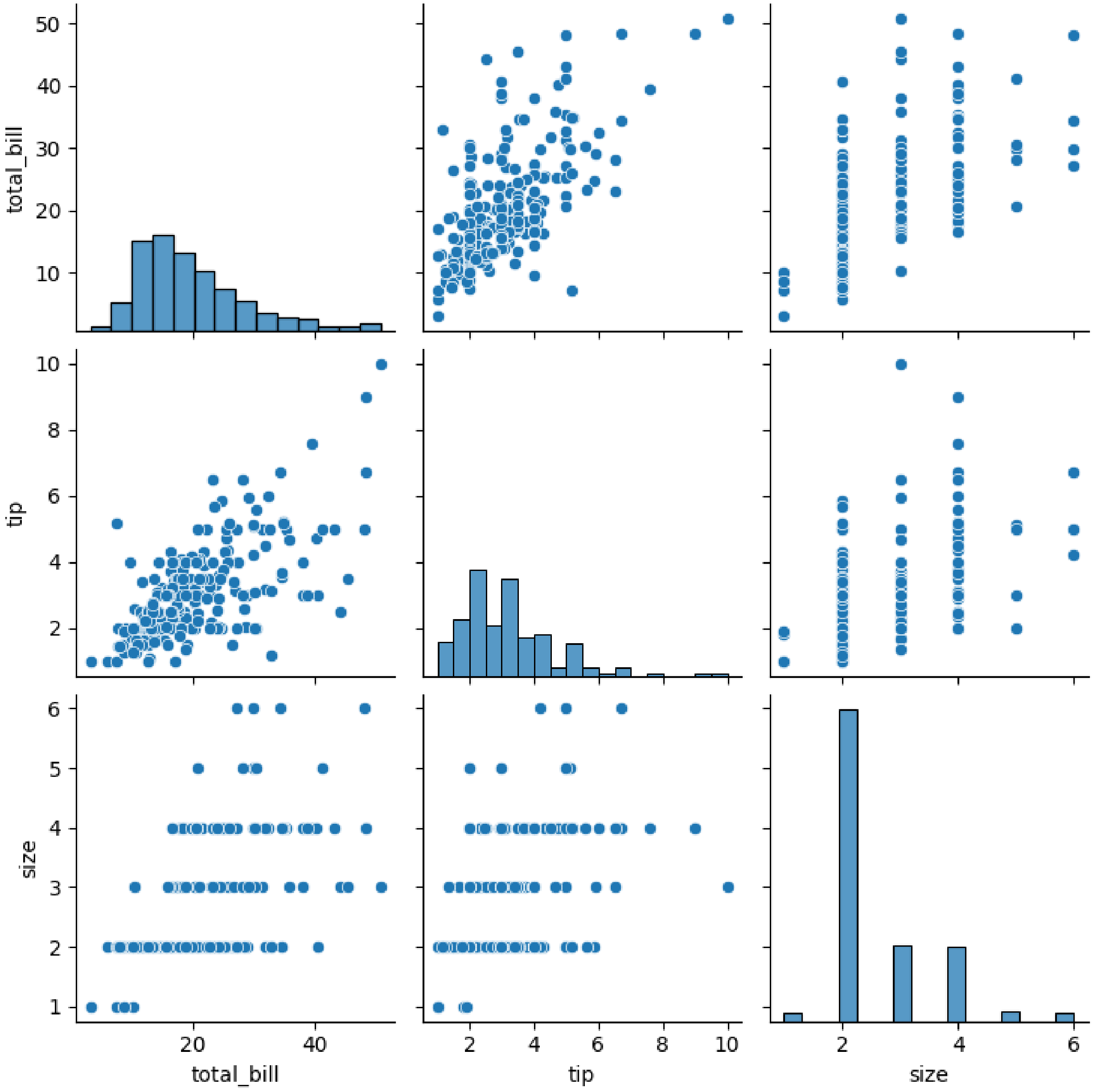
```
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
```

```
<seaborn.axisgrid.JointGrid at 0x132f26f5d08>
```

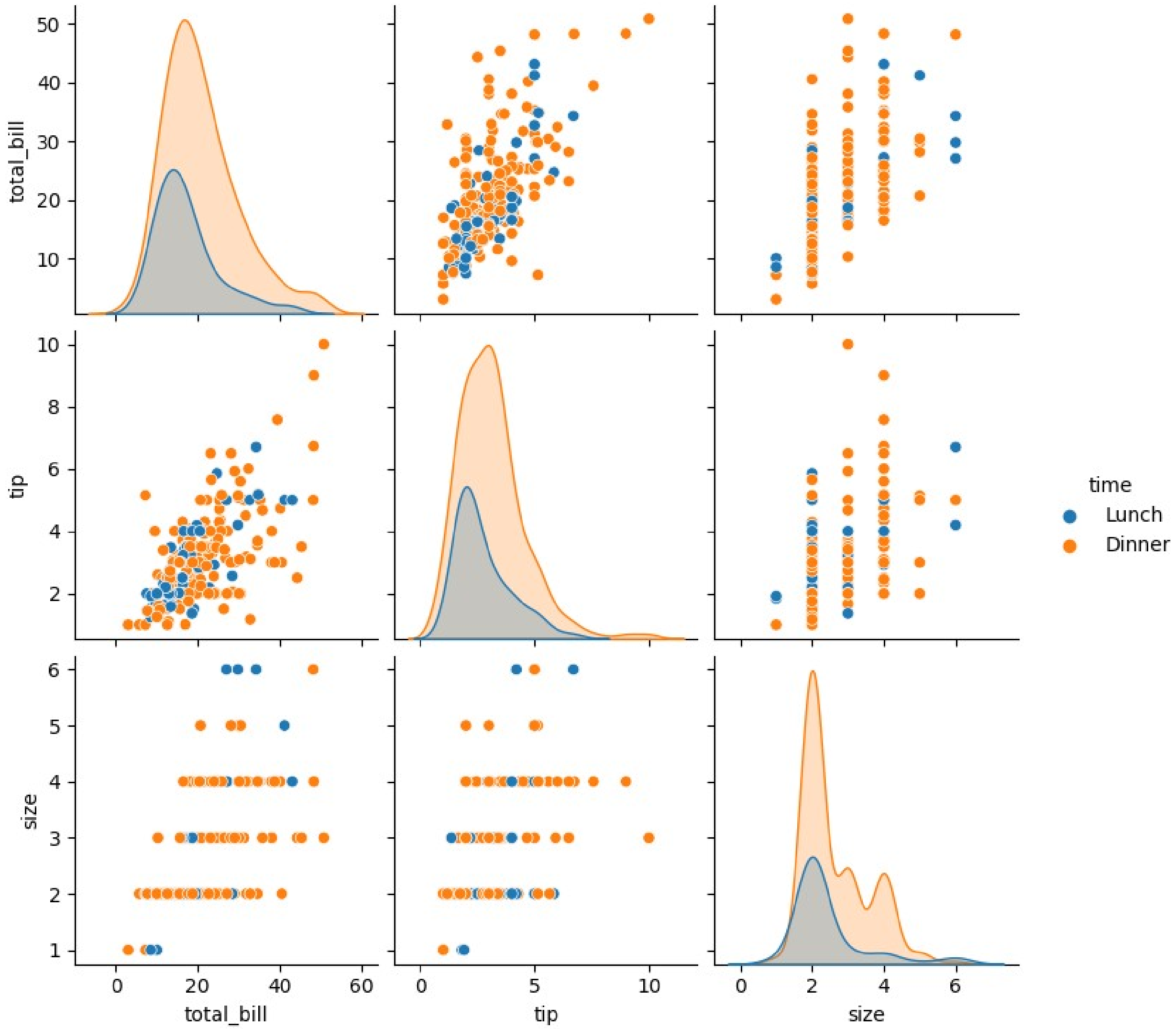


```
sns.pairplot(tips)
```

```
<seaborn.axisgrid.PairGrid at 0x132f26f7708>
```

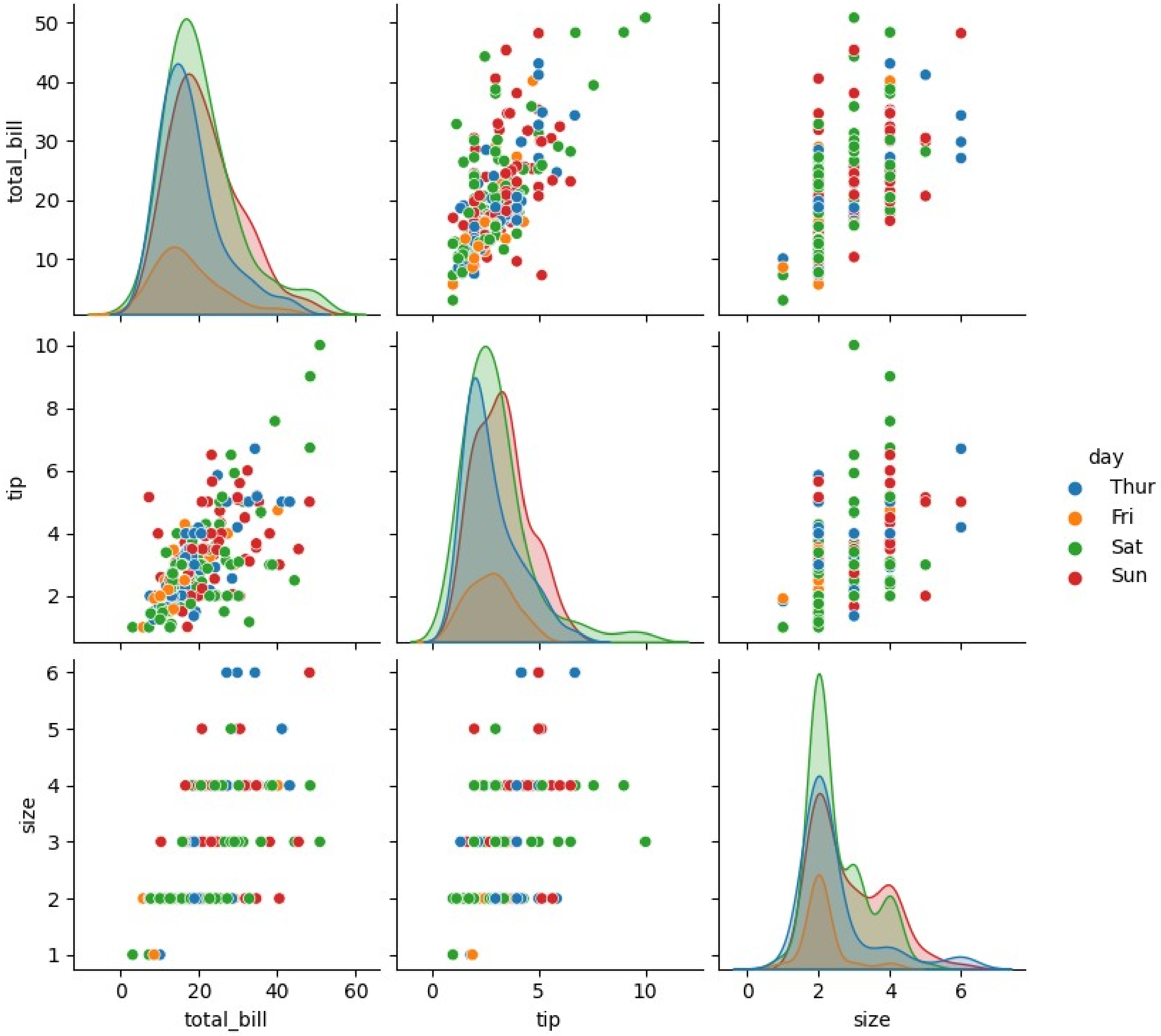


```
tips.time.value_counts()

Dinner    176
Lunch      68
Name: time, dtype: int64

sns.pairplot(tips,hue='time')

<seaborn.axisgrid.PairGrid at 0x132f3420d88>
```



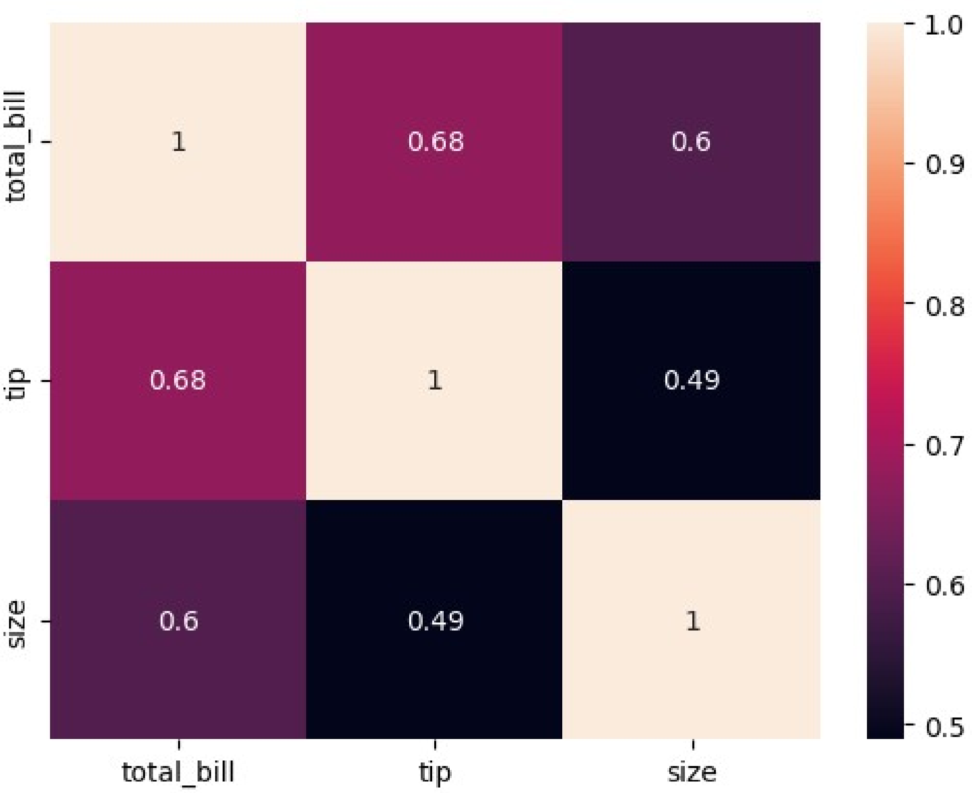
```
sns.pairplot(tips,hue='day')
```

```
<seaborn.axisgrid.PairGrid at 0x132f4c14088>
```



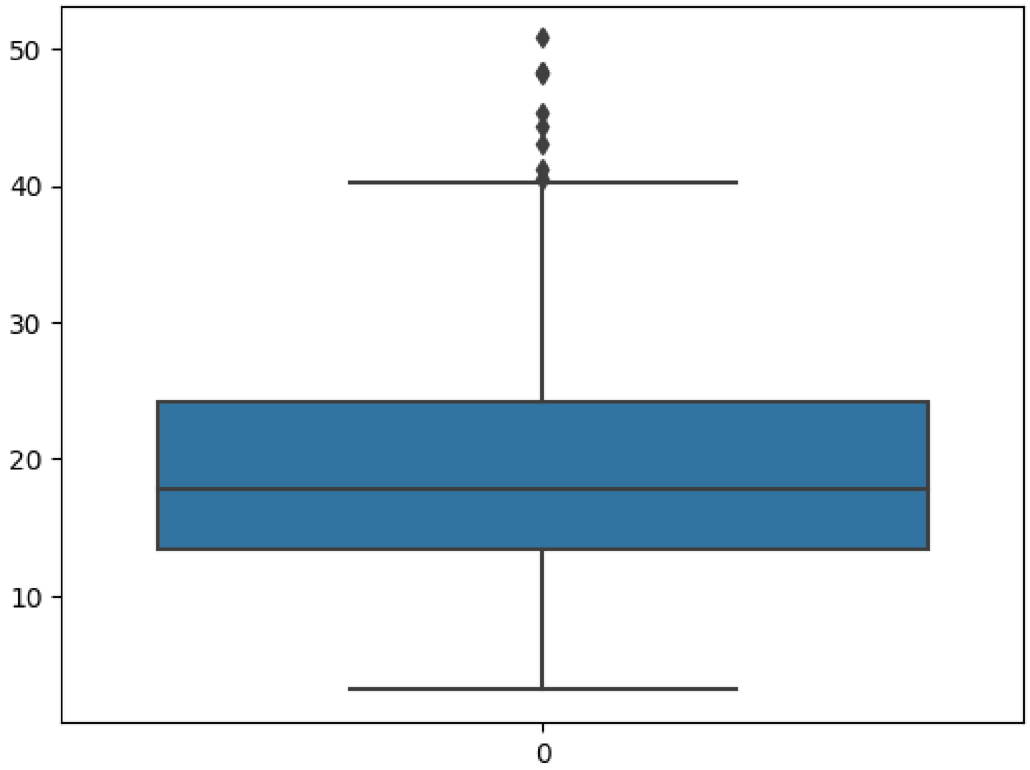
```
sns.heatmap(tips.select_dtypes(include=['number']).corr(), annot=True)
```

```
<AxesSubplot:>
```



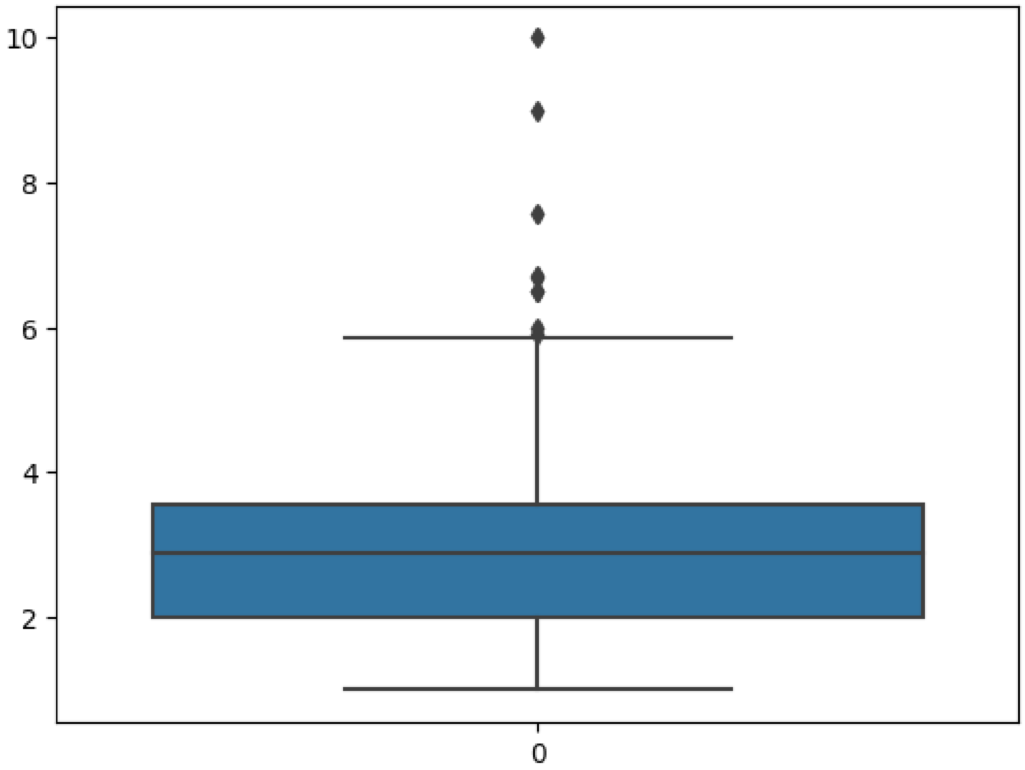
```
sns.boxplot(tips.total_bill)
```

```
<AxesSubplot:>
```



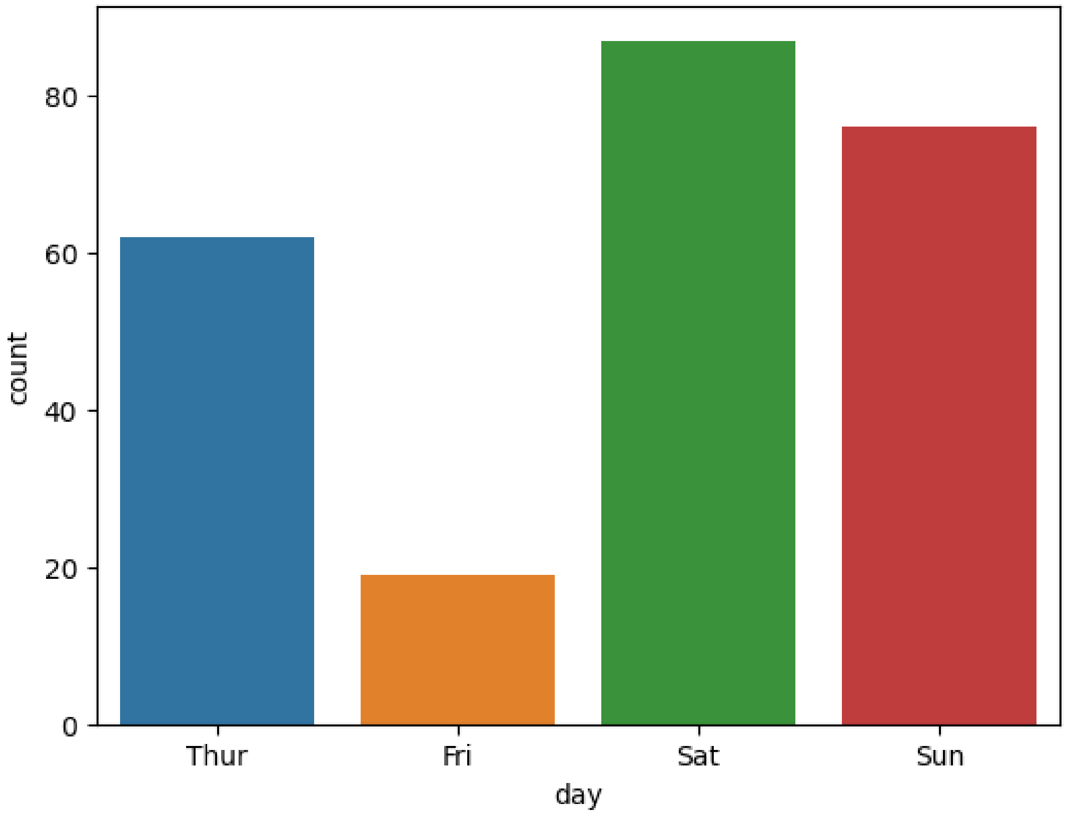
```
sns.boxplot(tips.tip)
```

```
<AxesSubplot:>
```



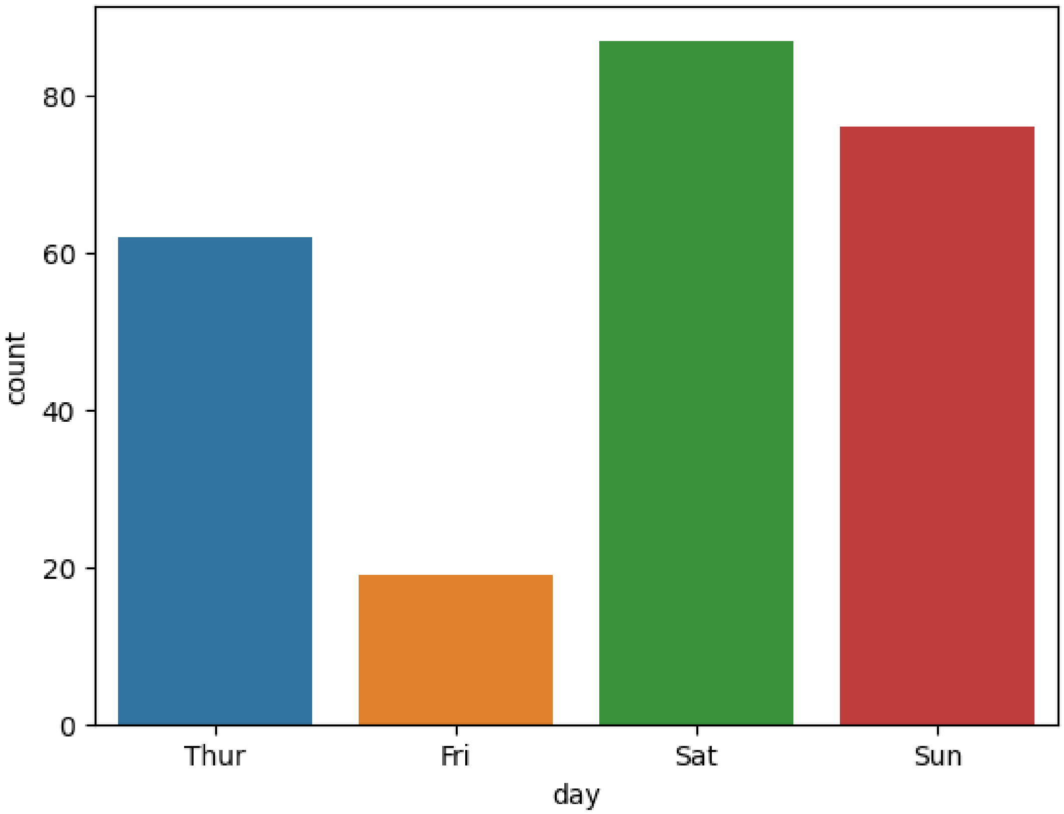
```
sns.countplot(x='day', data=tips)
```

```
<AxesSubplot:xlabel='day', ylabel='count'>
```



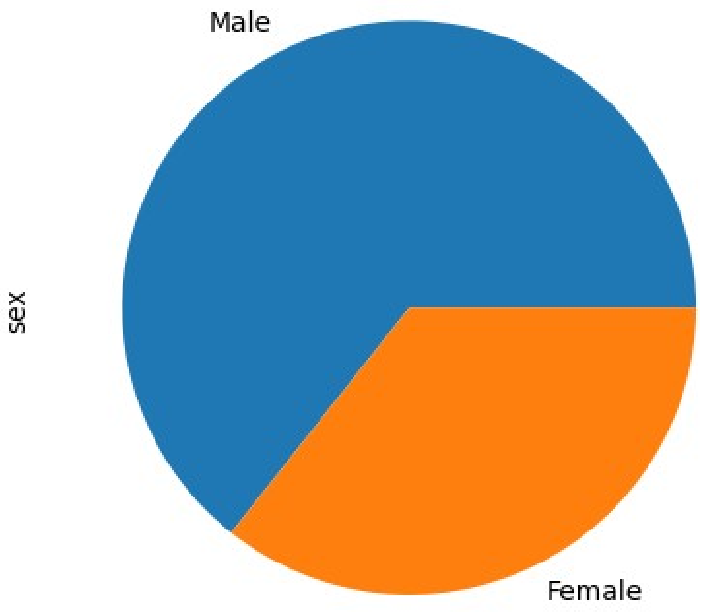
```
sns.countplot(x='day', data=tips)
```

```
<AxesSubplot:xlabel='day', ylabel='count'>
```



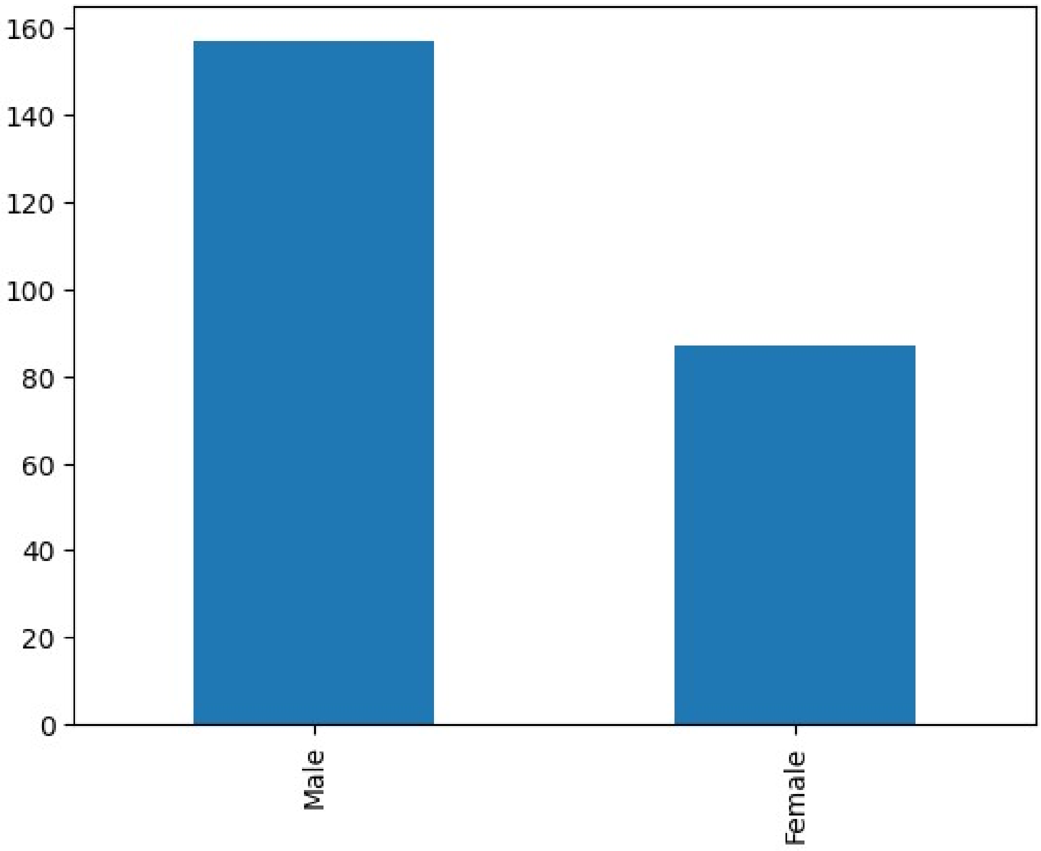
```
tips.sex.value_counts().plot(kind='pie')
```

```
<AxesSubplot:ylabel='sex'>
```



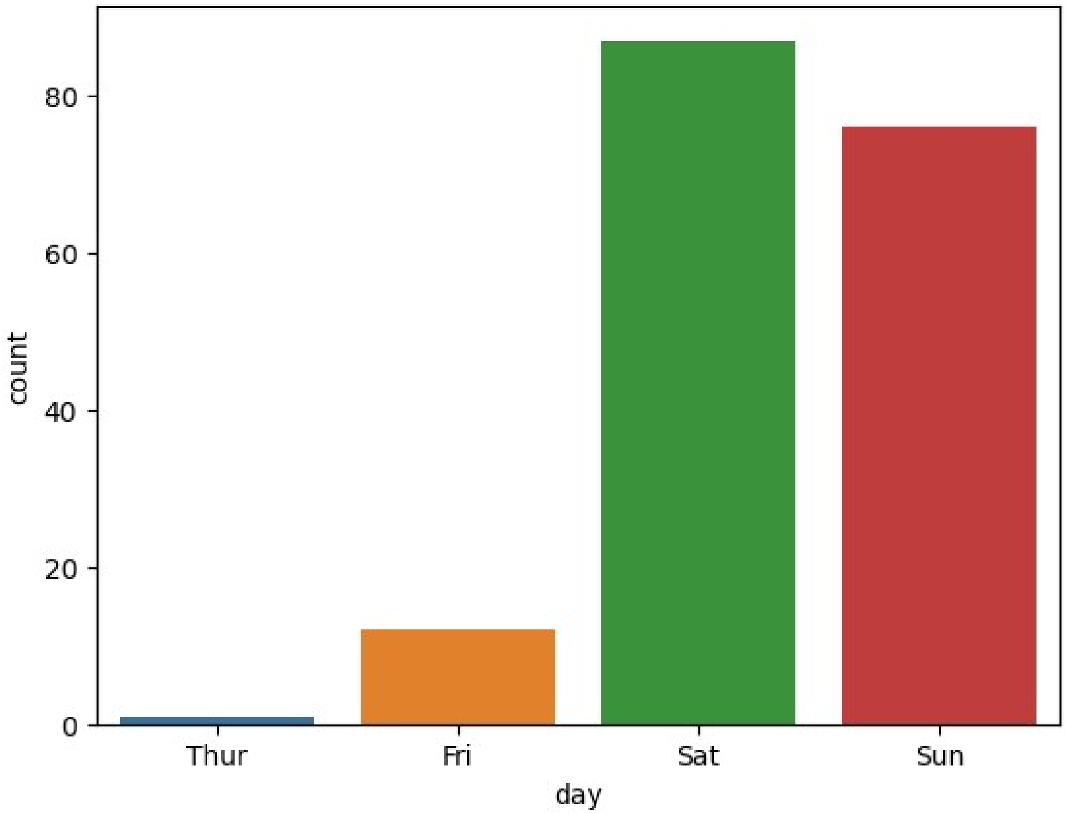
```
tips.sex.value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```

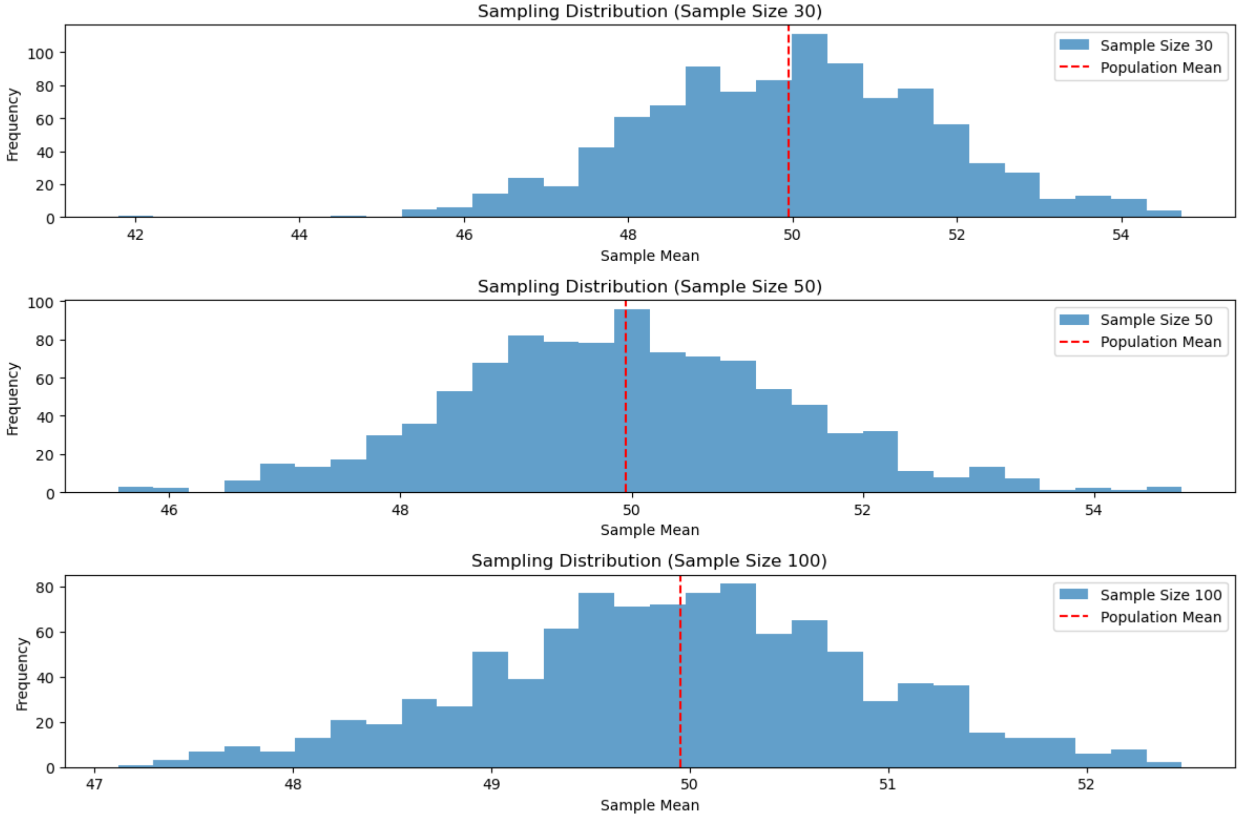


```
sns.countplot(x='day', data=tips[tips['time'] == 'Dinner'])
```

```
<AxesSubplot:xlabel='day', ylabel='count'>
```


80
60
40
20
0
count
Thur
Fri
Sat
Sun
day

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


Sampling Distribution (Sample Size 30)
Frequency
0
20
40
60
80
100
42
44
46
48
50
52
54
Sample Mean
Sample Size 30
Population Mean
Sampling Distribution (Sample Size 50)
Frequency
0
20
40
60
80
100
46
48
50
52
54
Sample Mean
Sample Size 50
Population Mean
Sampling Distribution (Sample Size 100)
Frequency
0
20
40
60
80
47
48
49
50
51
52
Sample Mean
Sample Size 100
Population Mean

```
import math
import numpy as np
from statsmodels.stats.weightstats import ztest
from scipy.stats import norm

sample_marks =
[650,730,510,670,480,800,690,530,590,620,710,670,640,780,650,490,800,6
00,510,700]

# Method 1 : Using Z-score

sample_mean = np.mean(sample_marks)
sample_size = np.count_nonzero(sample_marks)
population_mean = 600
population_std = 100
alpha = 0.05
z_score = (sample_mean-
population_mean)/(population_std/math.sqrt(sample_size))
critical_value = 1.645 # from z table
if(z_score<critical_value):
    print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is
accepted!')


# Method 2: Using built in function of ztest

ztest_score, pval =
ztest(sample_marks,value=population_mean,alternative='larger')
print('Z-test Score:',ztest_score,'\nP-value:',pval)
if(pval>alpha):
     print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is
accepted!')


# Method 3: Creating a function

def ztest(x,mu,sigma,n):
    deno = sigma/math.sqrt(n)
    z = (x-mu)/deno
    p = 2*(1-norm.cdf(abs(z)))
    return z,p

s_mean = np.mean(sample_marks)
p_mean = 600
p_std = 100
s_size = np.count_nonzero(sample_marks)
```

```
ztest(s_mean,p_mean,p_std,s_size)

ztest(641,600,100,20)
```

```
Null hypothesis is rejected.
Alternate hypothesis is accepted!
Z-test Score: 1.831744911595958
P-value: 0.03349471703839336
Null hypothesis is rejected.
Alternate hypothesis is accepted!
```

```
(1.8335757415498277, 0.06671699590108493)
```

```python
import math
import numpy as np
from statsmodels.stats.weightstats import ztest

sample_marks1 = [650,730,510,670,480,800,690,530,590,620,710,670,640,780,650,490,800,600,510,700]
sample_marks2 = [630,720,462,631,440,783,673,519,543,579,677,649,632,768,615,463,781,563,488,650]

sample_mean1 = np.mean(sample_marks1)
sample_mean2 = np.mean(sample_marks2)
sample_size1 = np.count_nonzero(sample_marks1)
sample_size2 = np.count_nonzero(sample_marks2)
population_mean_diff = 10
population_std1 = 100
population_std2 = 90
alpha = 0.05

# Method 1: Using built in function of ztest

z,p = ztest(x1=sample_marks1,x2=sample_marks2,value=population_mean_diff,alternative='larger')
print('Z-score:',z,'\nP-value:',p)

if(p>alpha):
    print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is accepted!')

# Method 2: Calculating Z-score

zscore = ((sample_mean1-sample_mean2)-(population_mean_diff))/(math.sqrt((population_std1**2/sample_size1)+(population_std2**2/sample_size2)))
critical_value = 1.645 # from z table

if(zscore<critical_value):
    print('Null hypothesis is accepted!')
else:
    print('Null hypothesis is rejected. \nAlternate hypothesis is accepted!')
```

```
Z-score: 0.5438117264622684
P-value: 0.293285519251652
Null hypothesis is accepted!
Null hypothesis is accepted!
```

```python
# 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.
```

```
# 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.
```

```python
import numpy as np
from scipy.stats import f_oneway

# Sample data: Exam scores for three teaching methods
np.random.seed(42)
method_A_scores = np.random.normal(loc=80, scale=10, size=30)
method_B_scores = np.random.normal(loc=85, scale=10, size=30)
method_C_scores = np.random.normal(loc=90, scale=10, size=30)

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(method_A_scores, method_B_scores,
method_C_scores)

print("F-Statistic:", f_statistic)
print("P-Value:", p_value)
```

```
F-Statistic: 12.20952551797281
P-Value: 2.1200748140507065e-05
```

```
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: 608.0 bytes
```

```
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
```

```
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_siz
e=0.2,random_state=42)
```

```
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]])
```

```
model.intercept_
```

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

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

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

```python
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
```

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

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

```
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 x 5 columns]
```

```
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
```

```
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],
       [    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)
```

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, 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, 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)
```

```
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_siz
e=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
```

```
x_train,x_test,y_train,y_test=train_test_split(features,label,test_siz
e=0.2,random_state=i)
finalModel=LogisticRegression()
finalModel.fit(x_train,y_train)
```

```
LogisticRegression()
```

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

```
0.625
0.7125
```

```
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
```

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

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

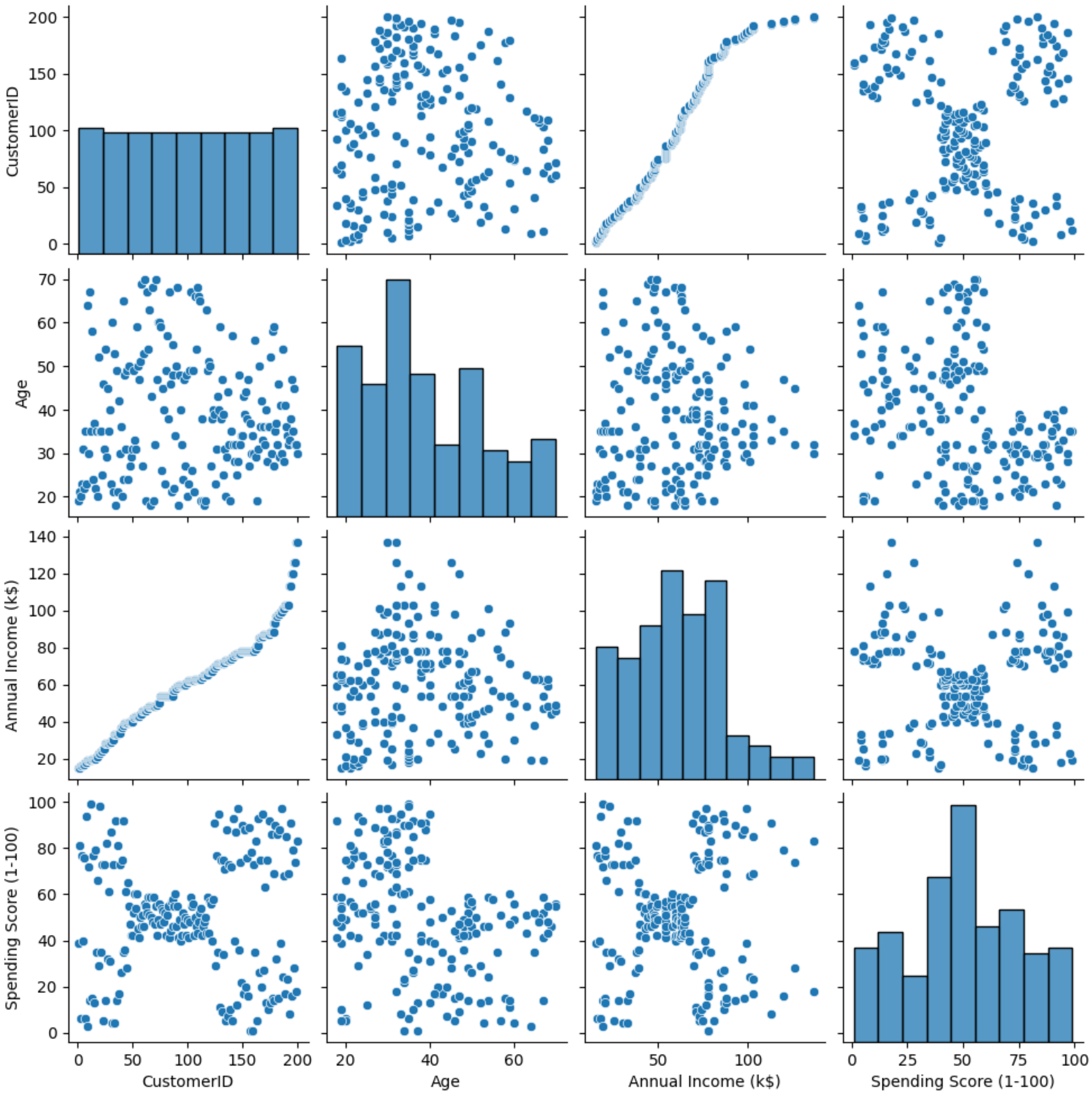
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    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
```

```python
df.head()
```

```
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

```python
sns.pairplot(df)
```

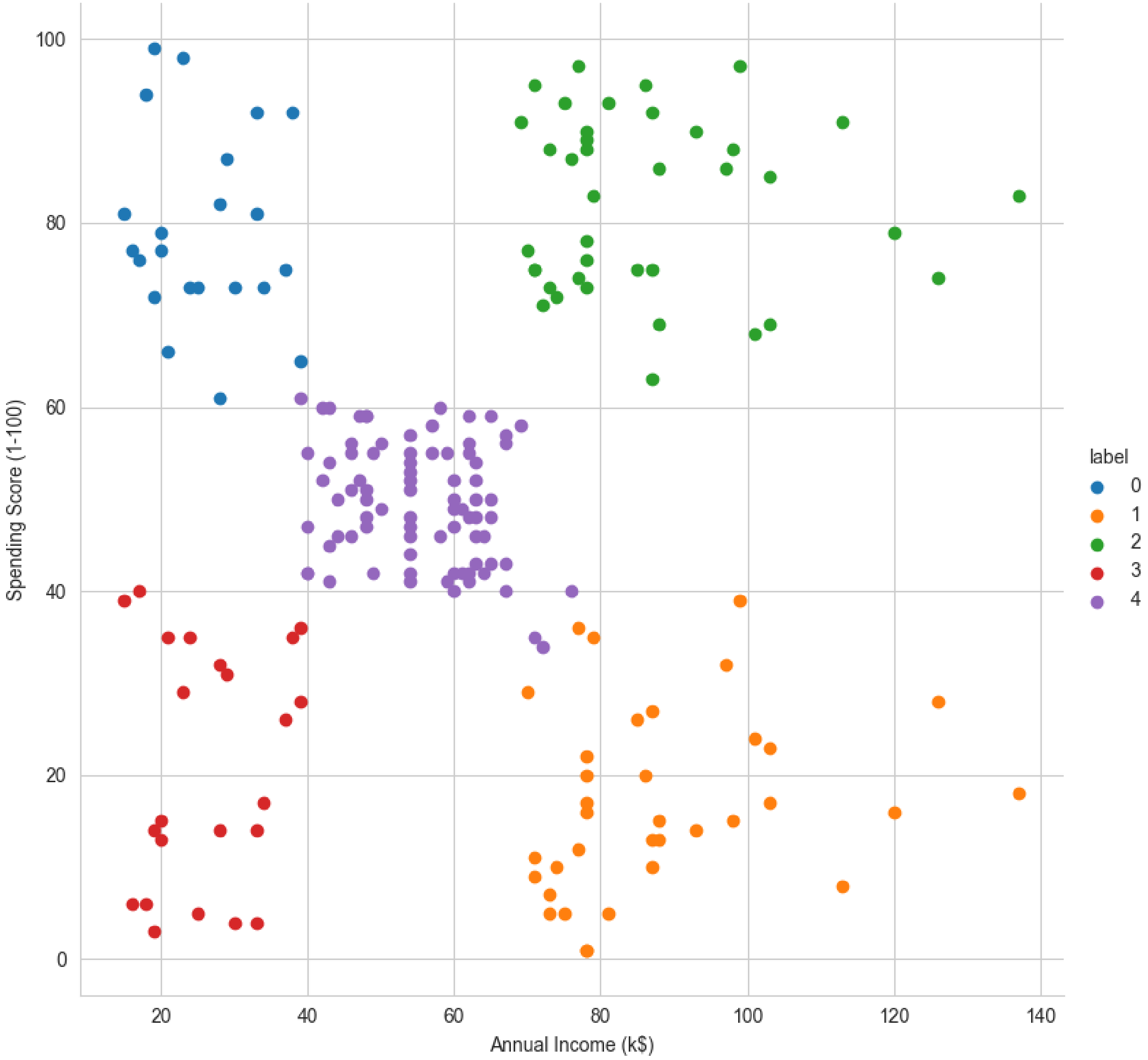
```
<seaborn.axisgrid.PairGrid at 0x11c2ceb0c48>
```



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

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

KMeans(n_clusters=5)

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

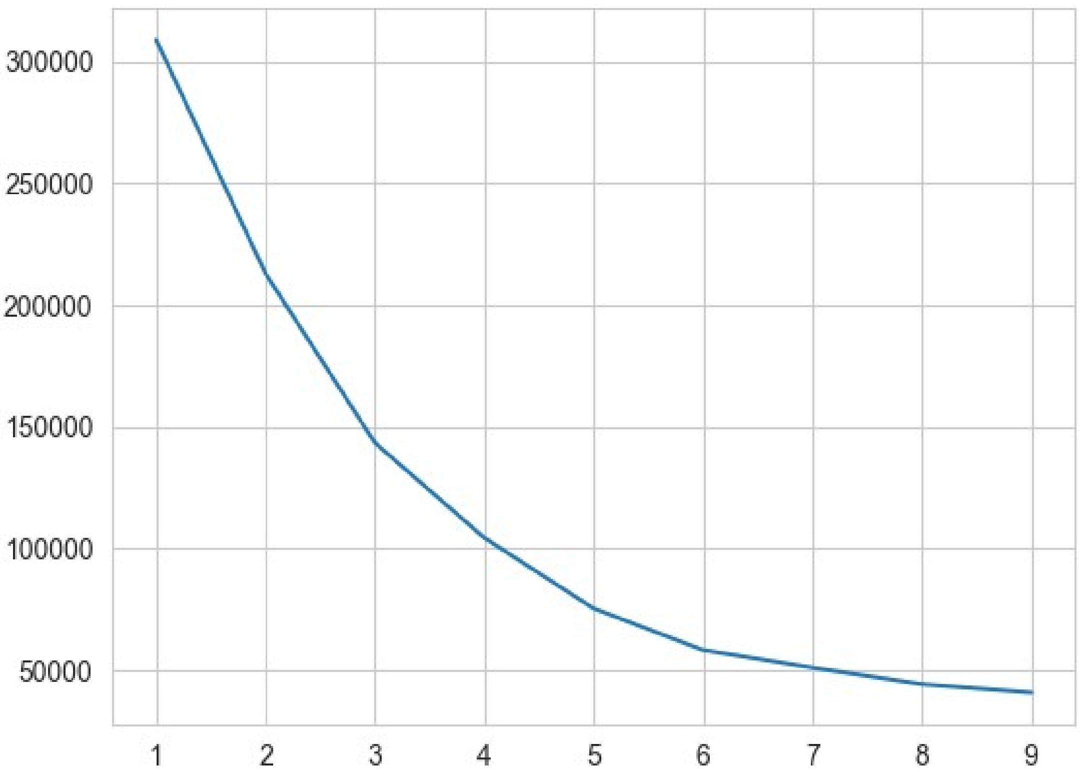
```
c:\users\asus\appdata\local\programs\python\python37\lib\site-
packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy


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

```
[<matplotlib.lines.Line2D at 0x11c30cd9a08>]
```


300000
250000
200000
150000
100000
50000
1
2
3
4
5
6
7
8
9

```
import string
import nltk
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

# Download necessary NLTK data
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Asus\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
# Define text preprocessing function
def textPreprocessing(data):
    if not isinstance(data, str):
        return ""
    remove_pun = [c for c in data if c not in string.punctuation]
    sentences = ''.join(remove_pun)
    words = sentences.split()
    return ' '.join(words)

# Load dataset
file_path = r"spam.csv" # Use raw string to handle backslashes in the
file path
df = pd.read_csv(file_path, sep='\t', names=['label', 'message'],
encoding='latin1')
df['message'] = df['message'].astype(str)

wordVector = CountVectorizer(analyzer=textPreprocessing)
finalWordVector = wordVector.fit(df['message'])
print(finalWordVector.vocabulary_)
bow = finalWordVector.transform(df['message'])

print(bow)
```

```
{'n': 1, 'a': 0}
  (0, 0)	1
  (0, 1)	2
  (1, 0)	1
  (1, 1)	2
  (2, 0)	1
  (2, 1)	2
  (3, 0)	1
  (3, 1)	2
  (4, 0)	1
  (4, 1)	2
  (5, 0)	1
  (5, 1)	2
  (6, 0)	1
  (6, 1)	2
  (7, 0)	1
  (7, 1)	2
  (8, 0)	1
  (8, 1)	2
  (9, 0)	1
  (9, 1)	2
  (10, 0)	1
  (10, 1)	2
  (11, 0)	1
  (11, 1)	2
  (12, 0)	1
  :	:
  (5562, 1)	2
  (5563, 0)	1
  (5563, 1)	2
  (5564, 0)	1
  (5564, 1)	2
  (5565, 0)	1
  (5565, 1)	2
  (5566, 0)	1
  (5566, 1)	2
  (5567, 0)	1
  (5567, 1)	2
  (5568, 0)	1
  (5568, 1)	2
  (5569, 0)	1
  (5569, 1)	2
  (5570, 0)	1
  (5570, 1)	2
  (5571, 0)	1
  (5571, 1)	2
  (5572, 0)	1
  (5572, 1)	2
  (5573, 0)	1
  (5573, 1)	2
  (5574, 0)	1
  (5574, 1)	2

# Transform to TF-IDF features
tfidfObject = TfidfTransformer().fit(bow)
final_feature = tfidfObject.transform(bow)

# Train the Naive Bayes model
model = MultinomialNB()
model.fit(final_feature, df['label'])
```

```
MultinomialNB()
```

```python
# Evaluate the model
score = model.score(final_feature, df['label'])
print("Model Accuracy: ", score)
```

```
Model Accuracy:  0.005381165919282515
```

```python
# Input SMS for prediction
inputSMS = input("Enter the SMS Content: ")
preprocessText = textPreprocessing(inputSMS)
```

```
Enter the SMS Content: random
```

```python
# Transform the input SMS to feature vector
vector = finalWordVector.transform([preprocessText])
finalFeature = tfidfObject.transform(vector)
```

```python
# Predict and print the result
pred = model.predict(finalFeature)[0]
print("Given SMS is", pred)
```

```
Given SMS is ham,"Sorry, I'll call later",,,
```