All about some famous Python Libraries

------------Numpy----------

```
In [1]:  #Numpy stands for Numerical python and is the core library for numeric and scientific compu
         #It consists of multidimensional array objects and a collection of routinces for processing
```

```
In [2]:  import numpy as np
         n1 = np.array([1,2,3,4])
         n1

Out[2]:  array([1, 2, 3, 4])
```

```
In [3]:  type(n1)

Out[3]:  numpy.ndarray
```

```
In [4]:  n2 = np.array([[5,6,7,8],[9,10,11,12]])
         n2

Out[4]:  array([[ 5,  6,  7,  8],
                [ 9, 10, 11, 12]])
```

```
In [5]:  type(n2)

Out[5]:  numpy.ndarray
```

```
In [6]:  #initializing numpy array with zeros
```

```
In [7]:  n1 = np.zeros((1,5))#1 denote the no of rows and 5 denotes the no of columns
         n1

Out[7]:  array([[0., 0., 0., 0., 0.]])
```

```
In [8]:  type(n1)

Out[8]:  numpy.ndarray
```

```
In [9]:  #another example
         n2 = np.zeros((5,5))
         n2

Out[9]:  array([[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.]])
```

```
In [10]: type(n2)

Out[10]: numpy.ndarray
```

```
In [11]: #full method is used to inintialize NumPy array with same number
```

```
In [12]: n1 = np.full((3,2),9)
         n1
```

Out[12]:
```
array([[9, 9],
       [9, 9],
       [9, 9]])
```

In [13]:
```python
#Initializing NumPy array within a range using arange method
```

In [14]:
```python
n1 = np.arange(10,20) #Note: 20 is exclusive
n1
```

Out[14]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [15]:
```python
#another example
n2 = np.arange(0,50,10)
n2
```

Out[15]:
```
array([ 0, 10, 20, 30, 40])
```

In [16]:
```python
#Initializing NumPy array with random numbers
```

In [17]:
```python
n1 = np.random.randint(1,100,10) #means between 1 and 100 give any 10 random values
n1
```

Out[17]:
```
array([79, 12, 20, 18, 36, 82, 87, 97, 41, 21])
```

In [18]:
```python
#Checking the shape of NumPy arrays and changing it
#You need to take care of the thing is the dimension converted must be equal like (4,4) arr
```

In [19]:
```python
n1 = np.array([[10,20,30],[40,50,60]])
n1.shape
```

Out[19]:
```
(2, 3)
```

In [20]:
```python
n1
```

Out[20]:
```
array([[10, 20, 30],
       [40, 50, 60]])
```

In [21]:
```python
n1.shape = (3,2)
n1
```

Out[21]:
```
array([[10, 20],
       [30, 40],
       [50, 60]])
```

In [22]:
```python
#Joining  NumPy arrays
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
n3 = np.array([70,80,90])

np.vstack((n1,n2,n3))
```

Out[22]:
```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

In [23]:
```python
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])

np.hstack((n1,n2))
```

Out[23]:
```
array([10, 20, 30, 40, 50, 60])
```

In [24]:
```python
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
```

```
np.column_stack((n1,n2))
```

Out[24]:
```
array([[10, 40],
       [20, 50],
       [30, 60]])
```

In [25]:
```
#Numpy Operations(Intersection and Difference)
```

In [26]:
```
n1 = np.array([10,20,30,40,50,60])
n2 = np.array([50,60,70,80,90])

np.intersect1d(n1,n2)
```

Out[26]:
```
array([50, 60])
```

In [27]:
```
np.setdiff1d(n1,n2)
```

Out[27]:
```
array([10, 20, 30, 40])
```

In [28]:
```
np.setdiff1d(n2,n1)
```

Out[28]:
```
array([70, 80, 90])
```

In [29]:
```
#NumPy Array Mathematics
```

In [30]:
```
#1. Addition of NumPy Arrays
n1 = np.array([10,20])
n2 = np.array([30,40])
np.sum([n1,n2])
```

Out[30]:
```
100
```

In [31]:
```
np.sum([n1,n2], axis = 0) #Axis 0 means vertically and 1 means horizontally
```

Out[31]:
```
array([40, 60])
```

In [32]:
```
np.sum([n1,n2], axis = 1)
```

Out[32]:
```
array([30, 70])
```

In [33]:
```
#Scaling values inside an array
```

In [34]:
```
#Basic Addition
n1 = np.array([10,20,30])
n1 = n1+1
n1
```

Out[34]:
```
array([11, 21, 31])
```

In [35]:
```
#Basic Subtraction
n1 = n1-1
n1
```

Out[35]:
```
array([10, 20, 30])
```

In [36]:
```
#Basic Multiplication
n1 = n1 * 2
n1
```

Out[36]:
```
array([20, 40, 60])
```

In [37]:

```
#Basic Division
n1 = n1 / 4
n1
```

Out[37]:

```
array([ 5., 10., 15.])
```

In [38]:

```
#Numpy Maths Functions
#1.Mean
n1 = np.array([10,20,30,40,50,60])
np.mean(n1)
```

Out[38]:

```
35.0
```

In [39]:

```
np.median(n1)
```

Out[39]:

```
35.0
```

In [40]:

```
np.std(n1)
```

Out[40]:

```
17.07825127659933
```

In [41]:

```
#Numpy Save & Load
n1 = np.array([10,20,30,40,50,60])
np.save('my_numpy',n1)
```

In [42]:

```
n2 = np.load('my_numpy.npy')
n2
```

Out[42]:

```
array([10, 20, 30, 40, 50, 60])
```

------------------------Pandas-------------------------

In [43]:

```
#Pandas stands for Panel Data and is the core library for data manipulation and data analys
#It consists of single and multi-dimensional data structures for data manipulation
```

In [44]:

```
#Single-dimensional data structures are known as Series Object and Multidimensional data st
```

In [45]:

```
#Series object is one-dimensional labeled array
```

In [46]:

```
import pandas as pd
s1 = pd.Series([1,2,3,4,5]) #Take care S is capital in series :)
s1
```

Out[46]:

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [47]:

```
type(s1)
```

Out[47]:

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

In [48]:

```
s2 = pd.Series([10,20,30,40,50], index = ['a', 'b', 'c', 'd', 'e'])
s2
```

```
Out[48]: a    10
         b    20
         c    30
         d    40
         e    50
         dtype: int64
```

```
In [49]: #Series Object from Dictionary
```

```
In [50]: pd.Series({'a':10, 'b':20, 'c':30})#key will be working as an index
```

```
Out[50]: a    10
         b    20
         c    30
         dtype: int64
```

```
In [51]: #Changing index position and its repositioning

         pd.Series({'a':10, 'b':20, 'c':30}, index = ['b', 'c', 'd', 'a']) #NaN stands for Not a Nu
```

```
Out[51]: b    20.0
         c    30.0
         d     NaN
         a    10.0
         dtype: float64
```

```
In [52]: #EXTRACTING INDIVIDUAL ELEMENTS
```

```
In [53]: #1. Extracting a single element
         s1 = pd.Series([10,20,30,40,50,60,70,80,90])
         s1
```

```
Out[53]: 0    10
         1    20
         2    30
         3    40
         4    50
         5    60
         6    70
         7    80
         8    90
         dtype: int64
```

```
In [54]: s1[3]
```

```
Out[54]: 40
```

```
In [55]: #2. Extracting a sequence of elements
         s1[:4]
```

```
Out[55]: 0    10
         1    20
         2    30
         3    40
         dtype: int64
```

```
In [56]: #3. Extracting elements from back
         s1[-3:]
```

```
Out[56]: 6    70
         7    80
         8    90
         dtype: int64
```

```
In [57]: #Adding a scalar value to Series elements
```

```
s1 + 5
```

Out[57]:

```
0    15
1    25
2    35
3    45
4    55
5    65
6    75
7    85
8    95
dtype: int64
```

In [58]:

```
#Adding two Series objects, we can also use -,*, / etc also
s1 = pd.Series([1,2,3,4,5,6,7,8,9])
s2 = pd.Series([10,20,30,40,50,60,70,80,90])
```

In [59]:

```
s1 + s2
```

Out[59]:

```
0    11
1    22
2    33
3    44
4    55
5    66
6    77
7    88
8    99
dtype: int64
```

In [60]:

```
#Pandas Dataframe -->> Dataframe is a 2 dimensional labelled data-structure comprises of ro
```

In [61]:

```
import pandas as pd

pd.DataFrame({"Name":['Nik', 'Sam' ,'Apu'], "Marks":[86,47,89]})
```

Out[61]:

	Name	Marks
0	Nik	86
1	Sam	47
2	Apu	89

In [62]:

```
#DataFrame in-built functions
```

In [63]:

```
iris = pd.read_csv('iris.csv')
```

In [64]:

```
iris.head()
```

Out[64]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [65]:

```
iris.tail()
```

Out[65]:

	150	4	setosa	versicolor	virginica
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

In [66]:

```
iris.shape
```

Out[66]:

```
(150, 5)
```

In [67]:

```
iris.describe()
```

Out[67]:

	150	4	setosa	versicolor	virginica
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [68]:

```
iris.head()
```

Out[68]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [69]:

```
iris.iloc[5:11, 2:] #5:11 gives us the index from 5 to 11 exclusive and 2: denotes all the
```

Out[69]:

	setosa	versicolor	virginica
5	1.7	0.4	0
6	1.4	0.3	0
7	1.5	0.2	0
8	1.4	0.2	0
9	1.5	0.1	0
10	1.5	0.2	0

In [70]:

```
iris.loc[0:3, ("setosa", "versicolor")]
```

Out[70]:

	setosa	versicolor
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2

In [71]:

```
#Dropping Columns
```

In [72]:

```
iris.drop('virginica', axis = 1)
```

Out[72]:

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

150 rows × 4 columns

In [73]:

```
#Dropping Rows
iris.drop([1,2,3], axis = 0)
```

Out[73]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

147 rows × 5 columns

In [74]:
```
#More pandas functions
iris.mean()
```

Out[74]:
```
150           5.843333
4             3.057333
setosa        3.758000
versicolor    1.199333
virginica     1.000000
dtype: float64
```

In [75]:
```
iris.median()
```

Out[75]:
```
150           5.80
4             3.00
setosa        4.35
versicolor    1.30
virginica     1.00
dtype: float64
```

In [76]:
```
iris.min()
```

Out[76]:
```
150           4.3
4             2.0
setosa        1.0
versicolor    0.1
virginica     0.0
dtype: float64
```

In [77]:
```
iris.max()
```

Out[77]:
```
150           7.9
4             4.4
setosa        6.9
versicolor    2.5
virginica     2.0
dtype: float64
```

In [78]:
```
iris.head()
```

Out[78]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [79]:

```
def double_make(s):
    return s*2
```

In [80]:

```
iris[['150','4']].apply(double_make)
```

Out[80]:

	150	4
0	10.2	7.0
1	9.8	6.0
2	9.4	6.4
3	9.2	6.2
4	10.0	7.2
...	...	...
145	13.4	6.0
146	12.6	5.0
147	13.0	6.0
148	12.4	6.8
149	11.8	6.0

150 rows × 2 columns

In [81]:

```
iris.head()
```

Out[81]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [82]:

```
iris['virginica'].value_counts()
```

Out[82]:

```
virginica
0    50
1    50
2    50
Name: count, dtype: int64
```

In [83]:

```
iris.sort_values(by = 'setosa')
```

Out[83]:

	150	4	setosa	versicolor	virginica
22	4.6	3.6	1.0	0.2	0
13	4.3	3.0	1.1	0.1	0
14	5.8	4.0	1.2	0.2	0
35	5.0	3.2	1.2	0.2	0
36	5.5	3.5	1.3	0.2	0
...	...	...	...	...	...
131	7.9	3.8	6.4	2.0	2
105	7.6	3.0	6.6	2.1	2
117	7.7	3.8	6.7	2.2	2
122	7.7	2.8	6.7	2.0	2
118	7.7	2.6	6.9	2.3	2

150 rows × 5 columns

----Matplotlib----

In [84]:
```
#Matplotlib is a python library used for data visualization
#You can create bar-plots, scatter-plots, histograms and a lot more with matplotlib
```

In [85]:
```
#Line Plot
```

In [86]:
```
import numpy as np
from matplotlib import pyplot as plt
```

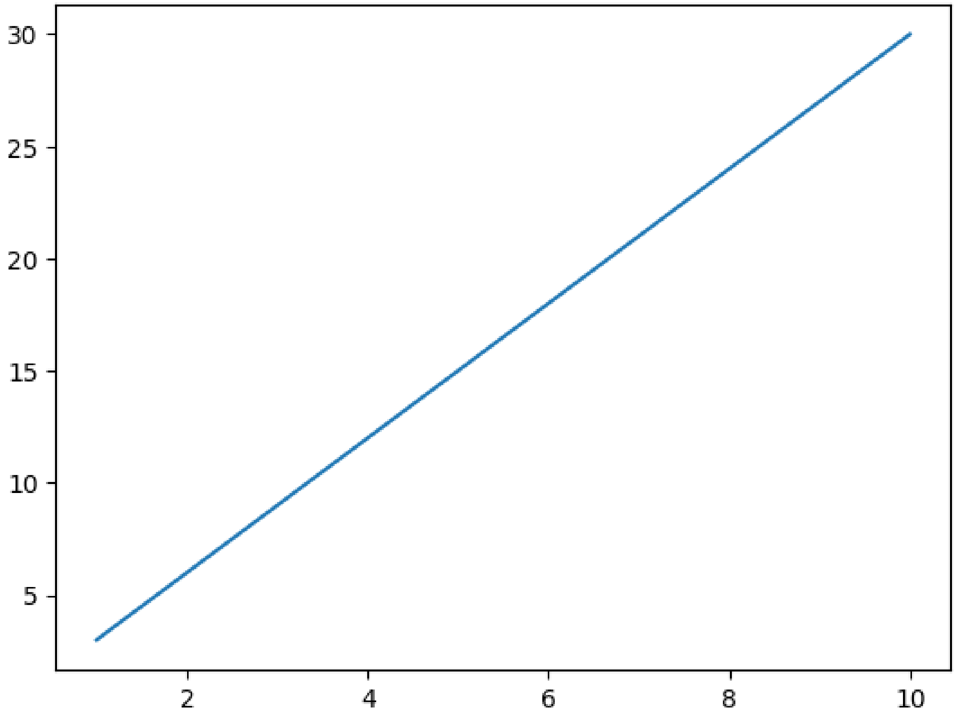
In [87]:
```
x = np.arange(1,11)
x
```

Out[87]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [88]:
```
y = 3 * x
y
```

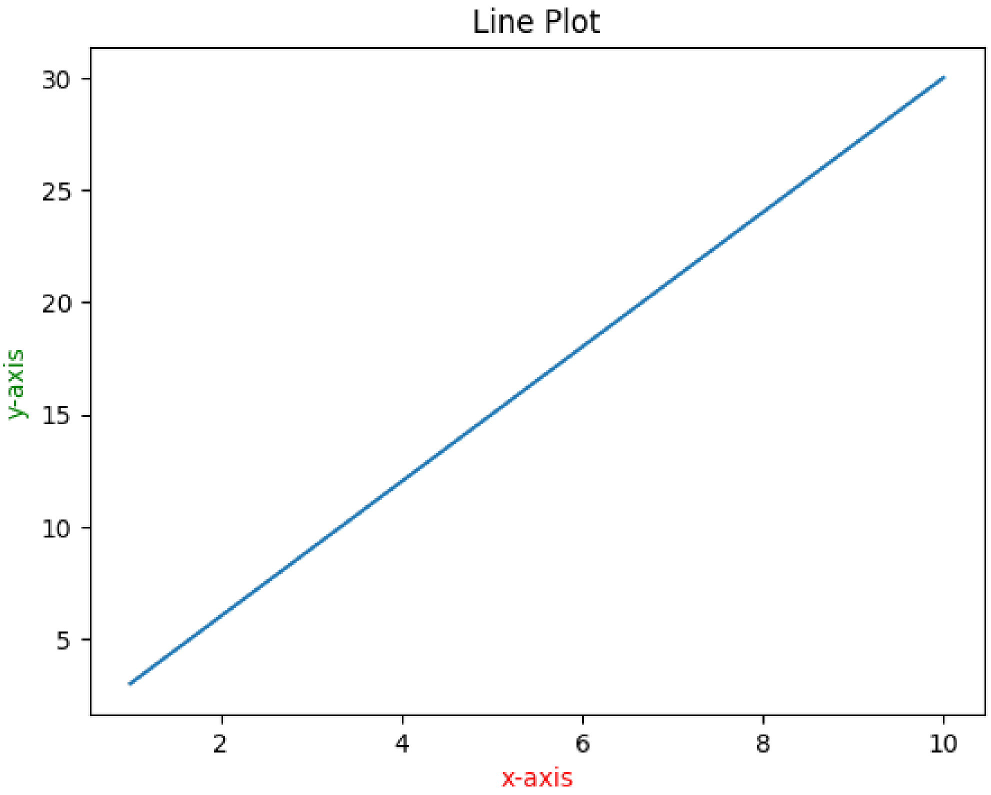
Out[88]:
```
array([ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30])
```

In [89]:
```
plt.plot(x,y)
plt.show()
```

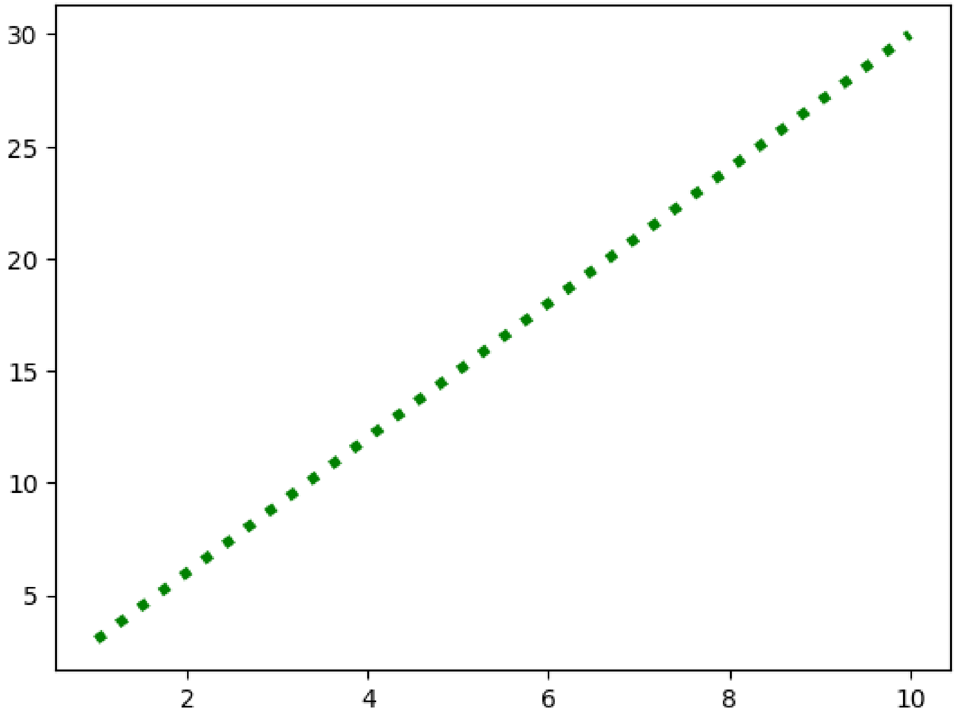


In [90]:
```
#Adding Title and Labels
```

In [91]:
```
plt.plot(x,y)
plt.title("Line Plot")
plt.xlabel("x-axis", c = "red")
plt.ylabel("y-axis", color = 'green')
plt.show()
```



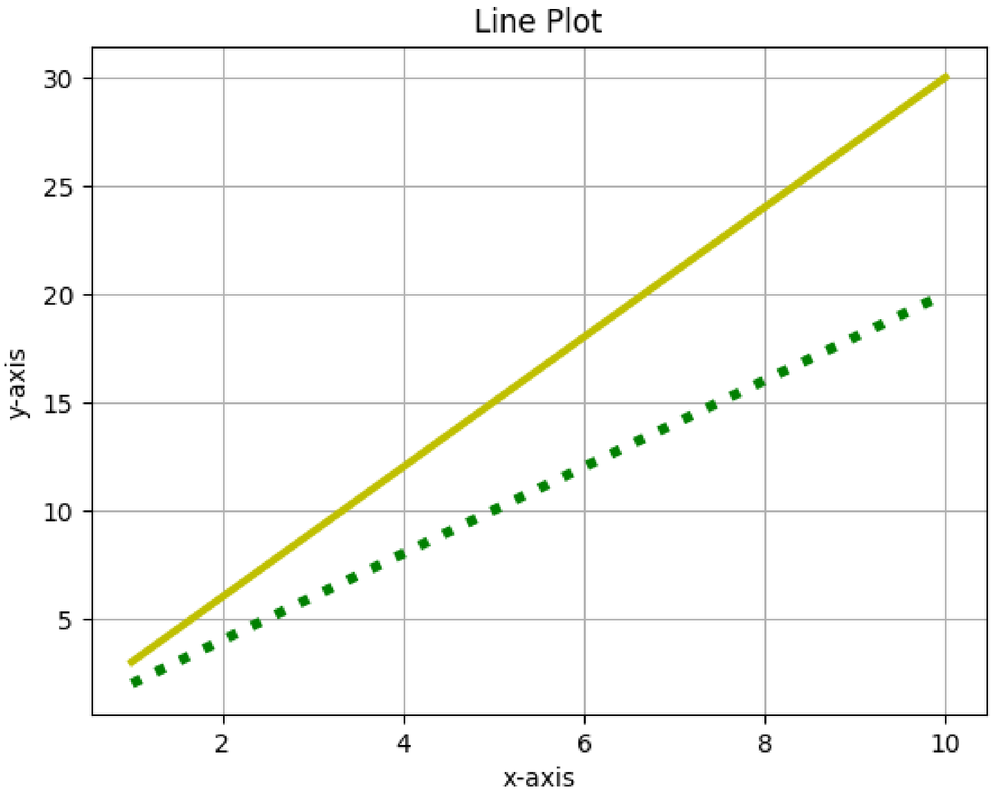
In [92]:
```
#Changing Line Aesthetics
plt.plot(x,y,color = 'g', linestyle = ':', linewidth =4 )
plt.show()
```



```
In [93]: #Adding two lines in the same plot
```

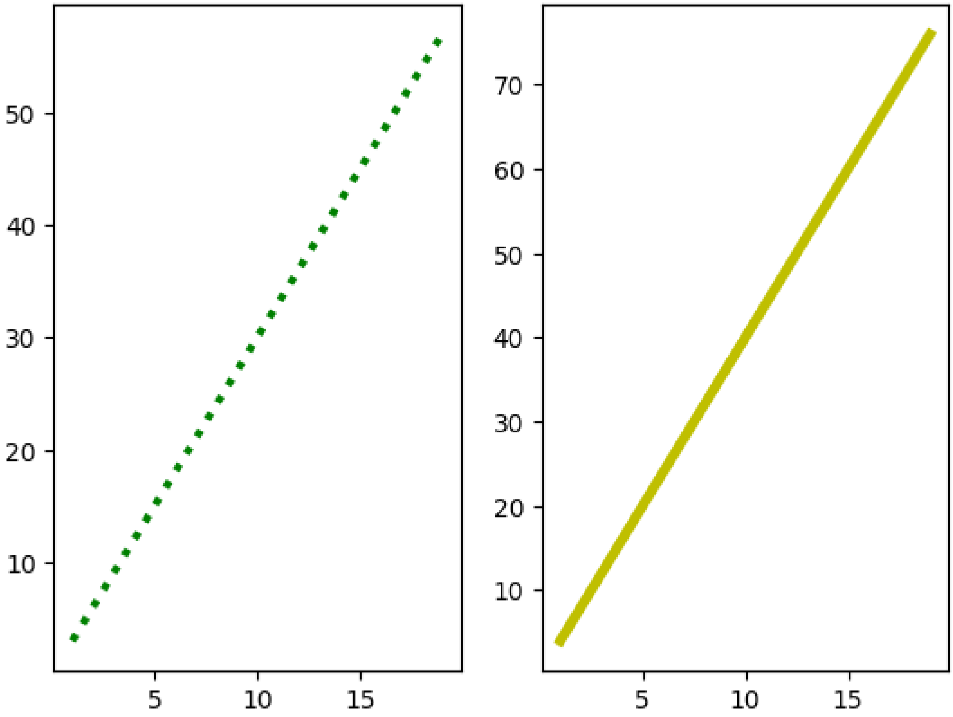
```
In [94]: x = np.arange(1,11)
         y1 = 2*x
         y2 = 3*x
```

```
In [95]: plt.plot(x,y1, color = 'g', linestyle = ':', linewidth = 4)
         plt.plot(x,y2, color = 'y', linestyle = '-', linewidth = 3)
         plt.title("Line Plot")
         plt.xlabel("x-axis")
         plt.ylabel("y-axis")
         plt.grid(True)
         plt.show()
```



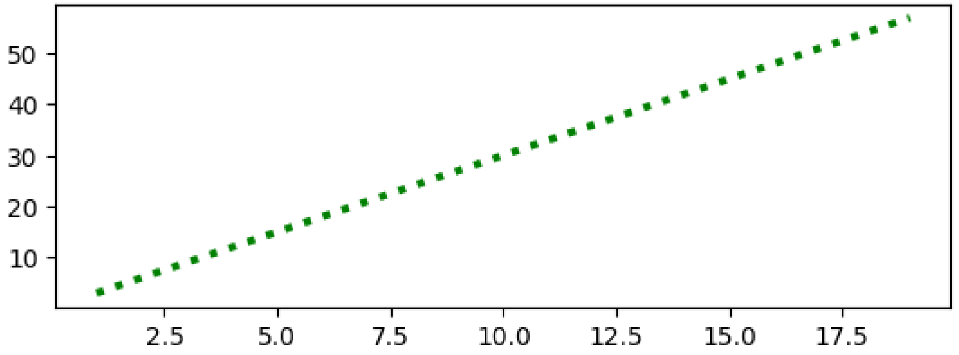
In [96]:

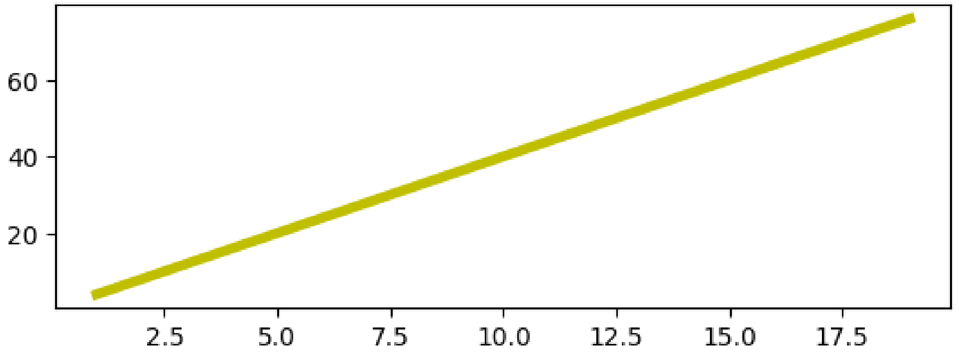
```
#Adding sub-plots in the line plot
x = np.arange(1,20)
y1 = 3 * x
y2 = 4 * x

plt.subplot(1,2,1) # means make 1 rows and 2 columns type subplots and last 1 indicates thi
plt.plot(x, y1, color = 'g', linestyle = ':', linewidth = 3)

plt.subplot(1,2,2)
plt.plot(x, y2, color = 'y', linestyle = '-', linewidth = 4)


plt.show()
```



In [97]:

```
#Above one with column wise
x = np.arange(1,20)
y1 = 3 * x
y2 = 4 * x

plt.subplot(2,1,1) # means make 1 rows and 2 columns type subplots and last 1 indicates thi
plt.plot(x, y1, color = 'g', linestyle = ':', linewidth = 3)

plt.subplot(2,1,2)
plt.plot(x, y2, color = 'y', linestyle = '-', linewidth = 4)


plt.show()
```





```
In [98]: #Bar Plot is used for categorical data
```

```
In [99]: student = {'golu':85, 'molu':96,'bholu':49, 'chhotu':68}
```

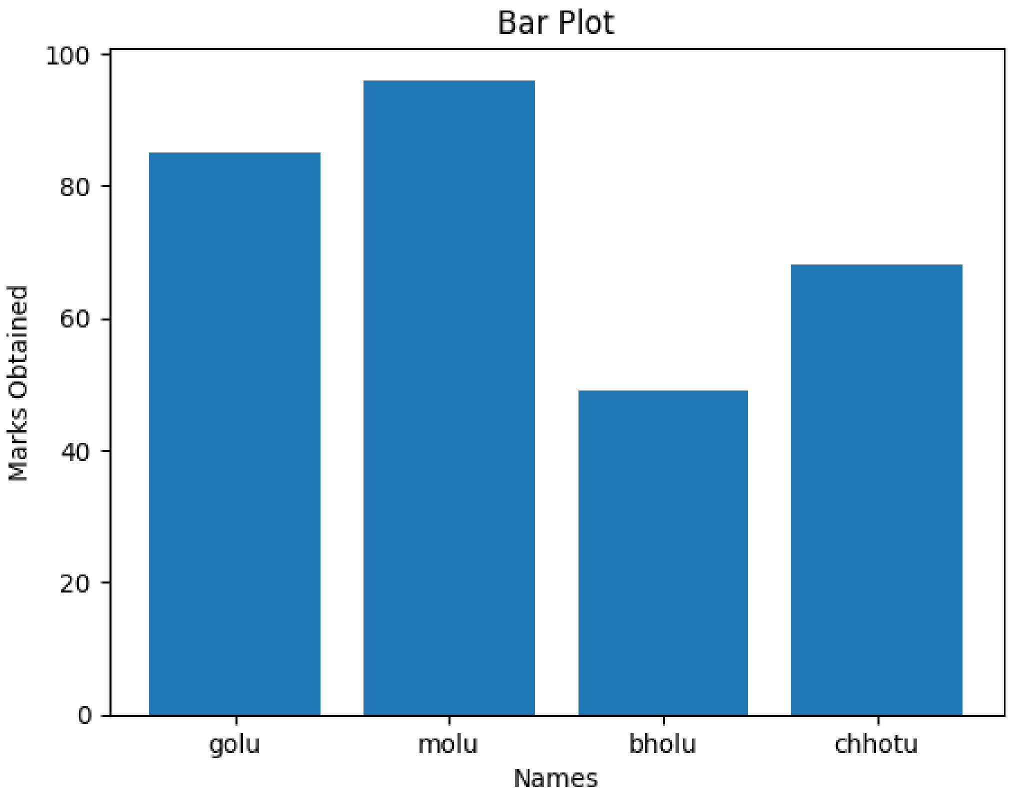
```
In [100… names = list(student.keys())
         names
```

```
Out[100]: ['golu', 'molu', 'bholu', 'chhotu']
```

```
In [101… values = list(student.values())
         values
```

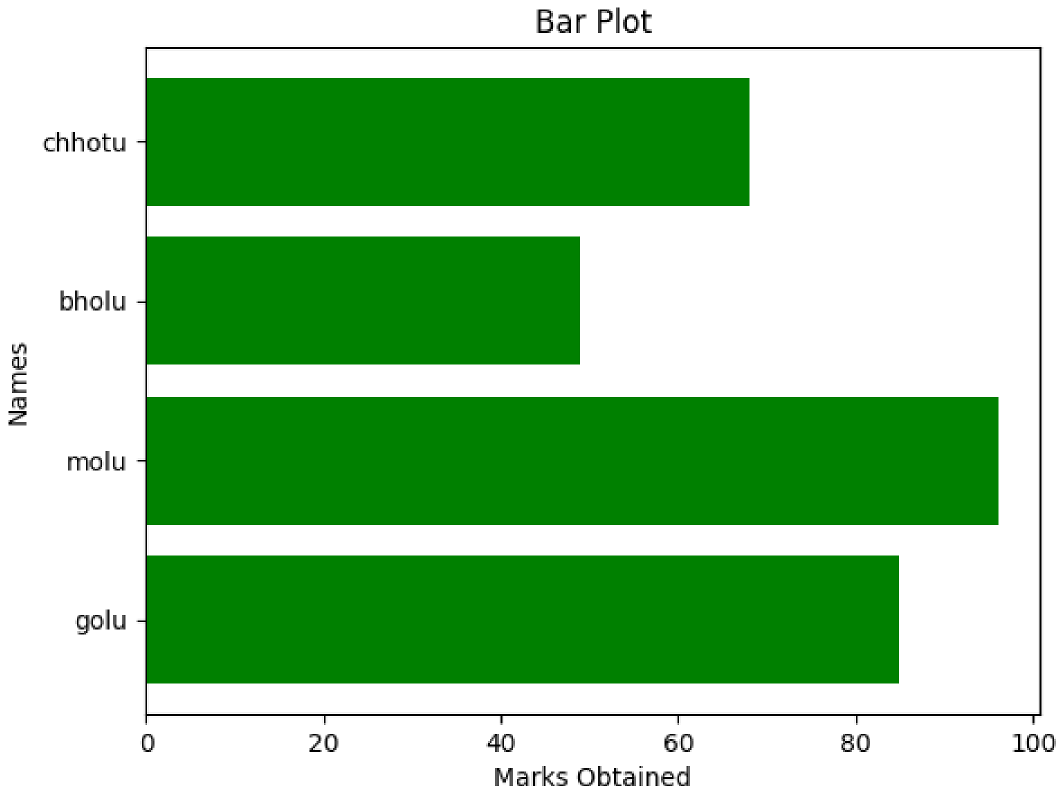
```
Out[101]: [85, 96, 49, 68]
```

```
In [102… plt.bar(names, values)
         plt.title('Bar Plot')
         plt.xlabel('Names')
         plt.ylabel('Marks Obtained')
         plt.show()
```



In [103…

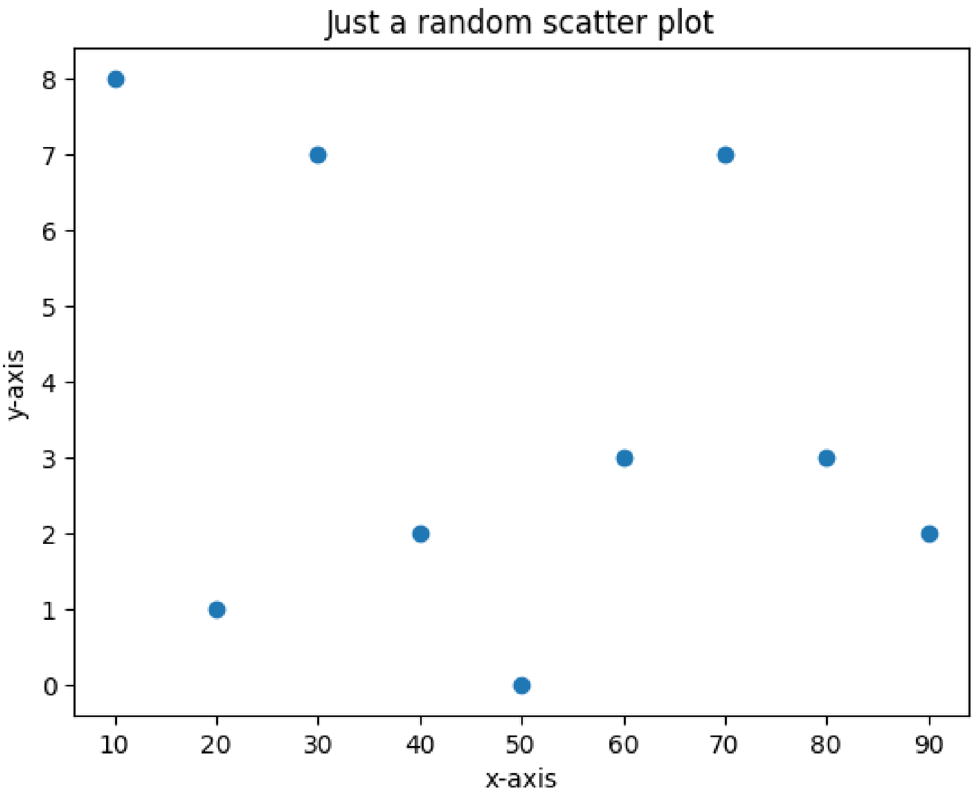
```
#Horizontal barchart
plt.barh(names, values, color = 'g')
plt.title('Bar Plot')
plt.xlabel('Marks Obtained')
plt.ylabel('Names')
plt.show()
```



In [104…

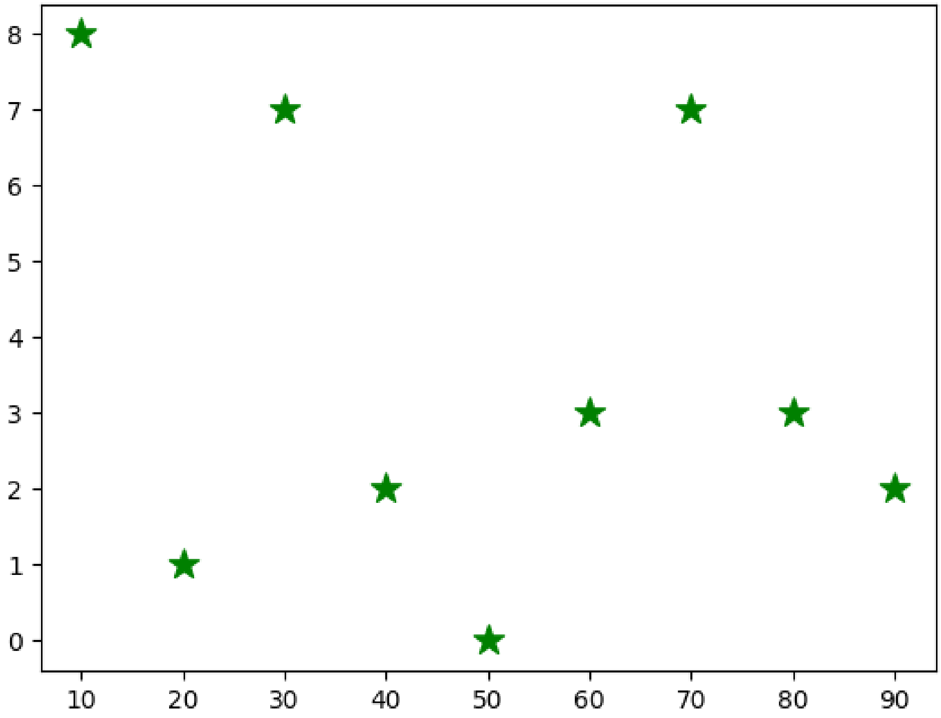
```
#creating a basic scatter-plot
x = np.arange(10,100,10)
a = [8,1,7,2,0,3,7,3,2]
```

```
plt.scatter(x,a)
plt.title('Just a random scatter plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



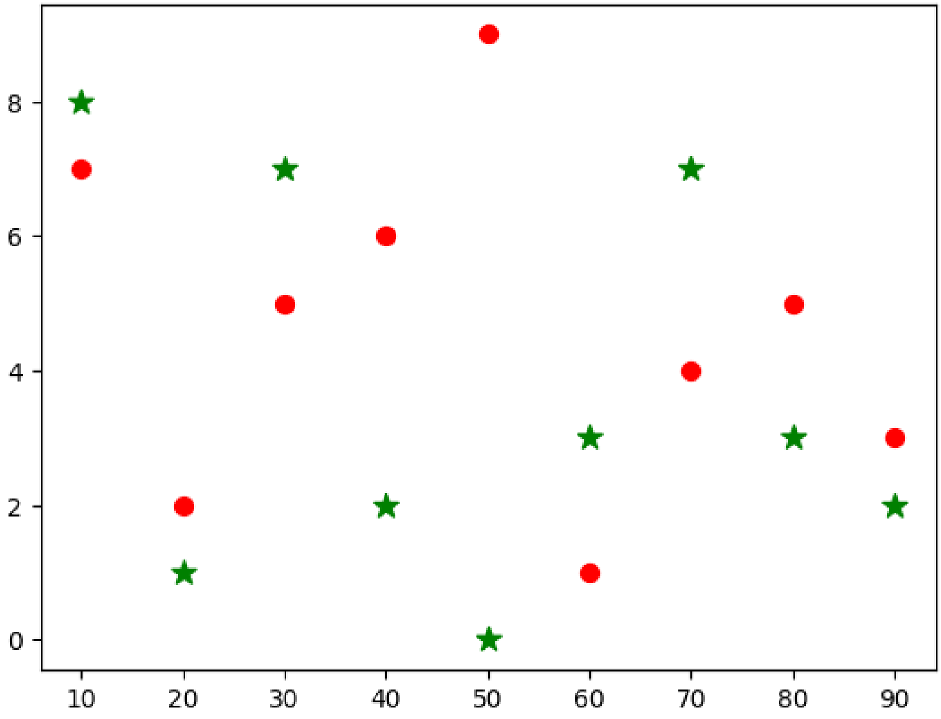
In [105…

```
plt.scatter(x,a,marker = "*", c = 'g', s = 150)
plt.show()
```



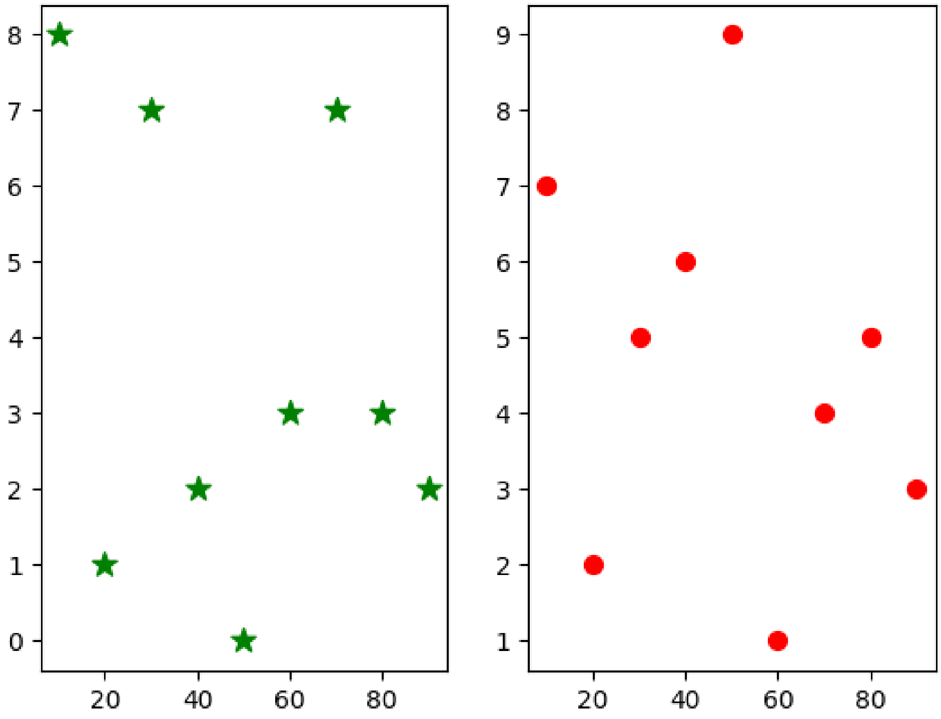
In [106…

```
#Adding two markers in the same plot
x = [10, 20, 30, 40, 50, 60, 70, 80, 90]
a = [8,1,7,2,0,3,7,3,2]
b = [7,2,5,6,9,1,4,5,3]
plt.scatter(x,a,marker = '*', c = 'g', s = 100)
plt.scatter(x,b,marker = '.', c = 'r', s = 200)
plt.show()
```



In [107…

```
plt.subplot(1,2,1)
plt.scatter(x,a,marker = "*", c = 'g', s = 100)

plt.subplot(1,2,2)
plt.scatter(x,b,marker = ".", c = 'r', s = 200)
plt.show()
```

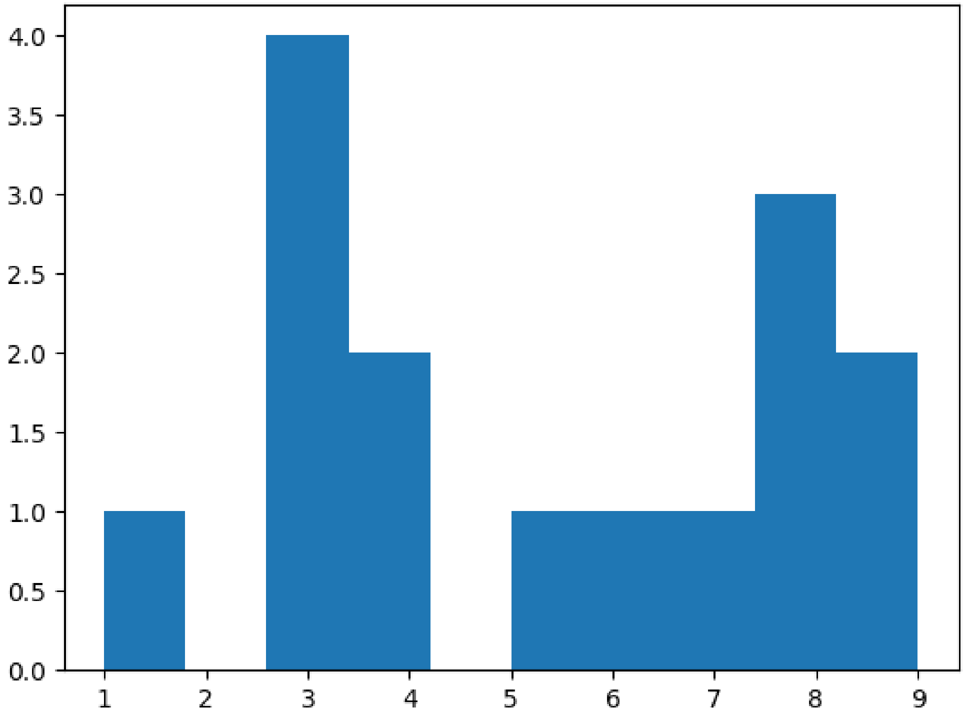


In [108…

```
#Histogram (it gives the frequency of the data how often it occurred)
```

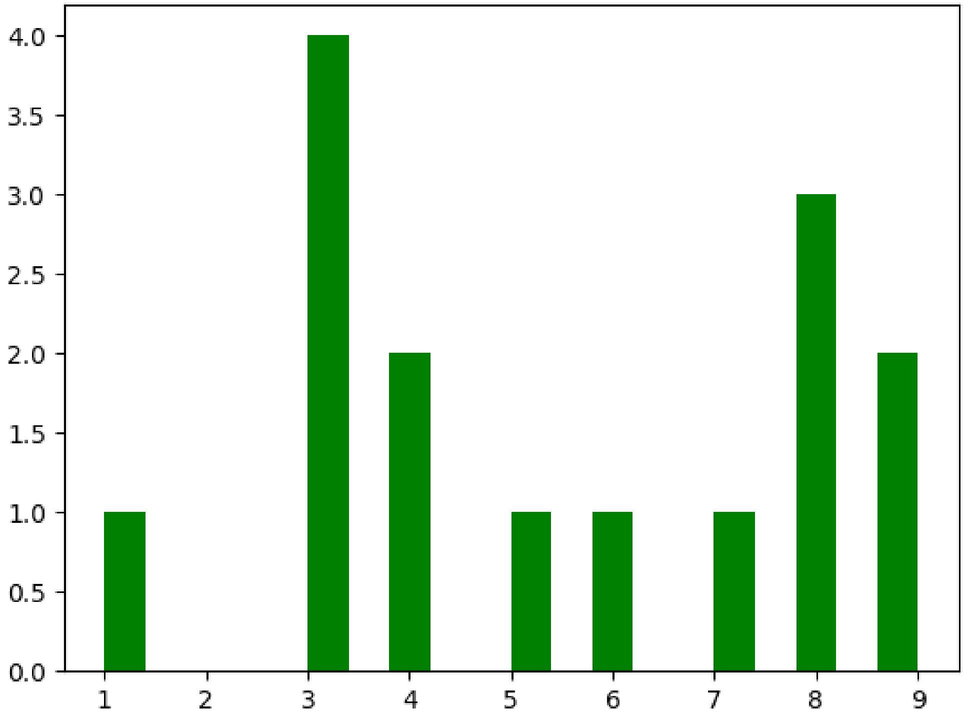
In [109…

```
data = [1,3,3,3,3,9,9,5,4,4,8,8,8,6,7]
plt.hist(data)
plt.show()
```



In [110…

```
#Changing Aesthetics
plt.hist(data, color = 'g', bins = 20)
plt.show()
```



In [111…

```
#working with a histogram in a dataset
```

In [112…

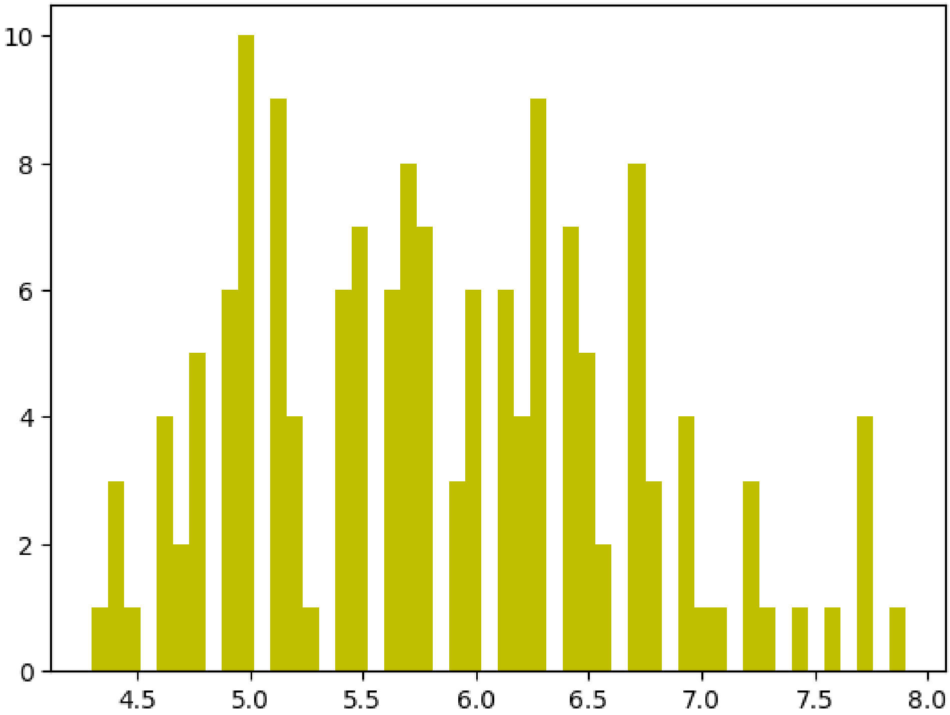
```
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[112]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [113…

```
plt.hist(iris['150'], bins = 50, color = 'y')
plt.show()
```

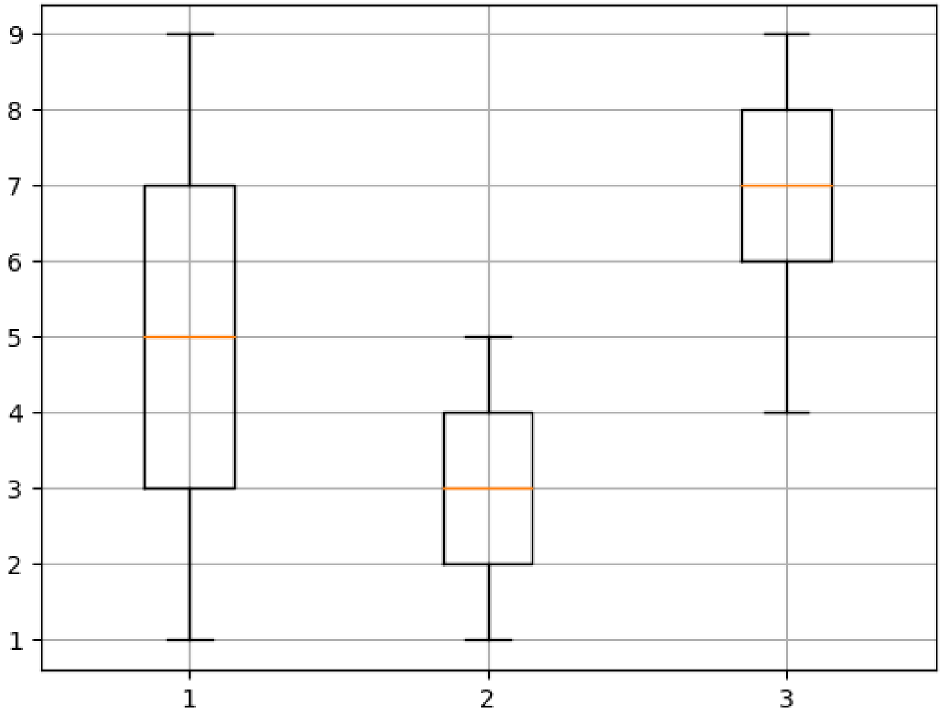


In [114…

```
#Creating data
one = [1,2,3,4,5,6,7,8,9]
two = [1,2,3,4,5,4,3,2,1]
three = [6,7,8,9,8,7,6,5,4]

data = list([one, two , three])
```

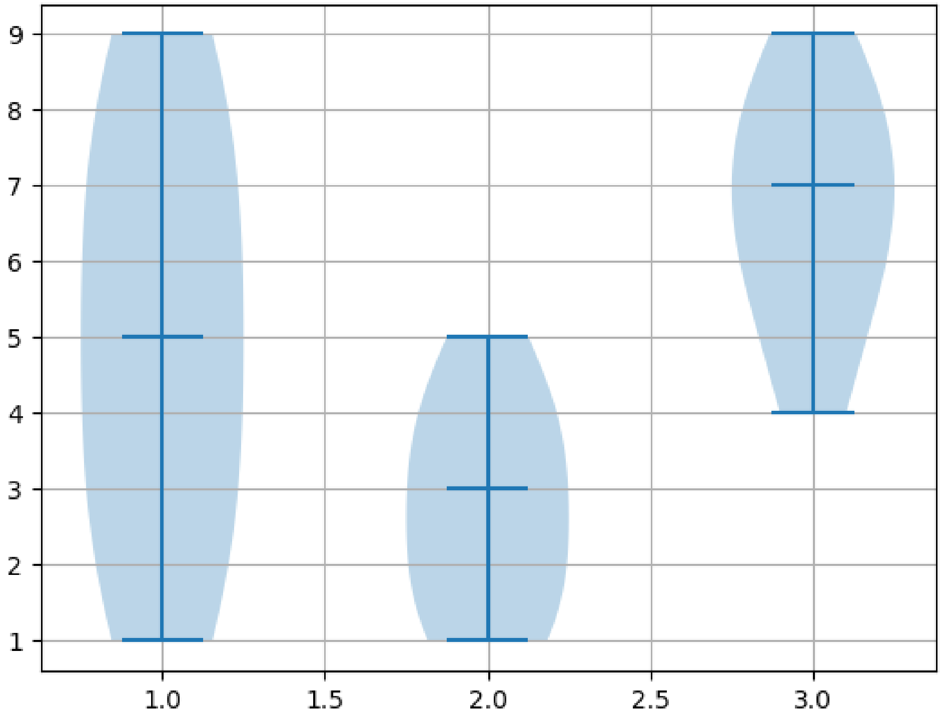
In [115…

```
plt.boxplot(data)
plt.grid(True)
plt.show()
```



In [116…

```
plt.violinplot(data, showmedians = True)
plt.grid(True)
plt.show()
```

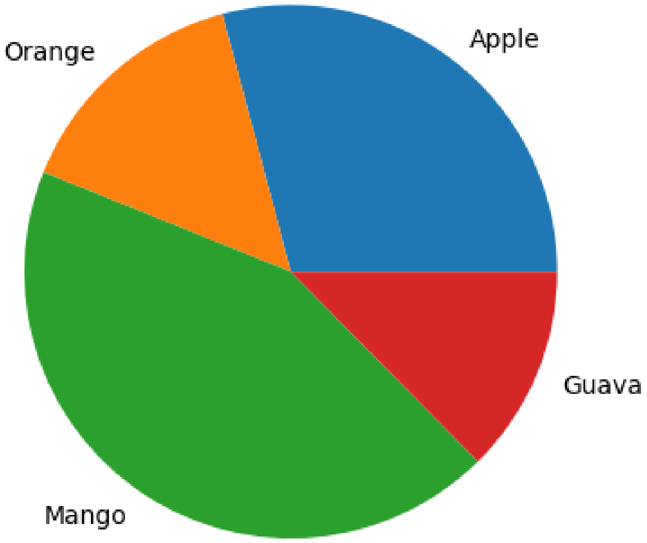


In [117…

```
#Pie-Chart
fruit = ['Apple', 'Orange', 'Mango', 'Guava']
quantity = [67,34,100,29]
```

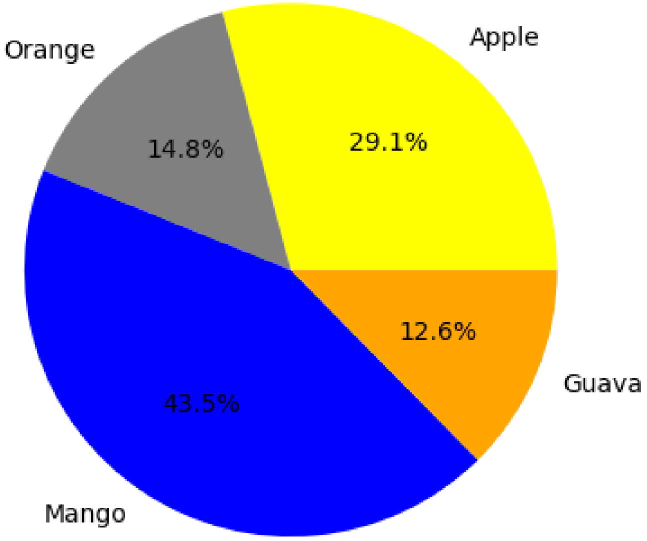
In [118…

```
plt.pie(quantity, labels = fruit)
plt.show()
```



In [119…

```
#Changing Aesthetics
plt.pie(quantity, labels = fruit, autopct = '%0.1f%%', colors = ['yellow', 'grey', 'blue',
plt.show()
```



In [120…

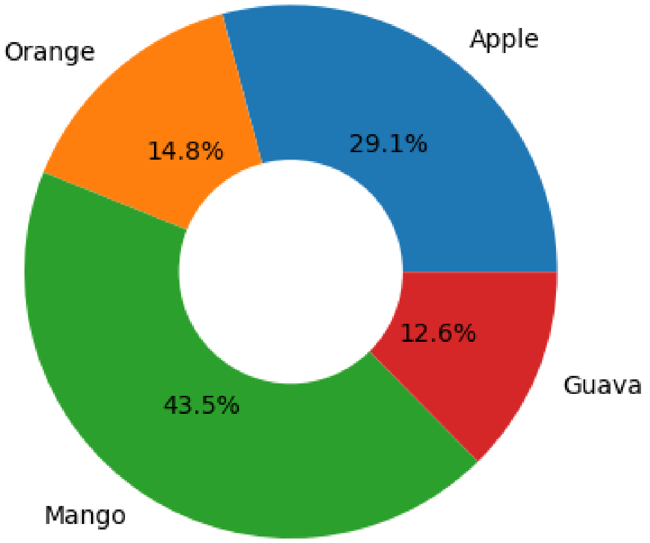
```
#DoughNut-Chart
```

In [121…

```
fruit = ['Apple', 'Orange', 'Mango', 'Guava']
quanitity = [67, 34, 100, 29]
```

In [122…

```
plt.pie(quantity,labels = fruit,autopct = '%0.1f%%' ,radius = 1)
plt.pie([1], colors =['w'], radius = 0.42)
plt.show()
```



-----Seaborn-----

In [123…

```
import seaborn as sns
from matplotlib import pyplot as plt
```

In [124…

```
fmri = sns.load_dataset("fmri")
fmri.head()
```

Out[124]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

In [125…

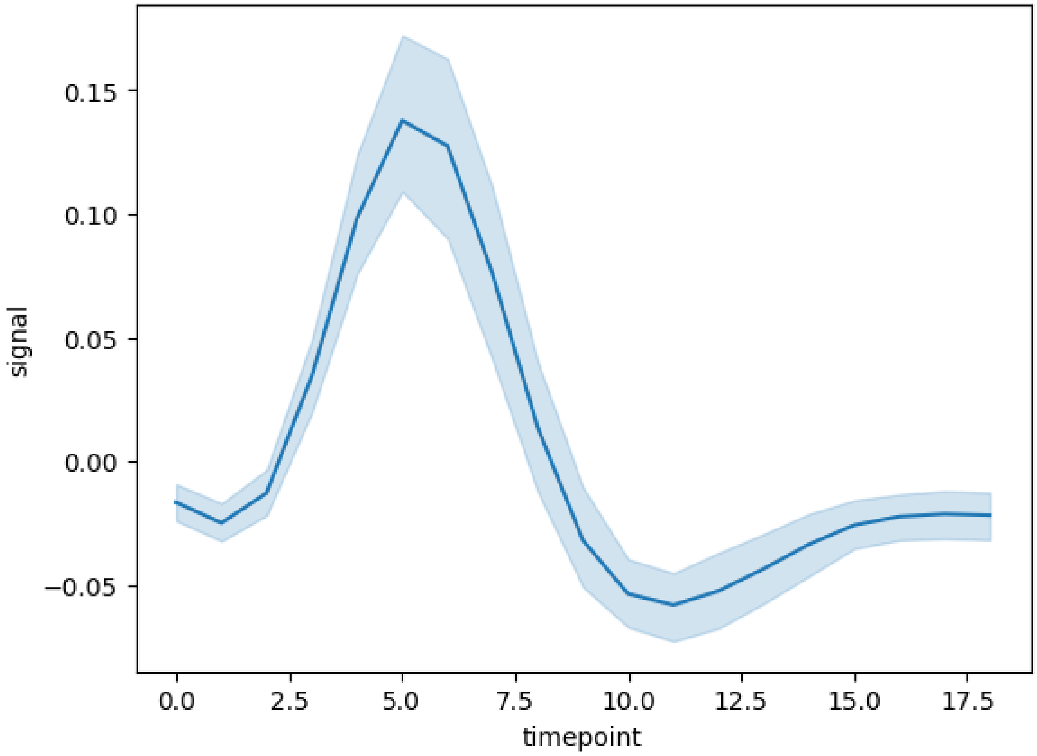
```
fmri.shape
```

Out[125]:

```
(1064, 5)
```

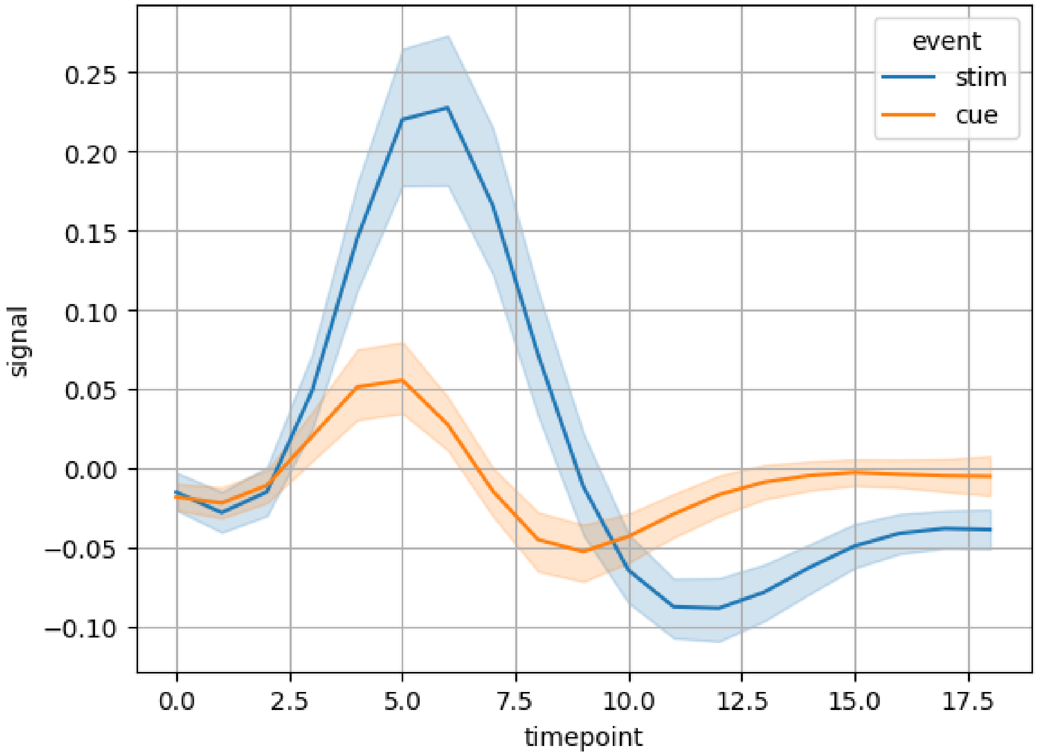
In [126…

```
sns.lineplot(x = 'timepoint', y = 'signal', data = fmri)
plt.show()
```



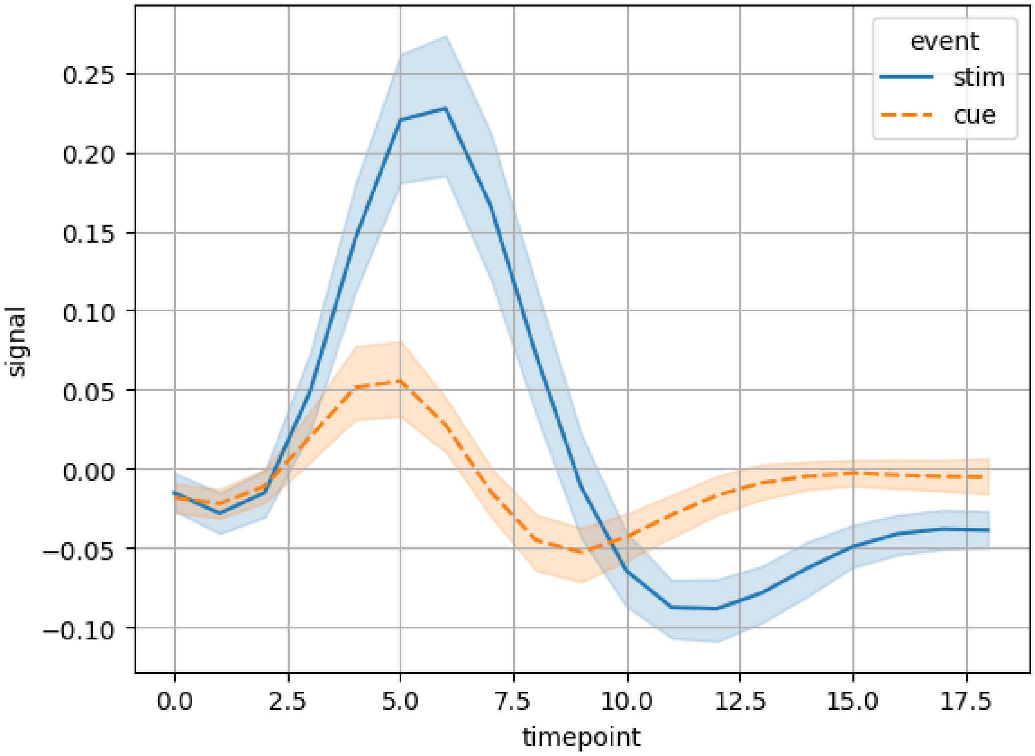
In [127…

```
sns.lineplot(x = 'timepoint', y = 'signal', data = fmri, hue = 'event')
plt.grid(True)
plt.show()
```



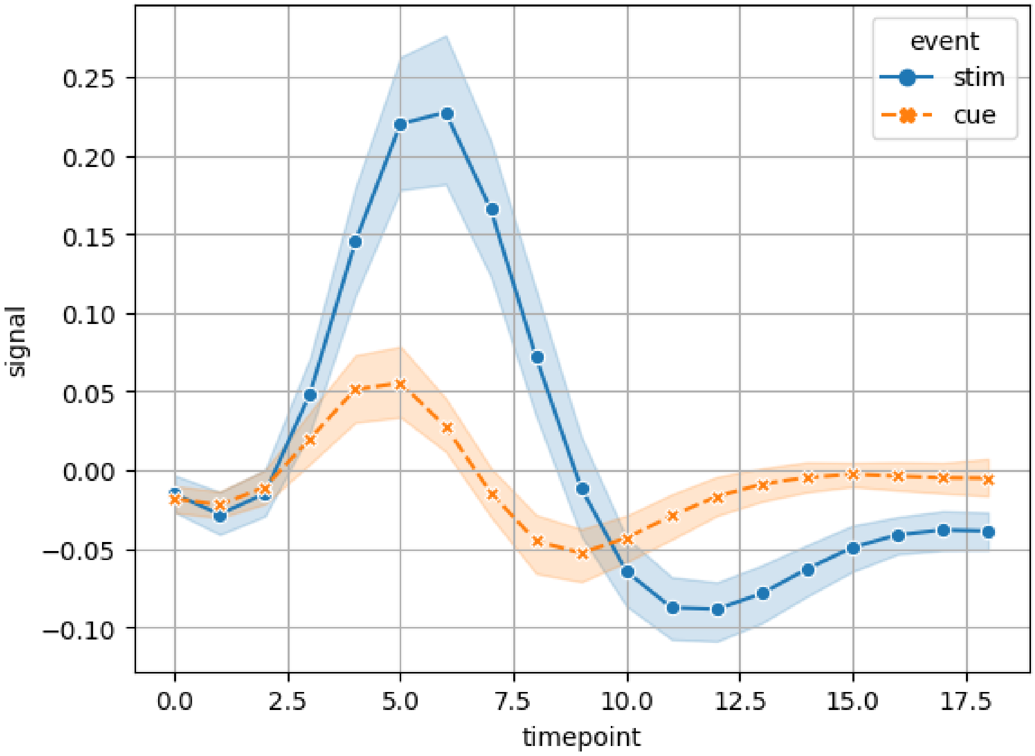
In [128…

```
sns.lineplot(x = 'timepoint', y = 'signal', data = fmri, hue = 'event', style = 'event')
plt.grid(True)
plt.show()
```



In [129…

```
sns.lineplot(x = 'timepoint', y = 'signal', data = fmri, hue = 'event', style = 'event', ma
plt.grid(True)
plt.show()
```



In [130…

```
#SeaBorn Bar Plot
```

In [131…

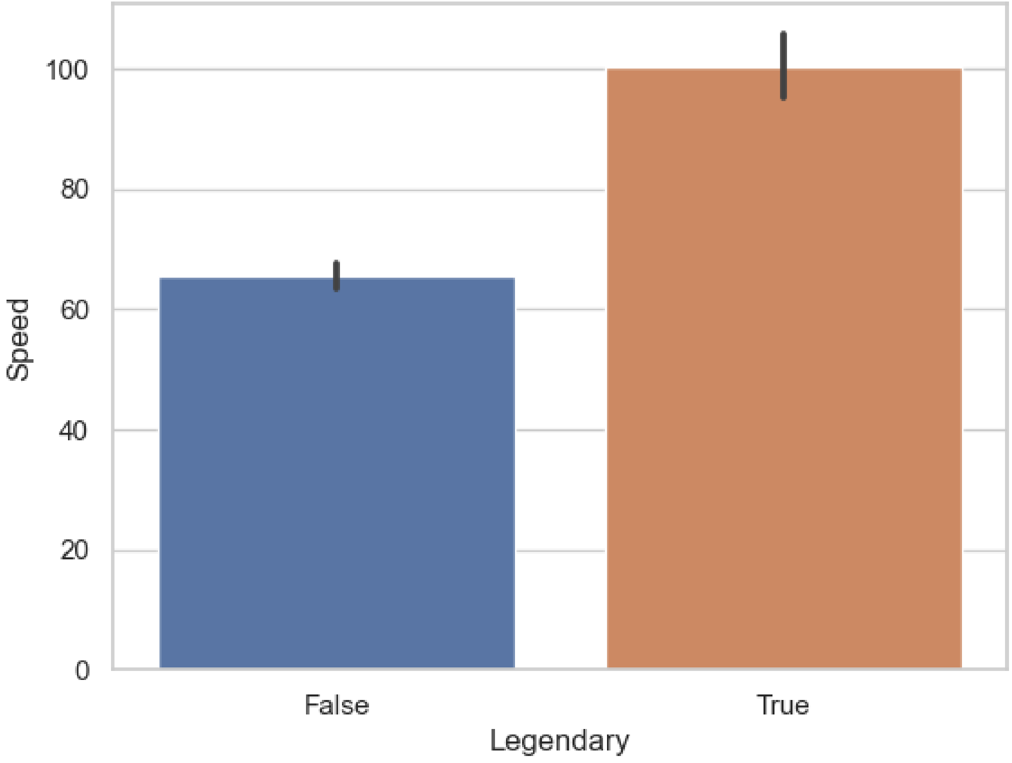
```
import pandas as pd
sns.set(style = 'whitegrid')
pokemon = pd.read_csv('pokemon.csv')
pokemon.head()
```

Out[131]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	

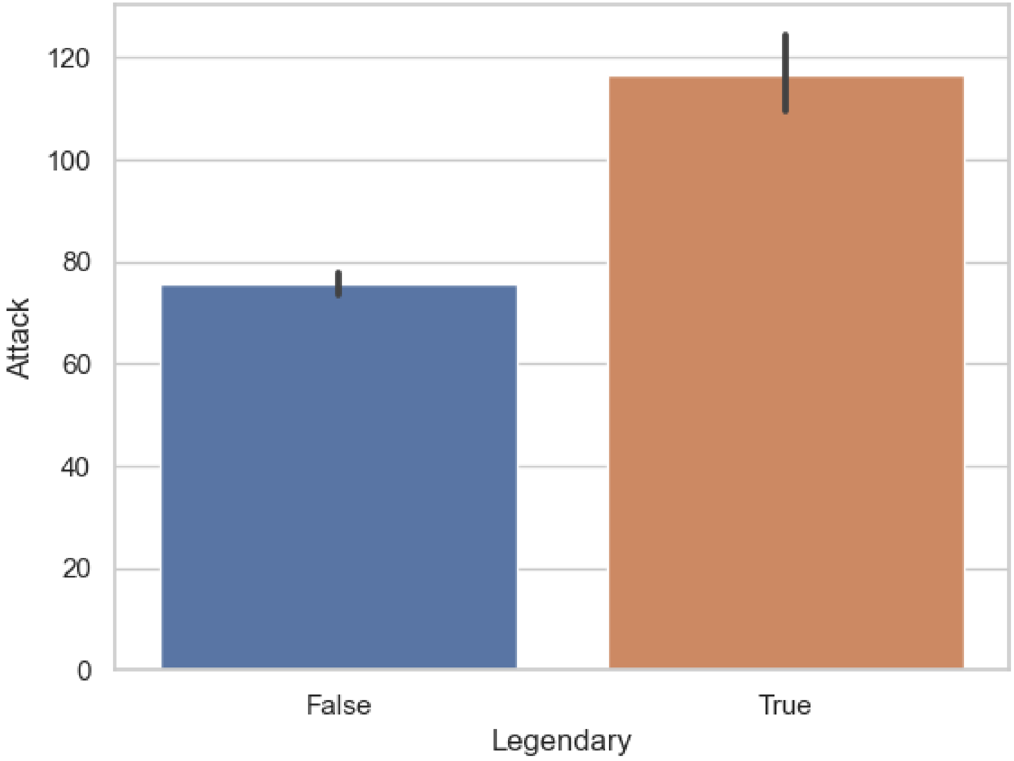
In [132…

```
sns.barplot(x = 'Legendary', y = 'Speed', data = pokemon)
plt.show()
#Note: The black line showing above the bars are specifying the maximum value of that parti
```



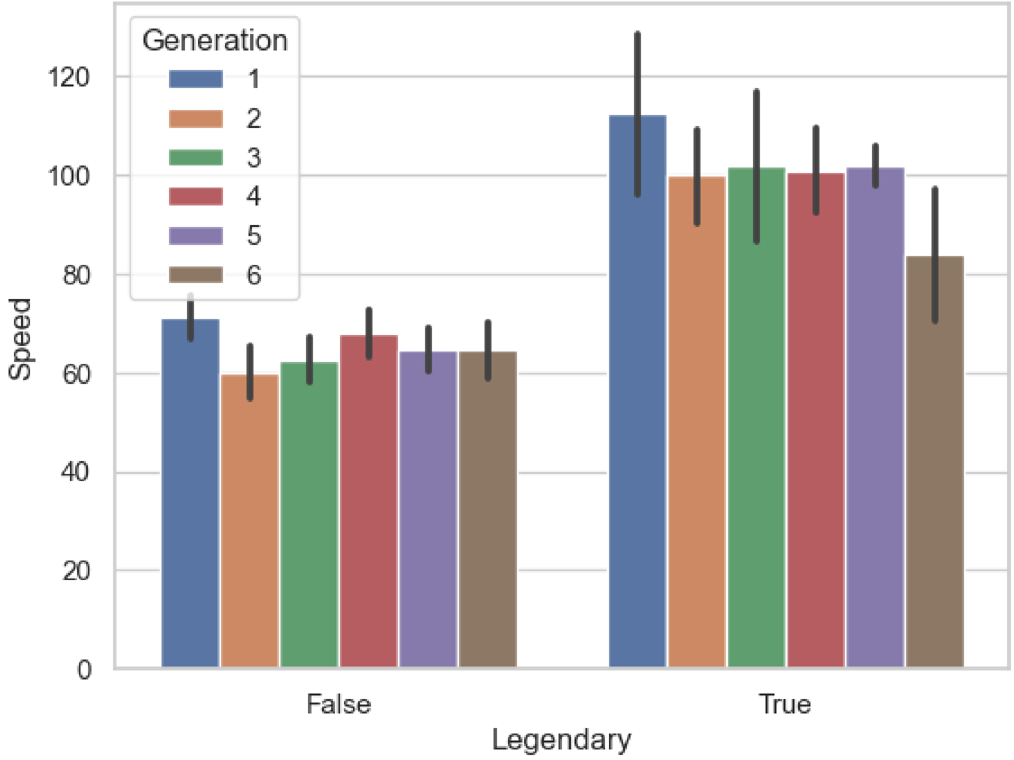
In [133…

```
sns.barplot(x = 'Legendary', y = 'Attack', data = pokemon)
plt.show()
```



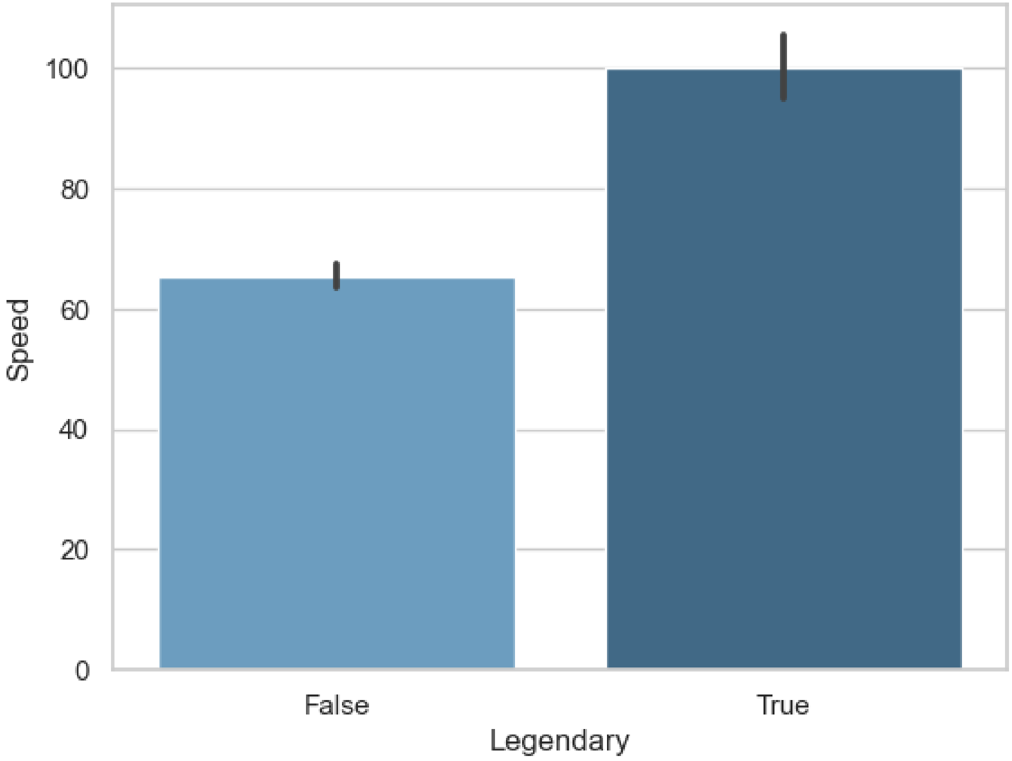
In [134…

```
sns.barplot(x = 'Legendary', y = 'Speed', hue = 'Generation', data = pokemon)
plt.show()
```



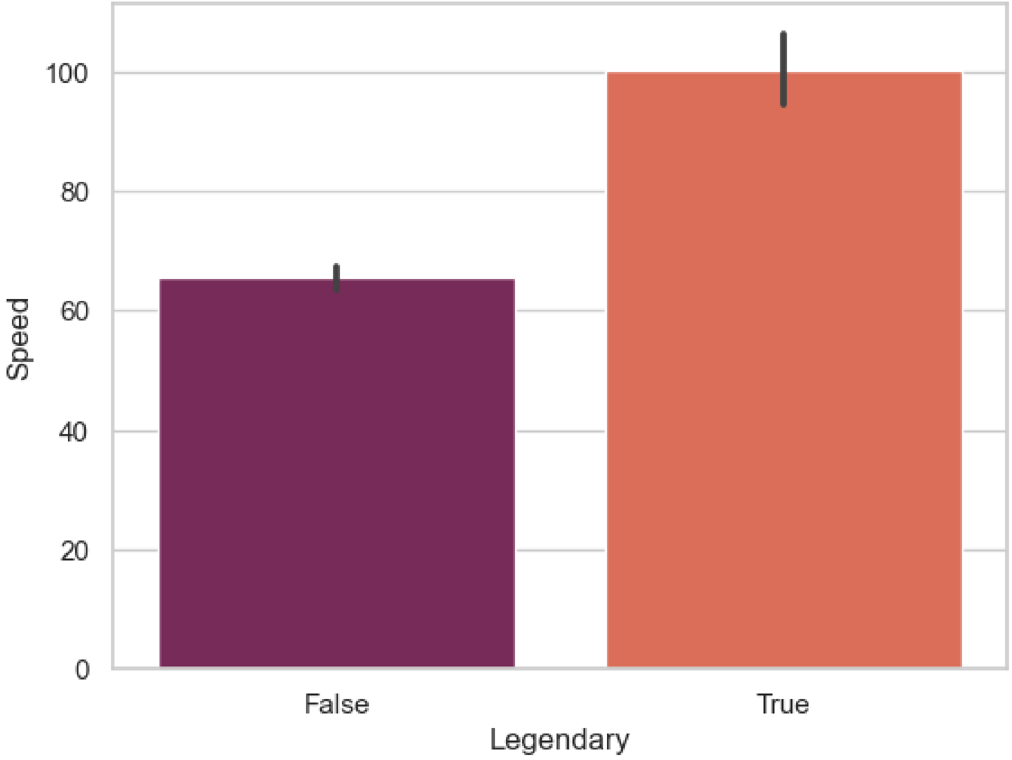
In [135…

```
#seaborn bar plot aesthetics
sns.barplot(x = 'Legendary', y = 'Speed', data = pokemon, palette = 'Blues_d')
plt.show()
```



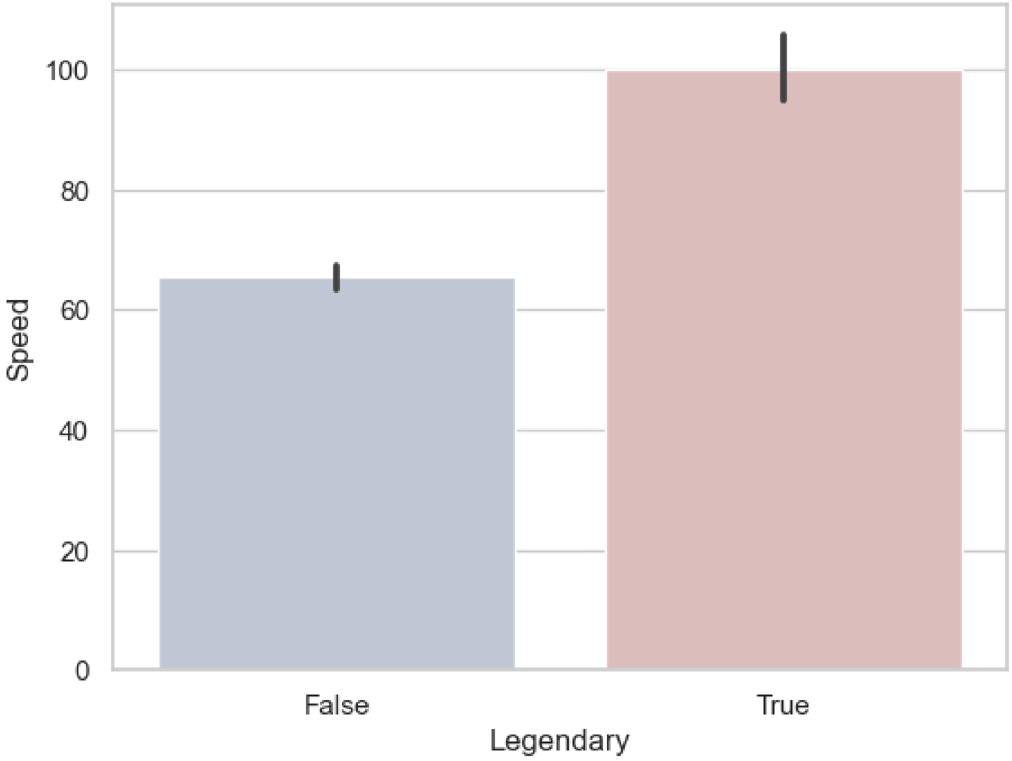
In [136…

```
sns.barplot(x = 'Legendary', y = 'Speed', data = pokemon, palette = 'rocket')
plt.show()
```



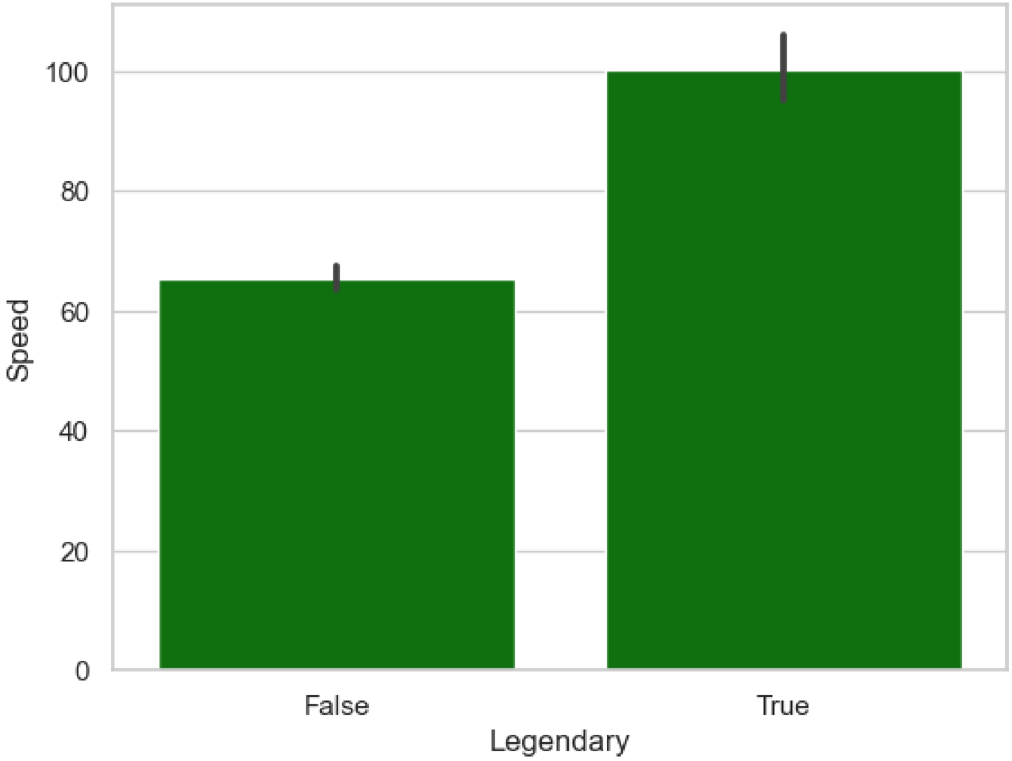
In [137…

```
sns.barplot(x = 'Legendary', y = 'Speed', data = pokemon, palette = 'vlag')
plt.show()
```



In [138…

```
# Note: We use color command to set the same color to both the bars
sns.barplot(x = 'Legendary', y = 'Speed', data = pokemon, color = "green")
plt.show()
```



In [139…

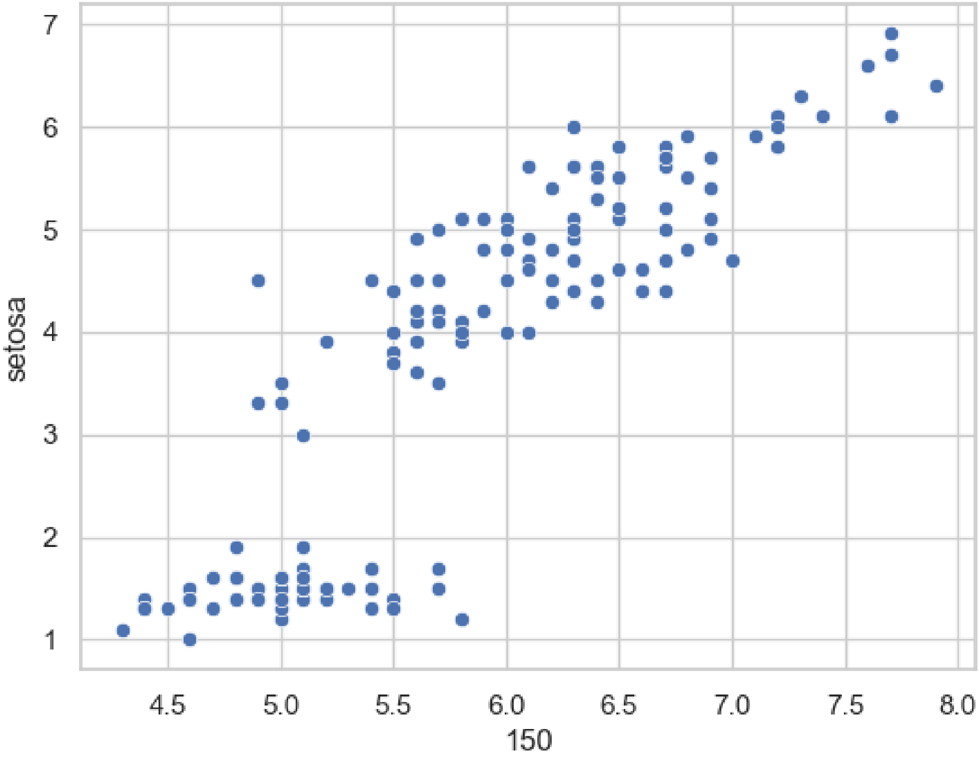
```
#SeaBorn Scatterplot
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[139]:

	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

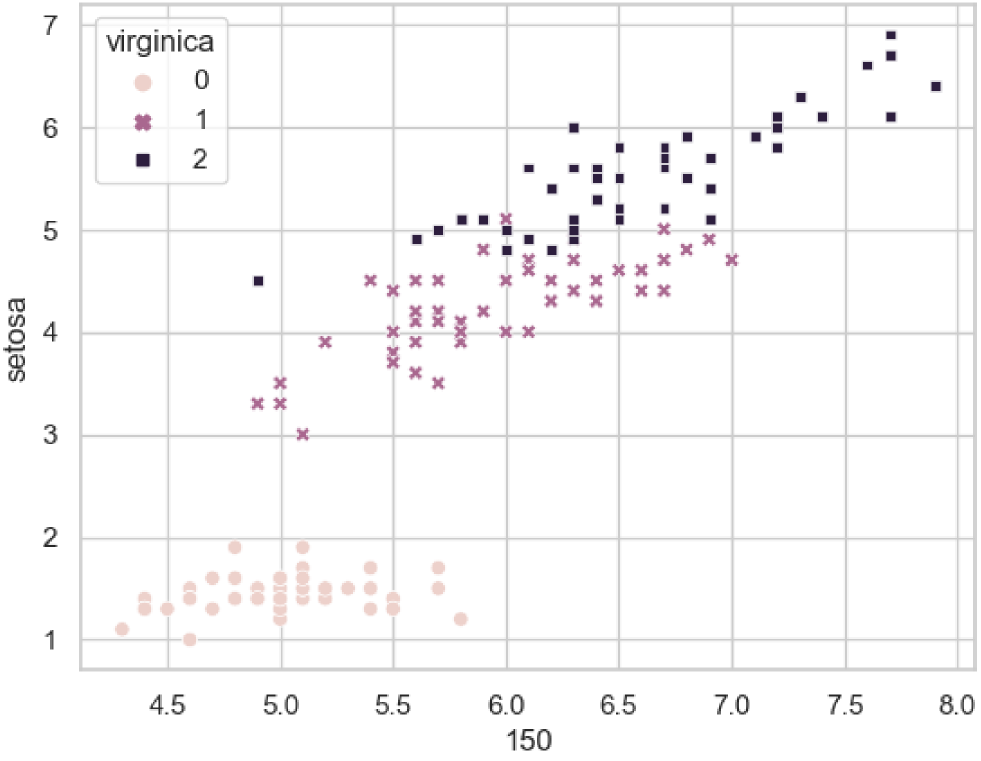
In [140…

```
sns.scatterplot(x = '150', y = 'setosa', data = iris)
plt.show()
```



In [141…

```
sns.scatterplot(x = '150', y = 'setosa', data = iris, hue = 'virginica', style = 'virginica
plt.show()
```



In [142…

```
#SeaBorn Histogram/Distplot

diamonds = pd.read_csv('diamonds.csv')
```

In [143…

```
diamonds.head()
```

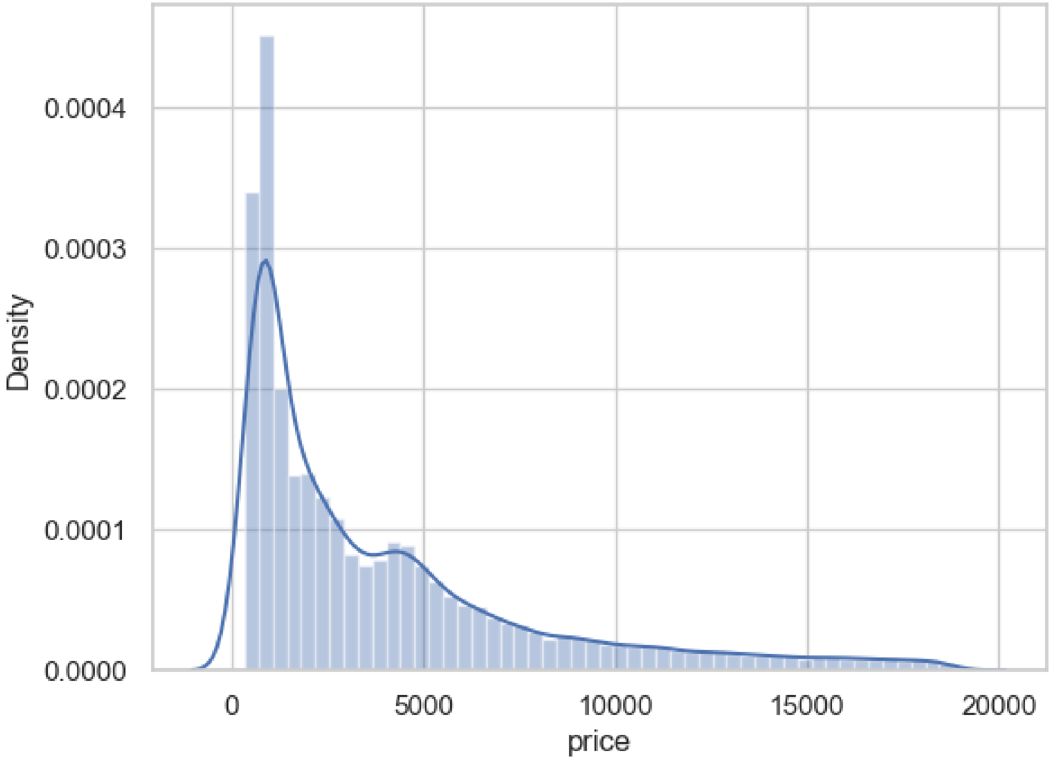
Out[143]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [144…

```
sns.distplot(diamonds['price'])
plt.show()
```

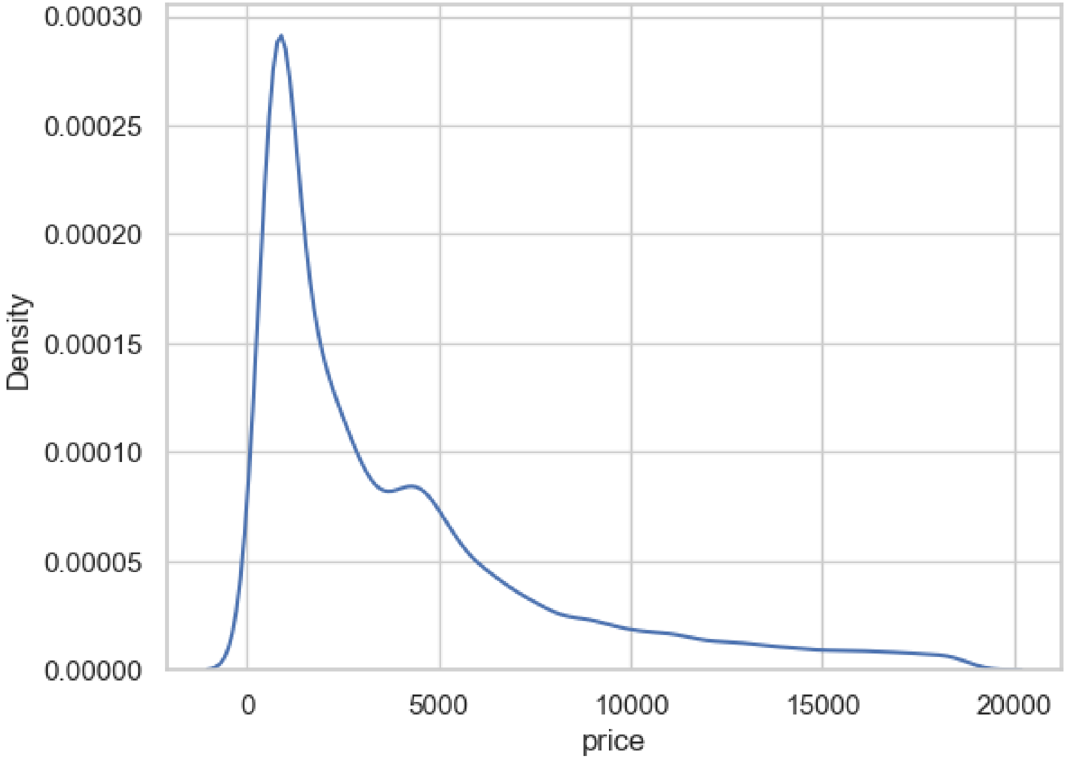
```
C:\Users\Nitish\AppData\Local\Temp\ipykernel_1700\893117346.py:1: UserWarning:

`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

  sns.distplot(diamonds['price'])
```



In [145…

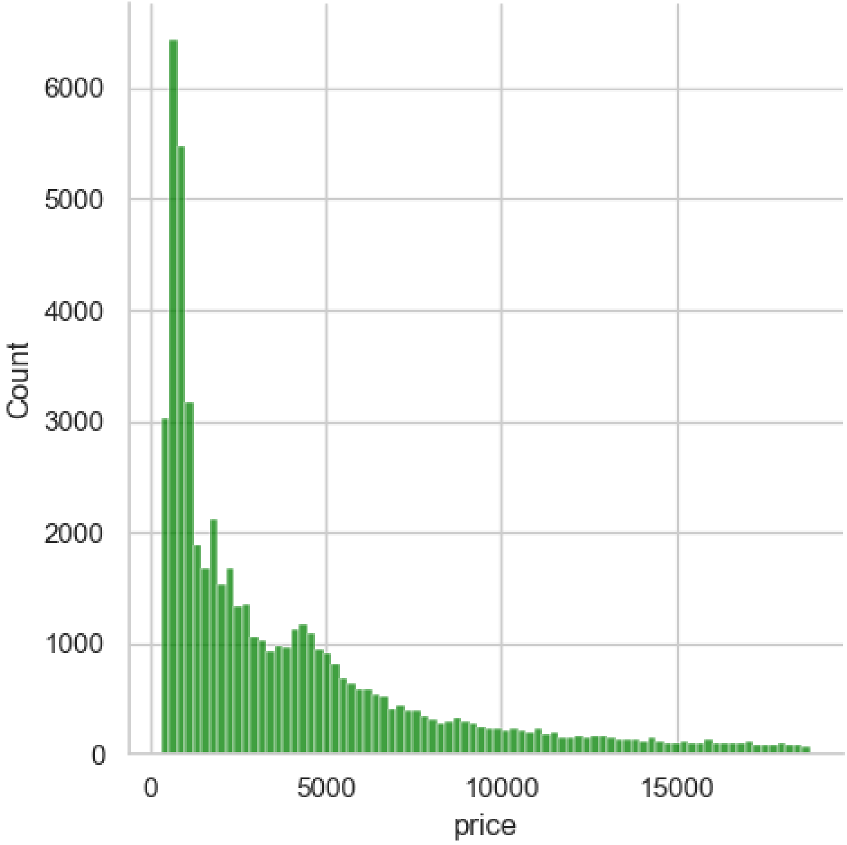
```
sns.distplot(diamonds['price'], hist = False)
plt.show()
```

```
C:\Users\Nitish\AppData\Local\Temp\ipykernel_1700\1567278561.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 `kdeplot` (an axes-level function for kernel density plots).

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

  sns.distplot(diamonds['price'], hist = False)
```



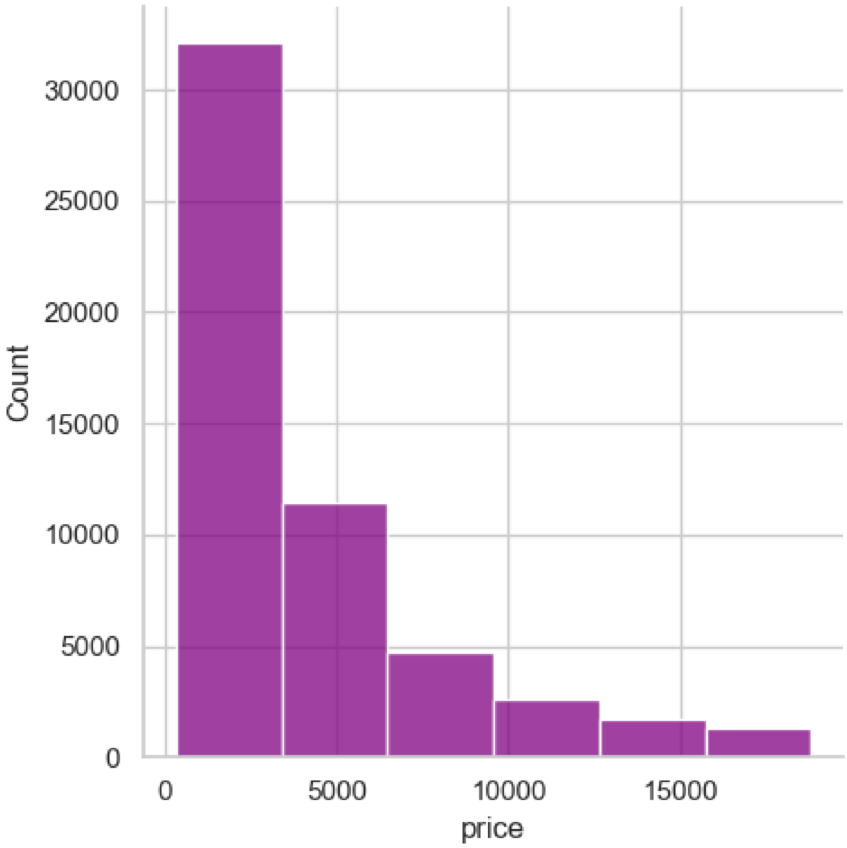
In [146…

```
sns.displot(diamonds['price'], color = "green")
plt.show()
```



In [147…

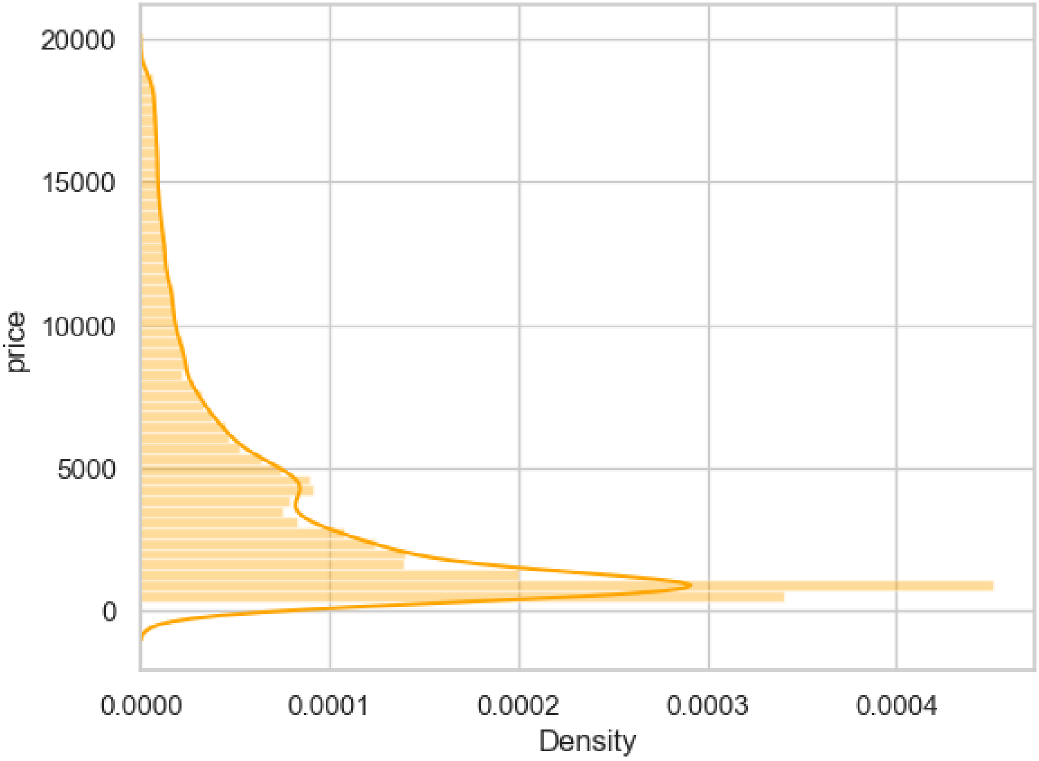
```
 sns.displot(diamonds['price'], color = "purple", bins = 6, kde = False)
plt.show()
```



In [148…

```
sns.distplot(diamonds['price'], color = 'orange', vertical = True)
plt.show()
```

```
C:\Users\Nitish\AppData\Local\Temp\ipykernel_1700\455992419.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

  sns.distplot(diamonds['price'], color = 'orange', vertical = True)
```

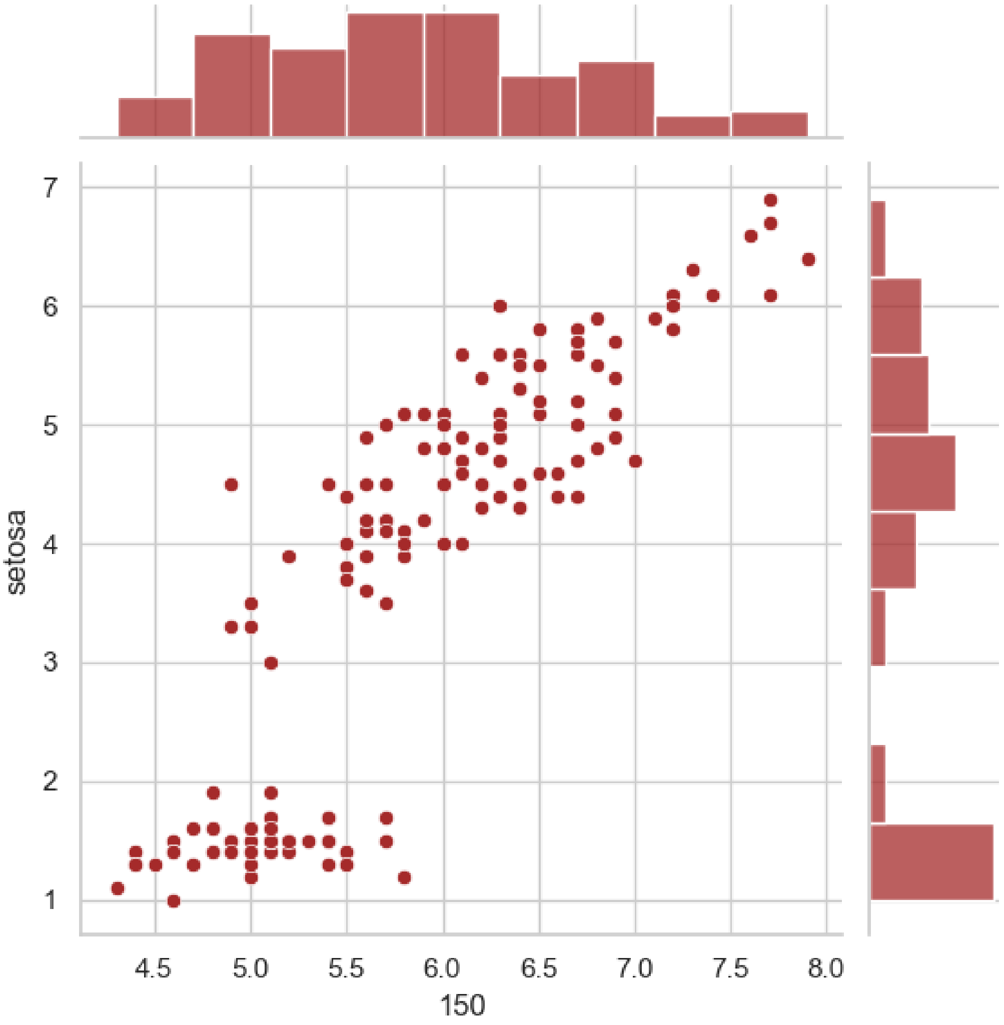


In [149…
```
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[149]:

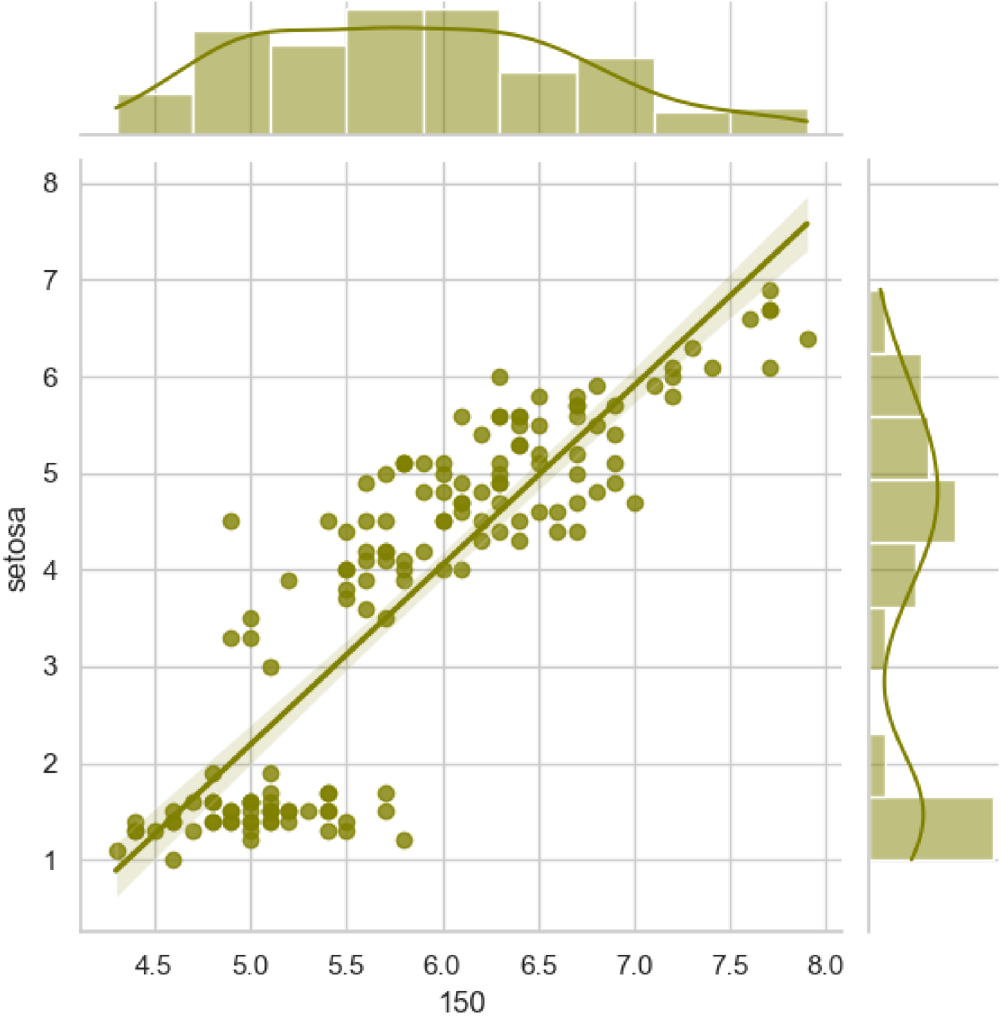
	150	4	setosa	versicolor	virginica
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [150…
```
sns.jointplot(x = '150', y = 'setosa', data = iris, color = 'brown')
plt.show()
```



In [151…

```
sns.jointplot(x = '150', y = 'setosa', data = iris, color = 'olive', kind = "reg")
plt.show()
```



In [152…

```
#SeaBorn BoxPlot
churn = pd.read_csv('churn.csv')
churn.head()
```

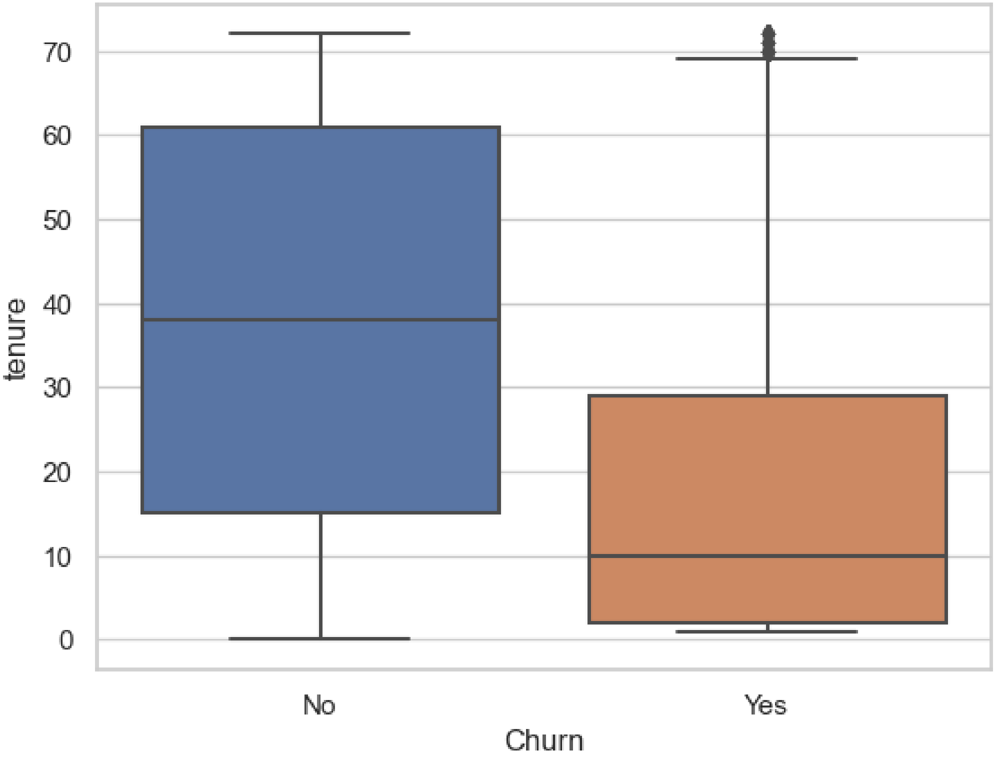
Out[152]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	I
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

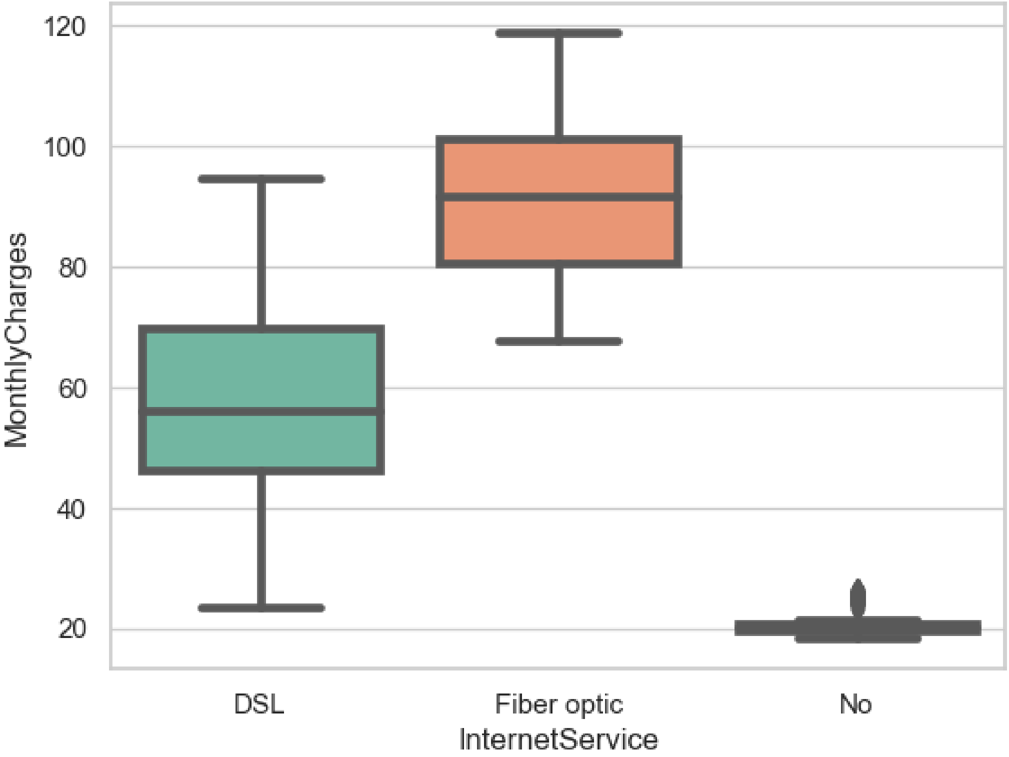
In [153…

```
sns.boxplot(x = 'Churn', y = 'tenure', data = churn)
plt.show()
```



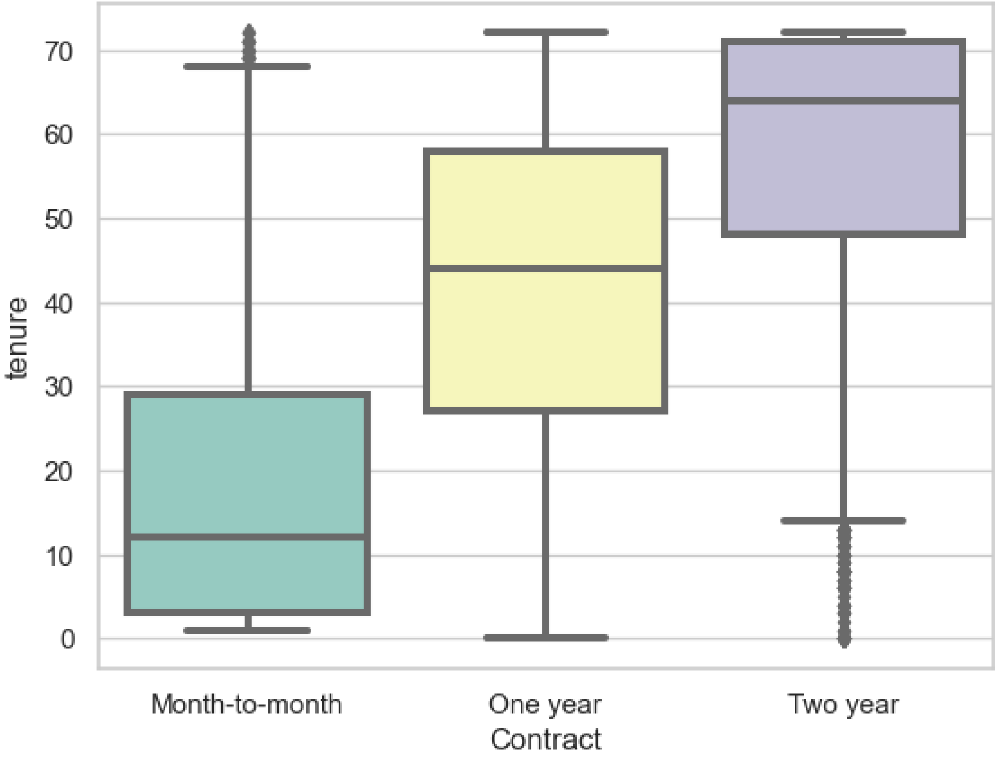
In [154…

```
sns.boxplot(x = 'InternetService', y = 'MonthlyCharges', data = churn, palette = 'Set2', li
plt.show()
```



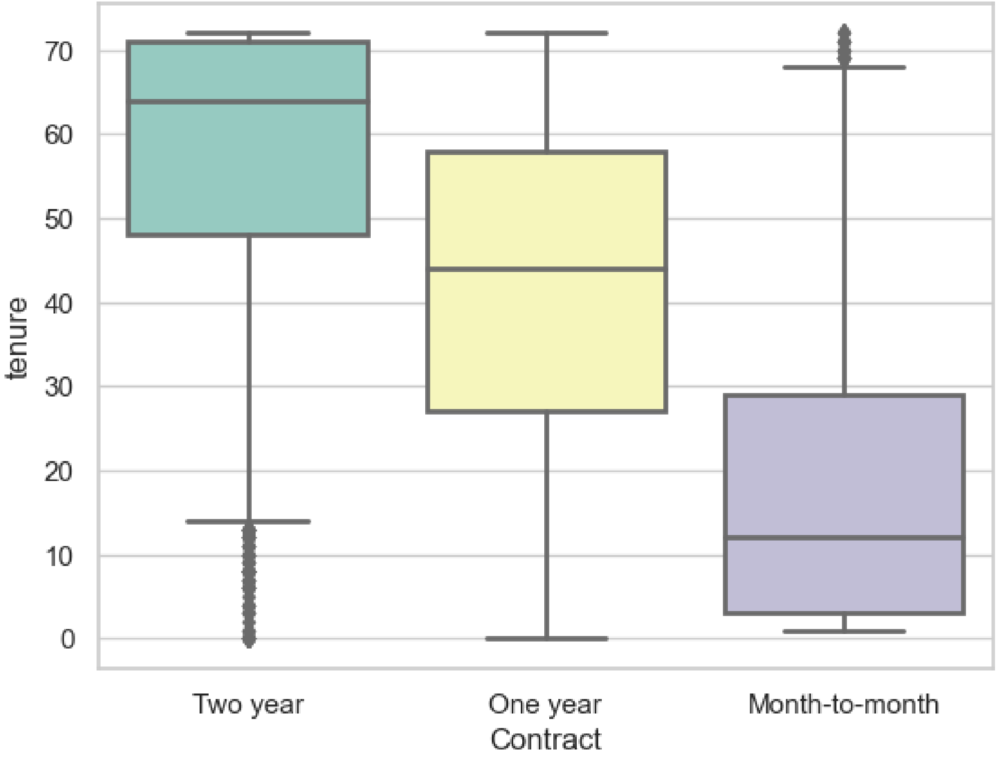
In [155…

```
sns.boxplot(x = 'Contract', y = 'tenure', data = churn, linewidth = 2.9, palette = 'Set3')
plt.show()
```



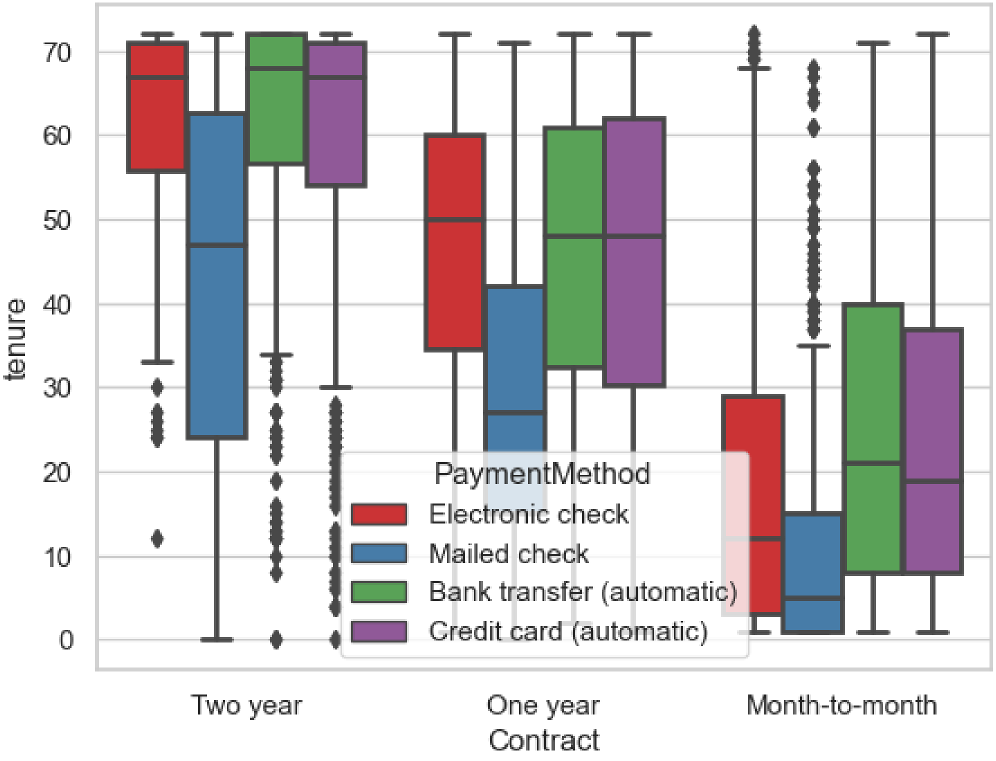
In [156…

```
sns.boxplot(x = 'Contract', y = 'tenure', data = churn, linewidth = 2, palette = 'Set3', or
plt.show()
```



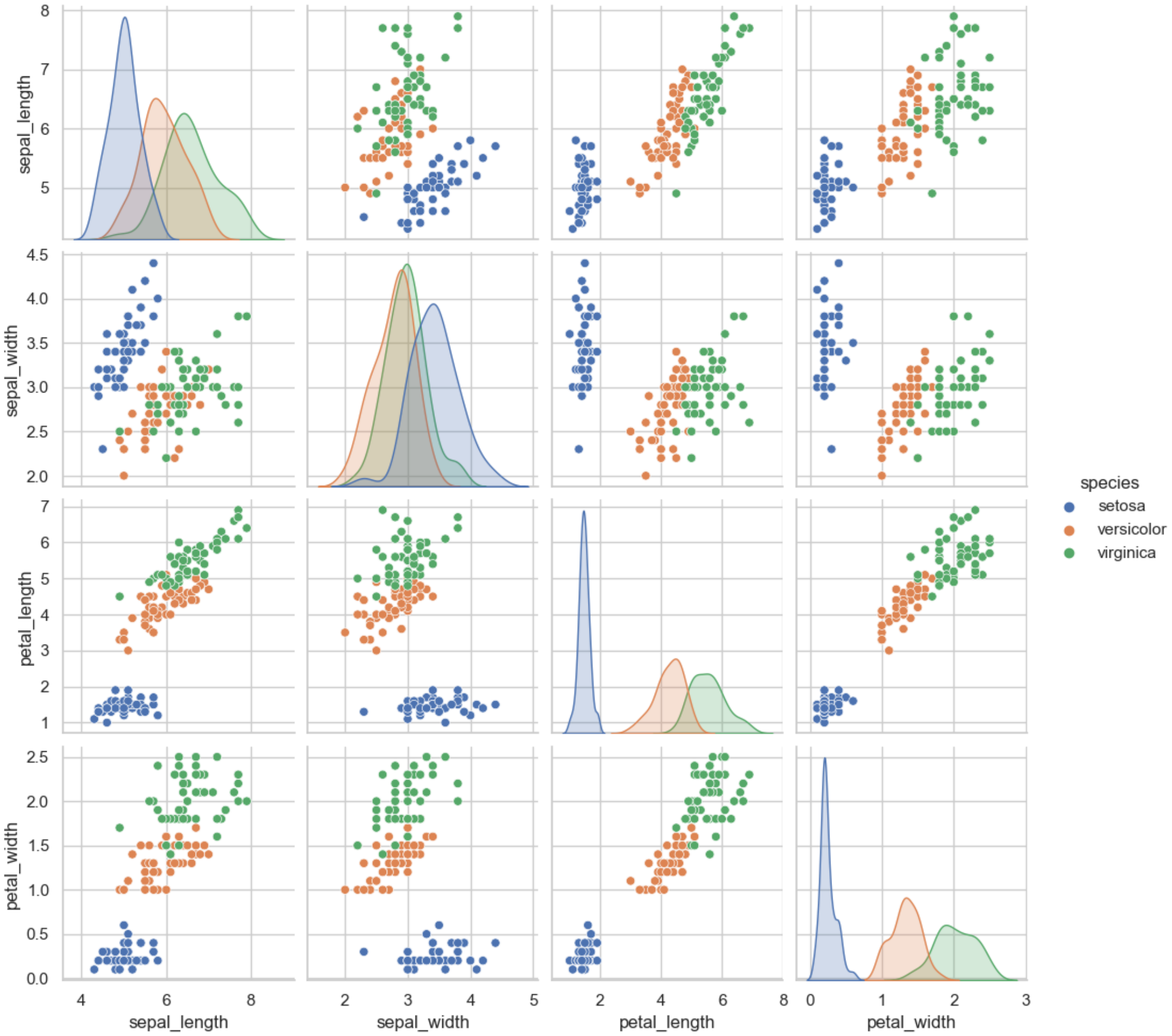
In [157…

```
sns.boxplot(x = 'Contract', y = 'tenure', data = churn, linewidth = 2, palette = 'Set1', or
plt.show()
```



```
In [158…  #SeaBorn Pair Plot
```

```
In [159…  df = sns.load_dataset('iris')
          sns.pairplot(df, hue = 'species')
          plt.show()
```



```
In [160…  #data analysis on ipl matches csv file
          #the implementation of all the libraries we have studied till now
```

```
In [161…  #loading the required libraries
          import pandas as pd
          from matplotlib import pyplot as plt
          import seaborn as sns
```

```
In [162…  ipl = pd.read_csv('matches.csv')
```

```
In [163…  ipl.head()
```

Out[163]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_ap
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	
4	5	2017	Bangalore	2017-04-08	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	

In [164…

```
ipl.shape
```

Out[164]:

```
(756, 18)
```

In [165…

```
#Getting the freq of most MOM awards
ipl['player_of_match'].value_counts()
```

Out[165]:

```
player_of_match
CH Gayle          21
AB de Villiers    20
RG Sharma         17
MS Dhoni          17
DA Warner         17
                  ..
PD Collingwood     1
NV Ojha            1
AC Voges           1
J Theron           1
S Hetmyer          1
Name: count, Length: 226, dtype: int64
```

In [166…

```
ipl['player_of_match'].value_counts()[0:10] #top 10 most times MOM Awards
```

Out[166]:

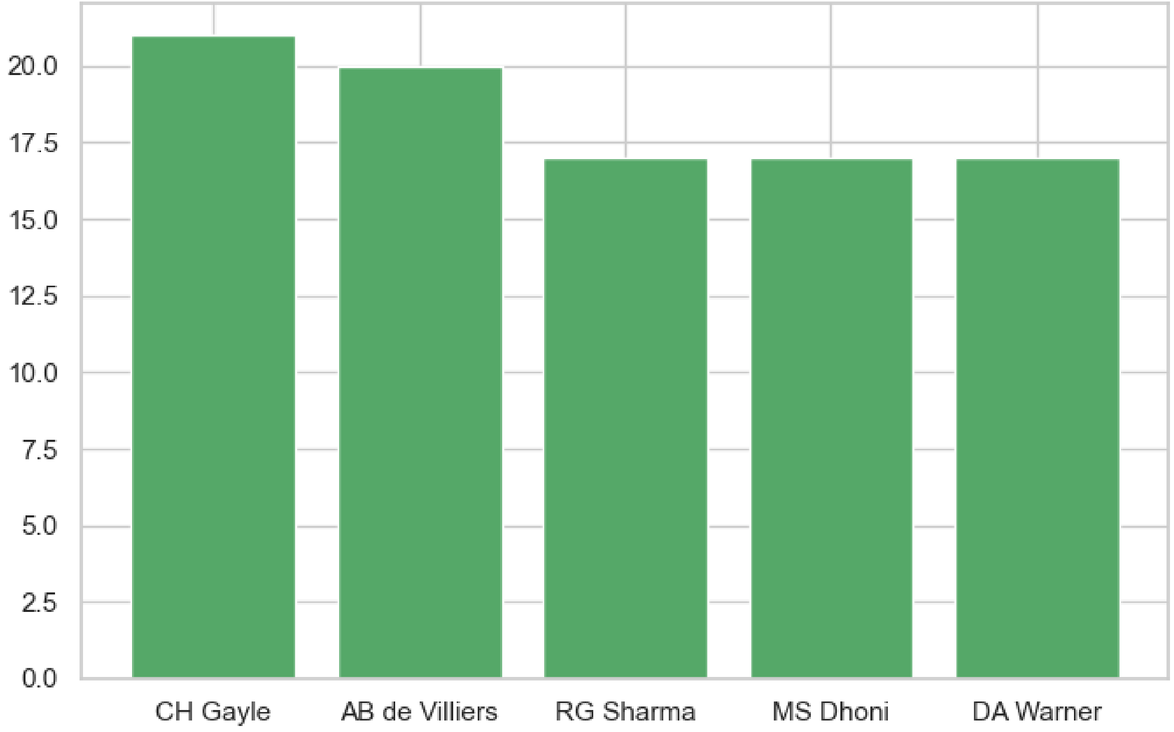
```
player_of_match
CH Gayle          21
AB de Villiers    20
RG Sharma         17
MS Dhoni          17
DA Warner         17
YK Pathan         16
SR Watson         15
SK Raina          14
G Gambhir         13
MEK Hussey        12
Name: count, dtype: int64
```

In [167…

```
#To get only the names of the players
list(ipl['player_of_match'].value_counts()[0:5].keys())
```

Out[167]:
```
['CH Gayle', 'AB de Villiers', 'RG Sharma', 'MS Dhoni', 'DA Warner']
```

In [168…
```
plt.figure(figsize = (8,5))
plt.bar(list(ipl['player_of_match'].value_counts()[0:5].keys()),list(ipl['player_of_match']
plt.show()
```



In [169…
```
#Getting the frequency of result column
ipl['result'].value_counts()
```

Out[169]:
```
result
normal       743
tie            9
no result      4
Name: count, dtype: int64
```

In [170…
```
#Finding out the number of toss wins w.r.t each team
ipl['toss_winner'].value_counts()
```

Out[170]:
```
toss_winner
Mumbai Indians                 98
Kolkata Knight Riders          92
Chennai Super Kings            89
Royal Challengers Bangalore    81
Kings XI Punjab                81
Delhi Daredevils               80
Rajasthan Royals               80
Sunrisers Hyderabad            46
Deccan Chargers                43
Pune Warriors                  20
Gujarat Lions                  15
Delhi Capitals                 10
Kochi Tuskers Kerala            8
Rising Pune Supergiants         7
Rising Pune Supergiant          6
Name: count, dtype: int64
```

In [171…
```
#Extracting the records where a team won batting first
batting_first = ipl[ipl['win_by_runs']!=0]
```

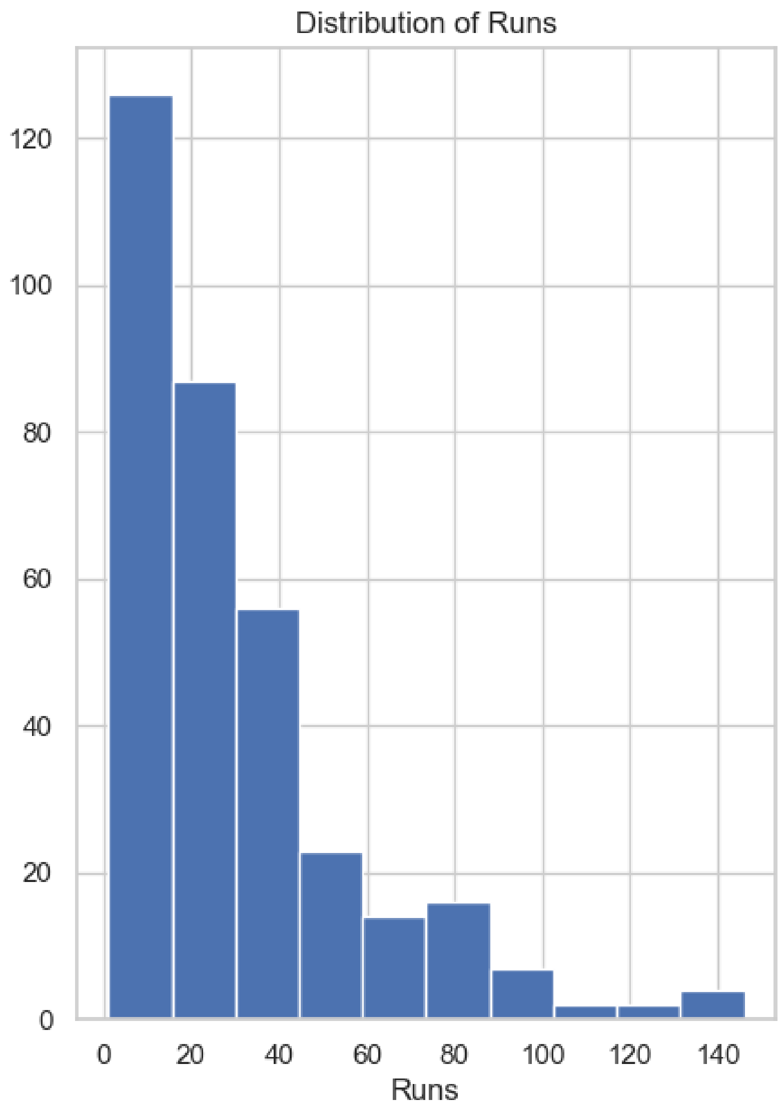
In [172…
```
batting_first.head()
```

Out[172]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_a
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	
4	5	2017	Bangalore	2017-04-08	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	
8	9	2017	Pune	2017-04-11	Delhi Daredevils	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	
13	14	2017	Kolkata	2017-04-15	Kolkata Knight Riders	Sunrisers Hyderabad	Sunrisers Hyderabad	field	normal	
14	15	2017	Delhi	2017-04-15	Delhi Daredevils	Kings XI Punjab	Delhi Daredevils	bat	normal	

In [173…

```
#Making a histogram
plt.figure(figsize=(5,7))
plt.hist(batting_first['win_by_runs'])
plt.title('Distribution of Runs')
plt.xlabel('Runs')
plt.show()
```



In [174…

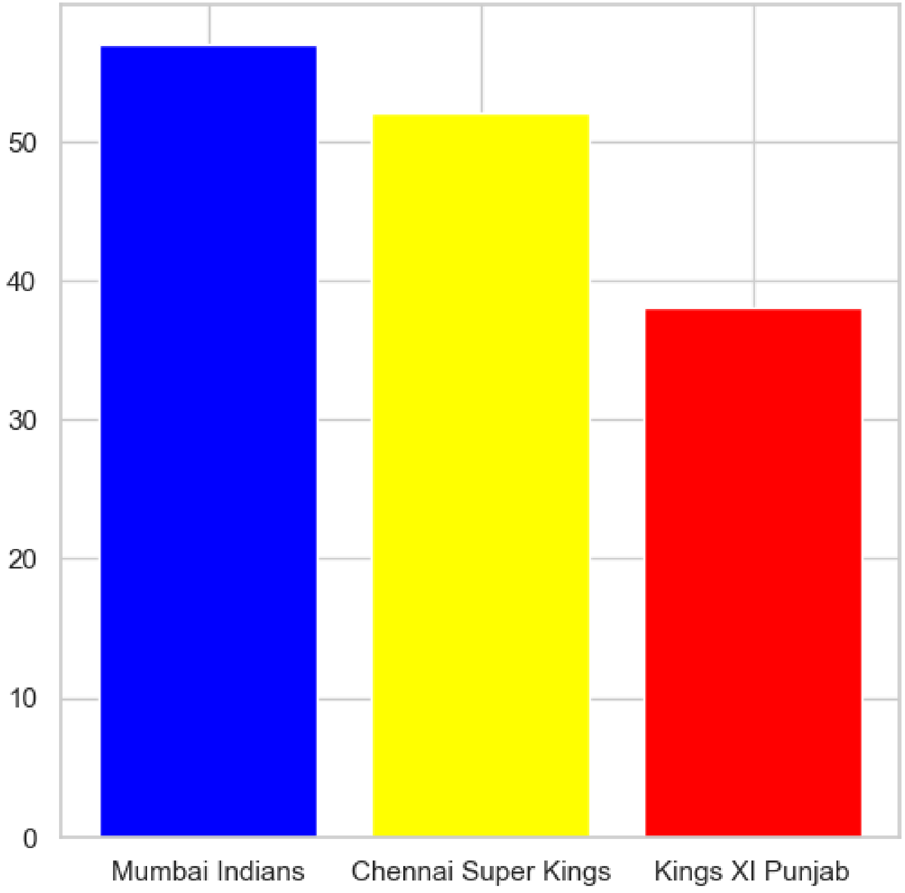
```
#Finding out the number of wins w.r.t each team after batting first
batting_first['winner'].value_counts()
```

Out[174]:

```
winner
Mumbai Indians                 57
Chennai Super Kings            52
Kings XI Punjab                38
Kolkata Knight Riders          36
Royal Challengers Bangalore    35
Sunrisers Hyderabad            30
Rajasthan Royals               27
Delhi Daredevils               25
Deccan Chargers                18
Pune Warriors                   6
Rising Pune Supergiant          5
Delhi Capitals                  3
Kochi Tuskers Kerala            2
Rising Pune Supergiants         2
Gujarat Lions                   1
Name: count, dtype: int64
```

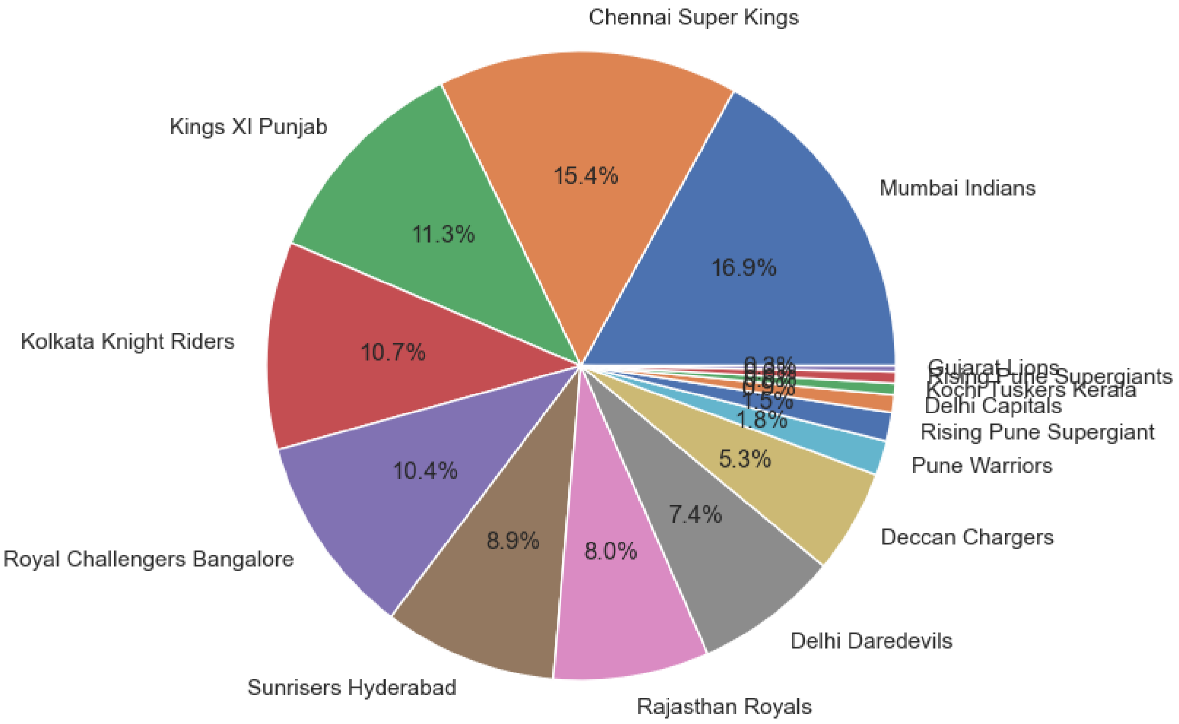
In [175…

```
#Making a bar-plot for top 3 teams with most wins after batting first
plt.figure(figsize=(6,6))
plt.bar(list(batting_first['winner'].value_counts()[0:3].keys()),list(batting_first['winne
plt.show()
```



In [176…

```
#Making a pie-chart
plt.figure(figsize= (7,7))
plt.pie(list(batting_first['winner'].value_counts()), labels = list(batting_first['winner'
plt.show()
```



In [177…

```
#Extracting those records where a team has won after batting second
```

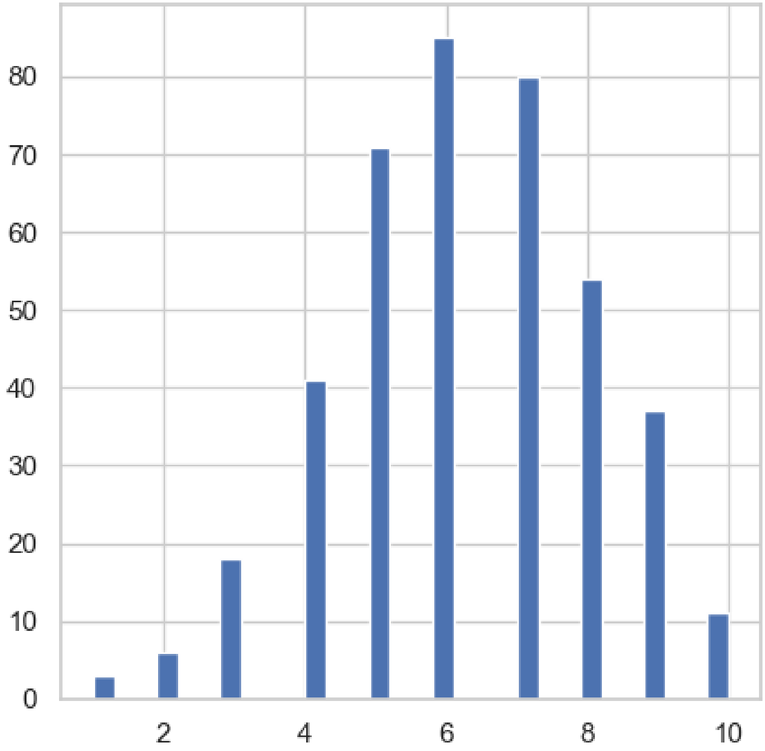
In [178…
```
batting_second = ipl[ipl['win_by_wickets']!=0]
```

In [179…
```
#Looking at the head
batting_second.head()
```

Out[179]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_app
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	
5	6	2017	Hyderabad	2017-04-09	Gujarat Lions	Sunrisers Hyderabad	Sunrisers Hyderabad	field	normal	
6	7	2017	Mumbai	2017-04-09	Kolkata Knight Riders	Mumbai Indians	Mumbai Indians	field	normal	

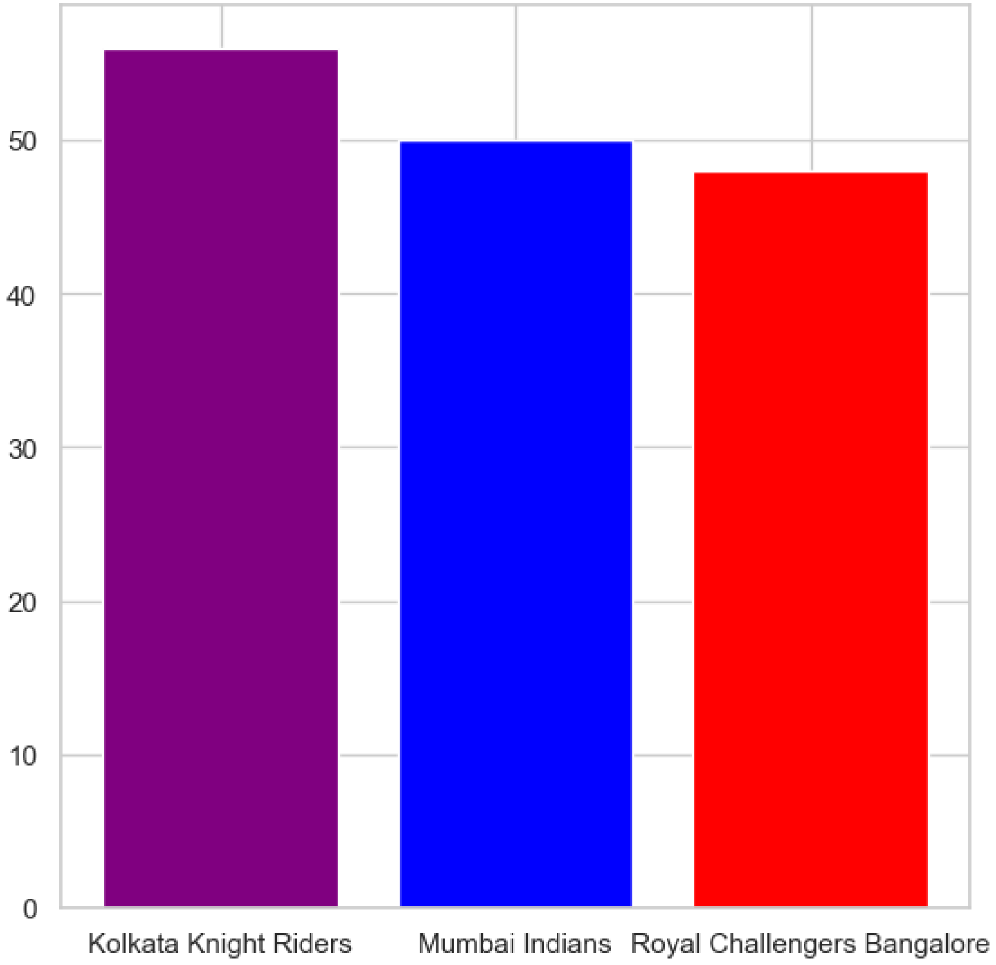
In [180…
```
#Making a histogram for frequency of wins w.r.t number of wickets
plt.figure(figsize=(5,5))
plt.hist(batting_second['win_by_wickets'],bins = 30)
plt.show()
```



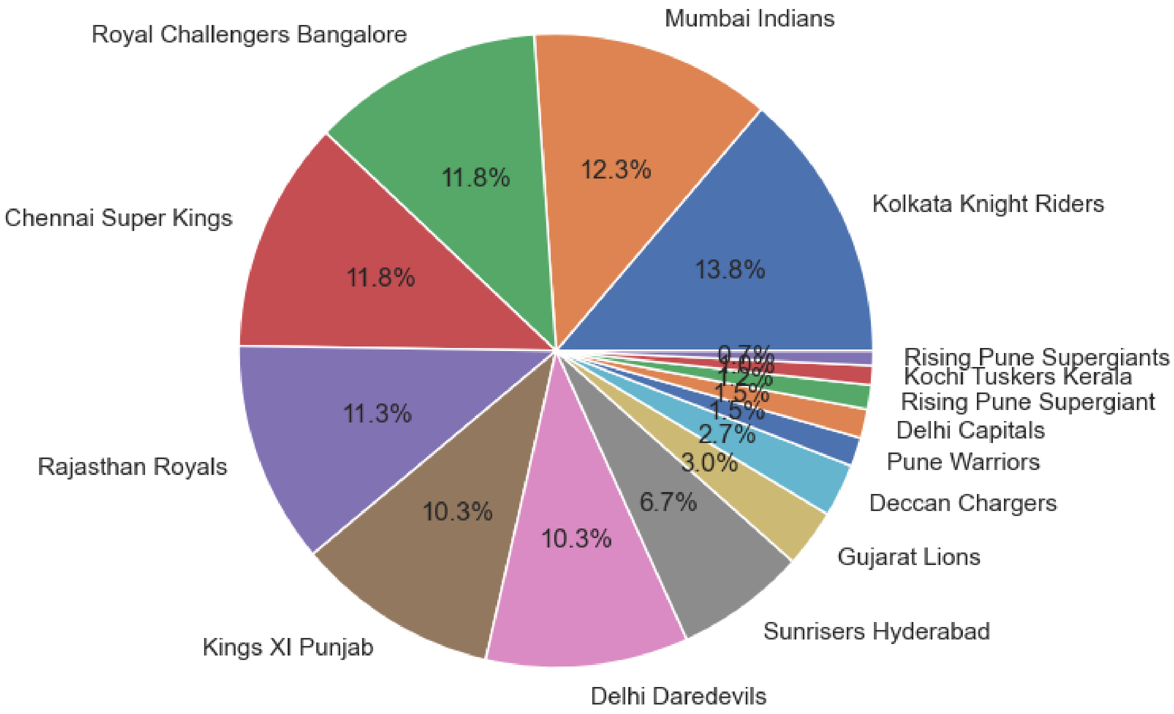
In [181…
```
#Finding out the frequency of number of wins w.r.t each tie after batting second
batting_second['winner'].value_counts()
```

```
Out[181]: winner
          Kolkata Knight Riders          56
          Mumbai Indians                 50
          Royal Challengers Bangalore    48
          Chennai Super Kings            48
          Rajasthan Royals               46
          Kings XI Punjab                42
          Delhi Daredevils               42
          Sunrisers Hyderabad            27
          Gujarat Lions                  12
          Deccan Chargers                11
          Pune Warriors                   6
          Delhi Capitals                  6
          Rising Pune Supergiant          5
          Kochi Tuskers Kerala            4
          Rising Pune Supergiants         3
          Name: count, dtype: int64
```

```
In [182…  #Making a bar plot for top-3 teams with most wins after batting second
          plt.figure(figsize = (6.5,6.5))
          plt.bar(list(batting_second['winner'].value_counts()[0:3].keys()),list(batting_second['winn
          plt.show()
```



```
In [183…  #Making a pie-chart
          plt.figure(figsize= (6.5,6.5))
          plt.pie(list(batting_second['winner'].value_counts()), labels = list(batting_second['winner
          plt.show()
```



```
In [184…: #Looking at the number of matches played each season
ipl['season'].value_counts()
```

```
Out[184]: season
2013    76
2012    74
2011    73
2010    60
2014    60
2016    60
2018    60
2019    60
2017    59
2015    59
2008    58
2009    57
Name: count, dtype: int64
```

```
In [185…: #Looking at the number of matches played in each city
ipl['city'].value_counts()
```

Out[185]:

```
city
Mumbai            101
Kolkata            77
Delhi              74
Bangalore          66
Hyderabad          64
Chennai            57
Jaipur             47
Chandigarh         46
Pune               38
Durban             15
Bengaluru          14
Visakhapatnam      13
Centurion          12
Ahmedabad          12
Rajkot             10
Mohali             10
Indore              9
Dharamsala          9
Johannesburg        8
Cuttack             7
Ranchi              7
Port Elizabeth      7
Cape Town           7
Abu Dhabi           7
Sharjah             6
Raipur              6
Kochi               5
Kanpur              4
Nagpur              3
Kimberley           3
East London         3
Bloemfontein        2
Name: count, dtype: int64
```

In [186…

```
#Finding out how many times a team has won the match after winning the toss
np.sum(ipl['toss_winner']==ipl['winner'])
```

Out[186]:

```
393
```

In [187…

```
393/756 #The ratio of winning toss and winning matches
```

Out[187]:

```
0.5198412698412699
```

Done with the Great Learning session on Python Libraries