YBI FOUNDATION INTERNSHIP

NAME : AYUSH KUMAR

ASSIGNMENT

```
import pandas as pd
```

```
titanic = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Titanic.csv')
```

```
titanic.head(5) #First n rows of the DataFrame
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	emba
0	1	1	Allen, Miss. Elisabeth Walton	1	29.00	0	0	24160	211.3375	B5	
1	1	1	Allison, Master. Hudson	0	0.92	1	2	113781	151.5500	C22 C26	

```
titanic.tail(5) # Last n rows of the DataFrame
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin
1304	3	0	Zabour, Miss. Hileni	1	14.5	1	0	2665	14.4542	NaN
1305	3	0	Zabour, Miss. Thamine	1	NaN	1	0	2665	14.4542	NaN
1306	3	0	Zakarian, Mr	0	26.5	0	0	2656	7.2250	NaN

```
titanic.info() #Index, Datatype and Memory information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   pclass     1309 non-null   object 
 1   survived   1309 non-null   int64  
 2   name       1309 non-null   object 
 3   sex        1309 non-null   int64  
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   int64  
 6   parch      1309 non-null   int64  
 7   ticket     1309 non-null   object 
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object 
 10  embarked   1307 non-null   object 
 11  boat       486 non-null    object 
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object 
```

```
dtypes: float64(3), int64(4), object(7)
```

```
titanic.describe() #Summary statistics for numerical columns
```

	survived	sex	age	sibsp	parch	fare	
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	12
mean	0.381971	0.355997	29.881138	0.498854	0.385027	33.295479	16
std	0.486055	0.478997	14.413493	1.041658	0.865560	51.758668	9
min	0.000000	0.000000	0.170000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	21.000000	0.000000	0.000000	7.895800	7
50%	0.000000	0.000000	28.000000	0.000000	0.000000	14.454200	15
75%	1.000000	1.000000	39.000000	1.000000	0.000000	31.275000	25
max	1.000000	1.000000	80.000000	8.000000	9.000000	512.329200	32

```
titanic['age'].nsmallest(3)
```

```
763     0.17
747     0.33
1240    0.42
Name: age, dtype: float64
```

```
titanic.age.nlargest(4)
```

```
14      80.0
61      76.0
1235    74.0
9       71.0
Name: age, dtype: float64
```

```
titanic.age.max()
```

```
80.0
```

```
titanic.age.min()
```

```
0.17
```

```
titanic['pclass']=titanic['pclass'].astype('object')
```

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   object
```

```
 1   survived   1309 non-null   int64
 2   name       1309 non-null   object
 3   sex        1309 non-null   object
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   int64
 6   parch      1309 non-null   int64
 7   ticket     1309 non-null   object
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object
 10  embarked   1307 non-null   object
 11  boat       486 non-null    object
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object
dtypes: float64(3), int64(3), object(8)
memory usage: 143.3+ KB
```

```
titanic.embarked
```

```
0       S
1       S
2       S
3       S
4       S
       ..
1304    C
1305    C
1306    C
1307    C
1308    S
Name: embarked, Length: 1309, dtype: object
```

```
titanic.embarked.unique()
```

```
array(['S', 'C', nan, 'Q'], dtype=object)
```

```
titanic.embarked.value_counts()
```

```
S    914
C    270
Q    123
Name: embarked, dtype: int64
```

```
titanic.age.value_counts()
```

```
24.00    47
22.00    43
21.00    41
30.00    40
18.00    39
         ..
0.33      1
22.50     1
70.50     1
0.67      1
26.50     1
Name: age, Length: 98, dtype: int64
```

```
titanic.replace({'sex':{'male':0,'female':1}})
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin
0	1	1	Allen, Miss. Elisabeth Walton	1	29.00	0	0	24160	211.3375	B5
1	1	1	Allison, Master. Hudson Trevor	0	0.92	1	2	113781	151.5500	C22 C26
2	1	0	Allison, Miss. Helen Loraine	1	2.00	1	2	113781	151.5500	C22 C26
3	1	0	Allison, Mr. Hudson Joshua Creighton	0	30.00	1	2	113781	151.5500	C22 C26
4	1	0	Allison, Mrs. Hudson J C (Bessie	1	25.00	1	2	113781	151.5500	C22

```
titanic.replace({'sex':{'male':0,'female':1}}, inplace = True)
```

```
titanic.iloc[3,4]
```

```
30.0
```

```
titanic.loc[3 ,'age']
```

```
30.0
```

```
titanic.iloc[:,4]
```

```
0       29.00
1        0.92
2        2.00
3       30.00
4       25.00
        ...
1304    14.50
1305      NaN
1306    26.50
1307    27.00
1308    29.00
Name: age, Length: 1309, dtype: float64
```

```
titanic.loc[2:6,['age','name','sex']]
```

	age	name	sex
2	2.0	Allison, Miss. Helen Loraine	1
3	30.0	Allison, Mr. Hudson Joshua Creighton	0
4	25.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	1
5	48.0	Anderson, Mr. Harry	0
6	63.0	Andrews, Miss. Kornelia Theodosia	1

```
titanic[titanic.age>75 ]
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarl
14	1	1	Barkworth, Mr. Algernon	0	80.0	0	0	27042	30.00	A23	

```
titanic[(titanic.age>75)&(titanic.sex=='male') ]
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat

```
titanic.apply(pd.Series.value_counts) # Unique values and counts for all columns
```

```
titanic.value_counts(dropna=False) #View unique values and counts
```

```
pclass  survived  name                                          sex  age   sibsp  p
1       0         Allison, Miss. Helen Loraine                  1    2.0   1      2
3       0         Lefebre, Miss. Ida                            1    NaN   3      1
                  Lobb, Mr. William Arthur                      0    30.0  1      0
                  Lithman, Mr. Simon                            0    NaN   0      0
                  Ling, Mr. Lee                                 0    28.0  0      0

2       0         Morley, Mr. Henry Samuel ("Mr Henry Marshall")  0  39.0  0      0
                  Moraweck, Dr. Ernest                          0    54.0  0      0
                  Montvila, Rev. Juozas                         0    27.0  0      0
                  Mitchell, Mr. Henry Michael                   0    70.0  0      0
3       1         de Mulder, Mr. Theodore                       0    30.0  0      0
Length: 1309, dtype: int64
```

```
titanic.isnull() #Checks for null Values, Returns Boolean Arrray
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embark
0	False	False	False	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	...	...	
1304	False	False	False	False	False	False	False	False	False	True	Fa
1305	False	False	False	False	True	False	False	False	False	True	Fa
1306	False	False	False	False	False	False	False	False	False	True	Fa
1307	False	False	False	False	False	False	False	False	False	True	Fa
1308	False	False	False	False	False	False	False	False	False	True	Fa

1309 rows × 14 columns

```
titanic.mean()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
  """Entry point for launching an IPython kernel.
pclass        2.294882
survived      0.381971
sex           0.355997
age          29.881138
sibsp         0.498854
parch         0.385027
fare         33.295479
```

```
body        160.809917
dtype: float64
```

```
titanic.corr()
```

	survived	sex	age	sibsp	parch	fare	body
survived	1.000000	0.528693	-0.055512	-0.027825	0.082660	0.244265	NaN
sex	0.528693	1.000000	-0.063645	0.109609	0.213125	0.185523	0.015903
age	-0.055512	-0.063645	1.000000	-0.243699	-0.150917	0.178740	0.058809
sibsp	-0.027825	0.109609	-0.243699	1.000000	0.373587	0.160238	-0.099961
parch	0.082660	0.213125	-0.150917	0.373587	1.000000	0.221539	0.051099
fare	0.244265	0.185523	0.178740	0.160238	0.221539	1.000000	-0.043110
body	NaN	0.015903	0.058809	-0.099961	0.051099	-0.043110	1.000000

```
titanic.std()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
  """Entry point for launching an IPython kernel.
pclass       0.837836
survived     0.486055
sex          0.478997
age         14.413493
sibsp        1.041658
parch        0.865560
fare        51.758668
body        97.696922
dtype: float64
```

```
titanic.mean()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
  """Entry point for launching an IPython kernel.
pclass        2.294882
survived      0.381971
sex           0.355997
age          29.881138
sibsp         0.498854
parch         0.385027
fare         33.295479
body        160.809917
dtype: float64
```

```
# MORE CHEAT SHEET


#Use these commands to import data from a variety of different sources and formats.
```

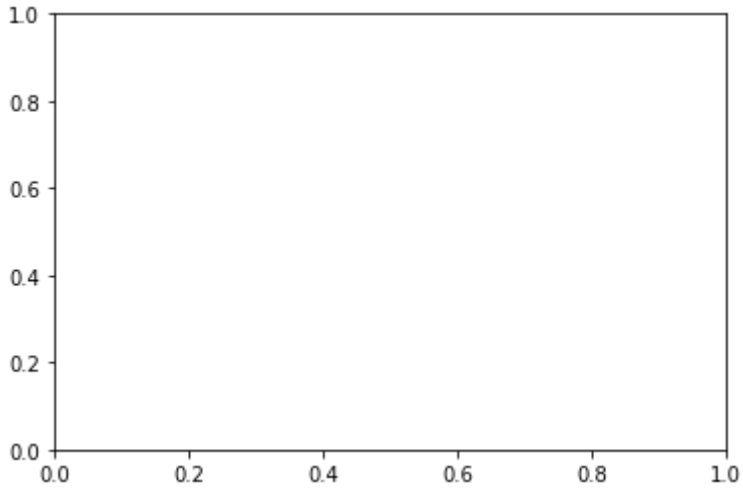
```
# pd.read_csv(filename) | From a CSV file
# pd.read_table(filename) | From a delimited text file (like TSV)
# pd.read_excel(filename) | From an Excel file
# pd.read_sql(query, connection_object) | Read from a SQL table/database
# pd.read_json(json_string) | Read from a JSON formatted string, URL or file.
# pd.read_html(url) | Parses an html URL, string or file and extracts tables to a list of
# pd.read_clipboard() | Takes the contents of your clipboard and passes it to read_table()
# pd.DataFrame(dict) | From a dict, keys for columns names, values for data as lists


# Use these commands to export a DataFrame to CSV, .xlsx, SQL, or JSON.

# df.to_csv(filename) | Write to a CSV file
# df.to_excel(filename) | Write to an Excel file
# df.to_sql(table_name, connection_object) | Write to a SQL table
# df.to_json(filename) | Write to a file in JSON format


# These commands can be useful for creating test segments.

# pd.DataFrame(np.random.rand(20,5)) | 5 columns and 20 rows of random floats
# pd.Series(my_list) | Create a series from an iterable my_list
# df.index = pd.date_range('1900/1/30', periods=df.shape[0]) | Add a date index

# Viewing/Inspecting Data


# Use these commands to take a look at specific sections of your pandas DataFrame or Serie

# df.head(n) | First n rows of the DataFrame
# df.tail(n) | Last n rows of the DataFrame
# df.shape | Number of rows and columns
# df.info() | Index, Datatype and Memory information
# df.describe() | Summary statistics for numerical columns
# s.value_counts(dropna=False) | View unique values and counts
# df.apply(pd.Series.value_counts) | Unique values and counts for all columns


# Use these commands to select a specific subset of your data.

# df[col] | Returns column with label col as Series
# df[[col1, col2]] | Returns columns as a new DataFrame
# s.iloc[0] | Selection by position
# s.loc['index_one'] | Selection by index
# df.iloc[0,:] | First row
# df.iloc[0,0] | First element of first column


# Use these commands to perform a variety of data cleaning tasks.

# df.columns = ['a','b','c'] | Rename columns
# pd.isnull() | Checks for null Values, Returns Boolean Arrray
# pd.notnull() | Opposite of pd.isnull()
# df.dropna() | Drop all rows that contain null values
# df.dropna(axis=1) | Drop all columns that contain null values
```

```
# df.dropna(axis=1,thresh=n) | Drop all rows have have less than n non null values
# df.fillna(x) | Replace all null values with x
# s.fillna(s.mean()) | Replace all null values with the mean (mean can be replaced with al
# s.astype(float) | Convert the datatype of the series to float
# s.replace(1,'one') | Replace all values equal to 1 with ‘one’
# s.replace([1,3],['one','three']) | Replace all 1 with ‘one’ and 3 with ‘three’
# df.rename(columns=lambda x: x + 1) | Mass renaming of columns
# df.rename(columns={'old_name': 'new_ name'}) | Selective renaming
# df.set_index('column_one') | Change the index
# df.rename(index=lambda x: x + 1) | Mass renaming of index


# Use these commands to filter, sort, and group your data.

# df[df[col] > 0.5] | Rows where the column col is greater than 0.5
# df[(df[col] > 0.5) & (df[col] < 0.7)] | Rows where 0.7 > col > 0.5
# df.sort_values(col1) | Sort values by col1 in ascending order
# df.sort_values(col2,ascending=False) | Sort values by col2 in descending order
# df.sort_values([col1,col2],ascending=[True,False]) | Sort values by col1 in ascending or
# df.groupby(col) | Returns a groupby object for values from one column
# df.groupby([col1,col2]) | Returns groupby object for values from multiple columns
# df.groupby(col1)[col2] | Returns the mean of the values in col2, grouped by the values i
# df.pivot_table(index=col1,values=[col2,col3],aggfunc=mean) | Create a pivot table that g
# df.groupby(col1).agg(np.mean) | Find the average across all columns for every unique col
# df.apply(np.mean) | Apply the function np.mean() across each column
# nf.apply(np.max,axis=1) | Apply the function np.max() across each row


# Use these commands to combine multiple dataframes into a single one.

# df1.append(df2) | Add the rows in df1 to the end of df2 (columns should be identical)
# pd.concat([df1, df2],axis=1) | Add the columns in df1 to the end of df2 (rows should be
# df1.join(df2,on=col1,how='inner') | SQL-style join the columns in df1 with the columns o


# Use these commands to perform various statistical tests. (These can all be applied to a

# df.describe() | Summary statistics for numerical columns
# df.mean() | Returns the mean of all columns
# df.corr() | Returns the correlation between columns in a DataFrame
# df.count() | Returns the number of non-null values in each DataFrame column
# df.max() | Returns the highest value in each column
# df.min() | Returns the lowest value in each column
# df.median() | Returns the median of each column
# df.std() | Returns the standard deviation of each column
```

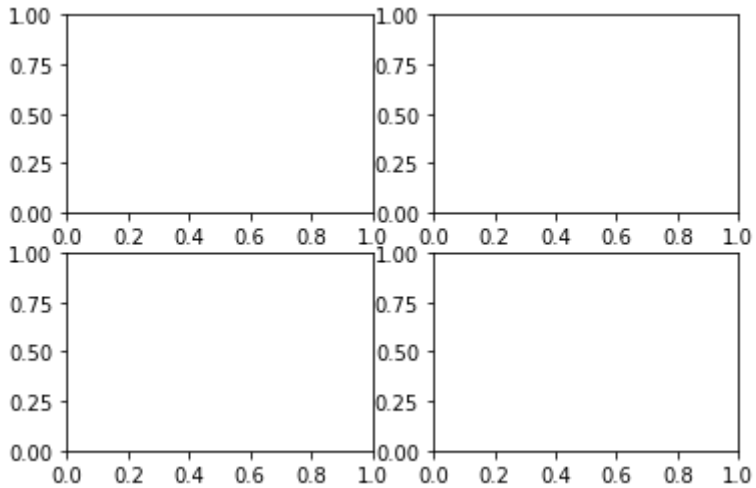
✓ 0s completed at 9:29 PM

```
import matplotlib.pyplot as plt
```

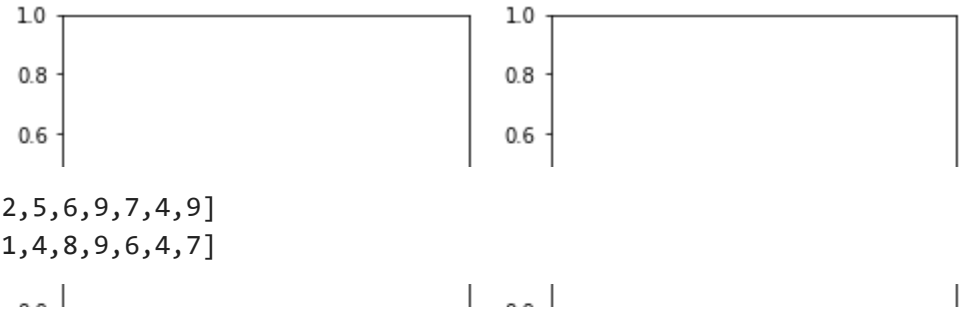
```
fig,ax = plt.subplots()
```




```
fig,ax = plt.subplots(nrows=2 , ncols=2)
```

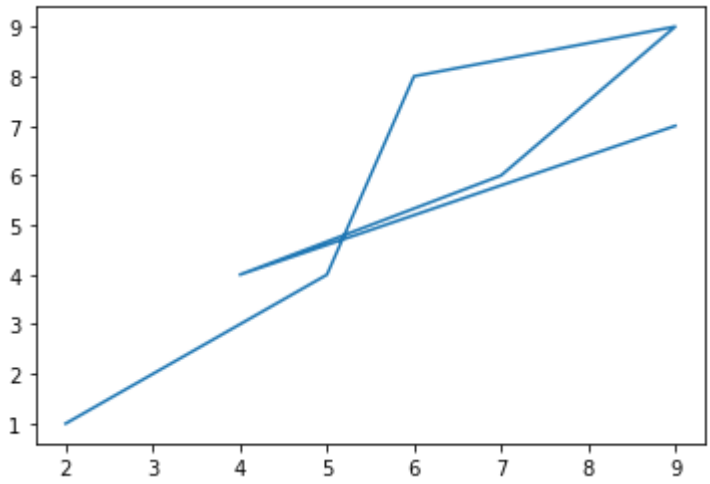



```
fig,ax = plt.subplots(nrows=2 , ncols=2 , figsize=(8, 6))
```

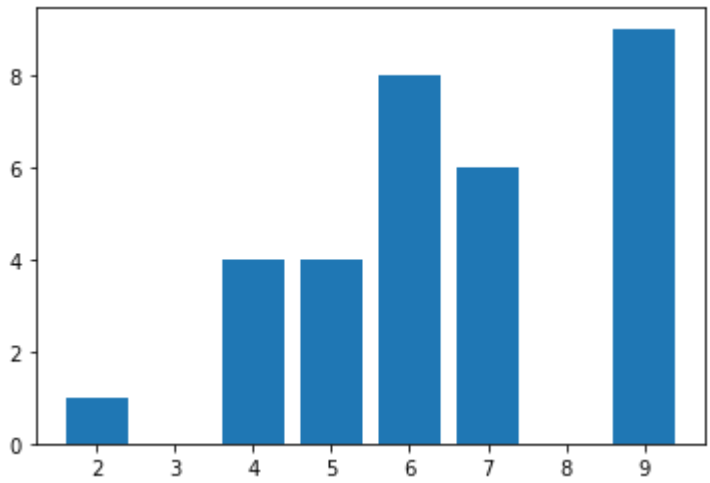



```
x = [2,5,6,9,7,4,9]
y = [1,4,8,9,6,4,7]
```

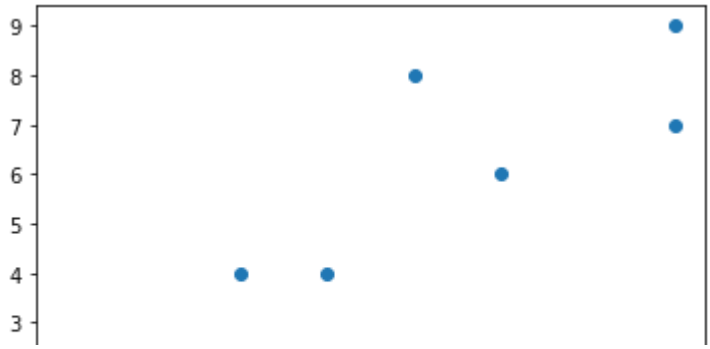
```
fig,ax = plt.subplots()
ax.plot(x,y)
plt.show()
```



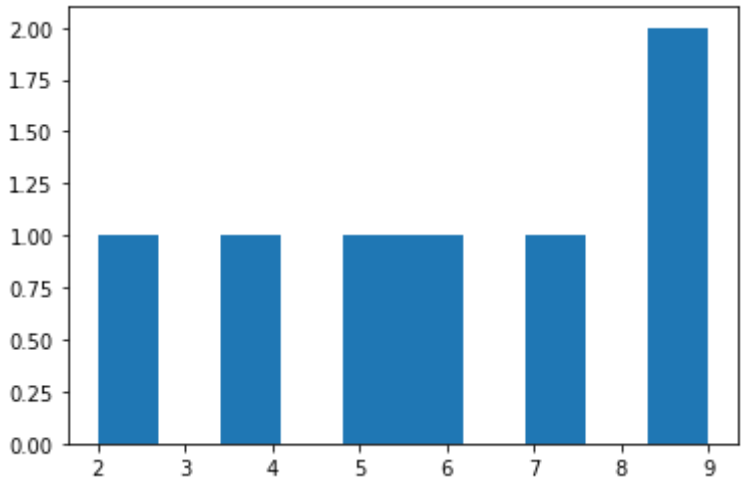

```
fig,ax = plt.subplots()
ax.bar(x,y)
plt.show()
```



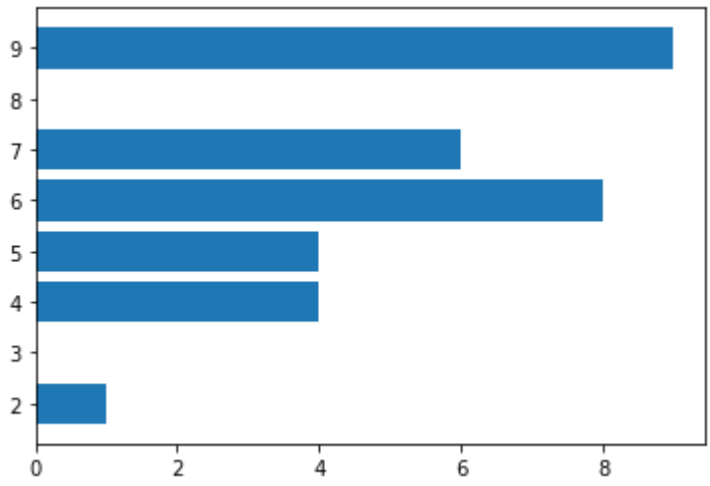

```
fig,ax = plt.subplots()
ax.scatter(x,y)
plt.show()
```



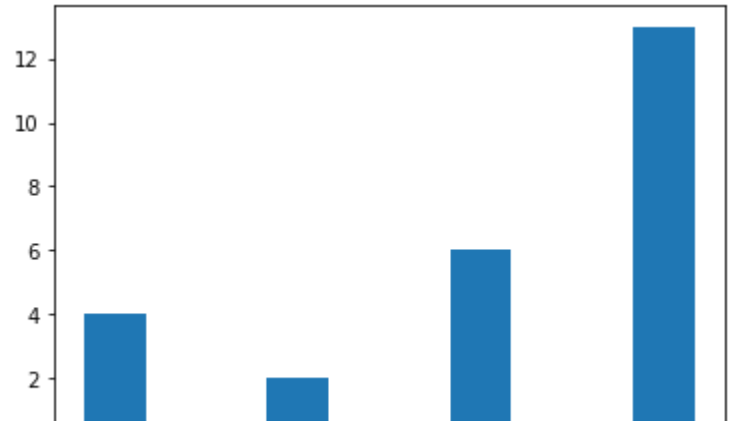

```
fig,ax = plt.subplots()
ax.hist(x)
plt.show()
```



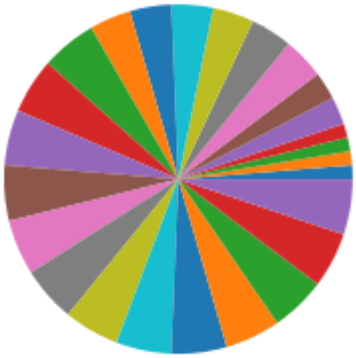

```
fig,ax = plt.subplots()
ax.barh(x,y)
plt.show()
```



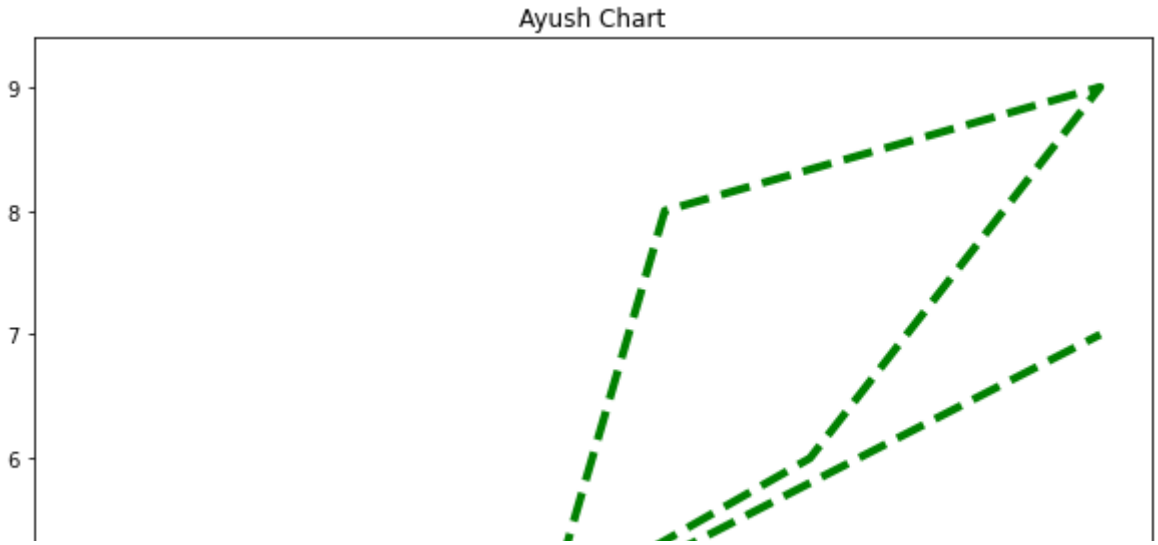

```
z = [1,1,1,1,2,2,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4]
fig,ax = plt.subplots()
ax.hist(z)
plt.show()
```




```
fig,ax = plt.subplots()
ax.pie(z)
plt.show()
```



```
fig,ax = plt.subplots(figsize=(10, 10))
ax.plot(x,y ,color='green',linestyle='--',linewidth='4')
ax.set_title('Ayush Chart')
ax.set_xlabel('Months')
ax.set_ylabel('Sales')
plt.show()
```



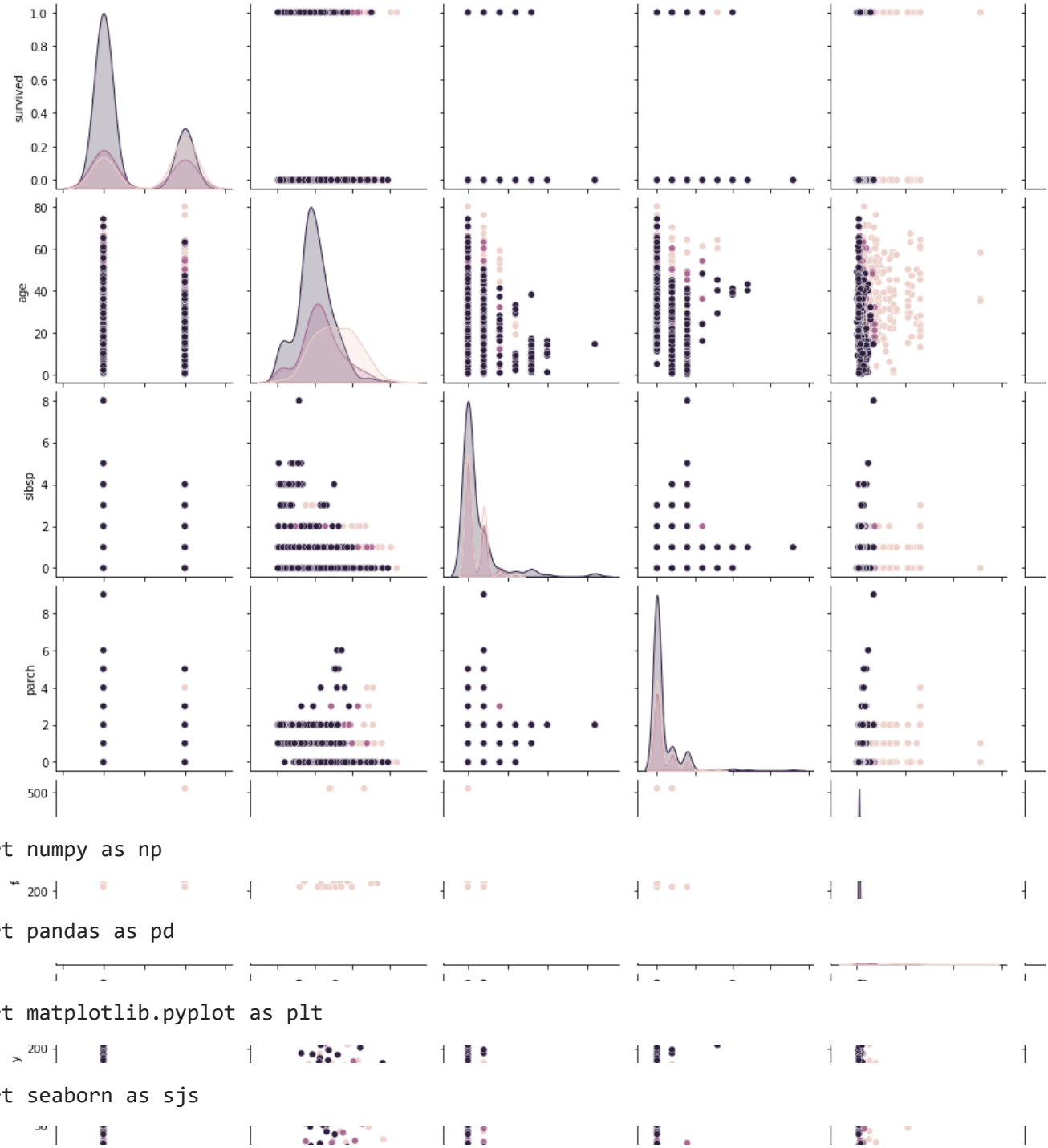

```
import seaborn as sns
```

```
titanic = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Titanic.csv')
```

```
import pandas as pd
```

```
sns.pairplot(titanic,hue='pclass')
```

```
<seaborn.axisgrid.PairGrid at 0x7f3fdaa26390>
```




```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sjs
```

```
salary = pd.read_csv('https://github.com/YBI-FOUNDATION/Dataset/raw/main/Salary%20Data.csv
```

```
salary.head()
```

	Experience Years	Salary
0	1.1	39343
1	1.2	42774
2	1.3	46205
3	1.5	37731
4	2.0	43525

```
salary.tail()
```

	Experience Years	Salary
35	9.0	105582
36	9.5	116969
37	9.6	112635
38	10.3	122391
39	10.5	121872

```
salary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Experience Years  40 non-null     float64
 1   Salary            40 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 768.0 bytes
```

```
salary.describe()
```

	Experience Years	Salary
count	40.000000	40.000000
mean	5.152500	74743.625000
std	2.663715	25947.122885
min	1.100000	37731.000000
25%	3.200000	56878.250000
50%	4.600000	64472.500000
75%	6.875000	95023.250000
max	10.500000	122391.000000

```
#Define X , Y
```

```
Y=salary['Salary']#Always use single bracket #Flat
```

```
Y.shape
```

```
(40,)
```

```
X=salary[['Experience Years']]#Always use double bracket
```

```
X.shape
```

```
(40, 1)
```

```
X
```

	Experience Years
0	1.1
1	1.2
2	1.3
3	1.5
4	2.0
5	2.2
6	2.5
7	2.9
8	3.0
9	3.2
10	3.2
11	3.5
12	3.7
13	3.8
14	3.9
15	4.0
16	4.0
17	4.1
18	4.3

Y

```
0     39343
1     42774
2     46205
3     37731
4     43525
5     39891
6     48266
7     56642
8     60150
9     54445
10    64445
11    60000
12    57189
13    60200
14    63218
15    55794
16    56957
17    57081
18    59095
19    61111
20    64500
```

```
21     67938
22     66029
23     83088
24     82200
25     81363
26     93940
27     91000
28     90000
29     91738
30     98273
31    101302
32    113812
33    111620
34    109431
35    105582
36    116969
37    112635
38    122391
39    121872
Name: Salary, dtype: int64
```

```
#Split Data
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size=0.8,random_state=2002)
```

```
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```

```
((32, 1), (8, 1), (32,), (8,))
```

```
#Model
```

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
```

```
model.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
model.intercept_
```

```
25666.243989366354
```

```
model.coef_
```

```
array([9400.90415121])
```

```
y_pred = model.predict(X_test)
```

```
y_pred
```

```
array([124375.73757705,  99933.38678391, 105573.92927463,  89592.39221758,
       122495.55674681,  62329.77017908,  82071.66889661,  92412.66346294])
```

```
from sklearn.metrics import mean_absolute_error
```

```
mean_absolute_error(Y_test,y_pred)
```

```
3848.1854386374953
```