Creating a data frame with pandas:

In [1]:
```
### Creating a dataframe
import pandas as pd
dataset = {'Fruits': ["Apple","Mango","Grapes","Strawberry","Orange"], 'Sup
# Create DataFrame
df = pd.DataFrame(dataset)

# Print the output.
df
```

Out[1]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Strawberry	25
4	Orange	20

Read the top element chart:

In [2]:
```
# print the first couple of elements
df.head()
```

Out[2]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Strawberry	25
4	Orange	20

Read the Bottom element chart:

In [3]:
```
# print the last couple of elements
df.tail()
```

Out[3]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Strawberry	25
4	Orange	20

Understanding the statistical information of the data:

In [4]:
```
# Understand all the essential features
df.describe()
```

Out[4]:

	Supply
count	5.000000
mean	20.000000
std	7.905694
min	10.000000
25%	15.000000
50%	20.000000
75%	25.000000
max	30.000000

In [5]:
```
df
```

Out[5]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Strawberry	25
4	Orange	20

Merging the values:

In [6]:
```
### Creating a dataframe
import pandas as pd
dataset1 = {'Fruits': ["Apple", "Mango","Grapes", "Strawberry", "Orange"],
dataset2 = {'Fruits': ["Melons", "banana"], 'Supply': [10, 20]}
# Create DataFrame
df1 = pd.DataFrame(dataset1)
df2 = pd.DataFrame(dataset2)

# Print the output.
df1.merge(df2, how = "outer")
```

Out[6]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Strawberry	25
4	Orange	20
5	Melons	10
6	banana	20

When how="outer",it performs an outer join,which means that it will retain all the rows from both dataframes, combining matching rows and filling in the missing values with NaN in the case of no match. In other words,it returns all rows from both dataframes, and if there's no matching data in either dataframe for a particular row,the result will have NaN values in those cells.

Grouping the values:

In pandas, the groupby functionn is used for grouping rows of data into groups based on the values in one or more colums.The groups created can be used for aggregation,transforming, or filtering the data.

In [7]:
```
dataset = {'Fruits': ["Apple", "Mango","Grapes", "Stawberry", "Oranges"], '
df = pd.DataFrame(dataset)
a = df.groupby('Fruits')
a.first()
```

Out[7]:

Fruits	Supply
Apple	30
Grapes	10
Mango	15
Oranges	20
Stawberry	25

a.first():This returns the first row of each group in the DataFrameGroupBy object a. The result is a new DataFrame that contains the first row of each group, grouped by the 'Fruits' column.

Accessing Specific Rows and Columns:

In [8]:
```
df
```

Out[8]:

	Fruits	Supply
0	Apple	30
1	Mango	15
2	Grapes	10
3	Stawberry	25
4	Oranges	20

In [9]:
```
# Access specific rows and columns by specific positions
df.iloc[1:4,[1]]
```

Out[9]:

	Supply
1	15
2	10
3	25

Accessing by labels:

In [10]:
```
# Access specific rows and columns by specific labels
df.loc[1:3,['Fruits']]
```

Out[10]:

	Fruits
1	Mango
2	Grapes
3	Stawberry

In [11]:
```
df.loc[0,['Fruits']]
```

Out[11]:
```
Fruits    Apple
Name: 0, dtype: object
```

In [12]:
```
df.loc[3,['Supply']]
```

Out[12]:
```
Supply    25
Name: 3, dtype: object
```

In [13]:
```
# # Accessing the first row of the dataframe
df.loc[0, :]
df.iloc[0, :]

# Accessing the first column of the dataframe
# df.loc[:, 'Fruits']
# df.iloc[:, 0]

# Accessing specific row and column value
# df.loc[0, 'Supply']
# df.iloc[0, 0.]
```

Out[13]:
```
Fruits    Apple
Supply       30
Name: 0, dtype: object
```

Sort the Values in a data frame:

In [14]:
```
dataset = {'Fruits': ["Apple", "Mango", "Grapes", "Strawberry", "Orange"],
df = pd.DataFrame(dataset)
df.sort_values(by = ["Supply"])
```

Out[14]:

	Fruits	Supply
2	Grapes	10
1	Mango	15
4	Orange	20
3	Strawberry	25
0	Apple	30

In [15]:
```
#Sort the Values in a  data frame:

dataset = {'Fruits':["Apple","Mango", "Grapes", "Strawberry", "Oranges"],

df = pd.DataFrame(dataset)

df.sort_values("Supply")
```

Out[15]:

	Fruits	Supply
2	Grapes	10
1	Mango	15
4	Oranges	20
3	Strawberry	25
0	Apple	30

Sample

In [16]:
```
# #this will give random 5 rows of the dataframe we can also get the randon
df.sample(3)
```

Out[16]:

	Fruits	Supply
1	Mango	15
0	Apple	30
4	Oranges	20

columns

In [17]:
```
#this will the all the name of attributes in the dataframe
df.columns
```

Out[17]:
```
Index(['Fruits', 'Supply'], dtype='object')
```

In [18]:
```
import pandas as pd
dataset={"Names":["beyonce","akcent","ayesha","ariana","christina","avril",
 "selena","adele","katy","jennifer","shakira","rihanna"]}

subjects = ['Subject1', 'Subject2', 'Subject3', 'Subject4', 'Subject5']

# Creating a dataframe with the names and empty columns for the marks in th
df = pd.DataFrame(dataset, columns=['Names'] + subjects)

# Add the marks for each subject and student
df['Subject1'] = [90, 80, 85, 70, 75, 95, 80, 85, 90, 70, 75, 80, 85, 90, 8
df['Subject2'] = [85, 70, 80, 90, 85, 95, 70, 80, 85, 90, 75, 85, 80, 90, 7
df['Subject3'] = [80, 85, 90, 70, 75, 95, 85, 80, 85, 90, 75, 80, 85, 90, 7
df['Subject4'] = [75, 90, 85, 80, 75, 95, 70, 85, 90, 70, 75, 85, 80, 90, 7
df['Subject5'] = [80, 85, 90, 70, 75, 95, 80, 85, 90, 70, 75, 80, 85, 90, 8

df
```

Out[18]:

	Names	Subject1	Subject2	Subject3	Subject4	Subject5
0	beyonce	90	85	80	75	80
1	akcent	80	70	85	90	85
2	ayesha	85	80	90	85	90
3	ariana	70	90	70	80	70
4	christina	75	85	75	75	75
5	avril	95	95	95	95	95
6	taylor	80	70	85	70	80
7	alicia	85	80	80	85	85
8	britney	90	85	85	90	90
9	mariah	70	90	90	70	70
10	selena	75	75	75	75	75
11	adele	80	85	80	85	80
12	katy	85	80	85	80	85
13	jennifer	90	90	90	90	90
14	shakira	80	75	75	75	80
15	rihanna	75	80	80	80	75

In [19]:
```
df.head()
```

Out[19]:

	Names	Subject1	Subject2	Subject3	Subject4	Subject5
0	beyonce	90	85	80	75	80
1	akcent	80	70	85	90	85
2	ayesha	85	80	90	85	90
3	ariana	70	90	70	80	70
4	christina	75	85	75	75	75

In [20]:
```
df.tail()
```

Out[20]:

	Names	Subject1	Subject2	Subject3	Subject4	Subject5
11	adele	80	85	80	85	80
12	katy	85	80	85	80	85
13	jennifer	90	90	90	90	90
14	shakira	80	75	75	75	80
15	rihanna	75	80	80	80	75

In [21]:
```
df.columns
```

Out[21]:
```
Index(['Names', 'Subject1', 'Subject2', 'Subject3', 'Subject4', 'Subject
5'], dtype='object')
```

In [22]:
```
df.describe()
```

Out[22]:

	Subject1	Subject2	Subject3	Subject4	Subject5
count	16.000000	16.000000	16.000000	16.000000	16.000000
mean	81.562500	82.187500	82.500000	81.250000	81.562500
std	7.465197	7.295832	6.831301	7.637626	7.465197
min	70.000000	70.000000	70.000000	70.000000	70.000000
25%	75.000000	78.750000	78.750000	75.000000	75.000000
50%	80.000000	82.500000	82.500000	80.000000	80.000000
75%	86.250000	86.250000	86.250000	86.250000	86.250000
max	95.000000	95.000000	95.000000	95.000000	95.000000

In [23]:
```
df.shape
```

Out[23]:
```
(16, 6)
```

df.isna().sum()returns the number of missing values in each column.

The df.isna().sum()code returns the number of missing values in each column of the dataframe df.It checks for missing values (represented by NaN) in each column, and returns the count of missing values for each column in the form of a Pandas series.if all values in a column are present and non-null,then the count for that column will be 0.

In [24]:
```
df.isna().sum()
```

Out[24]:
```
Names       0
Subject1    0
Subject2    0
Subject3    0
Subject4    0
Subject5    0
dtype: int64
```

In [25]:
```
# df[['Subject1', 'Subject2']].mean() returns the mean of the specified col
df[['Subject1', 'Subject2']].mean()
```

Out[25]:
```
Subject1    81.5625
Subject2    82.1875
dtype: float64
```

In [26]:
```
# df.sort_values(by='Subject1', ascending=False) sorts the dataframe by the
df.sort_values(by='Subject1', ascending=False)
```

Out[26]:

	Names	Subject1	Subject2	Subject3	Subject4	Subject5
5	avril	95	95	95	95	95
0	beyonce	90	85	80	75	80
8	britney	90	85	85	90	90
13	jennifer	90	90	90	90	90
2	ayesha	85	80	90	85	90
7	alicia	85	80	80	85	85
12	katy	85	80	85	80	85
1	akcent	80	70	85	90	85
6	taylor	80	70	85	70	80
11	adele	80	85	80	85	80
14	shakira	80	75	75	75	80

Here you can perform many aggregate functions. Here you can perform some statistical operations on a set of data. let's see the aggregate functions that are available in the Pandas package:

1.count() – Count the number of non-null observation.

2.sum() – sum of all the values / list

3.min() – Minimum value

4.max() – maximum value

5.mean() – Arthmetic Mean

6.median() – Arthmetic median()

7.mode() – Mode

8.std() – standard deviation

9.var() – variance . You can use the agg() function with the help of this you can perform several statistical computations at once.

In [27]:
```
df .count()
```

Out[27]:
```
Names       16
Subject1    16
Subject2    16
Subject3    16
Subject4    16
Subject5    16
dtype: int64
```

Visualizations using Pandas Dataframe Operation

Histogram:

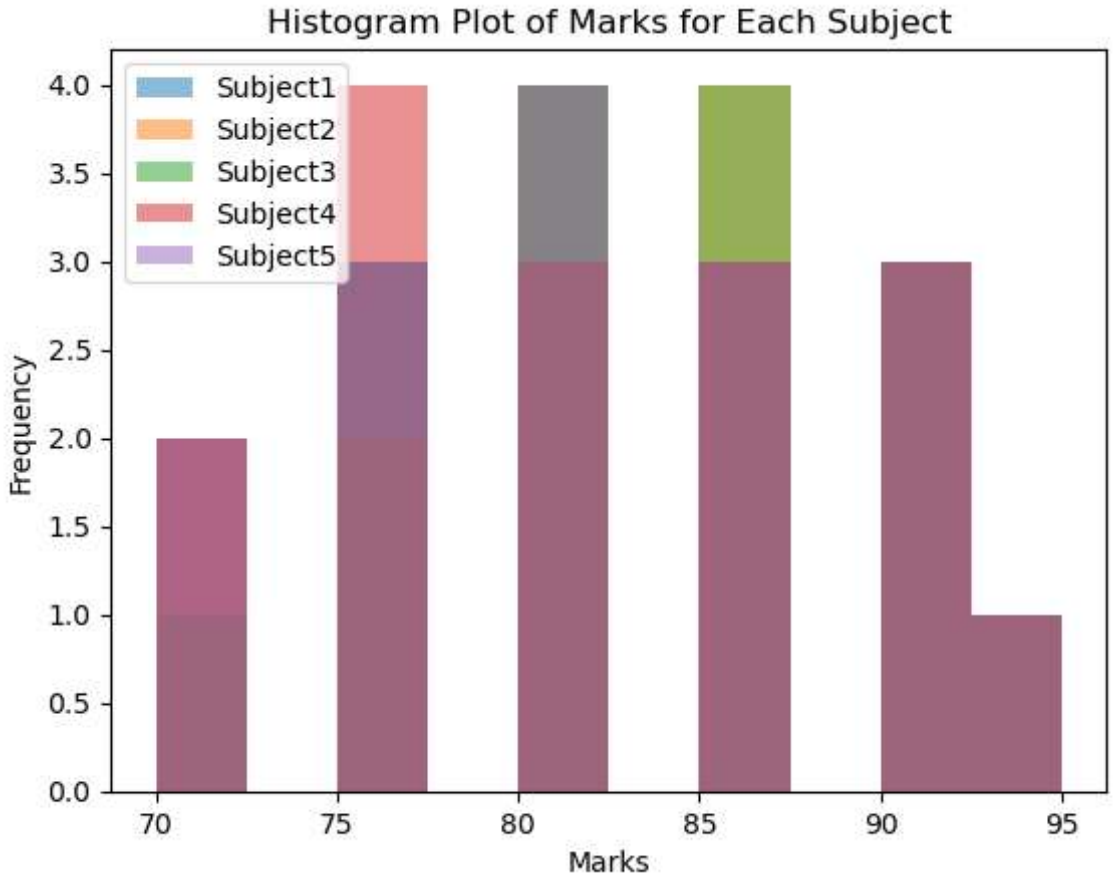
A histogram consists of rectangles whose area is proportional to the frequency of a variable and whose width is equal to the class interval.

x,y:- Takes the coordinates of the x and y-axis.

bins:- Bins are the numbers that represent the intervals into which you want to group the source data.

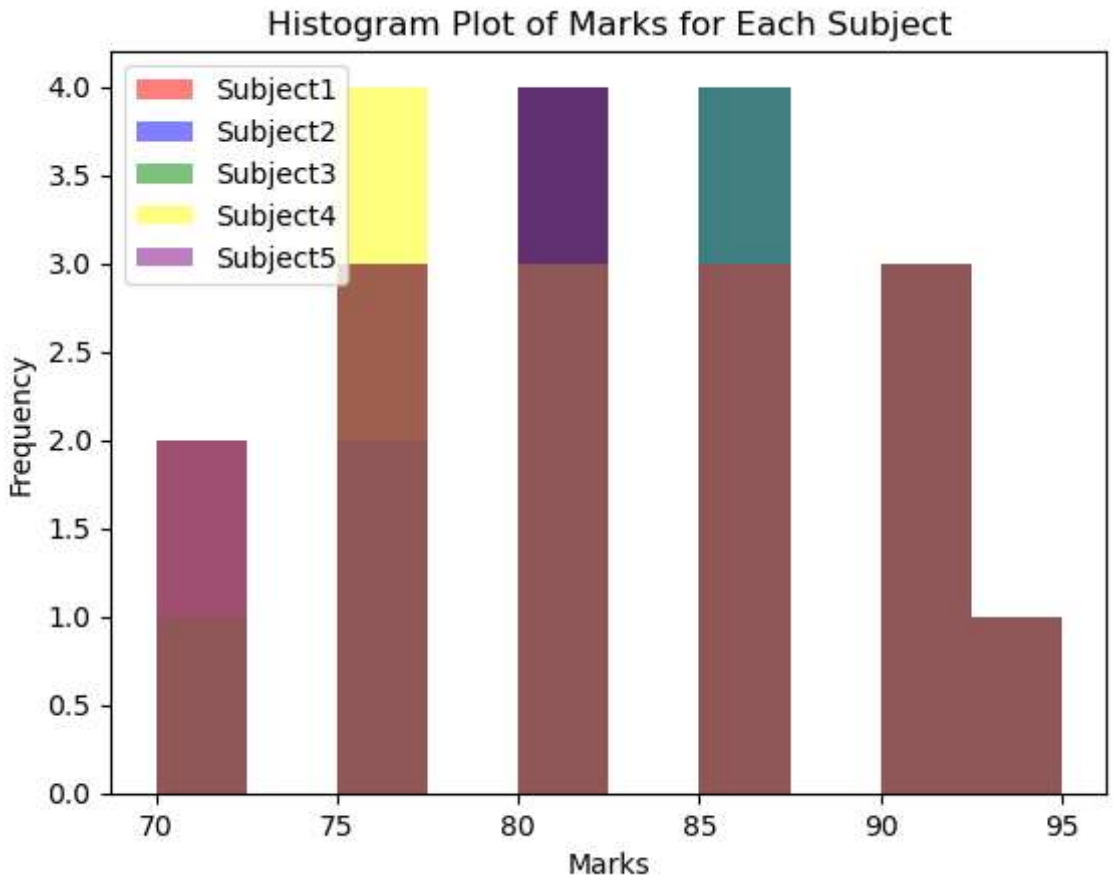
In [28]:
```
import matplotlib.pyplot as plt
import seaborn as np
import numpy as np
import pandas as pd
```

In [29]:
```
import matplotlib.pyplot as plt

# Plotting a histogram for each subject
for column in subjects:
    plt.hist(df[column], bins=10, alpha=0.5, label=column)

plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.title('Histogram Plot of Marks for Each Subject')
plt.legend()
plt.show()
```

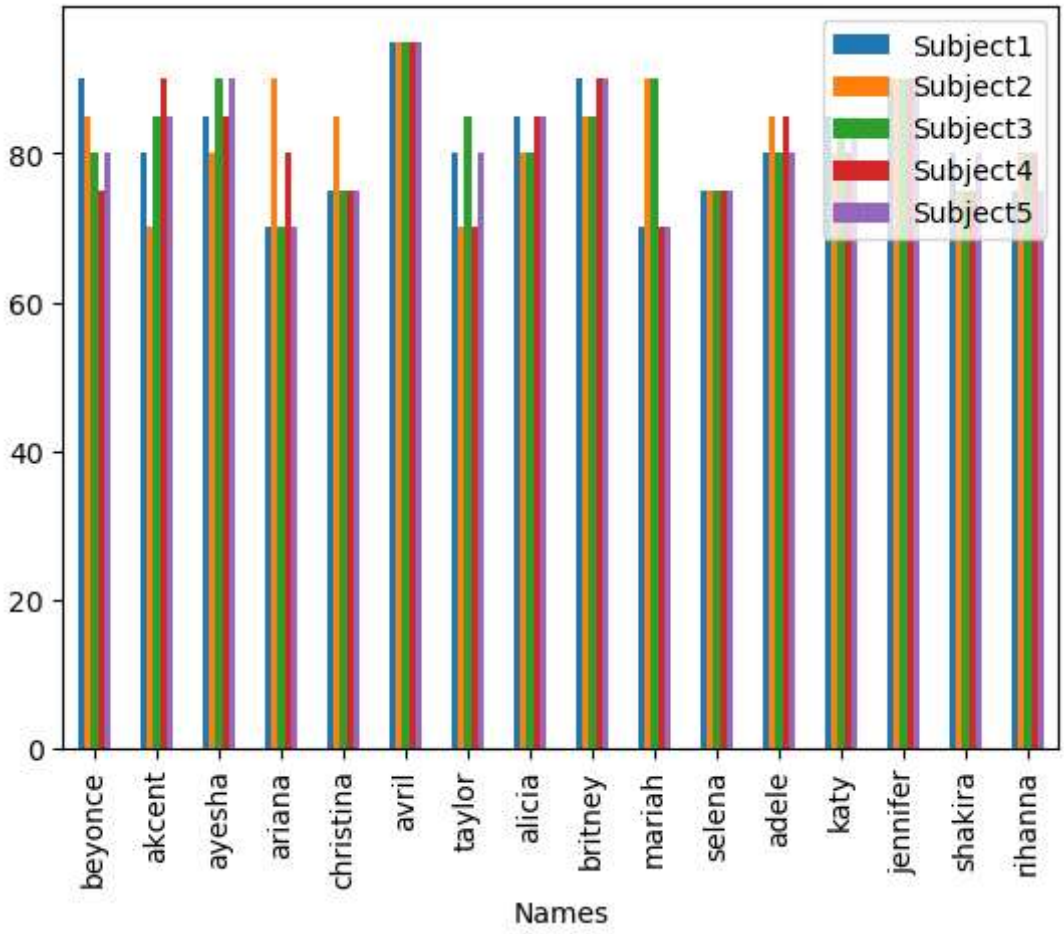


You can specify the color for each histogram plot by passing the color argument to the hist function. Here's an updated version of the code:

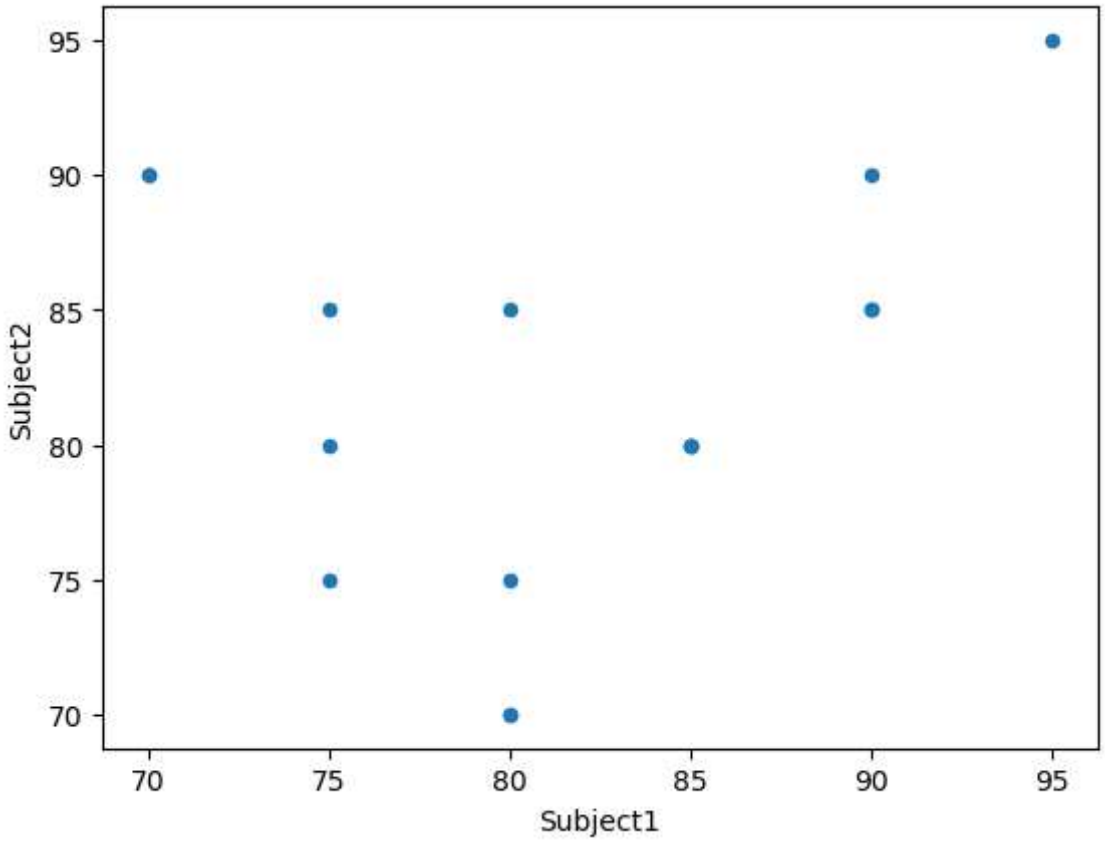
In [30]:
```
import matplotlib.pyplot as plt

# Plotting a histogram for each subject with different colors
colors = ['red', 'blue', 'green', 'yellow', 'purple']
for i, column in enumerate(subjects):
    plt.hist(df[column], bins=10, alpha=0.5, label=column, color=colors[i])

plt.xlabel('Marks')
plt.ylabel('Frequency')
plt.title('Histogram Plot of Marks for Each Subject')
plt.legend()
plt.show()
```



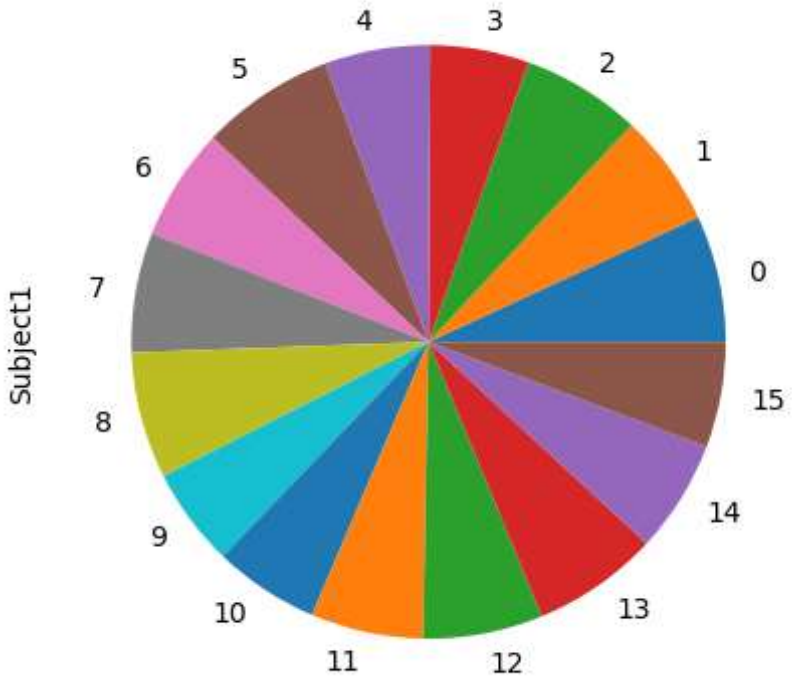
In [31]:
```
# Bar Plot
df.plot(kind='bar', x='Names', y=subjects)
plt.show()
```



In [32]:
```
# Scatter Plot
df.plot(kind='scatter', x='Subject1', y='Subject2')
plt.show()
```

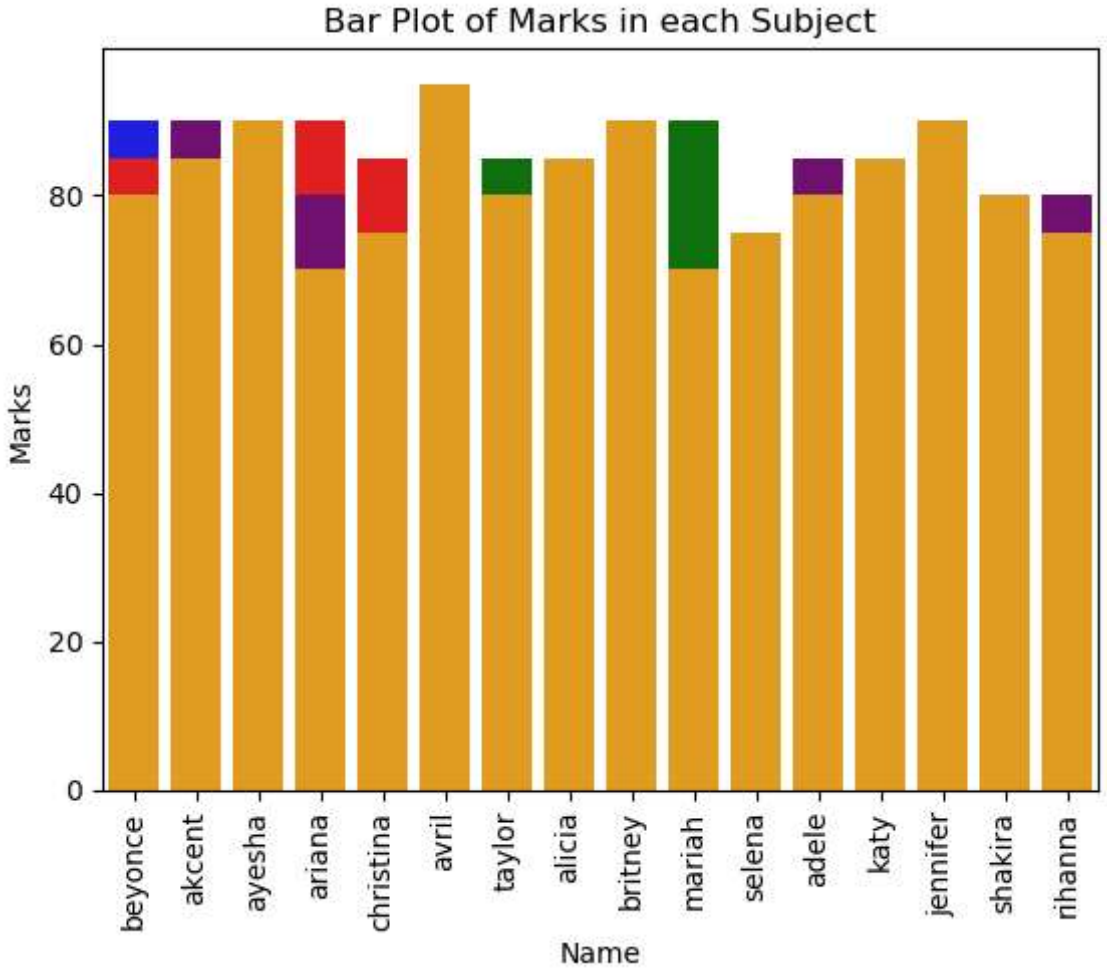


In [33]:
```
# Pie Chart
df['Subject1'].plot(kind='pie')
plt.show()
```

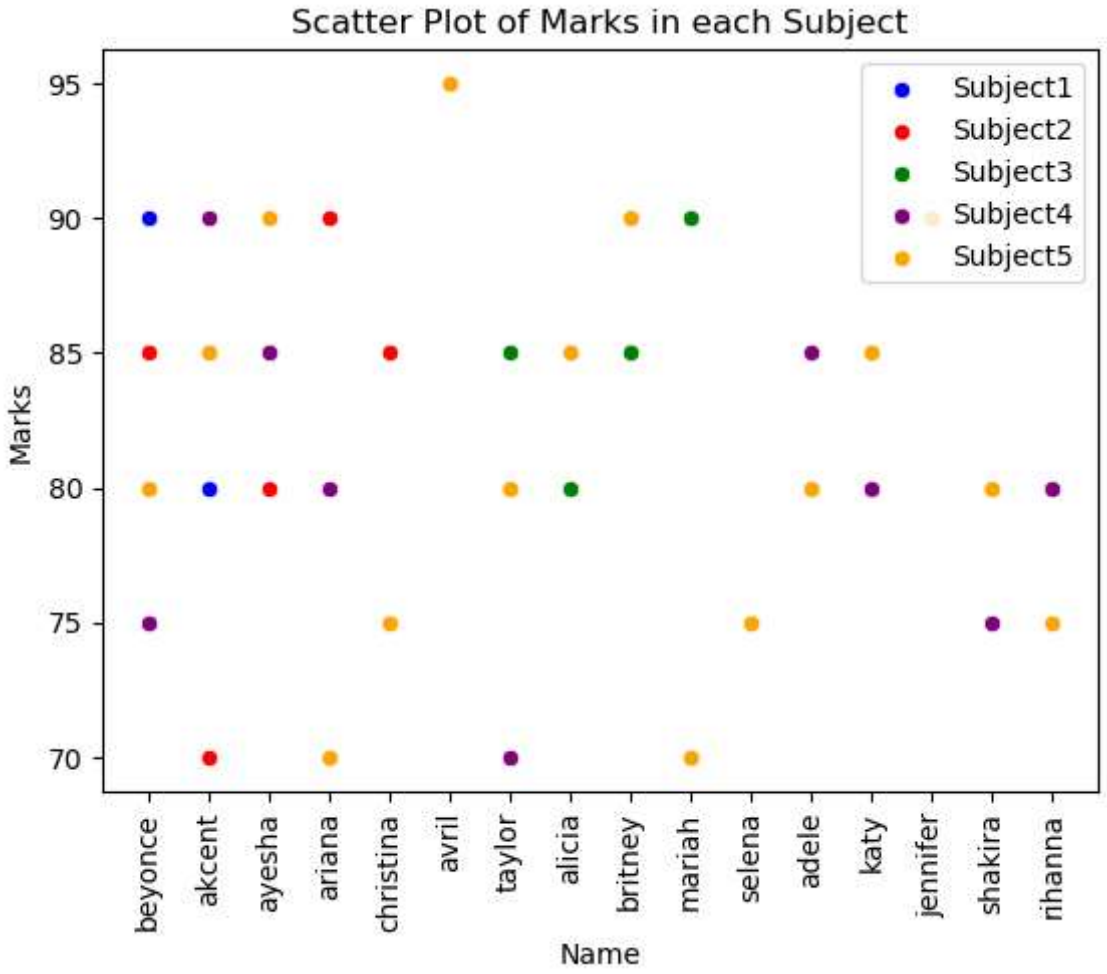


In [34]:

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

# Bar plot for all subjects
sns.barplot(x='Names', y='Subject1', data=df, color='blue')
sns.barplot(x='Names', y='Subject2', data=df, color='red')
sns.barplot(x='Names', y='Subject3', data=df, color='green')
sns.barplot(x='Names', y='Subject4', data=df, color='purple')
sns.barplot(x='Names', y='Subject5', data=df, color='orange')

plt.title("Bar Plot of Marks in each Subject")
plt.xlabel("Name")
plt.ylabel("Marks")
plt.xticks(rotation=90)
plt.show()
```



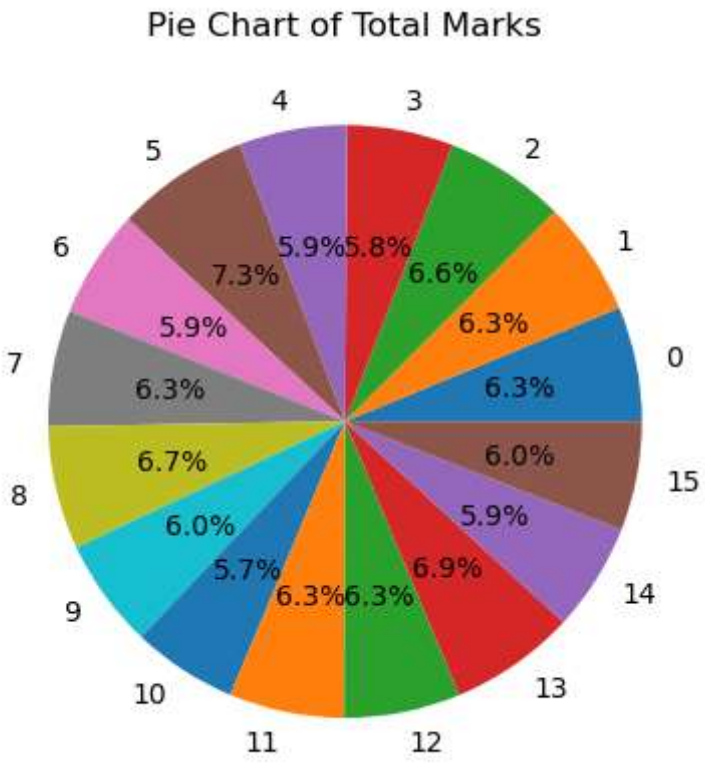
In [35]:
```
# Scatter plot for all subjects
sns.scatterplot(x='Names', y='Subject1', data=df, color='blue', label='Subj
sns.scatterplot(x='Names', y='Subject2', data=df, color='red', label='Subje
sns.scatterplot(x='Names', y='Subject3', data=df, color='green', label='Sub
sns.scatterplot(x='Names', y='Subject4', data=df, color='purple', label='Su
sns.scatterplot(x='Names', y='Subject5', data=df, color='orange', label='Su

plt.title("Scatter Plot of Marks in each Subject")
plt.xlabel("Name")
plt.ylabel("Marks")
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



In [36]:

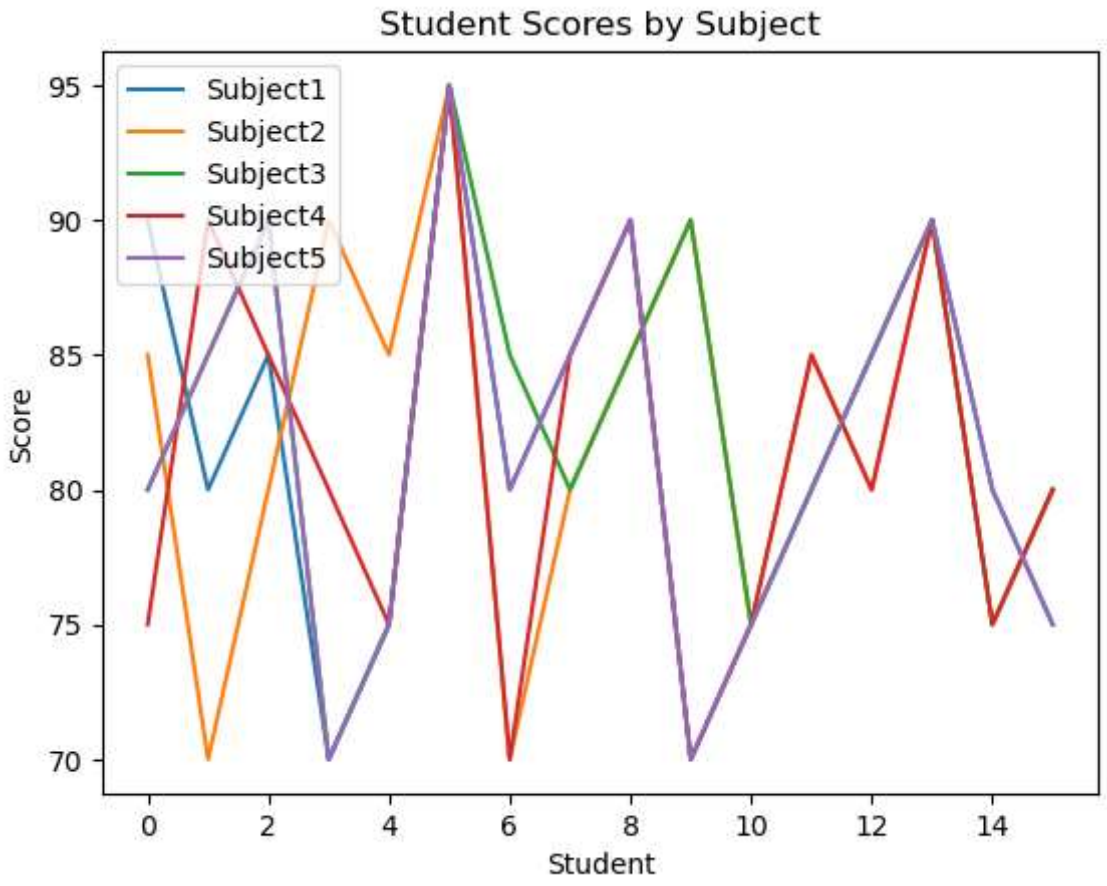
```
# Pie chart for a single student
subject_sum = df.sum(axis=1)
subject_sum.plot.pie(autopct='%1.1f%%')

plt.title("Pie Chart of Total Marks")
plt.xlabel("")
plt.ylabel("")
plt.show()
```

```
C:\Users\varal\AppData\Local\Temp\ipykernel_7312\2694078891.py:2: FutureW
arning: Dropping of nuisance columns in DataFrame reductions (with 'numer
ic_only=None') is deprecated; in a future version this will raise TypeErr
or.  Select only valid columns before calling the reduction.
  subject_sum = df.sum(axis=1)
```



In [37]:

```
import matplotlib.pyplot as plt

# create a list of x values (index of the data frame)
x = range(len(df))

# plot a line chart for each subject
for subject in subjects:
    y = df[subject]
    plt.plot(x, y, label=subject)

# add title, legend, x and y axis labels
plt.title("Student Scores by Subject")
plt.legend(loc="upper left")
plt.xlabel("Student")
plt.ylabel("Score")

# show the plot
plt.show()
```



Handle Missing Values using Pandas dataframe operations

In [38]:
```
import numpy as np
df = pd.DataFrame([[
 np.nan, 2, np.nan, 0],
 [3, 4, np.nan, 1],
 [np.nan, np.nan, np.nan, 5],
 [np.nan, 3, np.nan, 4]],
 columns=list("ABCD"))
print(df)
```

```
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4
```

In [39]:
```
df.isna().sum()
```

Out[39]:
```
A    3
B    1
C    4
D    0
dtype: int64
```

In [40]:
```
df.fillna(0)
```

Out[40]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	0.0	3.0	0.0	4

limit = 2 means only two NA values are filled with 0 or in a row if more than two NA values are present then only the first two Na values are filled with zero if we will pass the limit=2.

In [41]:
```
df.fillna(0,limit=2)
```

Out[41]:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	NaN	5
3	NaN	3.0	NaN	4

In [42]:
```
import numpy as np
df = pd.DataFrame([[
 np.nan, 2, np.nan, 0],
 [3, 4, np.nan, 1],
 [np.nan, np.nan, np.nan, 5],
 [np.nan, 3, np.nan, 4]],
 columns=list("ABCD"))
df1=pd.DataFrame({"E":[2,3,4,5]})
df2=pd.merge(df,df1,left_index=True, right_index=True)

print(df2)
```

```
     A    B   C  D  E
0  NaN  2.0 NaN  0  2
1  3.0  4.0 NaN  1  3
2  NaN  NaN NaN  5  4
3  NaN  3.0 NaN  4  5
```

Here, the left_index and right_index parameters are set to True to indicate that the merge should be based on the indices of the DataFrames rather than a specific column.

In [43]:
```
import numpy as np
import pandas as pd

df = pd.DataFrame([[np.nan, 2, np.nan, 0],
 [3, 4, np.nan, 1],
 [np.nan, np.nan, np.nan, 5],
 [np.nan, 3, np.nan, 4]],
 columns=list("ABCD"))

df1 = pd.DataFrame({'E': [2, 3, 4, 5]})

df2 = pd.merge(df, df1, left_index=True, right_index=True)

new_row = pd.DataFrame({'A': [7], 'B': [8], 'C': [9], 'D': [10], 'E': [11]})

df2 = df2.append(new_row, ignore_index=True)

print(df2)
```

```
     A    B    C   D   E
0  NaN  2.0  NaN   0   2
1  3.0  4.0  NaN   1   3
2  NaN  NaN  NaN   5   4
3  NaN  3.0  NaN   4   5
4  7.0  8.0  9.0  10  11

C:\Users\varal\AppData\Local\Temp\ipykernel_7312\3363002200.py:16: Future
Warning: The frame.append method is deprecated and will be removed from p
andas in a future version. Use pandas.concat instead.
  df2 = df2.append(new_row, ignore_index=True)
```

Drop NA

you can simply drop all the NA values by using the dropna() function.

This will drop all the rows if there is any NA value present.

In [47]:
```
df2.dropna(inplace=True)
```

In [45]:
```
df2
```

Out[45]:

	A	B	C	D	E
4	7.0	8.0	9.0	10	11

In [46]:
```
df2.count()
```

Out[46]:
```
A    1
B    1
C    1
D    1
E    1
dtype: int64
```

In [ ]:
```

```