Day 28- Facilitation Guide
(Pandas Data Analysis)

Index

(1.50 hrs) ILT

I. Recap

In our last session we learned:

- **Read and save data in a csv file:** To read data from a CSV file and save it to another CSV file using Pandas, you can use the pd.read_csv() function to read data from a CSV file and the to_csv() method to save data to a CSV file.
- **Cleaning Data:** Cleaning data is an essential step in data preprocessing and analysis. The Pandas library in Python provides various functions and methods to help you clean and transform your data.

In this session we are going to understand how we can analyze data using pandas and how to use pivot tables in pandas.

Note: Download customer_data.csv file from your LMS

II. Pandas data analysis

Calculating Basic statistical measurement

In the data analysis part, we need to calculate some statistical measurements. For calculating this pandas have multiple useful functions. The first useful function is describe() the function it will display most of the basic statistical measurements.
For this function, you can add .T for transforming the display. It will make it easy to look at when there are multiple columns.

df.describe().T

Example:

```
import pandas as pd

# Provide the path to your CSV file
```

```
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)

print(df.describe().T)
```

Output:

```
                count           mean         std        min          25%  \
Transaction_ID  2512.0  152443.931131  724.580482  151200.00  151815.7500
Age             2470.0      46.637652   18.186277      15.00      32.0000
Referal         2357.0       0.652100    0.476405       0.00       0.0000
Amount_spent    2270.0    1418.422577  878.507451       2.09     678.1925

                     50%          75%        max
Transaction_ID  152443.500  153071.2500  153699.00
Age                 47.000      62.0000      78.00
Referal              1.000       1.0000       1.00
Amount_spent      1341.435    2038.1025    2999.98
```

The above function only shows numerical column information.
count shows how many values are there. mean shows the average value of each column.
std shows the standard deviation of columns, which measures the amount of variation.
min is the minimum value of each column.
25%, 50%, and 75% show total values lie in those groups
max shows maximum values of those columns.

We already know the above code will display only numeric columns and basic statistical information.
For object or category columns we can use describe(include=object) .

df.describe(include=object).T

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
```

```
df = pd.read_csv(csv_file_path)

print(df.describe(include=object).T)
```

Output:

	count	unique	top	freq
Transaction_date	2512	810	8/29/2020	12
Gender	2484	2	Female	1356
Marital_status	2512	2	Married	1473
State_names	2512	50	Illinois	67
Segment	2512	5	Basic	1136
Employees_status	2486	4	Employees	946
Payment_method	2512	3	PayPal	1168

The above information,
count shows how many values are there.
unique is how many values are unique in that column.
top is the highest number of values lying in that category.
freq shows how many values frequently lie on those top values.

We can calculate the mean, median, mode, maximum values, minimum values of individual columns and simply use these functions.

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
# Calculate Mean
mean = df['Age'].mean()

# Calculate Median
median = df['Age'].median()

#Calculate Mode
mode = df['Age'].mode().iloc[0]

# Calculate standard deviation
std = df['Age'].std()
```

```
# Calculate Minimum values
minimum = df['Age'].min()

# Calculate Maximum values
maximum = df.Age.max()

print(f" Mean of Age : {mean}")
print(f" Median of Age : {median}")
print(f" Mode of Age : {mode}")
print(f" Standard deviation of Age : {std:.2f}")
print(f" Maximum of Age : {maximum}")
print(f" Minimum of Age : {minimum}")
```

Output:

```
Mean of Age : 46.63765182186235
Median of Age : 47.0
Mode of Age : 54.0
Standard deviation of Age : 18.19
Maximum of Age : 78.0
Minimum of Age : 15.0
```

What is correlation?

Correlation is a statistical measure that quantifies the degree to which two variables are related or move together. It indicates whether an increase in one variable is associated with an increase or decrease in another variable. Correlation values range from -1 to 1, where:

A correlation of 1 (positive correlation) indicates a perfect positive linear relationship. When one variable increases, the other also increases by a consistent amount.

A correlation of -1 (negative correlation) indicates a perfect negative linear relationship. When one variable increases, the other decreases by a consistent amount.

A correlation of 0 (no correlation) indicates no linear relationship between the variables. Changes in one variable do not affect the other.

In pandas, we can display the correlation of different numeric columns. For this, we can use .corr() function.

Syntax:

.corr()
DataFrame.corr(method='pearson', min_periods=1)

Parameters:

method (optional): Specifies the correlation method to be used. The default method is 'pearson', which calculates the Pearson correlation coefficient. Other options include 'spearman' for Spearman rank correlation and 'kendall' for Kendall Tau rank correlation. You can also pass a callable function that takes two one-dimensional arrays and returns a correlation. If not specified, 'pearson' is used.

min_periods (optional): The minimum number of non-null values required to compute a correlation. By default, it is set to 1, which means that a correlation is computed if there is at least one non-null value in both columns being compared. If you want to require a higher minimum, you can set this parameter to a different value.

Here's a simple example to illustrate correlation:

Let's consider two variables: "Hours Studied" and "Exam Scores" for a group of students.
We want to determine the correlation between the time students spend studying and their exam scores.
We'll assume that a positive correlation exists, meaning that as students study more, their exam scores tend to increase.

```
import pandas as pd

# Sample data
data = {
    'Hours Studied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Exam Scores': [50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Calculate the correlation
correlation = df['Hours Studied'].corr(df['Exam Scores'])
print("Correlation between Hours Studied and Exam Scores:", correlation)
```

Output:

```
Correlation between Hours Studied and Exam Scores: 1.0
```

In this example, we've created a DataFrame with two columns: "Hours Studied" and "Exam Scores." As the number of hours studied increases, we expect students to achieve higher exam scores. Therefore, we anticipate a positive correlation.

Number of unique values in the category column

To display the sum of count of all unique values we use nunique() function on the desired column name.

For example, display total unique values in State_names columns we use this function:

```
# for display how many unique values are there in State_names column
df['State_names'].nunique()
```

Complete Code:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
# Printing Unique values
print(df['State_names'].nunique())
```

Output:

```
50
```

Show all unique values

To display all unique values we use the unique() function with the desired column name.

```
# for display unique values of State_names column
df['State_names'].unique()
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
```

```
# Printing Unique values
print(df['State_names'].unique())
```

Output:

```
['Kansas' 'Illinois' 'New Mexico' 'Virginia' 'Connecticut' 'Hawaii'
 'Florida' 'Vermont' 'California' 'Colorado' 'Iowa' 'South Carolina'
 'New York' 'Maine' 'Maryland' 'Missouri' 'North Dakota' 'Ohio' 'Nebraska'
 'Montana' 'Indiana' 'Wisconsin' 'Alabama' 'Arkansas' 'Pennsylvania'
 'New Hampshire' 'Washington' 'Texas' 'Kentucky' 'Massachusetts' 'Wyoming'
 'Louisiana' 'North Carolina' 'Rhode Island' 'West Virginia' 'Tennessee'
 'Oregon' 'Alaska' 'Oklahoma' 'Nevada' 'New Jersey' 'Michigan' 'Utah'
 'Arizona' 'South Dakota' 'Georgia' 'Idaho' 'Mississippi' 'Minnesota'
 'Delaware']
```

Counts of unique values

To show unique values count we use the value_counts() method. This function will display unique values with a number of each value that occurs. For example, if we want to know how many unique values of Gender columns with value frequency number of then we use this method below.

df['Gender].value_counts()

Example:

```
import pandas as pd
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
# Counts of unique values
print(df['Gender'].value_counts())
```

Output:

```
Gender
Female    1356
Male      1128
Name: count, dtype: int64
```

If we want to show with the percentage of occurrence rather number than we use normalize=True argument in value_counts() function

Calculate percentage of each category
df['Gender'].value_counts(normalize=True)

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
# Calculate the gender wise percentage
print(df['Gender'].value_counts(normalize=True))
```

Output:

```
Gender
Female    0.545894
Male      0.454106
Name: proportion, dtype: float64
```

Sort values

If we want to sort data frames by particular columns, we need to use sort_values() the method. We can use sort by ascending or descending order.
By default, it's in ascending order. If we want to use descending order then simply we need to pass ascending=False argument in sort_values() the function.

Syntax:

```
# Sort Values by City
df.sort_values(by=['State_names']).head(6)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Gender','Age','Marital_status','State_names']
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)

# Sort Values by State Names
print(df.sort_values(by=['State_names']).head(6))
```

Output:

	Gender	Age	Marital_status	State_names
617	Male	43.0	Married	Alabama
1856	Female	18.0	Married	Alabama
2443	Male	60.0	Married	Alabama
2441	Male	18.0	Single	Alabama
1023	Female	30.0	Married	Alabama
1853	Male	18.0	Married	Alabama

For sorting our data frame by Amount_spent with ascending order:

Syntax:

```
# Sort Values Amount Spent with ascending order
df.sort_values(by=['Amount_spent']).head(3)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Transaction_ID','Gender','Age','Marital_status','Payment_method',
'Amount_spent' ]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)
```

```
# Sort Values Amount_spent with ascending order
print(df.sort_values(by=['Amount_spent']).head(10))
```

Output:

	Transaction_ID	Gender	Age	Marital_status	Payment_method	Amount_spent
2468	153656	Female	73.0	Married	PayPal	2.09
568	151756	Male	46.0	Single	PayPal	2.16
2401	153589	Female	60.0	Single	PayPal	2.84
962	152150	Female	56.0	Married	Other	5.31
860	152048	Male	26.0	Single	PayPal	5.55
958	152146	Female	64.0	Married	PayPal	6.79
304	151492	Female	78.0	Single	PayPal	7.44
1304	152492	Male	26.0	Married	PayPal	8.67
2243	153431	Male	32.0	Married	Other	8.79
1178	152366	Female	71.0	Single	PayPal	14.42

For sorting our data frame by Amount_spent with descending order:

Syntax:

```
# Sort Values Age  with descending order
df.sort_values(by=['Amount_spent'], ascending=False).head(3)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Transaction_ID','Gender','Age','Marital_status','Payment_method',
'Amount_spent' ]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)

# Sort Values Amount_spent  with descending order
print(df.sort_values(by=['Amount_spent'], ascending=False).head(10))
```

Output:

	Transaction_ID	Gender	Age	Marital_status	Payment_method	Amount_spent
17	151217	Female	77.0	Married	Card	2999.98
485	151673	Male	65.0	Married	PayPal	2998.62
2279	153467	Female	78.0	Single	PayPal	2997.21
589	151777	Male	51.0	Single	PayPal	2997.15
743	151931	Male	44.0	Single	Card	2996.82
2367	153555	Male	39.0	Single	Card	2995.73
101	151296	Female	33.0	Married	PayPal	2989.33
1254	152442	Female	16.0	Married	Card	2988.13
1123	152311	Male	35.0	Married	Other	2987.96
1177	152365	Female	30.0	Single	PayPal	2985.70

Alternatively, we can use nlargest() and nsmallest() functions for displaying largest and smallest values with desired numbers.
Syntax:

nlargest()

DataFrame.nlargest(n, columns, keep='first')

n: An integer representing the number of largest values you want to retrieve from the Series or DataFrame.

columns: In the DataFrame version, you specify the column(s) to consider when finding the largest values. This can be a single column name or a list of column names.

keep: (Optional) Determines how to handle duplicates if there are multiple values with the same highest value. It can be one of the following:
'first': Keep the first occurrence of duplicates (default).
'last': Keep the last occurrence of duplicates.

nsmallest()

DataFrame.nsmallest(n, columns, keep='first')

n: An integer representing the number of largest values you want to retrieve from the Series or DataFrame.

columns: In the DataFrame version, you specify the column(s) to consider when finding the largest values. This can be a single column name or a list of column names.

keep: (Optional) Determines how to handle duplicates if there are multiple values with the same highest value. It can be one of the following:
'first': Keep the first occurrence of duplicates (default).
'last': Keep the last occurrence of duplicates.

For example, If we want to display the 4 largest Amount_spent rows then we use this:

Syntax:

```
# nlargest
df.nlargest(4, 'Amount_spent ').head(10) # first argument is how many rows you want to display and second one is columns name
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Transaction_ID','Gender','Age','Marital_status','Payment_method', 'Amount_spent' ]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)

# nlargest
print(df.nlargest(4, 'Amount_spent')) # first argument is how many rows you want to display and second one is columns name
```

Output:

	Transaction_ID	Gender	Age	Marital_status	Payment_method	Amount_spent
17	151217	Female	77.0	Married	Card	2999.98
485	151673	Male	65.0	Married	PayPal	2998.62
2279	153467	Female	78.0	Single	PayPal	2997.21
589	151777	Male	51.0	Single	PayPal	2997.15

For 4 smallest Amount_spent rows

```
# nsmallest
df.nsmallest(4, 'Amount_spent')
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
```

```
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Transaction_ID','Gender','Age','Marital_status','Payment_method',
'Amount_spent' ]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)

# nsmallest
print(df.nsmallest(4, 'Amount_spent'))#first argument is how many rows you want to
display and second one is columns name
```

Output:

	Transaction_ID	Gender	Age	Marital_status	Payment_method	Amount_spent
2468	153656	Female	73.0	Married	PayPal	2.09
568	151756	Male	46.0	Single	PayPal	2.16
2401	153589	Female	60.0	Single	PayPal	2.84
962	152150	Female	56.0	Married	Other	5.31

Conditional queries on Data

If we want to apply a single condition then first we will give one condition then we pass the condition on the data frame.

For example, if we want to display top 4 rows where Payment_method is PayPal then we use this:

```
# filtering - Only show Paypal users
condition = df['Payment_method'] == 'PayPal'
df[condition].head(4)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Transaction_ID','Gender','Age','Marital_status','Payment_method',
'Amount_spent' ]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)
```

```
# filtering - Only show Paypal users
condition = df['Payment_method'] == 'PayPal'

# display top 4 rows
print(df[condition].head(4))
```

Output:

	Transaction_ID	Gender	Age	Marital_status	Payment_method	Amount_spent
2	151202	Male	63.0	Married	PayPal	1572.60
5	151205	Male	71.0	Single	PayPal	2922.66
6	151206	Female	34.0	Married	PayPal	1481.42
7	151207	Male	37.0	Married	PayPal	1149.55

We can apply multiple conditional queries like these.

For example, if we want to display all Married females who live in New York then we use the following:

```
# first create 3 condition
female_person = df['Gender'] == 'Female'
married_person = df['Marital_status'] == 'Married'
loc_newyork = df['State_names'] == 'New York'

# we passing condition on our dataframe
df[female_person & married_person & loc_newyork].head(4)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
col=['Gender','State_names','Age','Marital_status','Payment_method', 'Amount_spent'
]
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path,usecols=col)

# first create 3 condition
female_person = df['Gender'] == 'Female'
married_person = df['Marital_status'] == 'Married'
loc_newyork = df['State_names'] == 'New York'
```

```
# we passing condition on our dataframe
# printing top 4 records meeting the above criteria
print(df[female_person & married_person & loc_newyork].head(4))
```

Output:

	Gender	Age	Marital_status	State_names	Payment_method	Amount_spent
164	Female	64.0	Married	New York	PayPal	1581.77
180	Female	20.0	Married	New York	PayPal	2694.20
254	Female	78.0	Married	New York	PayPal	2959.54
282	Female	32.0	Married	New York	Other	522.24

Summarizing or grouping data

Group by
In Pandas, the group by function is more popular in data analysis parts. It allows you to split and group data, apply a function, and combine the results. We can understand this function and use by below example:

Grouping by one column: For example, if we want to find maximum values of Age and Amount_spent by Gender then we can use this:

```
df[['Age', 'Amount_spent']].groupby(df['Gender']).max()
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
print(df[['Age', 'Amount_spent']].groupby(df['Gender']).max())
```

Output:

Gender	Age	Amount_spent
Female	78.0	2999.98
Male	78.0	2998.62

To find mean, count, and max values of Age and Amount_spent by Gender then we can use the agg() function with groupby().

agg()

*DataFrame.agg(func, axis=0, *args, **kwargs)*

func: This is a function or list of functions that you want to apply to the data. You can use built-in functions like sum, mean, max, min, or custom functions that you define.

axis: (Optional) Specifies the axis along which the aggregation is performed. For Series, it is 0 (default) because there is only one dimension. For DataFrame, you can set it to 0 (apply functions column-wise) or 1 (apply functions row-wise).

***args and **kwargs: (Optional)** Additional arguments and keyword arguments that can be passed to the aggregation function(s).

```
# Group by one columns
state_gender_res =
df[['Age','Gender','Amount_spent']].groupby(['Gender']).agg(['count', 'mean', 'max'])
state_gender_res
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
# Group by one columns
print(df[['Age','Gender','Amount_spent']].groupby(['Gender']).agg(['count', 'mean',
'max']))
```

Output:

	Age			Amount_spent		
	count	mean	max	count	mean	max
Gender						
Female	1326	46.864253	78.0	1217	1429.031627	2999.98
Male	1116	46.475806	78.0	1027	1407.716397	2998.62

As can be seen, the output displays the count, mean and max of two fields Age and Amount_spent categorized by Gender.

Grouping by multiple columns:
To find total count, maximum and minimum values of Amount_spent by State_names, Gender, and Payment_method then we can pass these column names under groupby() function and add .agg() with count, mean, max argument.

```
#Group By multiple columns
state_gender_res =
df[['State_names','Gender','Payment_method','Amount_spent']].groupby([
'State_names','Gender', 'Payment_method']).agg(['count', 'min', 'max'])

# fetch top 12 records meeting the above condition
state_gender_res.head(12)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)

#Group By multiple columns
state_gender_res =
df[['State_names','Gender','Payment_method','Amount_spent']].groupby([
'State_names','Gender', 'Payment_method']).agg(['count', 'min', 'max'])

# fetch top 12 records meeting the above condition
print(state_gender_res.head(12))
```

Output:

			Amount_spent		
			count	min	max
State_names	Gender	Payment_method			
Alabama	Female	Card	6	413.11	2749.37
		Other	5	851.25	2789.52
		PayPal	5	77.90	2520.85
	Male	Card	6	221.17	2735.65
		Other	3	459.47	1691.62
		PayPal	10	87.88	2876.36
Alaska	Female	Card	6	141.50	1988.38
		Other	7	489.16	2970.00
		PayPal	8	462.96	2615.89
	Male	Card	1	2497.31	2497.31
		Other	8	588.88	2977.82
		PayPal	8	91.32	1834.95

As can be seen, the data displays the count, min and max of the Amount_spent field categorized first by State, then by Gender and then by payment method.

Cross Tabulation (Cross tab)

Cross tabulation(also referred to as cross tab) is a method to quantitatively analyze the relationship between multiple variables. Also known as contingency tables.

It will help to understand the correlation between different variables.

For creating this table pandas have a built-in function crosstab().

Syntax:

crosstab()

pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)

index: This is typically a Series, array, or a column name from your DataFrame. It represents the variable to be displayed on the rows of the cross-tabulation.

columns: This is another Series, array, or a column name from your DataFrame. It represents the variable to be displayed on the columns of the cross-tabulation.

values: (Optional) An array or a list of column names. If provided, this represents the values to be aggregated based on the intersections of the index and columns variables. The aggregation is determined by the aggfunc parameter.

rownames: (Optional) A list of names for the row index.

colnames: (Optional) A list of names for the column index.

aggfunc: (Optional) A function or a list of functions to be applied to the values. If values is not specified, aggfunc is applied to the entire dataset. Common aggregation functions include 'count' (default), 'sum', 'mean', etc.

margins: (Optional) If set to True, it adds row and column margins, displaying subtotals and the grand total.

margins_name: (Optional) The name to use for the margin label when margins is set to True.

dropna: (Optional) If set to True, it excludes missing values (NaN) from the cross-tabulation.

normalize: (Optional) If set to True, it normalizes the table, showing proportions or percentages instead of counts.

For creating a simple cross tab between Maritatal_status and Payment_method columns we just use crosstab() with both column names.

```
pd.crosstab(df.Marital_status, df.Payment_method)
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
print(pd.crosstab(df.Marital_status, df.Payment_method))
```

Output:

Payment_method	Card	Other	PayPal
Marital_status			
Married	441	362	670
Single	308	233	498

This shows the payment method frequency for the Married and Single individuals.

We can include subtotals by margins parameter:

Syntax:

```
pd.crosstab(df.Marital_status, df.Payment_method, margins=True,
margins_name="Total")
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)

print(pd.crosstab(df.Marital_status, df.Payment_method, margins=True,
margins_name="Total"))
```

Output:

Payment_method	Card	Other	PayPal	Total
Marital_status				
Married	441	362	670	1473
Single	308	233	498	1039
Total	749	595	1168	2512

If We want a display with percentage than normalize=True parameter help
Syntax:

```
pd.crosstab(df.Marital_status, df.Payment_method, normalize=True, margins=True,
margins_name="Total")
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file
```

```
# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
print(pd.crosstab(df.Marital_status, df.Payment_method, normalize=True,
margins=True, margins_name="Total"))
```

Output:

Payment_method	Card	Other	PayPal	Total
Marital_status				
Married	0.175557	0.144108	0.266720	0.586385
Single	0.122611	0.092755	0.198248	0.413615
Total	0.298169	0.236863	0.464968	1.000000

In these cross tab features, we can pass multiple columns names for grouping and analyzing data. For instance, If we want to see how the Payment_method and Employees_status are distributed by Marital_status then we will pass these columns' names in crosstab() function and it will show below.

Syntax:

```
pd.crosstab(df.Marital_status, [df.Payment_method, df.Employees_status])
```

Example:

```
import pandas as pd
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
print(pd.crosstab(df.Marital_status, [df.Payment_method, df.Employees_status]))
```

Output:

```
Payment_method        Card                                        Other  \
Employees_status Employees Unemployment self-employed workers Employees
Marital_status
Married                179          38            81     136       127
Single                 115          37            57      96        95

Payment_method                                      PayPal              \
Employees_status Unemployment self-employed workers Employees Unemployment
Marital_status
Married                    41            79     114       260           67
Single                     17            44      76       170           55

Payment_method
Employees_status self-employed workers
Marital_status
Married                    120     217
Single                     106     159
```

IV. Data Visualization using Panda

Visualization is the key to data analysis. The most popular python package for visualization is matplotlib and seaborn but sometimes pandas will be handy for you. Pandas also provide some visualization plots easily. For the basic analysis part, it will be easy to use. For this section, we are exploring some different types of plots using pandas. Here are the plots.

Line plot

A line plot is the simplest of all graphical plots. A line plot is utilized to follow changes over a continuous-time and show information as a series. Line charts are ideal for comparing multiple variables and visualizing trends for single and multiple variables.

For creating a line plot in pandas we use .plot() function.

Syntax:
plot()

```
DataFrame.plot(
    x=None,           # The column to use for the x-axis
    y=None,           # The column to use for the y-axis
    kind='line',      # The type of plot (e.g., 'line', 'bar', 'scatter', 'hist', 'box', 'pie', etc.)
    ax=None,          # The Matplotlib axis object to use for plotting (optional)
    subplots=False,   # Create separate subplots for each column (default: False)
    layout=None,      # Specify the layout of subplots (e.g., (2, 2))
    title=None,       # Title for the plot (optional)
    xlim=None,        # Set the x-axis limits (e.g., (0, 10))
    ylim=None,        # Set the y-axis limits (e.g., (0, 100))
    logx=False,       # Use a logarithmic scale for the x-axis (default: False)
    logy=False,       # Use a logarithmic scale for the y-axis (default: False)
```

```
    legend=True,        # Show the legend (default: True)
    style=None,         # Line style and marker (e.g., 'ro--')
    colormap=None,      # Colormap for coloring points (e.g., 'viridis')
    table=False,        # Add a table to the plot (default: False)
    yticks=None,        # Specify y-axis tick locations
    xticks=None,        # Specify x-axis tick locations
    secondary_y=False,  # Plot secondary y-axis (default: False)
    mark_right=True,    # Mark ticks and limits on the right axis (default: True)
    **kwargs            # Additional keyword arguments passed to the Matplotlib plot function
)
```

Some of the common parameters include:

x and y: Specify the columns to use for the x-axis and y-axis, respectively. If not provided, the DataFrame's index and all numeric columns will be used.

kind: Specify the type of plot you want to create, such as 'line', 'bar', 'scatter', 'hist', 'box', 'pie', etc.

title: Set the title for the plot.

legend: Control whether to display the legend.

xlim and ylim: Set the limits for the x-axis and y-axis.

logx and logy: Use a logarithmic scale for the x-axis and/or y-axis if set to True.

colormap: Specify the colormap to use for coloring points in the plot.

secondary_y: If set to True, create a secondary y-axis.

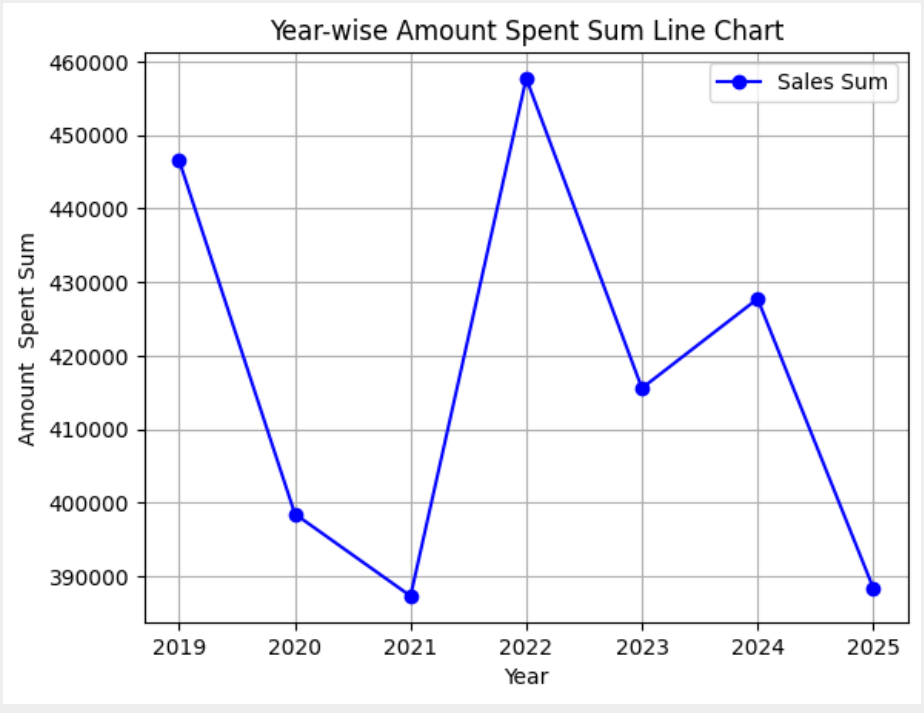
****kwargs:** Additional keyword arguments that can be passed to customize the Matplotlib plot.

For example, we create a line plot from one dummy dataset.
Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
```

```
df = pd.read_csv(csv_file_path)

df.dropna(inplace=True)

df['Date']=pd.to_datetime(df['Transaction_date'], format='%d-%m-%Y')

df['Year'] =  pd.DatetimeIndex(df['Date']).year

yearly_sales = df.groupby('Year')['Amount_spent'].sum().reset_index()

plt.plot(yearly_sales['Year'], yearly_sales['Amount_spent'], marker='o', linestyle='-',
color='b', label='Sales Sum')

plt.xlabel('Year')
plt.ylabel('Amount  Spent Sum')
plt.title('Year-wise Amount Spent Sum Line Chart')

plt.grid(True)
plt.legend()
plt.show()
```

Output:



The above graph shows that maximum purchases have happened in the year 2022. Then there is a sudden dip in purchase indicating a shift in mindset of the customers. Need to deep dive into the cause of the dip in purchase pattern.

Bar plot

A bar plot is also known as a bar chart that shows quantitative or qualitative values for different category items. In a bar, plot data are represented in the form of bars. Bars length or height are used to represent the quantitative value for each item. Bar plot can be plotted horizontally or vertically. For creating these plots look below.

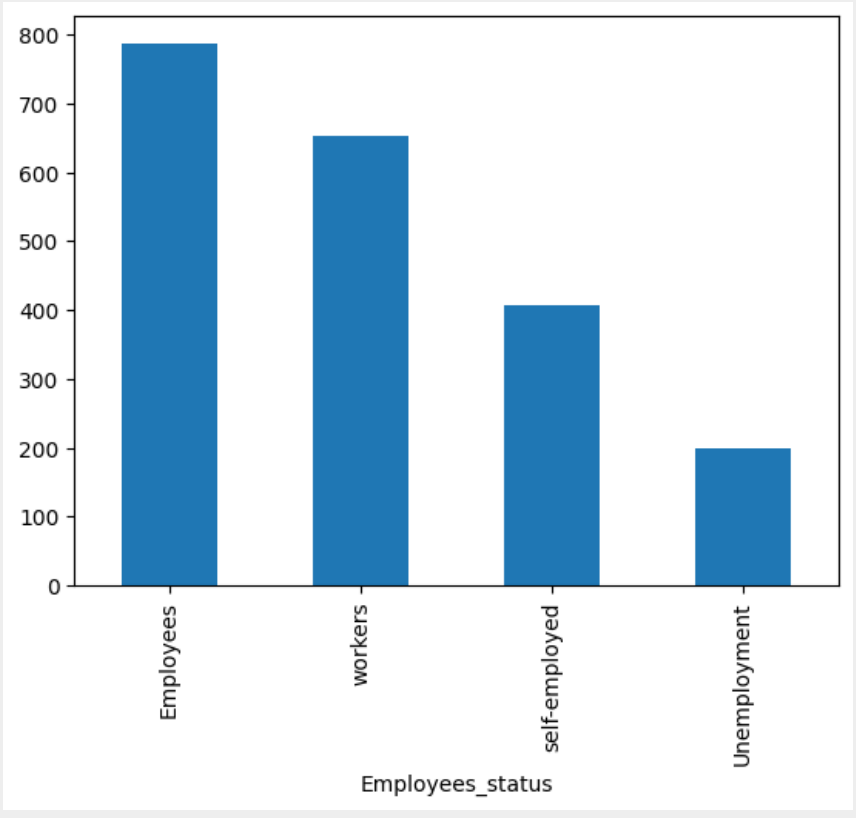
For vertical bar:
Syntax:

```
df['Employees_status'].value_counts().plot(kind='bar');
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)

df['Employees_status'].value_counts().plot(kind='bar');
```

Output:



Conclusion:
From the above chart we can see that in our data set most of the customers are employees.

For horizontal bar:
Syntax:

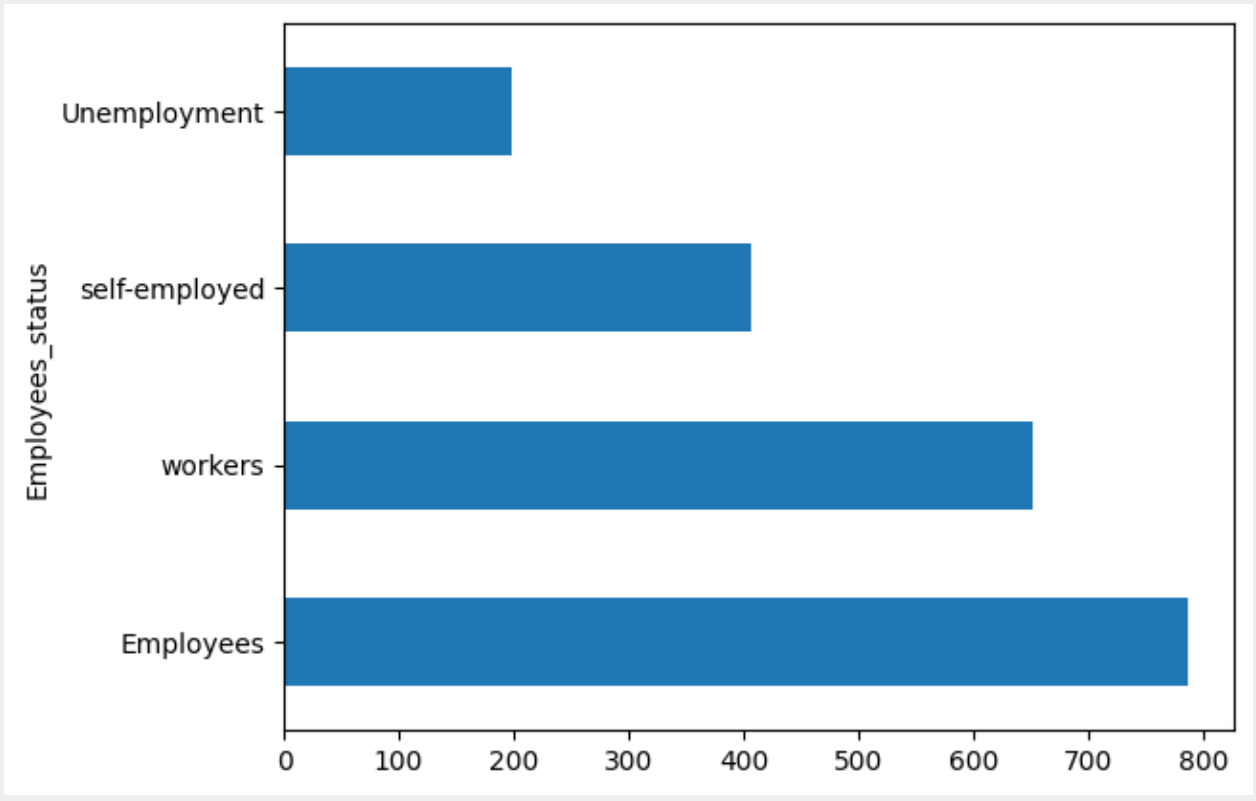
```
df['Employees_status'].value_counts().plot(kind='barh');
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
```

```
df['Employees_status'].value_counts().plot(kind='barh');
```

Output:



Conclusion:
From the above chart we can see that in our data set most of the consumers are employees .

Pie plot

A pie plot is also known as a pie chart. A pie plot is a circular graph that represents the total value with its components. The area of a circle represents the total value and the different sectors of the circle represent the different parts. In this plot, the data are expressed as percentages. Each component is expressed as a percentage of the total value.

In pandas for creating pie plots. We use kind=pie in plot() function in data frame column or series.

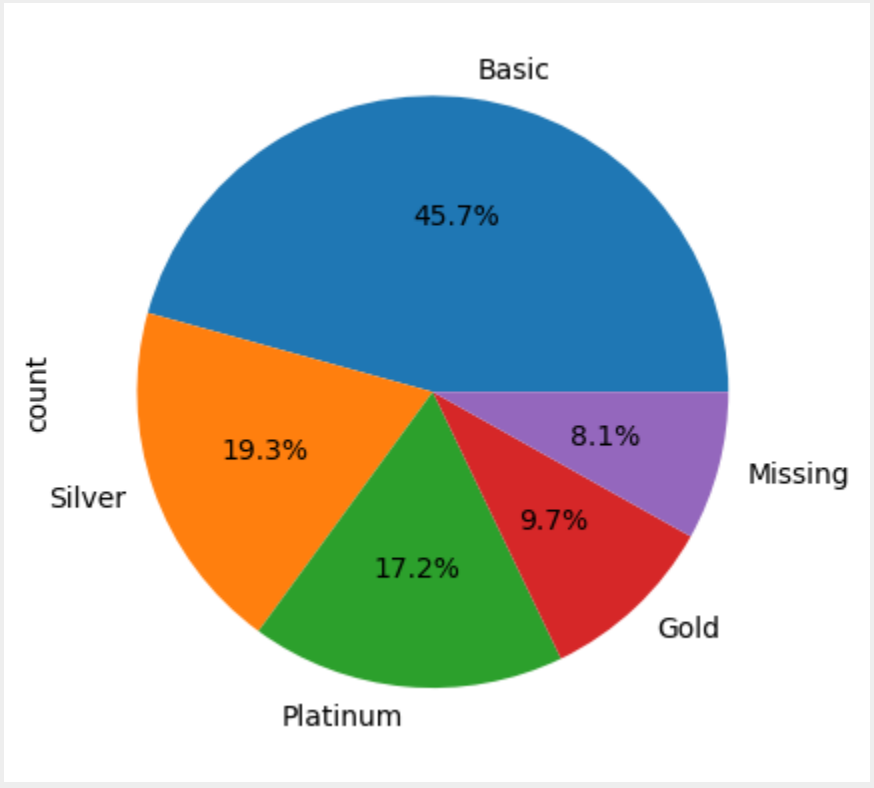
Syntax:

```
df['Segment'].value_counts().plot(kind='pie');
```

Example:

```
import pandas as pd

# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
df['Segment'].value_counts().plot(kind='pie',autopct='%1.1f%%',);
```

Output:



Conclusion:
From the above chart we can see that most of the consumer spent amount under Basic segment

Box Plot

A box plot is also known as a box and whisker plot. This plot is used to show the distribution of a variable based on its quartiles. Box plot displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum. It will also be popular to identify outliers.

We can plot this by one column or multiple columns. For multiple columns, we need to pass column name in the y axis variable as a list.

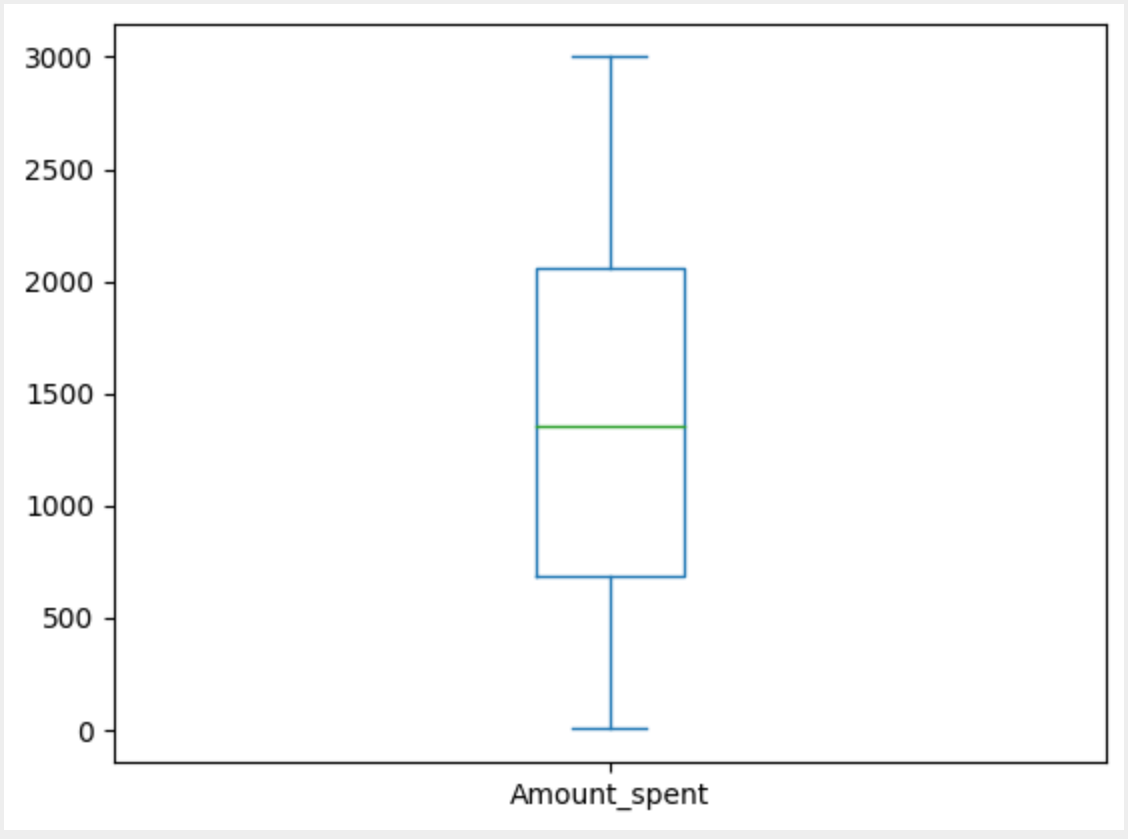
Syntax:

```
df.plot(y=['Amount_spent'], kind='box');
```

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
df.plot(y=['Amount_spent'], kind='box');
```

Output:



Conclusion:
From the above chart we can see that overall spent amounts are equal from the mean value.

In a box plot, we can plot the distribution of categorical variables against a numerical variable and compare them.

Let's plot it with the Employees_status and Amount_spent columns with pandas boxplot() method:

Syntax:

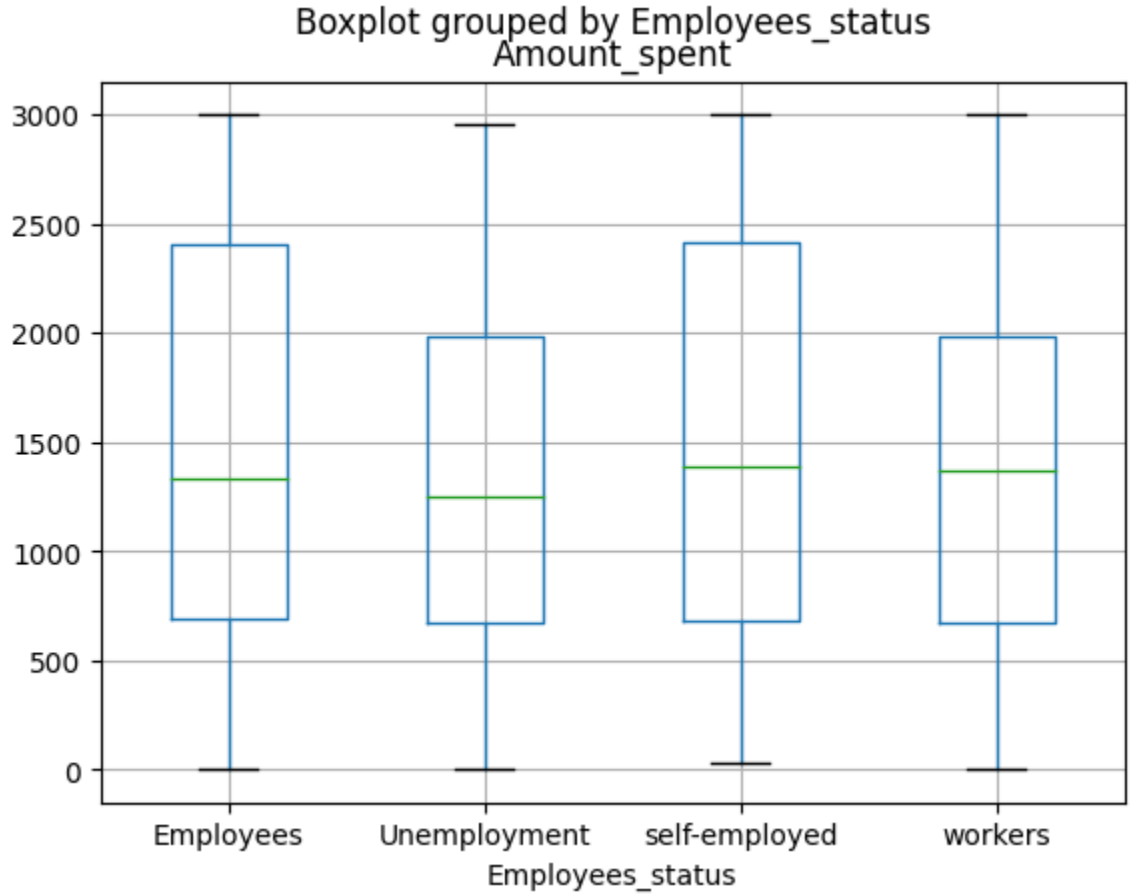
```
df.boxplot(by ='Employees_status', column =['Amount_spent'],ax=ax, grid = False);
```

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
```

```
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
df.boxplot(by ='Employees_status', column =['Amount_spent']);
```

Output:



Conclusion:
From the above we can see that Employees and self-employed spending is higher than unemployment and workers' customers.

Histogram

A histogram shows the frequency and distribution of quantitative measurement across grouped values for data items. It is commonly used in statistics to show how many of a certain type of variable occurs within a specific range or bucket.
Below we will plot a histogram for looking at Age distribution.

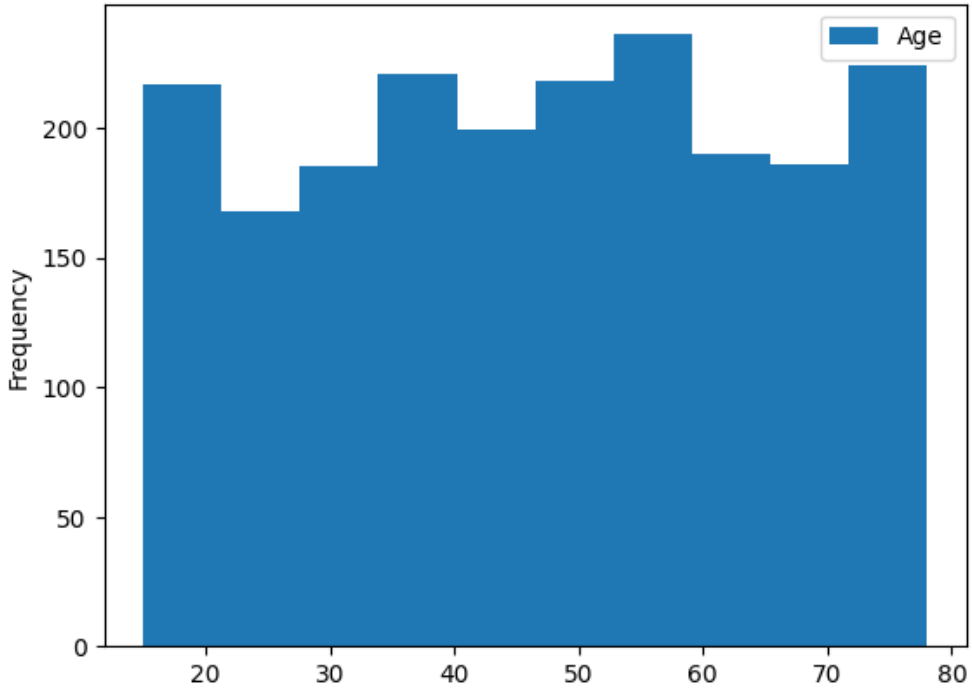
Syntax:

```
df.plot(
    y='Age',
    kind='hist',
    bins=10
);
```

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# Provide the path to your CSV file
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
df.plot(
    y='Age',
    kind='hist',
    bins=10
);
```

Output:



Conclusion:
From the above chart we can see that the distribution of customers within the age group 55 to 60 is highest in our customer spending pattern.

Scatter plot

A scatter plot is used to observe and show relationships between two quantitative variables for different category items.
Each member of the data set gets plotted as a point whose x-y coordinates relate to its values for the two variables.
Below we will plot a scatter plot to display relationships between Age and Amount_spent columns.

Syntax:

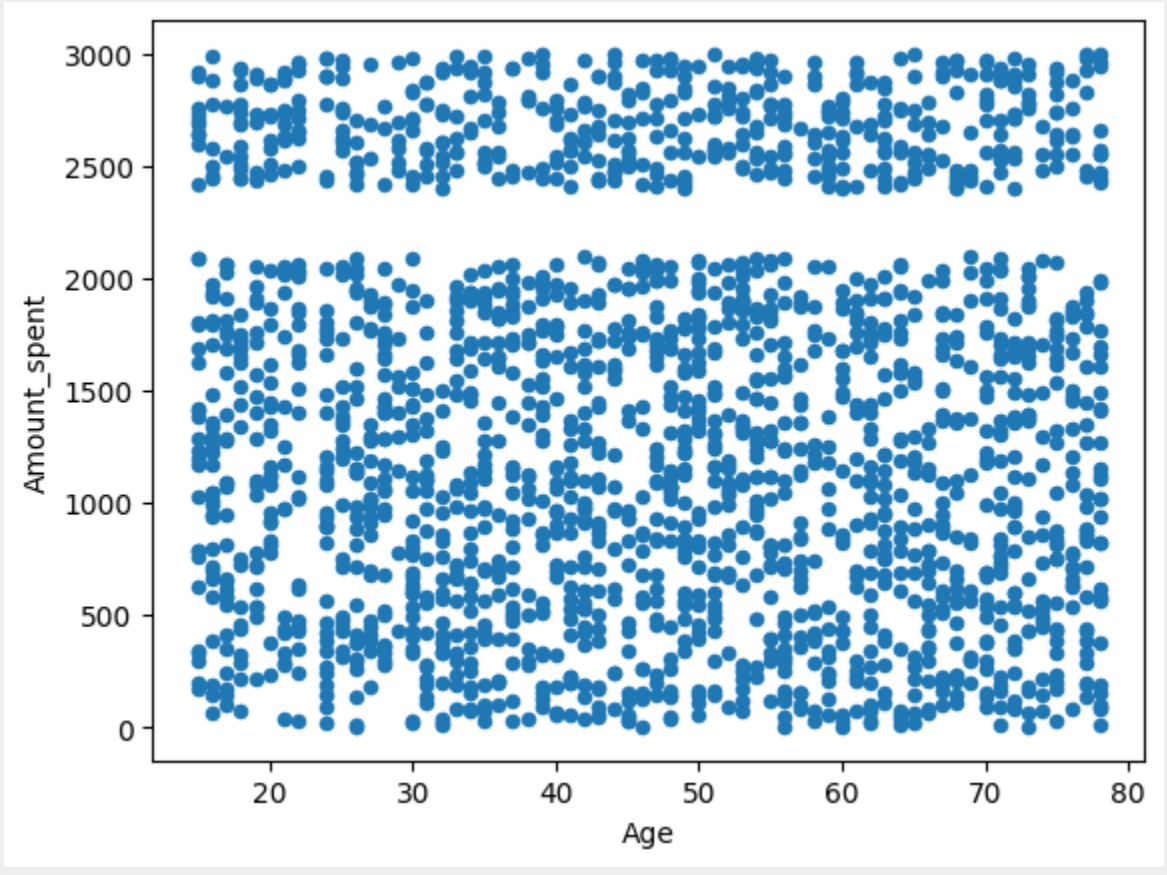
```
df.plot(
    x='Age',
    y='Amount_spent',
    kind='scatter'
);
```

Example:

```
import pandas as pd
# Provide the path to your CSV file
```

```
csv_file_path = 'customer_data.csv'  # Replace with the actual path to your CSV file

# Use read_csv to read the data from the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df.dropna(inplace=True)
df.plot(
    x='Age',
    y='Amount_spent',
    kind='scatter'
);
```

Output:



Conclusion:

According to the above chart we can see that most of the age groups spent amounts below 2000 thousand.

Exercise

Use GPT to write a program:

1.Hi! I have sales data for an online store and want to gain insights from the data.I want to explore how to load, clean, analyze, and visualize the data using Pandas.I want to use some dummy data. Can you please generate a complete code for me?