Pandas Cheat Sheet for Data Science in Python

Last Updated : 01 May, 2024

Pandas is a powerful and versatile library that allows you to work with data in Python. It offers a range of features and functions that make data analysis fast, easy, and efficient. Whether you are a data scientist, analyst, or engineer, Pandas can help you handle large datasets, perform complex operations, and visualize your results.

This Pandas Cheat Sheet is designed to help you master the basics of Pandas and boost your data skills. It covers the most common and useful commands and methods that you need to know when working with data in Python. You will learn how to create, manipulate, and explore data frames, how to apply various functions and calculations, how to deal with missing values and duplicates, how to merge and reshape data, and much more.

If you are new to Data Science using Python and Pandas, or if you want to refresh your memory, this cheat sheet is a handy reference that you can use anytime. It will save you time and effort by providing you with clear and concise examples of how to use Pandas effectively.

Pandas Cheat Sheet

This *Pandas Cheat Sheet* will help you enhance your understanding of the Pandas library and gain proficiency in working with DataFrames, importing/exporting data, performing functions and operations, and utilizing visualization methods to explore DataFrame information effectively.

What is Pandas?

Python's Pandas open-source package is a tool for data analysis and management. It was developed by Wes McKinney and is used in various fields, including data science, finance, and social sciences. Pandas' key features encompass the use of DataFrame and Series objects, efficient indexing capabilities, data alignment, and swift handling of missing data.

Installing Pandas

If you have Python installed, you can use the following command to install Pandas:

```
pip install pandas
```

Importing Pandas

Once Pandas is installed, you can import it into your Python script or Jupyter Notebook using the following import statement:

import pandas as pd

Data Structures in Pandas

Pandas provides two main data structures: Series and DataFrame.

- **Series**: A one-dimensional labelled array capable of holding any data type.
- **DataFrame**: A two-dimensional tabular data structure with labelled axes (rows and columns).

Command	Execution
`Import pandas as pd`	Load the Pandas library as custom defined name pd
`pd.__version__`	Check the Pandas version

Pandas Read and Write to CSV

Command	Execution Tasks
`pd.read_csv('xyz.csv')`	Read the .csv file
`df.to_csv('xyz.csv')`	Save the Pandas data frame as "xyz.csv" form in the current folder

Command	Execution Tasks

`pd.ExcelFile('xyz.xls' )`
`pd.read_excel(file, 'Sheet1')`
	Read the Sheet1 of the Excel file ‘xyz.xls’

`df.to_excel('xyz.xlsx', sheet_name='Sheet1')`	Save the dataset to xyz.xlsx as Sheet1
`pd.read_json('xyz.json')`	Read the xyz.json file
`pd.read_sql('xyz.sql')`	Read the xyz.sql file
`pd.read_html('xyz.html')`	Read the xyz.html file

Create Pandas Series and Dataframe

Command	Execution Tasks
`pd.Series(data=Data)`	Create a Pandas Series with Data like {10: ‘DSA’, 20: ‘ML’, 30: ‘DS’}
`pd.Series(data = ['Geeks','for','geeks'], index = ['A','B','C'])`	Create a Pandas Series and add custom defined index

Command	Execution Tasks
`pd.DataFrame(data)`	Create Pandas Data frame with Data like {‘Fruits’: [‘Mango’, ‘Apple’, ‘Banana’, ‘Orange’], ‘Quantity’: [40, 20, 25, 10], ‘Price’: [80, 100, 50, 70] }
`df.dtypes`	Give Data types
`df.shape`	Give shape of the data
`df['Column_Name'].astype('int32')`	Change the data type to integer 32 bit
`df['Column_Name'].astype('str')`	Change the data type to string
`df['Column_Name'].astype('float')`	Change the data type to float
`df.info()`	Check the data information
`df.values`	Give the data into the NumPy array

Pandas Dataframe

	Fruits	Quantity	Price
0	Mango	40	80
1	Apple	20	100
2	Banana	25	50
3	Orange	10	70

Pandas Sorting, Reindexing, Renaming, Reshaping, Dropping

Sorting by values	
`df.sort_values('Price', ascending=True)`	Sort the values of ‘Price’ of data frame df in Ascending order
`df.sort_values('Price', ascending=False)`	Sort the values of ‘Price’ of data frame df in Descending order
Sorting by Index	
`df.sort_index(ascending=False)`	Sort the index of data frame df in Descending order
Reindexing	

`df.reset_index(drop=True, inplace=True)`	Reset the indexes to default - **inplace = True** # make changes to the original data frame - **drop = True** # Drop the initial indexes, if False then the previous index is assigned in a column.
Renaming	
`df.rename(columns={'Fruits': 'FRUITS', 'Quantity': 'QUANTITY', 'Price': 'PRICE'}, inplace=True)`	Rename the column name with its respective values: In the given code 'Fruits' will be replaced by 'FRUITS', 'Quantity' will be replaced 'QUANTITY' and 'Price' will be replaced by 'PRICE'
Reshaping	
`pd.melt(df)`	Gather columns into rows
`pivot = df.pivot(columns='FRUITS', values=['PRICE', 'QUANTITY'])`	Create a Pivot Table
Dropping	

`df1 = df.drop(columns=['QUANTITY'], axis=1)`	Drop Column • Drop the ‘QUANTITY’ from the data frame df, Here axis = 1 is for the column.
`df2 = df.drop([1, 3], axis=0)`	Drop Rows • Drop 2nd and 4th rows of data frame df, Here axis = 0 is for row

Dataframe Retrieving Series/DataFrame Information and Slicing

Observation	
`df.head()`	Print the first 5 rows
`df.tail()`	Print the last 5 rows
`df.sample(n)`	Select randomly n rows from the data frame df and print it.
`df.nlargest(2, 'QUANTITY')`	Select the largest top 2 rows of the numerical column name ‘QUANTITY’ by its values.
`df.nsmallest(2, 'QUANTITY')`	Select the smallest 2 rows of the numerical column name ‘QUANTITY’ by its values.

<table>
<tr><td colspan="2"><b>Observation</b></td></tr>
<tr><td><code>df[df.PRICE > 50]</code></td><td>Select the rows having ‘PRICE’ values > 50</td></tr>
<tr><td colspan="2"><b>Selection Column data</b></td></tr>
<tr><td><code>df['FRUITS']</code></td><td>Select a single column value with the name of the column I.E ‘FRUITS’</td></tr>
<tr><td><code>df[['FRUITS', 'PRICE']]</code></td><td>Select more than one column with its name.</td></tr>
<tr><td><code>df.filter(regex='F|Q')</code></td><td>Select the column whose names match the patterns of the respective regular expression
I.E ‘FRUITS’ & ‘QUANTITY’</td></tr>
<tr><td colspan="2"><b>Getting Subsets of rows or columns</b></td></tr>
<tr><td><code>df.loc[:, 'FRUITS':'PRICE']</code></td><td>Select all the columns between Fruits and Price</td></tr>
<tr><td><code>df.loc[df['PRICE'] < 70, ['FRUITS', 'PRICE']]</code></td><td>Select FRUITS name having PRICE <70</td></tr>
</table>

Observation	
`df.iloc[2:5]`	Select 2 to 5 rows
`df.iloc[:, [0, 2]]`	Select the columns having 0th & 2nd positions
`df.at[1, 'PRICE']`	Select Single PRICE value at 2nd row of the 'PRICE' column
`df.iat[1, 2]`	Select the single values by their position i.e at the 2nd row and 3rd column.
Filter	
`df.filter(items=['FRUITS', 'PRICE'])`	Filter by column name • Select the 'FRUITS' and 'PRICE' column of the data frame
`df.filter(items=[3], axis=0)`	Filter by row index • Select the 3rd row of the data frame • Here axis = 0 is for row
`df['PRICE'].where(df['PRICE'] > 50)`	Returns a new Series object with the same length as the original 'PRICE' column.

Observation	
	But where() function will replace values where the condition is False with NaN (missing value) or another specified value.
`df.query('PRICE>70')`	Filter a DataFrame based on a specified condition • Return the rows having PRICE > 70

Combine Two data sets:

Merge two data frame	
`pd.merge(df1, df2, how='left', on='Fruits')`	Left Join • Merge the two data frames df1 and df2 based on the ‘Fruits’ column of the left data frame i.e df1
`pd.merge(df1, df2, how='right', on='Fruits')`	Right Join • Merge the two data frames df1 and df2 based on the ‘Fruits’ column of the right data frame i.e df2

`pd.merge(df1, df2, how='inner', on='Fruits')`	Inner Join • Merge the two data frames df1 and df2 based on the common 'Fruits' name of both data frame
`pd.merge(df1, df2, how='outer', on='Fruits')`	Outer Join • Merge the two data frames df1 and df2 based on the common 'Fruits' name • In this case 'Fruits' of both data frames will be arranged accordingly
Concatenation	
`concat_df = pd.concat([df, df1], axis=0, ignore_index=True)`	Row-Wise Concatenation • axis = 0 : denotes that the data frame df and df1 will join vertically • Ignore_index = True : ensures that the resulting DataFrame has a new index, starting from zero and incrementing sequentially • concat_df has the rows of df followed by df1
`concat_df = pd.concat([df, df2], axis=1)`	Row-Wise Concatenation • axis = 1 : denotes that the data frame df and df1 will join horizontally • concat_df has the column of df followed by df2, • If the lengths of the DataFrames don't match, NaN values will be

	assigned to the missing elements.

Data Analysis:

Describe dataset	
`df.describe()`	Descriptive statistics of a data frame Return • count: Number of rows for each numerical column • mean: Average values of each numerical column • std: Standard deviation of each numerical column • min: Minimum value of each numerical column • 25%, 50%, 75%: 25, 50 & 75 percentile of each numerical column • max: Maximum values of each numerical column
`df.describe(include=['O'])`	Descriptive statistics of Object data types of the data frame • include =[‘O’] : Signifies the Object data types column

	• count: Number of rows for each object datatypes column • unique: Count of unique values for each object datatypes column • top: Top row value each object datatypes column • freq: Frequecy of the unique value
`df.FRUITS.unique()`	• Check the unique values of ‘FRUITS’ column in the dataset
`df.FRUITS.value_counts()`	Frequency the unique values in ‘FRUITS’ column
`df['PRICE'].sum()`	Return the sum of ‘PRICE’
`df['PRICE'].cumsum()`	Return the cumulative sum of ‘PRICE’ values
`df['PRICE'].min()`	Return the minimum value of ‘PRICE’ column
`df['PRICE'].max()`	Return the maximum value of ‘PRICE’ column

`df['PRICE'].mean()`	Return the mean value of 'PRICE' column
`df['PRICE'].median()`	Return the median value of 'PRICE' column
`df['PRICE'].var()`	Return the variance value of 'PRICE' column
`df['PRICE'].std()`	Return the standard deviation value of 'PRICE' column
`df['PRICE'].quantile([0.25, 0.75])`	Return the 25 and 75 percentile value of 'PRICE' column

`df.apply(summation)`	Apply any custom function with pandas
	
	`def summation(col):`
	`if col.dtypes != 'int64':`
	`return col.count()`
	`else:`
	`return col.sum()`

`df.cov(numeric_only=True)`	Compute the Covariance for numerical columns

`df.corr(numeric_only=True)`	Compute the Correlation for numerical columns
Missing Values	
`df.isnull()`	Check for null values • Return True or False, Having True means data is missing
`df.isnull().sum()`	Return the row-wise count of null values
`df['DISCOUNT'] = df['DISCOUNT'].fillna(value=VALUE)`	Fill the null values with the specified values ‘VALUE’. The value can be Mean, median, mode or any specified value.
`df1 = df.dropna()`	**Drop the null values**
Add a new column to the Data frame	
`df['COL_NAME'] = COL_DATA`	Add a column to the Existing dataset Note: The length of COL_DATA should be equal to the number of rows of existing dataset

`df = df.assign(Paid_Price=lambda df:`
`(df.QUANTITY *`
`df.PRICE))`
	Add a column using the existing columns values

Group By	
`grouped = df.groupby(by='COL_NAME')`	Group the dataframe w.r.t unique values of the specified column Name i.e ‘COL_NAME’
`grouped.agg(['count','sum', 'mean'])`	Return the count, sum and mean value as per grouped of column i.e ‘COL_NAME’
Graph with Pandas	

`grouped = df.groupby(['Origin'])`
`grouped.sum().plot.pie(y='Paid_Price',`
`subplots=True)`
	Pie Chart
	• Plot the Pie Chart showing group by sum of values in ‘Paid_Price’ as per group of ‘Origin’

`df.plot.scatter(x='PRICE',`
`y='DISCOUNT')`
	Scatter Plot
	• Scatter Plot between ‘PRICE’ and ‘DISCOUNT’

`df.plot.bar(x='FRUITS', y=['QUANTITY',`
`'PRICE', 'DISCOUNT'])`
	Bar Chart
	• Bar chart having horizontal axis with Fruit Names and the

	respective ‘QUANTITY,’PRICE’ and ‘DISCOUNT’ values.
`df['QUANTITY'].plot.hist(bins=3)`	Histogram Plot - Histogram plot of ‘QUANTITY’ column with specified bins value i.e 3 here.
`df.boxplot(column='PRICE', grid=False)`	Box Plot - Box plot of ‘PRICE’ column - It is used for outlier detection

Hands-on Practice on Pandas

Load the pandas libraries

```
import pandas as pd
# Print the Pandas version
print(pd.__version__)
```

Output:

```
1.5.2
```

I/O Pandas Series and Dataframe

Creating Pandas Series.

```
# Create series with Pandas
series = pd.Series(data = ['Geeks','for','geeks'],
                    index = ['A','B','C'])
series
```

Output:

```
A    Geeks
B      for
C    geeks
dtype: object
```

Create Pandas Dataframe

Creating Pandas Dataframe.

```
data = {'Fruits': ['Mango', 'Apple', 'Banana', 'Orange'],
        'Quantity': [40, 20, 25, 10],
        'Price': [80, 100, 50, 70]
        }

# Create Pandas Dataframe with dictionary
df = pd.DataFrame(data)
print(df)
```

Output:

```
   Fruits  Quantity  Price
0   Mango        40     80
1   Apple        20    100
2  Banana        25     50
3  Orange        10     70
```

Check the Data Types

We will check data types with the help of dtypes() function.

```
# check Data types
df.dtypes
```

Output:

```
Fruits       object
Quantity      int64
Price         int64
dtype: object
```

Check the dataframe shape

We will check data types with the help of shape() function.

```
# check the shape of dataset
df.shape
```

Output:

```
(4, 3)
```

Check the data info

df.info() methods return the all information of your dataset.

```
# check info
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, a to d
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Fruits    4 non-null      object
 1   Quantity  4 non-null      int64
 2   Price     4 non-null      int64
dtypes: int64(2), object(1)
memory usage: 128.0+ bytes
```

Change the Data type

```
df['Quantity'] = df['Quantity'].astype('int32')
df['Fruits'] = df['Fruits'].astype('str')
df['Price'] = df['Price'].astype('float')

df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, a to d
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Fruits    4 non-null      object
 1   Quantity  4 non-null      int32
 2   Price     4 non-null      float64
dtypes: float64(1), int32(1), object(1)
memory usage: 112.0+ bytes
```

Print data frame values as NumPy array

```
df.values
```

Output:

```
array([['Mango', 40, 80],
       ['Apple', 20, 100],
       ['Banana', 25, 50],
       ['Orange', 10, 70]], dtype=object)
```

Sorting, Reindexing, Renaming, Reshaping, Dropping

Sorting by values

```
# Sorting in Ascending order
print(df.sort_values('Price', ascending=True))
```

Output:

```
   Fruits  Quantity  Price
c  Banana        25     50
d  Orange        10     70
a   Mango        40     80
b   Apple        20    100
```

```
# Sorting in Descending order
print(df.sort_values('Price', ascending=False))
```

Output:

```
   Fruits  Quantity  Price
b   Apple        20    100
a   Mango        40     80
d  Orange        10     70
c  Banana        25     50
```

Sorting by Index

```
print(df.sort_index(ascending=False))
```

Output:

```
   Fruits  Quantity  Price
d  Orange        10     70
c  Banana        25     50
b   Apple        20    100
a   Mango        40     80
```

Reindexing

```
# Reset the indexes to default
# inplace = True will make changes to the orginal dataframe
# drop =True will drop the initial indexes
df.reset_index(drop=True, inplace=True)
print(df)
```

Output:

```
   Fruits  Quantity  Price
0   Mango        40     80
1   Apple        20    100
2  Banana        25     50
3  Orange        10     70
```

Renaming

```
df.rename(columns={'Fruits': 'FRUITS',
                   'Quantity': 'QUANTITY',
                   'Price': 'PRICE'},
          inplace=True)
print(df)
```

Output:

```
   FRUITS  QUANTITY  PRICE
0   Mango        40     80
1   Apple        20    100
2  Banana        25     50
3  Orange        10     70
```

Reshaping

A. Gather columns into rows.

```
# Gather columns into rows.
print(pd.melt(df))
```

Output:

```
    variable   value
0     FRUITS   Mango
1     FRUITS   Apple
2     FRUITS  Banana
3     FRUITS  Orange
4   QUANTITY      40
5   QUANTITY      20
6   QUANTITY      25
7   QUANTITY      10
8      PRICE      80
9      PRICE     100
10     PRICE      50
11     PRICE      70
```

B. Create a Pivot Table

```
# Pivot table
pivot = df.pivot(columns='FRUITS',
                 values=['PRICE', 'QUANTITY'])
print(pivot)
```

Output:

```
        PRICE                          QUANTITY
FRUITS  Apple Banana Mango Orange     Apple Banana Mango Orange
0         NaN    NaN  80.0    NaN       NaN    NaN  40.0    NaN
1       100.0    NaN   NaN    NaN      20.0    NaN   NaN    NaN
2         NaN   50.0   NaN    NaN       NaN   25.0   NaN    NaN
3         NaN    NaN   NaN   70.0       NaN    NaN   NaN   10.0
```

Dropping

A. Drop column

```
# Drop the DISCOUNT Columns
df1 = df.drop(columns=['QUANTITY'], axis=1)
```

```
print(df1)
```

Output:

```
    FRUITS  PRICE
0    Mango     80
1    Apple    100
2   Banana     50
3   Orange     70
```

B. Drop rows

```
# Drop 2nd and 4th rows
df2 = df.drop([1, 3], axis=0)
print(df2)
```

Output:

```
    FRUITS  QUANTITY  PRICE
0    Mango        40     80
2   Banana        25     50
```

Dataframe Slicing and Observation

A. Observation

We can view top 5 rows with head() methods

```
# Print first 5 rows
print(df.head())
```

Output:

```
    FRUITS  QUANTITY  PRICE
0    Mango        40     80
1    Apple        20    100
2   Banana        25     50
3   Orange        10     70
```

We can view the top last 5 rows with tail() methods.

```
# Print Last 5 rows
print(df.tail())
```

Output:

```
   FRUITS  QUANTITY  PRICE
0   Mango        40     80
1   Apple        20    100
2  Banana        25     50
3  Orange        10     70
```

sample() methods return the ith number of rows.

```
# Randomly select n rows
print(df.sample(3))
```

Output:

```
   FRUITS  QUANTITY  PRICE
2  Banana        25     50
0   Mango        40     80
1   Apple        20    100
```

```
# Select top 2 Highest QUANTITY
print(df.nlargest(2, 'QUANTITY'))
```

Output:

```
   FRUITS  QUANTITY  PRICE
0   Mango        40     80
2  Banana        25     50
```

```
# Select Least 2 QUANTITY
print(df.nsmallest(2, 'QUANTITY'))
```

Output:

```
   FRUITS  QUANTITY  PRICE
3  Orange        10     70
1   Apple        20    100
```

```
# Select the price > 50
print(df[df.PRICE > 50])
```

Output:

```
   FRUITS  QUANTITY  PRICE
0   Mango        40     80
1   Apple        20    100
3  Orange        10     70
```

B. Select Column data

```
# Select the FRUITS name
print(df['FRUITS'])
```

Output:

```
0      Mango
1      Apple
2     Banana
3     Orange
Name: FRUITS, dtype: object
```

```
# Select the FRUITS name and
# their corresponding PRICE
print(df[['FRUITS', 'PRICE']])
```

Output:

```
   FRUITS  PRICE
0   Mango     80
1   Apple    100
2  Banana     50
3  Orange     70
```

```
# Select the columns whose names match
# the regular expression
print(df.filter(regex='F|Q'))
```

Output:

```
   FRUITS  QUANTITY
0   Mango        40
1   Apple        20
2  Banana        25
3  Orange        10
```

C. Subsets of rows or columns

```
# Select all the columns between Fruits and Price
print(df.loc[:, 'FRUITS':'PRICE'])
```

Output:

```
   FRUITS  QUANTITY  PRICE
0   Mango        40     80
1   Apple        20    100
2  Banana        25     50
3  Orange        10     70
```

```
# Select FRUITS name having PRICE <70
print(df.loc[df['PRICE'] < 70,
             ['FRUITS', 'PRICE']])
```

Output:

```
   FRUITS  PRICE
2  Banana     50
```

```
# Select 2:5 rows
print(df.iloc[2:5])
```

Output:

```
   FRUITS  QUANTITY  PRICE
2  Banana        25     50
3  Orange        10     70
```

```
# Select the columns having ) 0th & 2nd positions
print(df.iloc[:, [0, 2]])
```

Output:

```
   FRUITS  PRICE
0   Mango     80
1   Apple    100
2  Banana     50
3  Orange     70
```

For more please refer to this article Indexing and Selecting data

Dataframe

	FRUITS	QUANTITY	PRICE
0	Mango	40	80
1	Apple	20	100
2	Banana	25	50
3	Orange	10	70

```
# Select Single PRICE value at 2nd Postion
df.at[1, 'PRICE']
```

Output:

```
100
```

```
# Select the single values by their position
df.iat[1, 2]
```

Output:

```
100
```

Filter

Filter by column name

```
print(df.filter(items=['FRUITS', 'PRICE']))
```

Output:

```
   FRUITS  PRICE
0   Mango     80
1   Apple    100
2  Banana     50
3  Orange     70
```

Filter by row index

```
# Filter by row index
print(df.filter(items=[3], axis=0))
```

Output:

```
   FRUITS  QUANTITY  PRICE
3  Orange        10     70
```

Where

```
df['PRICE'].where(df['PRICE'] > 50)
```

Output:

```
0     80.0
1    100.0
2      NaN
3     70.0
4     60.0
5      NaN
Name: PRICE, dtype: float64
```

Query

Pandas query() methods return the filtered data frame.

```
# QUERY
print(df.query('PRICE>70'))
```

Output:

```
  FRUITS  QUANTITY  PRICE
0  Mango        40     80
1  Apple        20    100
```

```
# Price >50 & QUANTITY <30
print(df.query('PRICE>50 and QUANTITY<30'))
```

Output:

```
    FRUITS  QUANTITY  PRICE
1    Apple        20    100
3   Orange        10     70
```

```
# FRUITS name start with 'M'
print(df.query("FRUITS.str.startswith('M')", ))
```

Output:

```
  FRUITS  QUANTITY  PRICE
0  Mango        40     80
```

Combine Two data sets

Create 1st dataframe

```
df1 = pd.DataFrame({'Fruits': ['Mango', 'Banana',
                              'Grapes', 'Apple',
                              'Orange'],
                    'Price':  [60, 40, 75, 100, 65]})
print(df1)
```

Output:

```
    Fruits  Price
0    Mango     60
1   Banana     40
2   Grapes     75
3    Apple    100
4   Orange     65
```

Create second dataframe

```
df2 = pd.DataFrame({'Fruits': ['Apple', 'Orange',
                              'Papaya',
                              'Pineapple', 'Mango', ],
                    'Price':  [120, 60, 30, 70, 50]})
print(df2)
```

Output:

```
       Fruits  Price
0       Apple    120
1      Orange     60
2      Papaya     30
```

```
3  Pineapple     70
4      Mango     50
```

Merge two dataframe

A. Left Join

```
print(pd.merge(df1, df2,
               how='left', on='Fruits'))
```

Output:

```
   Fruits  Price_x  Price_y
0   Mango       60     50.0
1  Banana       40      NaN
2  Grapes       75      NaN
3   Apple      100    120.0
4  Orange       65     60.0
```

B. Right Join

```
print(pd.merge(df1, df2,
               how='right', on='Fruits'))
```

Output:

```
      Fruits  Price_x  Price_y
0      Apple    100.0      120
1     Orange     65.0       60
2     Papaya      NaN       30
3  Pineapple      NaN       70
4      Mango     60.0       50
```

C. Inner Join

```
print(pd.merge(df1, df2,
               how='inner', on='Fruits'))
```

Output:

```
   Fruits  Price_x  Price_y
0   Mango       60       50
1   Apple      100      120
2  Orange       65       60
```

D. Outer Join

```
print(pd.merge(df1, df2,
               how='outer', on='Fruits'))
```

Output:

```
      Fruits  Price_x  Price_y
0      Mango     60.0     50.0
1     Banana     40.0      NaN
2     Grapes     75.0      NaN
3      Apple    100.0    120.0
4     Orange     65.0     60.0
5     Papaya      NaN     30.0
6  Pineapple      NaN     70.0
```

Concatenation

A. Row-wise Concatenation having the same column name

```
data = {'FRUITS': ['Grapes', 'Pineapple'],
        'QUANTITY': [23, 17],
        'PRICE': [60, 30]
        }

# Create Pandas Dataframe with dictionary
df1 = pd.DataFrame(data)

# Concatenate df and df1
df2 = pd.concat([df, df1], axis=0,
                ignore_index=True)
print(df2)
```

Output:

```
      FRUITS  QUANTITY  PRICE
0      Mango        40     80
1      Apple        20    100
2     Banana        25     50
3     Orange        10     70
4     Grapes        23     60
5  Pineapple        17     30
```

B. Column-wise Concatenation having the same column name

```
data = {'DISCOUNT': [5, 7, 10, 8, 6]}

# Create Pandas Dataframe with dictionary
```

```
discount = pd.DataFrame(data)

# Concatenate df2 and discount
df = pd.concat([df2, discount], axis=1)
print(df)
```

Output:

```
     FRUITS  QUANTITY  PRICE  DISCOUNT
0     Mango        40     80       5.0
1     Apple        20    100       7.0
2    Banana        25     50      10.0
3    Orange        10     70       8.0
4    Grapes        23     60       6.0
5 Pineapple        17     30       NaN
```

Descriptive Analysis Pandas

Describe dataset

A. For numerical datatype

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

Output:

```
       QUANTITY       PRICE   DISCOUNT
count   6.00000    6.000000   5.000000
mean   22.50000   65.000000   7.200000
std    10.05485   24.289916   1.923538
min    10.00000   30.000000   5.000000
25%    17.75000   52.500000   6.000000
50%    21.50000   65.000000   7.000000
75%    24.50000   77.500000   8.000000
max    40.00000  100.000000  10.000000
```

B. For object datatype

```
print(df.describe(include=['O']))
```

Output:

```
       FRUITS
count       6
unique      6
top     Mango
freq        1
```

Unique values

```
# Check the unique values in the dataset
df.FRUITS.unique()
```

Output:

```
array(['Mango', 'Apple', 'Banana', 'Orange', 'Grapes',
'Pineapple'],
      dtype=object)
```

```
# Count the total unique values
df.FRUITS.value_counts()
```

Output:

```
Mango        1
Apple        1
Banana       1
Orange       1
Grapes       1
Pineapple    1
Name: FRUITS, dtype: int64
```

Sum values

```
print(df['PRICE'].sum())
```

Output:

```
360
```

Cumulative Sum

```
print(df['PRICE'].cumsum())
```

Output:

```
0     80
1    180
2    230
3    300
4    360
Name: PRICE, dtype: int64
```

Minimum Values

```
# Minimumn PRICE
df['PRICE'].min()
```

Output:

```
30
```

Maximum Values

```
# Maximum PRICE
df['PRICE'].max()
```

Output:

```
100
```

Mean

```
# Mean PRICE
df['PRICE'].mean()
```

Output:

```
65.0
```

Median

```
# Median PRICE
df['PRICE'].median()
```

Output:

```
65.0
```

Variance

```
# Variance
df['PRICE'].var()
```

Output:

```
590.0
```

Standard Deviation

```
# Stardard Deviation
df['PRICE'].std()
```

Output:

```
24.289915602982237
```

Quantile

```
# Quantile
df['PRICE'].quantile([0, 0.25, 0.75, 1])
```

Output:

```
0.00     30.0
0.25     52.5
0.75     77.5
1.00    100.0
Name: PRICE, dtype: float64
```

Apply any custom function

```
# Apply any custom function
def summation(col):
    if col.dtypes != 'int64':
        return col.count()
    else:
        return col.sum()


df.apply(summation)
```

Output:

```
FRUITS         6
QUANTITY     135
PRICE        390
DISCOUNT       5
dtype: int64
```

Covariance

```
print(df.cov(numeric_only=True))
```

Output:

```
          QUANTITY  PRICE  DISCOUNT
QUANTITY     101.1   53.0     -10.4
PRICE         53.0  590.0     -18.0
DISCOUNT     -10.4  -18.0       3.7
```

Correlation

```
print(df.corr(numeric_only=True))
```

Output:

```
          QUANTITY     PRICE  DISCOUNT
QUANTITY  1.000000  0.217007 -0.499210
PRICE     0.217007  1.000000 -0.486486
DISCOUNT -0.499210 -0.486486  1.000000
```

Missing Values

Check for null values using isnull() function.

```
# Check for null values
print(df.isnull())
```

Output:

```
   FRUITS  QUANTITY  PRICE  DISCOUNT
0   False     False  False     False
1   False     False  False     False
2   False     False  False     False
3   False     False  False     False
4   False     False  False     False
5   False     False  False      True
```

Column-wise null values count

```
# Total count of null values
print(df.isnull().sum())
```

Output:

```
FRUITS      0
QUANTITY    0
PRICE       0
DISCOUNT    1
dtype: int64
```

Fill the null values with mean()

```
Mean = df.DISCOUNT.mean()

# Fill the null values
df['DISCOUNT'] = df['DISCOUNT'].fillna(Mean)
print(df)
```

Output:

```
      FRUITS  QUANTITY  PRICE  DISCOUNT
0      Mango        40     80       5.0
1      Apple        20    100       7.0
2     Banana        25     50      10.0
3     Orange        10     70       8.0
```

```
4     Grapes        23     60        6.0
5  Pineapple        17     30        7.2
```

We can also drop null values rows using the below command

```
# Drop the null values
df.dropna(inplace=True)
```

Add a column to the Existing dataset

```
# Values to add
Origin = pd.Series(data=['BH', 'J&K',
                         'BH', 'MP',
                         'WB', 'WB'])

# Add a column in dataset
df['Origin'] = Origin
print(df)
```

Output:

```
      FRUITS  QUANTITY  PRICE  DISCOUNT Origin
0      Mango        40     80       5.0     BH
1      Apple        20    100       7.0    J&K
2     Banana        25     50      10.0     BH
3     Orange        10     70       8.0     MP
4     Grapes        23     60       6.0     WB
5  Pineapple        17     30       NaN     WB
```

Add a column using the existing columns values

```
# Add a column using the existing columns values
df = df.assign(Paid_Price=lambda df:
               (df.QUANTITY * df.PRICE)\
               -(df.QUANTITY * df.PRICE)\
               *df.DISCOUNT/100)
print(df)
```

Output:

```
      FRUITS  QUANTITY  PRICE  DISCOUNT Origin  Paid_Price
0      Mango        40     80       5.0     BH      3040.0
1      Apple        20    100       7.0    J&K      1860.0
2     Banana        25     50      10.0     BH      1125.0
```

```
3      Orange        10      70         8.0      MP        644.0
4      Grapes        23      60         6.0      WB       1297.2
5   Pineapple        17      30         NaN      WB          NaN
```

Group By

Group the DataFrame by the 'Origin' column using groupby() methods

```
# Group the DataFrame by 'Origin' column
grouped = df.groupby(by='Origin')

# Compute the sum as per Origin State
# All the above function can be
# applied here like median, std etc
print(grouped.agg(['sum', 'mean']))
```

Output:

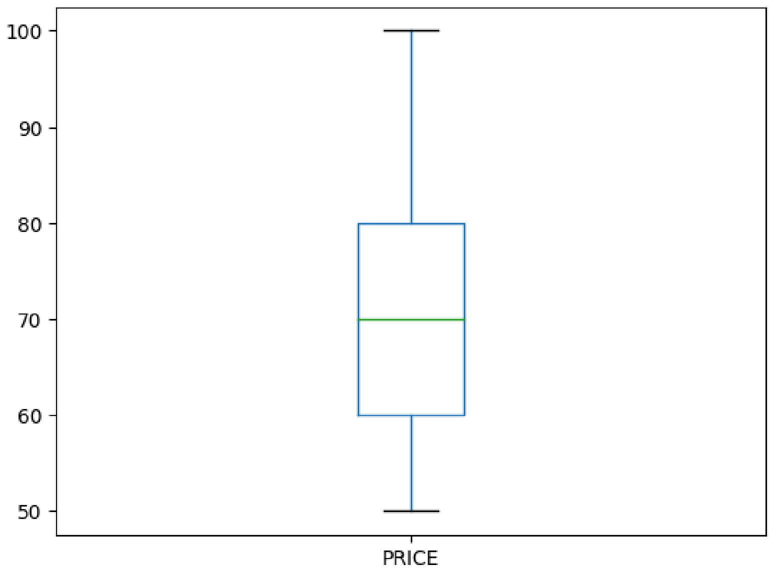
```
        QUANTITY       PRICE          DISCOUNT       Paid_Price
             sum  mean   sum   mean        sum mean          sum
mean
Origin
BH            65  32.5   130   65.0       15.0  7.5       4165.0
2082.5
J&K           20  20.0   100  100.0        7.0  7.0       1860.0
1860.0
MP            10  10.0    70   70.0        8.0  8.0        644.0
644.0
WB            40  20.0    90   45.0        6.0  6.0       1297.2
1297.2
```

Outlier Detection using Box plot

we can use a boxplot for Detection of the outliers.

```
# Box plot
df.boxplot(column='PRICE', grid=False)
```

Output:

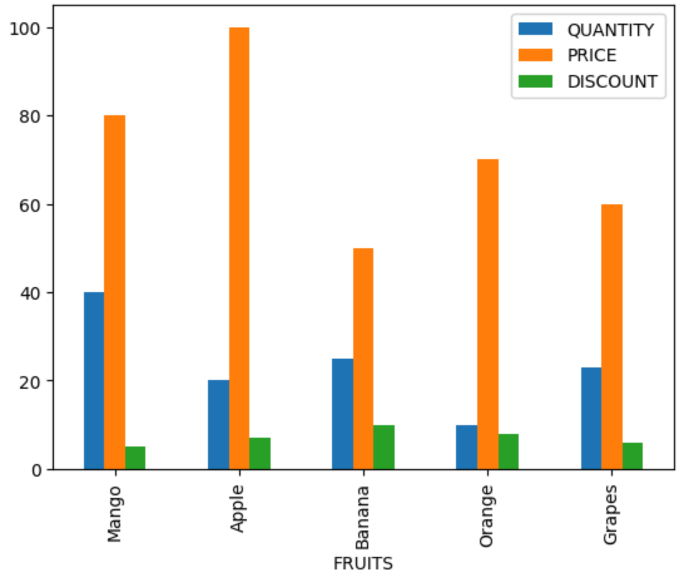


Bar Plot with Pandas

plot.bar() method is used to plot bar in pandas.

```
df.plot.bar(x='FRUITS', y=['QUANTITY', 'PRICE', 'DISCOUNT'])
```

Output:

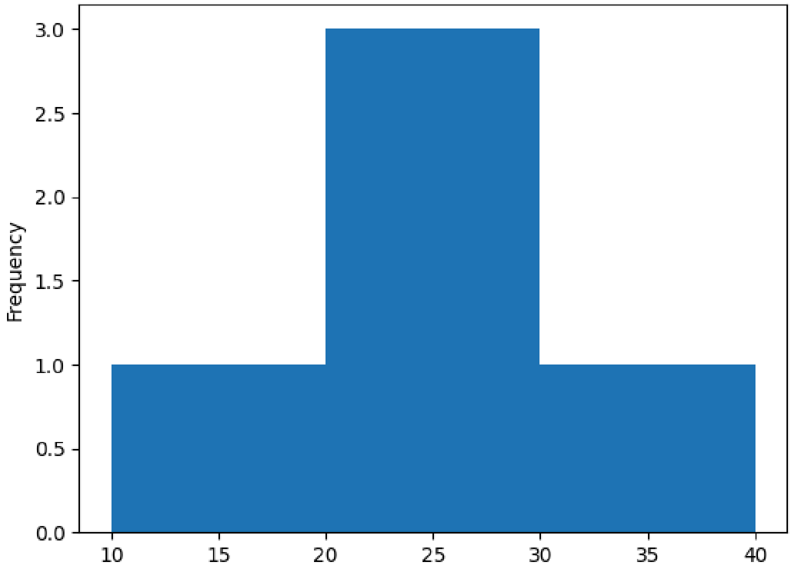


Histogram with pandas

plot.hist() methods is used to create a histogram.

```
df['QUANTITY'].plot.hist(bins=3)
```

Output:

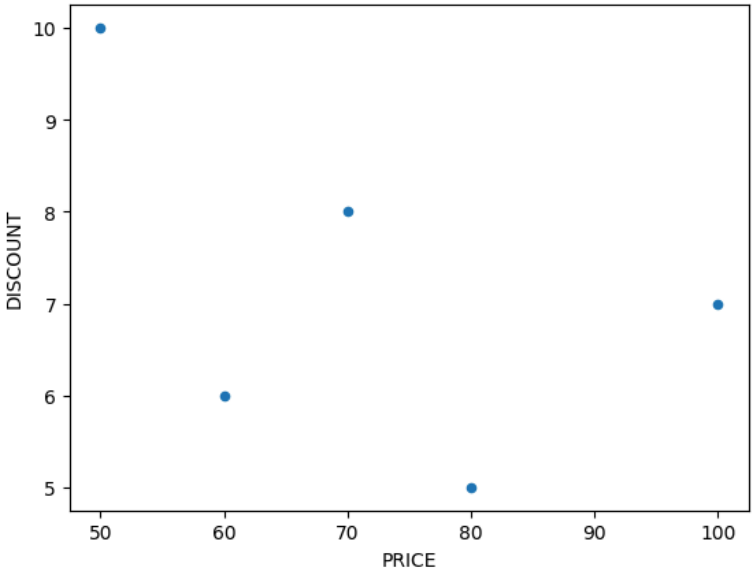


Scatter Plot with Pandas

scatter() methods used to create a scatter plot in pandas.

```
df.plot.scatter(x='PRICE', y='DISCOUNT')
```

Output:

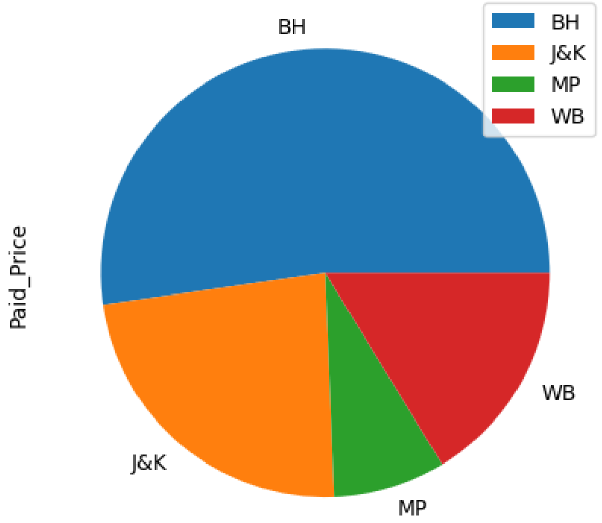


Pie Chart with Pandas

plot.pie() methods used to create pie chart.

```
grouped = df.groupby(['Origin'])
grouped.sum().plot.pie(y='Paid_Price', subplots=True)
```

Output:



Conclusion

In conclusion, the **Pandas Cheat Sheet** serves as an invaluable resource for data scientists and Python users. Its concise format and practical examples provide quick access to essential Pandas functions and methods. By leveraging this pandas cheat sheet, users can streamline their data manipulation tasks, gain insights from complex datasets, and make informed decisions. Overall, the Pandas Cheat Sheet is a must-have tool for enhancing productivity and efficiency in data science projects.

Pandas Cheat Sheet – FAQs

1. What is a Pandas cheat sheet?

> *A Pandas cheat sheet is a reference document that provides a quick overview of the most commonly used Pandas functions and methods. It is a valuable resource for anyone who is learning to use Pandas or who wants to brush up on their skills.*

2. What are the most important functions and methods in Pandas?

> *Some of the most important functions and methods in Pandas include:*

Code snippet:

- *df.head(): Returns the first few rows of a DataFrame.*
- *df.tail(): Returns the last few rows of a DataFrame.*
- *df.info(): Provides information about the DataFrame, such as the number of rows and columns, the data types of the columns, and the missing values.*
- *df.describe(): Provides summary statistics for the numerical columns in a DataFrame.*
- *df.loc[row_index, column_name]: Returns the value at a specific row and column in a DataFrame.*
- *df.iloc[row_index, column_index]: Returns the value at a specific row and column index in a DataFrame.*
- *df.sort_values(by='column_name'): Sorts the DataFrame by the values in a specific column.*
- *df.groupby('column_name'): Groups the DataFrame by the values in a specific column.*

3. How can I use a Pandas cheat sheet?

*A Pandas cheat sheet can be used as a reference document when you are working with Pandas. You can look up the function or method that you need and then use the documentation to learn how to use it. You can also use a **Pandas cheat sheet** to learn about the different features of Pandas.*

4. Is Pandas suitable for big data?

While Pandas is excellent for small to medium-sized datasets, it may not be the best choice for big data due to memory constraints. In such cases, alternatives like Dask or Apache Spark are recommended.

5. Can I perform machine learning with Pandas?

> *Pandas are primarily designed for data manipulation and analysis. For machine learning tasks, you can use libraries like Scikit-learn, which seamlessly integrates with Pandas.*

Comment More info

Advertise with us

Next Article

Java Cheat Sheet

Similar Reads

Python OpenCV Cheat Sheet

The Python OpenCV Cheat Sheet is your complete guide to mastering computer vision and image processing using Python. It's designed to be...

15+ min read

Tkinter Cheat Sheet

Tkinter, the standard GUI library for Python, empowers developers to effortlessly create visually appealing and interactive desktop application...

8 min read

Statistics Cheat Sheet

In the field of data science, statistics serves as the backbone, providing the essential tools and techniques for extracting meaningful insights from...

14 min read

Calculus Cheat Sheet

Calculus is a branch of mathematics that studies the properties and behavior of functions, rates of change, limits, and infinite series. Calculus...

15+ min read

Ansible Cheat Sheet

Ansible is a powerful open-source automation tool that is meant for configuration management and application deployment. It works with...

10 min read

NumPy Cheat Sheet: Beginner to Advanced (PDF)

NumPy stands for Numerical Python. It is one of the most important foundational packages for numerical computing & data analysis in Pytho...

15+ min read

ggplot2 Cheat Sheet

Welcome to the ultimate ggplot2 cheat sheet! This is your go-to resource for mastering R's powerful visualization package. With ggplot2, you can...

13 min read

Git Cheat Sheet

Git Cheat Sheet is a comprehensive quick guide for learning Git concepts, from very basic to advanced levels. By this Git Cheat Sheet, our aim is to...

10 min read

Linux Commands Cheat Sheet

Linux, often associated with being a complex operating system primarily used by developers, may not necessarily fit that description entirely. Whi...

13 min read

Subnet Mask Cheat Sheet

A Subnet Mask is a numerical value that describes a computer or device's how to divide an IP address into two parts: the network portion and the...

9 min read

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate
Tower, Sector- 136, Noida, Uttar Pradesh
(201305)

Registered Address:

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305

Company

About Us
Legal
Privacy Policy
Careers
In Media
Contact Us
GFG Corporate Solution
Placement Training Program

Explore

Job-A-Thon Hiring Challenge
Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP
GeeksforGeeks Videos
Geeks Community

Languages

Python
Java
C++

DSA

Data Structures
Algorithms
DSA for Beginners

PHP

GoLang

SQL

R Language

Android Tutorial

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning

ML Maths

Data Visualisation

Pandas

NumPy

NLP

Deep Learning

Python Tutorial

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

Web Scraping

OpenCV Tutorial

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Databases

SQL

MYSQL

PostgreSQL

Basic DSA Problems

DSA Roadmap

DSA Interview Questions

Competitive Programming

Web Technologies

HTML

CSS

JavaScript

TypeScript

ReactJS

NextJS

NodeJs

Bootstrap

Tailwind CSS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

Commerce

Accountancy

Business Studies

Economics

Management

HR Management

Finance

Income Tax

Preparation Corner

Company-Wise Recruitment Process

Resume Templates

Aptitude Preparation

PL/SQL

MongoDB

Competitive Exams

JEE Advanced

UGC NET

UPSC

SSC CGL

SBI PO

SBI Clerk

IBPS PO

IBPS Clerk

Free Online Tools

Typing Test

Image Editor

Code Formatters

Code Converters

Currency Converter

Random Number Generator

Random Password Generator

DSA/Placements

DSA - Self Paced Course

DSA in JavaScript - Self Paced Course

DSA in Python - Self Paced

C Programming Course Online - Learn C with Data Structures

Complete Interview Preparation

Master Competitive Programming

Core CS Subject for Interview Preparation

Mastering System Design: LLD to HLD

Tech Interview 101 - From DSA to System Design [LIVE]

DSA to Development [HYBRID]

Placement Preparation Crash Course [LIVE]

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]

Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]

Data Science Training Program - [LIVE]

Mastering Generative AI and ChatGPT

Data Science Course with IBM Certification

Clouds/Devops

DevOps Engineering

AWS Solutions Architect Certification

Salesforce Certified Administrator Course

Puzzles

Company-Wise Preparation

Companies

Colleges

More Tutorials

Software Development

Software Testing

Product Management

Project Management

Linux

Excel

All Cheat Sheets

Recent Articles

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships

Development/Testing

JavaScript Full Course

React JS Course

React Native Course

Django Web Development Course

Complete Bootstrap Course

Full Stack Development - [LIVE]

JAVA Backend Development - [LIVE]

Complete Software Testing Course [LIVE]

Android Mastery with Kotlin [LIVE]

Programming Languages

C Programming with Data Structures

C++ Programming Course

Java Programming Course

Python Full Course

GATE

GATE CS & IT Test Series - 2025

GATE DA Test Series 2025

GATE CS & IT Course - 2025

GATE DA Course 2025

GATE Rank Predictor