Pandas Cheat Sheet

Section 01

Importing the Pandas Library:

We need to import the library before we get started.
import pandas as pd

Pandas Data Structure:

We have two types of data structures in Pandas, Series and DataFrame.

Series

A series is a one-dimensional labeled array that can hold any data type.

DataFrame

DataFrame is a two-dimensional, potentially heterogeneous tabular data structure.
Or we can say Series is the data structure for a single column of a DataFrame.
Now let us see some examples of Series and DataFrames for better understanding.

Series:s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])

```
Series:s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
```

Data Frame:
data_mobile = {'Mobile': ['iPhone', 'Samsung', 'Redmi'], 'Color': ['Red', 'White', 'Black'], 'Price': [High, Medium, Low]}
df = pd.DataFrame(data_mobile, columns=['Mobile', 'Color', 'Price'])

```
data_mobile = {'Mobile': ['iPhone', 'Samsung', 'Redmi'], 'Color': ['Red', 'White', 'Black'],
'Price': [High, Medium, Low]}
df = pd.DataFrame(data_mobile, columns=['Mobile', 'Color', 'Price'])
```

Importing Data Convention:

The Pandas library offers a set of reader functions that can be performed on a wide range of file formats that return a Pandas object. Here we have mentioned a list of reader functions.

pd.read_csv("filename")
pd.read_table("filename")
pd.read_excel("filename")
pd.read_sql(query, connection_object)
pd.read_json(json_string)

```
pd.read_csv("filename")
pd.read_table("filename")
pd.read_excel("filename")
pd.read_sql(query, connection_object)
pd.read_json(json_string)
```

Similarly, we have a list of write operations that are useful while writing data into a file.

Exporting Data Convention:

df.to_csv("filename")
df.to_excel("filename")
df.to_sql(table_name, connection_object)
df.to_json("filename")

```
df.to_csv("filename")
df.to_excel("filename")
df.to_sql(table_name, connection_object)
df.to_json("filename")
```

Create Test/Fake Data:

The Pandas library allows us to create fake or test data in order to test our code segments. Check out the examples given below.

df = DataFrame() - Create an empty DataFrame
pd.DataFrame(np.random.rand(4,3)) – 3 columns and 4 rows of random floats
pd.Series(new_series) – Creates a series from an iterablenew_series

```
df = DataFrame() //Create an empty DataFrame
pd.DataFrame(np.random.rand(4,3)) //3 columns and 4 rows of random floats
pd.Series(new_series) //Creates a series from an iterablenew_series
```

DataFrame Retrieving: Column Data

df['Pet'] - Return a single column value with the name of the column and the name is 'Pet'
df[['Pet', 'Vehicle']] - Return more than a single column with its name.
df.filter(regex='TIM') - Return the column name whose names correspond to the corresponding regular expression patterns.

```
df['Pet']   //Return a single column value with the name of the column and the name is 'Pet'
df[['Pet', 'Vehicle']] //Return more than a single column with its name.
df.filter(regex='TIM') //Return the column name whose names correspond to the corresponding regular expression patterns.
```

Section 02

Operations: Viewing, Selecting, Sorting, and Groupby

Here we have mentioned various inbuilt functions and their operations.

View DataFrame contents:

df.head(n) – look at the first n rows of the DataFrame.
df.tail(n) – look at the last n rows of the DataFrame.
df.sample(n) - look at randomly n rows from the DataFrame.
df.nlargest(n, 'value') - Return and arrange the top n entries.
df.nsmallest(n, 'value') - Return and arrange bottom n entries
df[df.HEIGHT > 100] - Return the rows having 'HEIGHT' values > 100
df.drop_duplicates() - Remove duplicate rows according to all column
df.shape() – Gives the number of rows and columns.
df.info() – Information of Index, Datatype, and Memory.
df.describe() – Summary statistics for numerical columns.

```
df.head(n) //look at the first n rows of the DataFrame.
df.tail(n) //look at the last n rows of the DataFrame.
df.sample(n) //look at randomly n rows from the DataFrame.
df.nlargest(n, 'value') //Return and arrange the top n entries.
df.nsmallest(n, 'value') //Return and arrange bottom n entries
df[df.HEIGHT > 100] //Return the rows having 'HEIGHT' values > 100
df.drop_duplicates() //Remove duplicate rows according to all column
df.shape() //Gives the number of rows and columns.
df.info() //Information of Index, Datatype, and Memory.
df.describe() //Summary statistics for numerical columns.
```

Selecting:

We want to select and have a look at a chunk of data from our DataFrame. There are two ways of achieving both.
First, select by position, and second, select by label.

- Selecting by position using iloc:

df.iloc[0] – Select first row of the data frame
df.iloc[1] – Select second row of the data frame
df.iloc[-1] – Select last row of the data frame
df.iloc[:,0] – Select first column of the data frame
df.iloc[:,1] – Select second column of the data frame

```
df.iloc[0] //Select first row of the data frame
df.iloc[1] //Select second row of the data frame
df.iloc[-1] //Select last row of the data frame
df.iloc[:,0] //Select first column of the data frame
df.iloc[:,1] //Select second column of the data frame
```

- Selecting by label, using loc:

df.loc([0], [column labels]) - Select single value by row position & column labels
df.loc['row1':'row3', 'column1':'column3'] - Select and slicing on labels

```
df.loc([0], [column labels]) //Select single value by row position & column labels
df.loc['row1':'row3', 'column1':'column3'] //Select and slicing on labels
```

Sorting:

Another very simple yet useful feature offered by Pandas is the sorting of DataFrame.

df.sort_index() -Sorts by labels along an axis
df.sort_values(column1) – Sorts values by column1 in ascending order
df.sort_values(column2,ascending=False) – Sorts values by column2 in descending order
df.reset_index() - Allows to reset the index back to the default.

```
df.sort_index() //Sorts by labels along an axis
df.sort_values(column1) //Sorts values by column1 in ascending order
df.sort_values(column2,ascending=False) //Sorts values by column2 in descending order
df.reset_index() //Allows to reset the index back to the default.
```

Groupby:

Using the groupby technique, you can create a grouping of categories. It will be helpful while applying a function to the categories. This simple yet valuable technique is used widely in data science.

df.groupby(column) – Returns a groupby object for values from one column
df.groupby([column1,column2]) – Returns a groupby object values from multiple columns
df.groupby(column1)[column2].mean() – Returns the mean of the values in column2, grouped by the values in column1
df.groupby(column1)[column2].median() – Returns the mean of the values in column2, grouped by the values in column1

```
df.groupby(column) //Returns a groupby object for values from one column
df.groupby(column1,column2]) //Returns a groupby object values from multiple columns
df.groupby(column1)[column2].mean() //Returns the mean of the values in column2, grouped by the values in column1
df.groupby(column1)[column2].median() //Returns the mean of the values in column2, grouped by the values in column1
```

Merging Data Sets

For combining different datasets, you can use the merging technique. Parameters to be followed: {'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'

Inner Join: pd.merge(df1, df2, how='inner', on='Apple')
Outer Join: pd.merge(df1, df2, how='outer', on='Orange')
Left Join: pd.merge(df1, df2, how='left', on='Animals')
Right Join: pd.merge(df1, df2, how='right', on='Vehicles')
Cross Join: df1.merge(df2, how='cross')

```
pd.merge(df1, df2, how='inner', on='Apple') //Inner Join
pd.merge(df1, df2, how='outer', on='Orange') //Outer Join
pd.merge(df1, df2, how='left', on='Animals') //Left Join
pd.merge(df1, df2, how='right', on='Vehicles') //Right Join
df1.merge(df2, how='cross') //Cross Join
```

Renaming Data

If we want to rename the labels in the DataFrame, then use the df.rename(index=None, columns=None, axis=None) function; the values of column names and index labels can be replaced.

df.rename(columns={'ferrari': 'FERRARI', 'mercedes': 'MERCEDES', 'bently': 'BENTLY'}, inplace=True) - Rename the column name according to its values. If True then the value of the copy is ignored.
df.rename(columns={"": "a", "B": "c"}) - Rename columns by mapping them.
df.rename(index={0: "london", 1: "newyork", 2: "berlin"}) - Rename index by mapping them.

```
df.rename(columns={'ferrari': 'FERRARI', 'mercedes': 'MERCEDES', 'bently': 'BENTLY'},
inplace=True) //Rename the column name according to its values.
df.rename(columns={"": "a", "B": "c"}) //Rename columns by mapping them.
df.rename(index={0: "london", 1: "newyork", 2: "berlin"}) //Rename index by mapping them.
```

DataFrame with Duplicates

To remove duplicate data from the datasets, use duplicates(). It helps to find and eliminate the repeated labels in a DataFrame.

df.duplicated() - method used to identify duplicate rows in a DataFrame
df.index.duplicated - Remove duplicates by index value
df.drop_duplicates() - Remove duplicate rows from the DataFrame

```
df.duplicated() //method used to identify duplicate rows in a DataFrame
df.index.duplicated //Remove duplicates by index value
df.drop_duplicates() //Remove duplicate rows from the DataFrame
```

Section 03

Reshaping Data

Python Pandas offers a technique to manipulate/reshape a DataFrame and Series in order to change how the data is represented.

pivot = df.pivot(columns='Vehicles', values=['BRAND', 'YEAR']) - Create a table with various data types.
pd.melt(df) - Collect columns into a row.
pd.pivot_table(df, values="10", index=["1", "3"], columns=["1"]) - Pivoting with the aggregation of numeric data.

```
pivot = df.pivot(columns='Vehicles', values=['BRAND', 'YEAR']) //Create a table with various data types.
pd.melt(df) //Collect columns into a row.
pd.pivot_table(df, values="10", index=["1", "3"], columns=["1"]) //Pivoting with the aggregation of numeric data.
```

Concatenating Data

The concatenation function is performed when combining objects along a shared index or column.

df = pd.concat() - Allows for a full copy of the data.
df = pd.concat([df3,df1]) - Combining two DataFrame.
df = pd.concat([S3,S1]) - Combining two series.
df = pd.concat([df3,S1], axis=1) - Combining the DataFrame and series together.

```
df = pd.concat() //Allows for a full copy of the data.
df = pd.concat([df3,df1]) //Combining two DataFrame.
df = pd.concat([S3,S1]) //Combining two series.
df = pd.concat([df3,S1], axis=1 ) //Combining the DataFrame and series together.
```

Filtering

To refine specific values from a large dataset with multiple conditions, using the filter() method helps filter the DataFrame and returns only the rows or columns that are specified.

df = df.filter(items=['City', 'Country']) - Return 'City' and 'Country' columns from the DataFrame
df = df.filter(like='tion', axis=1) - Return columns that present 'tion' in names.
df = df.filter(regex='Quest') - Return the column name whose names correspond to the corresponding regular expression patterns.
df = df.query('Speed>70') - Returns the value from a row whose speed is more than 70.

```
df = df.filter(items=['City', 'Country']) //Return 'City' and 'Country' columns from the DataFrame
df = df.filter(like='tion', axis=1) //Return columns that present 'tion' in names.
df = df.filter(regex='Quest') //Return the column name whose names correspond to the corresponding regular expression patterns.
df =  df.query('Speed>70') //Returns the value from a row whose speed is more than 70.
```

Working with Missing Data

Drop columns: df.drop(columns=['column_name'], inplace=True) - Drop rows containing null data in any column.
Filling Data: cbd["London"].fillna("Newyork, inplace = True) - Filling na values in London with Newyork in DataFrame.
Replacing: df_filled.replace([2, 30], [1, 10]) - Replacing more than one value in the DataFrame.
Interpolation: df.interpolate(method ='linear', limit_direction ='backward', axis=0) - Interpolate the missing values filled in by columns in the forward direction using the linear method.

```
df.dropna(columns=['column_name'], inplace=True) //Drop rows containing null data in any column.
cbd["London"].fillna("Newyork", inplace = True) //Filling na values in London with Newyork in DataFrame.
df_filled.replace([2, 30], [1, 10]) //Replacing more than one value in the DataFrame.
df.interpolate(method ='linear', limit_direction ='backward',
axis=0) //Interpolate the missing values filled in by columns in the forward direction using the linear method.
```

Pandas: Statistical Functions

There are some special methods available in Pandas that make our calculations easier. Let's have a look at:

Mean:df.mean() – mean of all columns
Median:df.median() – median of each column
Standard Deviation:df.std() – standard deviation of each column
Max:df.max() – highest value in each column
Min:df.min() – lowest value in each column
Count:df.count() – number of non-null values in each DataFrame column
Describe:df.describe() – Summary statistics for numerical columns
apply those methods in our Product_ReviewDataFrame

```
df.mean() //mean of all columns
df.median() //median of each column
df.std() //standard deviation of each column
Max:df.max() //highest value in each column
Min:df.min() //lowest value in each column
df.count() //number of non-null values in each DataFrame column
df.describe() //Summary statistics for numerical columns apply those methods in our Product_ReviewDataFrame
```

Dropping Data

While working with a variety of datasets, there is a possibility we have to remove certain data. Using the Dropping() method, specific data can be removed from rows and columns.

Drop Columns: df.drop(['Nike'], axis=1) - Drop the 'Nike' from the columns.
Drop row by index: df.drop(['Size'], axis=0) - Drop the 'Size' from the row.
Dropping the multindex DataFrame - df.drop(index='offers', columns='location') - Drop the multiple labels 'offers' and 'location' from rows and columns, respectively.

```
df.drop(['Nike'], axis=1) //Drop the 'Nike' from the columns.
df.drop(['Size'], axis=0) //Drop the 'Size' from the row.
df.drop(index='offers', columns='location') //Drop the multiple labels  'offers' and 'location' from rows and columns, respectively.
```

Pandas Indexing

In Pandas, indexing returns specific rows and columns of data from the DataFrame.

Retrieving data frame from csv file: detail =
pd.read_csv("employee_db.csv", index_col ="Contact")
Set Index can replace the existing indexes or columns from the DataFrame:
detail.set_index('Name')
Multiple indexing - MultiIndex(levels=[['2024-01-01', '2024-01-22', '2024-02-10'], ['mathew', 'linda']])
Reset Index- df.reset_index(level = 3, inplace = True, col_level = 2)

```
detail = pd.read_csv("employee_db.csv", index_col ="Contact") //Retrieving data frame from csv file
detail.set_index('Name') //Set Index can replace the existing indexes or columns from the DataFrame
multindex(levels=[['2024-01-01', '2024-01-22', '2024-02-10'], ['mathew', 'linda']]) //Multiple indexing
df.reset_index(level = 3, inplace = True, col_level = 2) //Reset Index
```

Plotting:

Data Visualization with Pandas is carried out in the following ways.

- Histogram
- Scatter Plot

Note: Call %matplotlib inline to set up plotting inside the Jupyter notebook.
Histogram: df.plot.hist()
Scatter Plot: df.plot.scatter(x='column1',y='column2')

```
df.plot.hist() //Histogram
df.plot.scatter(x='column1',y='column2') //Scatter Plot
```