Pandas

1. import pandas as pd # Import pandas library
2. df = pd.read_csv('filename.csv') or pd.read_csv(r'location/folder/filename.csv') or pd.read_excel('filename.xlsx') #Load data from a CSV file or Excel file into a DataFrame
3. df.head() or df.head(n) #Display the first 5 rows (by default) or size = n number of rows of the DataFrame
4. df.info() #Get a concise summary of the DataFrame
5. df.describe() #Generate descriptive statistics of the DataFrame
6. df['column_name'] or df.column_name OR df['column_name][index_number] #Access a specific column OR access specific column's row value
7. df[['col1', 'col2']] #Access multiple columns
8. df.loc[row_index] or df.loc[: , 3] or df.loc[0, 'column_name'] #Access a row by index or all "rows" and "4rth" column or 1st row and column_name column intersecting value
9. df.iloc[row_index] or df.iloc[[0,1,2],0] or df.iloc[-5:] #Access a row by integer position or 1st 2nd 3rd row and 1st column or last 5 rows with all columns

Choosing between `loc` and `iloc` ¶

`iloc` uses the Python stdlib indexing scheme, where the **first element** of the range is **included** and the **last one excluded**. So `0:10` will select entries `0,...,9`.

`loc`, meanwhile, **indexes inclusively.** So `0:10` will select entries `0,...,10`. loc can index any stdlib type: **strings**

10. df.isnull().sum() #Check for missing values in the DataFrame
11. df.set_index("Title") #This is useful if you can come up with an index for the dataset which is better than the current one
12. df.dropna() #Remove rows with missing values
13. df.fillna(value) #Fill missing values with a specific value

14. df['column'] = df['column'].astype('int') # Convert column data type
15. df.groupby('column').mean() # Group by column and calculate the mean
16. df.sort_values('column') # **Sort DataFrame by a specific column**
17. df.merge(other_df, on='key') # Merge DataFrames on a key column
18. pd.concat([df1, df2]) # Concatenate DataFrames
19. df['date'] = pd.to_datetime(df['date']) # Convert column to datetime
20. df.resample('M').mean() # Resample time-series data to monthly frequency
21. df.shape #(number of rows x number of columns)
22. df.duplicated() and df.duplicated().sum()
23. df.drop_duplicates() #Drops/deletes duplicate rows

CONDITIONAL SELECTION

24. df.loc[df.Country == “Italy”] #df.Country == “Italy” This operation produced a Series of `True` / `False` booleans based on the `country` of each record. This result can then be used inside of `loc` to select the **relevant data.**
26. df.loc[(df.Country == “Italy”) & (df.points≥90)] # and &, or |

CONDITIONAL SELECTORS

26. isin() # `isin` lets you select data whose value "is in" a list of values

eg df.loc[df.Country.isin([”Italy”,”France”])] #select wines only from Italy or France

27. isnull() ,notnull() #These methods let you highlight values which are (or are not) empty (`NaN`)

eg df.loc[df.price.notnull()]

ASSIGNING DATA

28. df[”Country”] =”India” #Every value of column is changed to “India”

SUMMARY FUNCTIONS

29. df.country.describe() #This method generates a high-level summary of the attributes of the **given column**. for only NUMERICAL DATA
30. df.column_name.mean()

31. df.col_name.unique() #To see a list of unique values we can use the `unique()` function
32. df.col_name.value_counts() #To see a list of unique values *and* how often they occur in the dataset, we can use the `value_counts()` method: