# 1.kaggle-PANDA

## What is Pandas?

- Pandas is a Python library used for working with data sets.

It provides functions for cleaning, exploring, and manipulating datasets efficiently.

- Created by Wes McKinney in 2008, its name is a blend of "Panel Data" and "Python Data Analysis."
- here is the list of things to do with pandas:
    1. Data set cleaning, merging, and joining.
    2. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
    3. Columns can be inserted and deleted from DataFrame and higher dimensional objects.
    4. Powerful group by functionality for performing split-apply-combine operations on data sets.

## *Importing Pandas

*To start with the code write-*

In[1]: *import pandas as pd*

## *Creating Data

There are 2 objects in pandas-

1. DataFrame
2. Series

## *DataFrame

Pandas DataFrame is a two-dimensional data structure with labeled axes (rows and columns).

Example-

In[2]: *pd.DataFrame({'A': [5, 2], 'B': [13, 12]})*

Out[2]:

| | A | B |
|---|---|---|
| 0 | 5 | 13 |

| | | |
|---|---|---|
| 1 | 2 | 12 |

The list of row labels used in a DataFrame is known as an **Index**. We can assign values to it by using an index parameter in our constructor:

*In[3]:* *pd.DataFrame({'A': ['Good', 'Fantastic.'],*
*'B': ['Pretty good.', 'Bland.']},*
*index=['Product A', 'Product B'])*

*Out[3]:*

| | A | B |
|---|---|---|
| Product A | Good | Pretty good |
| Product B | Fantas-tic | Bland |

# *Series

A Series, by contrast, is a sequence of data values. If a DataFrame is a table, a Series is a list:

In[4]: *pd.Series([1, 2, 3, 4, 5])*

Out[4]:
0 1
1 2
2 3
3 4
4 5
dtype: int64

A Series in Pandas represents a single column of a DataFrame. You can assign row labels using an index parameter, but a Series does not have a column name; it only has an overall name.
In[5]: *pd.Series([10, 20, 30], index=['2010 Sales', '2011 Sales', '2012 Sales'], name='Product A')*
Out[5]:
2010 Sales 10
2011 Sales 20
2012 Sales 30
Name: Product A, dtype: int64

# *CSV file

CSV files are the Comma Separated Files. To access data from the CSV file, we require a function read_csv() from Pandas that retrieves data in the form of the data frame.
*wine_reviews = pd.read_csv("../input/wine-reviews/winemag-data-130k-v2.csv")*

We can examine the contents of the resultant DataFrame using the head() command,

*wine_reviews.head()*

To make pandas use that column for the index (instead of creating a new one from scratch), we can specify an index_col

*wine_reviews = pd.read_csv("../input/wine-reviews/winemag-data-130k-v2.csv", index_col=0)*

Pandas describe() is used to view some basic statistical details like percentile, mean, std, etc. of a data frame or a series of numeric values
*print(data.descibe())*

# *Indexing, Selecting and Assigning

For a given large DataFrame, we can simplify it to one of it's parameters by typing the name of the DataFrame, followed by the property that we desire. Eg:
*review.country  OR  review['country']*

We can utilize each supplementary parameter to eliminate one dimension from the desired outcome. For instance, with an additional dimension, we could pinpoint the precise location of a data point. For example:

*review['country'][0]*
#Output in this case is 'Italy', which is a singular data point.

# *Iloc & loc

There are two parameters for indexing in Pandas, those being ‘ loc ’ and ‘ iloc ’. loc is useful in situations where we know the name of the parameter we desire, whereas iloc is used when we know the numerical index of the parameter.
As stated earlier, iloc utilizes the integral value of the datapoint, so it is called as follows:
*test.iloc[0]*

loc is called in the following way:
*test.loc[0, ‘parameter’]*

**NOTE:** Both loc and iloc are row-first, column-second. This is the opposite of what we do in native Python, which is column-first, row-second.

iloc includes the first element in a range, but not the final. So in the case of df.[0:1000], it would return 1000 results, (not counted the 1000th element) whereas loc doesn’t have this quirk and would return 1001 values.

The index of a dataset can be changed to whatever value which allows sorting by that parameter by the function:
*code.set_index("parameter")*
*#.set_index("") being the function.*

Pandas offers enhanced indexing functionalities tailored for Series and DataFrames, particularly excelling in datasets featuring multiple parameters. It provides advanced capabilities for indexing, such as position-based indexing.
Position-based indexing: Enables access to rows and columns based on integral positions, akin to the functionality of the **loc** function.

We can use boolean operators such as **“&, I,”** to take the union/intersection of various parameters.

## *Renaming and Combining

We can rename the index name or column which helps us in sorting better by the function:
*code.rename()*

While dealing with larger datasets, we will have to combine them in order to perform meaningful analysis. We do that by either the **concat()** , **join()** or **merge()** methods. **merge()** is redundant as most of its jobs can be performed by the join function.

The **concat()** function simply concatenates two strings; i.e., joins the end of one string with the beginning of another. This is useful when two datasets have nearly similar parameters.

The **join()** function combines two DataFrame objects which have an index in common.

## *Summary Functions and Maps

The term “Summary Function” is an unofficial term used to structure the data in more useful ways.

In order to gain an overview of the data presented in a column, we can use the describe() method, which tells us the count, mean, standard deviation (std), minimum, 25th percentile 25%, 50th percentile 50% or median, 75th percentile 75% and maximum.
*count — 129971.000000*
*mean — 88.447138..*
*75% — 91.000000*
*max — 100.000000*
*Name: points, Length: 8, dtype: float64*
Note: The **describe()** method only works for numerical data.

To see the mean of some values we can use the **mean()** function, similarly, to see a list of unique values we can use the unique() function.

To see a list of unique values and how often they occur in the dataset, we can use the **value_counts()** method:

| | |
|---|---|
| Name 1 | 3125125 |
| Name 2 | 23251 |
| Name 3 | 2799 |
| Name 4 | 22 |

The term “Map” relates to a mathematical function, i.e., that takes a parameter in the x- coordinate and “maps” it onto the f(x). In other words, it takes one set of values and “maps” it to
another set of values. In this case, you can create a function with custom parameters, and “map” it onto the columns/rows of the given dataset, or use a dictionary and map the key-value pairs onto it.

## *Grouping and Sorting

To group and categorize DataFrames, we use the **groupby()** function that performs utility similar to that of value_counts() function, by then adding in additional prompts as required.

For additional clarity, the **groupby()** function splits a DataFrame into groups based on specified criteria, such as values in one or more columns, and then applies a function (such as aggregation, transformation, or filtering) to each group independently.
*function.groupby('parameter_cost').price.min()*

In order to get a functionality similar to that of the value_counts() function, we can use the agg() function that allows us to pass in multiple functions in our DataFrame simultaneously. Eg:
*function.groupby(['parameter_cost']).price.agg([len, min, max])*
*#applying the length, minimum and maximum of all the points in the dataset.*

Multi-indexing is a prevalent technique in Pandas aimed at enhancing data efficiency. It involves converting or reducing a higher-dimensional input into a lower dimension, thus optimizing data organization.

When sorting a DataFrame, custom parameters can be passed to specify the sorting criteria using the **sort_values()** method.

## *Data Types and Missing Values

The Data Type simply refers to the data type used in a DataFrame or a Series, represented by **dtype .** We can use it to check the datatype in a particular column in a dataset.

Note: Strings get the object  type, and not their own type.

Conversions between datatypes are possible, eg a int64  data type can be converted into a float64  data type.

Entries missing values are given the value **NaN** , short for "Not a Number".

We can target specifically these NaN  values with the handy **fillna()** function, eg:

*reviews.region_2.fillna("Unknown")*

Which replaces every “NaN” value with a “Unknown” warning instead.

**Backfill Strategy:** The idea of filling each missing value with the first non-null value that appears after the given record in the database.

# 2.GIT AND GITHUB

Works on Command line interface, it is case sensitive
GIT Commands:

**Clone**-> Bring a repository that is hosted somewhere like Github into a folder on your local machine
**add** -> Track your files and changes in Git
**commit** -> Save your files in Git
**push**-> Upload Git commits to a remote repo, like Github
**pull** -> Download changes from remote repo to your local machine, the opposite of push

**Readme.md** - a file containing essential project/repository information, typically in text format.

Each commit is identified by a unique address.

When importing into our code editor, utilize the HTTPS link rather than the SSH link unless SSH keys are set up.

**LS** - shorthand for la-ls, used to list folders, files, and hidden files within directories.

**git status** - a command displaying history and indicating uncommitted files. An untracked file signifies it hasn't been updated in Git.

To commit an untracked file, use "git add ." to track both untracked files and update modified files. For committing a specific file, employ "git add <File name>".

```
Last login: Wed Feb 28 22:45:01 on ttys000
aashirvad@aashirvads-MacBook-Pro ~ % ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/aashirvad/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/aashirvad/.ssh/id_rsa
Your public key has been saved in /Users/aashirvad/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:cjuXeZGLQLUmBMexVCnjyaH1L/mUZcbGT0HYGXFgX1o aashirvad@aashirvads-MacBook-Pro.local
The key's randomart image is:
+---[RSA 3072]----+
|      .o=oo.  *=E|
|       +*o.. o Bo|
|       =+*o  o. o|
|      ..+o.  .B .|
|      . S  oo* o |
|       o oo++o  .|
|        o =+o    |
|         o ..    |
|                 |
+----[SHA256]-----+
aashirvad@aashirvads-MacBook-Pro ~ %
```

Machine connection to GitHub is established via SSH keys.

This can be run on cmd prompt, GIT and on VSCode.
The public key given by <testkey>.pub is a proof that this public key exists because of private key

To copy anything to our clipboard, we use pbcopy < ~/testkey.
Just use ~ to copy previous text to clipboard

Use git push origin main to save file to repository in GITHUB. It requires authentication then you can upload

**To save files into the repository in GitHub we do**

1. **git add -** Tracking files and changes
2. **git status -**
3. **git commit -m "<Title>" -m "<Description>" -** Saves files and changes on git
4. **git push origin main -** Upload the git commands to GitHub. Here, the origin could be the location of the git repository, and the master refers to the branch where we want to push it.
5. **Refresh GitHub to access the files**

- **cd demo-repo:**
  - Changes the current directory to the "demo-repo" repository, allowing navigation within its file structure.
- **git status:**
  - Verifies the status of the current repository, indicating whether it is up to date or if there are pending changes requiring commitment or pushing.
- ***git remote:***
  - *Facilitates access to repositories located elsewhere, enabling*

    *local interaction with remote repositories through commands.*
- ***SSH Key Generation:***
  - SSH (Secure Shell) is a protocol for secure remote login, offering an alternative to password authentication.

- SSH key pairs provide a secure and convenient method for accessing remote servers without passwords.
- SSH keys can be generated using the ssh-keygen command in the terminal, VS Code, or PowerShell.
- The public key, generated alongside the private key, is added to GitHub to establish a seamless connection between the local Git and GitHub repositories.
- The private key, essential for authentication, must be securely stored on the PC and safeguarded from unauthorized access.

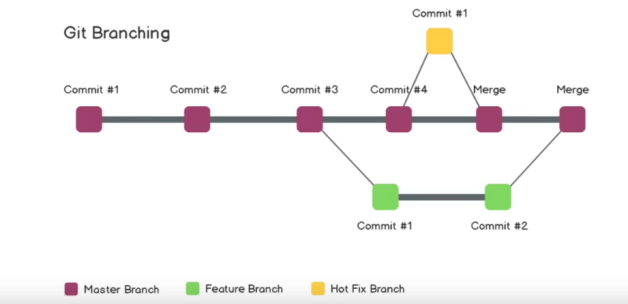
**Remote repositories-** refer to repositories hosted on servers, enabling collaboration and access from multiple locations.
They allow users to push and pull changes to and from a central repository, facilitating teamwork and version control.

**Workflow Difference between Git and GitHub:**

In Git, code changes are periodically saved, committed, and pushed to the Git repository. Pull requests are then made to incorporate changes.
However, in GitHub, code changes are committed first, followed by a push request. The use of git add and git push commands is not necessary in this workflow.

**Git Branching:**

Branching is a fundamental concept in Git, facilitating collaboration and parallel development.
The main branch, typically named "master," contains the stable version of the code and serves as the base for development.
Team members create feature branches from the master branch to work on specific features or fixes without affecting the main codebase.
Hotfix branches are used to quickly address urgent bugs on the master branch before merging changes back into the main codebase.




Some git branching commands:

- **git branch**: Used to determine the current branch in Git.

- **git checkout**: Enables switching between branches. By using **git checkout -b <name-of-branch>**, a new branch can be created. Using **git checkout -d <name-of-branch>** deletes the specified branch.
- **git diff**: Compares two versions of the code and highlights the differences. Syntax: **git diff <file-name>**.
- **git merge**: Merges changes made to the same file from two or more branches. Typically, after merging, the feature branch is deleted, and changes are saved to the master branch.

**Undoing in Git:**

- **git reset <file-name>**: Reverts unwanted changes committed previously in Git.
- If changes are not the previous change, use: git reset <file-name> HEAD~<number of iterations ago>.
- If the commit ID is unknown, use: **git reset <commit-id>**.

**Other Git commands:**

- **git log**: Displays the log of all changes committed in Git.

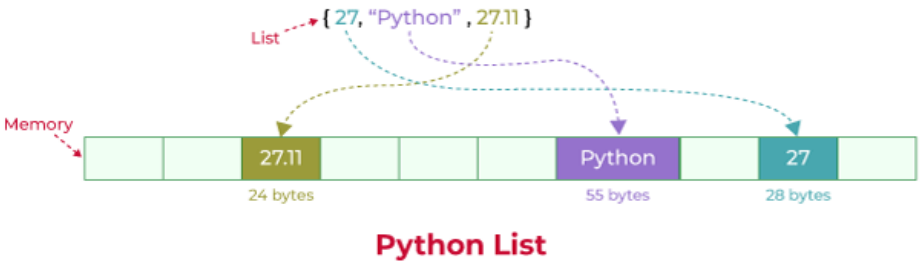
**Forking in Git:**

- Forking involves creating a personal copy of a repository on your own GitHub account, maintaining control.
- The forked copy allows making changes without affecting the original repository, facilitating experimentation and contribution.
- To contribute changes back to the original repository, create a pull request to propose the changes to the original repository owner. The owner can then review and decide whether to merge them.

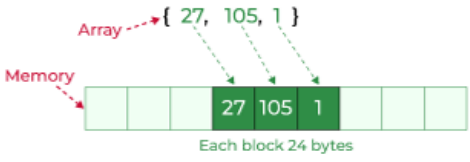
# 3.Numpy

## *Difference between numpy and lists

NumPy arrays are faster than lists due to their homogeneity, while lists are heterogeneous. NumPy arrays use more efficient data types like int16 or int8, compared to the int32 typically used by lists.
NumPy utilizes contiguous memory storage, ensuring data is stored continuously, unlike lists which lack contiguity.
It employs Single Instruction Multiple Data (SIMD) or vector processing, enhancing computational efficiency.
NumPy also facilitates effective cache utilization, further optimizing performance.




Python List




NumPy Array

- Memory size in bytes are in Mac OS X.

## *Uses of Numpy

### 1. Mathematics

As a replacement of MATLAB

Useful math operations-

numpy.add(), numpy.subtract(), numpy.multiply(), numpy.divide(): Perform

element-wise arithmetic operations.

numpy.dot(): Compute dot product of two arrays.

numpy.sum(), numpy.mean(), numpy.max(), numpy.min(): Basic statistical operations.

## 2.Creating NumPy arrays:

NumPy arrays can be 1-dimensional or 2-dimensional, depending on the data passed.
For a 1D array: np.array([1, 2, 3, 4, 5])
For a 2D array: np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
It's also possible to pass a dictionary into a NumPy array, but it needs to be converted into a list first.

## 3. Copying arrays:

When assigning array a to b, modifications to either array will affect the other. To avoid this, use b = a.copy().
Note:
In NumPy, mathematical operations are vectorized.
Array operations are element-wise, requiring arrays to have matching shapes.
Matrix multiplication:

Matrix multiplication isn't achieved through the * operator. Instead, use np.matmul(matrix1, matrix2).
New Concepts:
Kronecker product:

The Kronecker product combines two matrices into a larger matrix containing all possible products of their entries. It's a fundamental operation for eigen operations on matrices.

## 4. Some functions

- **np.random.rand():**

Used to generate random values between 0 and 1.
Syntax: np.random.rand(<no of random values required>).
Can also be used to create a matrix of random values. Syntax: np.random.rand(<dimensions of matrix>).

- **slicing and indexing in NumPy:**

For 1-D arrays: Use array[start_index:stop_index]. Negative indices like array[7:-1] are valid.
Values can be modified using slicing. For example: array[0:4] = [1, 2, 3, 4].

Extracting a row from a 2-D array: a[2].
Extracting a column from a 2-D array: a[:, 2].

**5.Array Manipulation**
ndarray.shape(): Get the shape of an array.

ndarray.reshape(): Reshape an array.

# *Pandas:

Pandas is a Python library built on top of NumPy, offering data structures like data frames, which organize data into tabular form with rows and columns.
Methods to create a data frame:

- **Using a dictionary:**
- *pd.DataFrame({'Temperature': [25, 30, 27], 'Humidity': [60, 55, 70]})*

Nested Lists:

*data = [[25, 60], [30, 55], [27, 70]]*

*pd.DataFrame(data, columns=['Temperature', 'Humidity'])*

Nested Dictionary:

*nested_dict = {'Temperature': {'City1': 25, 'City2': 30}, 'Humidity': {'City1':*

*60, 'City2': 55}}*

*pd.DataFrame(nested_dict)*

**A series:**

- A series is a one-dimensional array-like object with an index.

Creating a series object:

*pd.Series([25, 30, 27], index=['City1', 'City2', 'City3'], name='Temperature')*

# Reading and Exporting Data:

**read_csv:** Imports data from CSV files into Pandas DataFrames.
*df = pd.read_csv('data.csv')*

**to_csv**: Exports DataFrame back into CSV files.
*df.to_csv('results.csv')*

*Reading Data through HTML:*

- ***read_html**: Reads tables from an HTML file.*
- *df = pd.read_html('<url-with-an-html-tag>', flavor='lxml')*

**Accessing and Modifying Data:**

loc[]: Accesses and modifies DataFrame based on labels.
iloc[]: Accesses and modifies DataFrame using integer indices.
Applying Functions:

apply: Applies a function to a DataFrame.
Sorting Data:

sort_values(): Sorts DataFrame based on specified columns.
Concatenation and Merging:

pd.concat: Concatenates strings, series, or dataframes.
pd.merge: Merges DataFrames based on specified column(s)