Pandas

What is Pandas?

Python library used to work with data sets.
has functions to analyze, clean, manipulating data.
name has reference to panel data.

Why pandas?

helps to analyze data and make conclusions.
can clean messy data sets and make them readable.
The data is made relevant.

What can it do?

check for correlation.
Avg, min, max value.
delete rows.
check for null or empty values.

All this is called cleaning data.

To import

```
import pandas as pd.
```

Alias

```
import pandas as pd.
```

To check version

```
prind (pd.__version__)
```

Series

one dimensional array holding any data type.
It is like column in a table.

series from list

```
import pandas as pd.
a = [1, 7, 2]
myvar = pd.series (a)
print(myvar).
```

Note if nothing is specified then values are labeled with their index.

we can acess values by mvar([0])

To create labels

we can create labels using 'index' arguement

```
arr = np.array([1, 2, 3])
x = pd.Series(a, index = ["x", "y", "z"])
print (x)
```

```
=>      x   1
        y   2
        z   3
```

Note we can also pass numpy array inplace of index of same length.

Teacher's Signature..............

we can access the values through labels.

```
=) print (x["y"])
   => 2
```

Note we can also use dictionary to create series.
This way we will not have to give index.

```
Eg →   import pandas as pd
       dict1= {"day":1, "day2" : 2, "day3" : 3}
       x = pd.Series (dict)
        print(x)
```

keys of dictionary become labels.

we can also chose which keys to take from dictionary.

=> for above example:

```
x = pd.Series(dict, index = ["day1", "day2"]
```

Dataframe

2 dimensional data structure
table with rows and columns.

create

```
data = {
      "calories": [420,1090,1350],
      "diet" : ["food", "juice", "smoothie"]}
```

```
x = pd.DataFrame(data)
print(x)
```

Locate row

we can see each row by using index

⇒ · ~~print(x[0]~~
```
print(x.loc[0])
```

This is done through loc() function.

→ if single row then Series is returened.

To see multiple rows.

⇒ print(x.loc([0,1])

This returns dataframe.

Note | Index arguemen can be used in dataframe as well

Locate row through index

we can use the loc() to see specific rows through the named index.

~~x=~~ print(x.loc(["day1"])

Loading files

To load csv files to dataframe we use 'read_csv()'.

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df).
```

Note By default if dataset is big enough then pandas return first and last 5 rows.

To_string

to_string() is used to print the whole data.

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
```

=> This prints whole data.

Maximum rows

Maximum rows that can be displayed by the df are predefined in system.
If no. of rows exceed that, then only first and last five rows are used.

To check systems limit:

```
=> print(pd.options.display.max_rows)
```

we can also change this no as:-

```
pd.options.display.max_rows = 1000
print(df)
```

JSON files

- Big data sets are often stored or extracted as JSON.
- plain text, has format of an object.

To load

```
pd.read_json()
```

Note JSON has same formal as python Dictionarys.

if your json code is in a python dictionary, then it can be directly loaded to dataframe.

Analyzing Data frames

i viewing data

most used for quick overview is head().
returns a specified and headers from the top

```
print(df.head(10))
```

10-> returns top 10 rows including header.

tail()

same as head.

Only difference it returns last rows.

Info()

This gives info about the dataset.

-> No of rows and colums

-> no of nulls in colums.

-> name of column with datatype.

Null values

empty values or null values can be bad analyzing data.

This helps in cleaning data.

Data cleaning

it means fixing bad data in the data set.

- empty cells
- data in wrong format
- wrong data
- duplicates.

i) cleaning empty cells

can give wrong result when analyzing data.

a) Remove the rows.

This data cleaning feature will not affect big datasets

We use dropna() to achaive this.

```
df= pd.read_csv('data.csv')
new df = df.dropna()

print (new_df.to string)
```

-> This removes rows with at least one missing valu

Note: if you want to remove rows with all null value then use.

```
new_df.dropna(how='all')
```

-> if want to remove rows with missing values in column then.

```
new_df.dropna(subset=['column'])
```

Note we can use multiple columns also.

dropna() function does not change original dataset. if want to change original dataset then use

-> inplace = True.

b) Replace empty values

Another way is to replace new values with empty cells. This way we do not have to delete entire rows.

Fillna() is used.

- df = pd.read_csv('data.csv')
  df.fillna(130, inplace = True)

→ replace only for specified columns.

df['calories'].fitlna(30, inplace = True)

→ replace using mean(), median(), mode().

we can replace with mean, median or mode values. To do this

```
df= pd.read_csv('data.csv')
x = df['calories'].mean()
df['calories'].fillna(x, inplace = True)
```

Same syntax for median

For mode

mode returns series of values like list. so we use.

x = df['calories'].mode()[0]

This tells to use the first value in the list.

mean → Average of all values.
median → centre value after sorting.
mode → most occuring value.

Cleaning Wrong format

cells with wrong format data make it difficult to analyze

To clean data their are two methods.

- remove the rows.
- convert all columns in same format.

To convert to date to desired type.

```
df = pd.read_csv('data.csv')
df['date'] = pd.to_datetime(df['date'])
```

Cleaning wrong data

sometimes the data could just be wrong.
To fix this we use following methods.

i) replace with something else.

```
df.loc[7, 'Duration'] = 45
```

This works for small datasets.

Note → For large datasets we can set conditions for it

```
For x in df.index:
    if df.loc[x, 'Duration'] > 120:
        df.loc[x, 'Duration'] = 120
```

ii) remove the rows.

```
For x in df.index:
    if df.loc[x, 'Duration'] >120
        df.drop(x, inplace=True)
```

drop() → drop(index to remove , inplace)
This is used to drop specific rows.
useful when filtering on a condition.

dropna() → dropna()
drops rows and columns with empty values

dropna (axis=0, how='any')

any → drops rows or columns with at least 1 missing value
all → drops rows or columns with all missing values.

Cleaning Duplicates
To discover duplicates we use function duplicated()
This returns a boolean value for each row.
True if duplicate.

df.duplicated() → return true for every duplicated rows.

removing
we use 'drop_duplicates()'

⇒ df. drop_duplicates (inplace = True)

Pandas Correlations

corr() method calculate relationship between each column.

df.corr()

Note: This function ignores non numeric column.

result of corr() is a table with a lot of numbers It varies from -1 to 1.

1 → perfect correlation.

at least 0.6 or higher is good correlation.

Pandas plotting

we can use plot() to make diagrams.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()  → To plot

plt.show() → To show.
```

PAGE NO. :

Scatter plot

needs a x and y axis.

can be specified using kind='scatter'

df.plot(kind='scatter', x='Duration', y='Calories')

x = independent variable

y = dependent variable.

histogram

kind=hist

It needs only one column.

df['duration'].plot(kind=hist)

Take arguments like

bins=10 → Tells no of bins

color=skyblue color of bars.

edgecolor=black

To convert categoricat to numerical

pd.get_dummies() → one hot encoding.

return different columns on based of no of different values.

Teacher's Signature.................

df_encoded = pd.get_dummies (df, columns = ['city'

Note: we can also change new colum names by

pd.get_dummies (df, columns = ['city'], prefic = 'c', prefi

If only want city names then

pd.get_dummies (df['city'])

How to concatenate back to original dataset.

df_combined = pd.concat ([df, columns], axis=1)

To handle multicolinearity

use drop_first - True.