Data Analysis with python

* Pre-processing data in python

Data pre-processing :- The process of converting or mapping data from the initial "raw" form into another format, in order to prepare the data for further analysis.

Also known as: Data cleaning, Data Wrangling

→ Identify and handle missing values

(Missing value: whenever a data entry is left empty

→ Data formatting: Data from different sources maybe in various formats in different units, or in various conventions

python pandas :- to standardize the values into the same format, or unit, or convention.

→ Data Normalization (centering/ scaling)

columns will have different ranges direct comparison is often not meaningful

Normalization is a way to bring all data into a similar range for more useful comparison.

→ Data Binning :- Binning creates bigger categories from a set of numerical values

It is useful for comparison btn groups of data.

→ Turning categorical values to numeric variables.

Simple Dataframe operations

df ["symboling"]
df [" body-style"]
} Each of these column is a Panda series.

There are many ways to manipulate Dataframes in Python.

Add one to each column,

df [" symboling"] = df ["symboling"] +1

This will change the data frame column by adding one to the current value.

* Dealing with missing values in python

Missing values :-

→ What is missing value ?

→ missing values occur when no data value is stored for variable (feature) in an observation.

→ could be represented as "?", "N/A", 0 or just a blank cell.

How to deal with missing data ?

→ i) check with the data collection source (... can go back and find what the actual value should be)

ii) drop the missing values
→ drop the variable
→ drop the data entry
} If u dont have a lot of observations with missing data usually dropping the particular entry is the best

Replace the missing values

→ Replace it with an average (of similar datapoints)

* But what if the values cannot be averaged as with categorial variables?

→ Replace it by frequency

→ Replace it based on other functions (because the data gathered knows something additional about the missing data. For eg, he may know that the missing value tend to be. . . .

* Leave it as missing data

* How to drop missing values in python

- Use dataframes.dropna():

built-in method

you can choose to drop rows or columns that contain missing values like NaN.

Specification: axis=0 drops the entire row
axis=1 drops the entire column

```
df.dropna(subset=["price"], axis=0, inplace=True)
```

It allows the modification to be done on the data set directly. Just write result back into data frame

```
df = df.dropna(subset=["price"], axis=0)
```

df.dropna (subset = ["price"], axis = 0)

↓

doesn't change dataframe

To modify the dataframe :-

df.dropna (subset = ["price"], axis = 0,
inplace = True)

ii) To replace with actual values :-

replace → built-in method

Use dataframe.replace (missing_value, new_value)

(1) calculate mean :-

mean = df ["normalized-losses"].mean()

(2) Use method Replace :-

df ["normalized-losses"].replace (np.nan, mean)

Data formatting in Python

Data is in different units, and conventions and the pandas methods that help us to deal with this issue.

* Data formatting :-

→ Data are usually collected from different places and stored in different formats.

→ Bringing data into a common standard of expression allows user to make meaningful comparison.

Data formatting ensures the data is consistent and easily understandable.

Non-formatted :-

→ confusing

→ hard to aggregate

→ hard to compare

city
NY
New York
NY
NY

Formatted :

→ more clear

→ easy to aggregate

→ easy to compare

city
New York
New York
New York
New York

Applying calculations to an entire column

- convert "mpg" to "L/100km" in Car dataset
  (miles per gallon)
- df ["city-mpg"] = 235/ df ["city-mpg"]
  df.rename (columns = {"city-mpg" : city-L/100km"},
  inplace = True)

Incorrect data types

→ sometimes the wrong data type is assigned to a feature.

df ["price"] . tail (5)

... Here dtype : object

- Expected data type should be an integer or float type.

It should be corrected otherwise, the developed model may behave strangely and totally valid data may end up being treated like missing data.

* Data types in Python and Pandas :-

→ There are many data types in pandas

- objects = "A", "Hello" ....
- Int 64 = 1, 3, 5
- Float 64 = 2.123, 632.31, 0.12

* Correcting data types :-

To identify data types:

- Use dataframe.dtypes() to identify the data type.

To convert data types:

- Use dataframe.astype() to convert data type.

Example: convert data type to integer in column "price"

```
df["price"] = df["price"].astype("int")
```

Data Normalization in Python

→ Uniform the features value with different range.

normalization :- Enables a fair comparison b/n the different features making sure that they have same impact.

Eg. NoL-normalized

- age and income are in different range.
- hard to compare
- income will influence the result more

Age	income
20	100000
30	20000
40	50000

⇓

Normalized

- similar value range
- similar intrinsic influence on analytical model.

Age	income
0.2	0.2
0.3	0.04
0.4	1

Several approaches for normalization :-

① simple feature scaling

$$x_{new} = \frac{x_{old}}{x_{max}} \qquad 0 < range < 1$$

② Min-max

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}} \qquad 0 < range < 1$$

③ Z-score/ standard score.

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

μ ← average of the feature

σ ← standard deviation

typically 0 $\qquad -3 < range < 3$

With pandas :-

① df["length"] = df["length"] / df["length"].max()

② df["length"] = (df["length"] - df["length"].min()) / (df["length"].max() - df["length"].min())

③ df["length"] = (df["length"] - df["length"].mean()) / df["length"].std()

Lect : Binning in Python

- Binning :- Grouping of values into "bins"
- Convert numeric into categorical variables
- Group a set of numerical values into a set of "bins"

Binning can improve ~~efficiency~~ accuracy of the predictive models

- "price" is a feature range from 5,000 to 45,500 (in order to have a better representation of price)

price : 5000, 12000, 30000, 31000, 39000, 44500

bins : low | mid | High

Binning in python pandas :-

```
bins = np.linspace(min(df["price"]), max(df["price"]), 4)
group_names = ["Low", "medium", "High"]
df["price_binned"] = pd.cut(df["price"], bins, labels = group_names, include_lowest = True)
```

Lect : Turning categorical variables into quantitative variables in Python

Categorical Variables :-

Problem :- Most statistical models cannot take in the objects / strings as i/p and for model training only take the numbers as inputs.

Car	Fuel	...
A	gas	...
B	diesel	...
C	gas	...

Solution :- Add dummy variables for each unique category

- Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

"one-hot encoding"

* Dummy variables in Python pandas
- Use pandas.get_dummies() method
- Convert categorical variables to dummy variables (0 or 1)

```
pd.get_dummies(df['fuel'])
```

fuel
gas
diesel
gas
gas

The get_dummies method automatically generates a list of numbers

↓↓

gas	diesel
1	0
0	1
1	0
1	0