Module 2 - Python P.2

Pre-Processing Data in Python

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

=

Data cleaning

=

Data Wrangling

1. Identify and handle missing values
2. Data formatting
3. Data Normalization (centering / scaling)
4. Data Binning
5. Turning categorical values to numeric variables

Simple Dataframe Operations - Access a column

df["Symboling"]

df[Column name] } Access a column by specifying the name of the column.

How to drop missing values in Python

Use dataframe.dropna()

Dealing with missing values

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

- ✓ Check with the data collection source
- ✓ Drop the missing values } Remove
  - Drop the variable
  - Drop the data entry
- ✓ Replace the missing values } Always is the prefered option
  - Replace it with an average (of similar datapoints)
  - Replace it by frequency
  - Replace it based on other functions

Categorical values can be replaced by moda (frequency)
Numerical values can be replaced by mean (promedio, min max, etc)

How to drop missing values

Use df.dropna() Quitar filas o columnas

Axis = 0 drops the entire ROW
Axis = 1 drops the entire COLUMN

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

"In place = True" just write the result back into the dataframe
To modify the dataframe, you have to set the parameter "inplace" equal to true.

How to replace missing values

Use df.replace (missing-value, new-value):

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

Especificamos el valor que queremos reemplazar

El valor por el que queremos sustituir

normalized losses	make
164	audi
164	audi
NaN	audi
158	audi
...	

⇒ Replace it by mean

Data Formatting

- Bringing data into a common standard of expression allows users to make meaningful comparison.

City
NY
New York
N.Y

⇒

New York
New York
New York

Formatted
- More clear
- Easy to aggregate
- Easy to compare

Applying calculations to entire column

Ej. Convertir millas a Km

City-mpg
21
21
19

⇒

City L/100 km
11.2
11.2
12.4

Rename the name of the column

```
df ["city-mpg"] = 235 / df ["city-mpg"]
df.rename (columns = {"city-mpg": "city-L/100 km"}, inplace=True)
```

Norma

Incorrect data types

Sometimes the wrong data type is assigned to a feature

```
df["price"].tail(5)
```

```
... ...
... ...
... ...
... ...
204 2265
Name: price, dtypes: object
```

Can be letters or words
Int64 are Integers
float are real numbers
+ many other

```
df.dtypes()
```

Correcting data types

To identify data types

- Use `df.dtypes()` to identify data type.
- Use `df.astype()` to convert data type.

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

Data Normalization

age	income
20	10 000
30	2 000
40	500 000

Not-normalized

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

After normalization, both variables have a similar influence on the model

age	income
0.2	0.2
0.3	0.04
0.4	1

Normalized

Normalized variables into values that range from 0 to 1

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

Norma

Methods of normalizing data

Several approaches for normalization:

① $x_{new} = \frac{x_{old}}{x_{max}}$ Simple feature scaling

② $x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$ Min-Max

③ $x_{new} = \frac{x_{old} - \mu}{\sigma}$ μ → average of a feature, σ → standard deviation .. Z-score

New value range between 0 and 1

length
168.8
168.8
180.0

⇄

length
0.81
0.81
0.87

Simple feature scaling

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

length
168.8
168.8
180.0

⇄

length
0.41
0.41
0.58

Min-Max

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

length
168.8
168.8
180.0

length
-0.034
-0.034
0.089

Z-score

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

Binning

- Binning : Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins"
- "price" is a feature range from 5000 to 45500
  (in order to have a better representation of price)

Price : 5000, 10000, 12000, 12000 | 30000, 31000 | 39000, 44000, 44500

Bins: [Low] [Mid] [High]

Binning in Python

bins = np.linspace(min(df["price"]), max(df["price"]), 4)

We use linspace to return the array "bins" that contains 4 equally spaced numbers over the specified interval of the price.

group_names = ["low", "Medium", "High"]

df["price-binned"] = pd.cut(df["price"], bins, labels = group_names, include_lowest = True)

We use function "cut" to segment and sort the data values into bins.

Turning categorical ~~values~~ variables into quantitative variables in Python

Categorical → Numeric

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"

Categorical Values

Problem:

- Most statistical models cannot take in the objects / strings as input

Car	Fuel		gas (feature)	diesel (feature)
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

- Use `pd.get_dummies()` method
- Convert categorical variables to dummy variables (0 or 1)

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

Automatically generates a list of numbers, each one corresponding to a particular category of the variable.