PART 1 - Data Preprocessing.

#workflow

Data Preprocessing
- import data
- clean data -* important step IRL.
- split into train & test sets.

Modelling
- Build Model
- Train Model
- Make Predictions.

Evaluation
- Calculate performance matrix
- Make verdict.

Train-Test split.

usually 80/20 percent
↓ Training ↳ → Use trained model on test set that has known results to compare with models result.

Feature-Scaling

Remember (always applied to columns & never on rows).

(1) Normalization $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$ [Most values b/w 0 & 1]

where x is the value of column
x min is lowest & x max is highest

(2) Standardization $x' = \frac{x - \mu}{\sigma}$ where μ is mean/average & σ is std. deviation

[Most values except [+3, -3] extremes or outliers.]

- with unscaled features, the lower valued/unit column become irrelevant & make small difference to large scale values.

Practical.

(1) Import libraries → allows to import a library or function or any module.

① numpy - works with arrays for inputs.
② matplotlib - for charts/graph/visualizations.
③ pandas - import dataset / create matrix of features & dependent variable vector.

- Libraries - ensumble of module containing ~~modules~~ functions & classes.

- import numpy as np → alias.
  we can call with np ~~wod~~ now.
- import matplotlib.pyplot as plt.
  → interested in this module.
- impat pandas as pd.

(2) Import dataset

① dataset = pd.read_csv('filename.extension').
- create a dataframe with all values in dataset.

② create 2 new entities
① matrix of features & ② dependent variable vector

* features (independent variables) - the column with which we are going to predict dependent variable.
* dependent variables - last column of dataset that need to be predicted

full form - index locate

feature of pandas, allows us to take index of columns / rows we wanna extract.

* `x = dataset.iloc[:, :-1].values`

takes all rows (in python it means a range without UB or UB means all).

Here `[:, :-1]`

Row: all; column: (all except last).

so,

* `y = dataset.iloc[:, -1]`

(3) Handling Missing Values.

Method I: Just delete the row of missing value.
- works good for large datasets with low % of missing data.

Method II: Replace missing value by average of all data value in column.

* Scikitlearn - has lot of tools & preprocessing tools for this task.
* we use a module called impute in scikitlearn & ~~a class~~ import the SimpleImputer from it.

value to replace

`imputer = SimpleImputer(missing_value = np.nan, strategy = mean)`

to be replaced with

- Now, this object class has functions that will be used to find missing values & then impute it.
  - `imputer.fit(x[:, 1:3])` (upper bound is excluded in python (3-1) = 2 cols.)

expects all the columns of x with numeric value

* `imputer.transform(x[:, 1:3])` ← returns new updated matrix of features

So to change original as it have feature matrix
- X [: , 1:3] = imputer.transform(X[:, 1:3])

④ Encoding Categorical Data.

* It will be difficult to co-relate / compute these features with category string data, Thus we need to turn these categories to numbers.

~~Method~~ Idea I: Encode France to 0
Spain to 1
Germany to 2

- However model may interpret as order / numeric order & interpret as the order matters.

Better To-Do: ~~Turn~~ Turn to one hot encoding i.e. turn the category into total columns with total no. of distinct category.

So for France spain & germany we ge.

~~for~~ France	1	0	0
Spain	0	1	0
germany	0	0	1

① Encode InDependent Variable

- For this we use 2 classes
  1. column transformer class from composed module of scikit
  2. one hot encoder class from preprocess module of scikit

① ct = ColumnTransformer(transformers, remainder)

- transformers → what kind of transformation on which indexes; to specify we are doing encoding transformations
- remainder → columns that won't be applied for transformations.

* transformers = [('encoder', OneHotEncoder(), [0])]
  - 'encoder' → class name to encode to
  - [0] → indexes to transform / encode

codename to say we wannaa keep columns as it is & wont apply transformation.

remainders = 'passthrough'

↑ If we dont apply the remaining columns wont be added.

- with this class, we can fit and transform at once with function

x = ct.fit_transform(x)

↑ input to transform.

↑ Machine learning use numpy array so we need to convert it to np array

We call x = np.array(ct.fit_transform(x))

(2) Encode Dependent variable.

- we use LabelEncoder class from preprocessing from scikit

lbl = LabelEncoder()

↑ only one column with text value, so its obvious what needs to be performed

y = lbl.fit_transform(y).

↑ doesnt need to be numpy array since its single dependent var. array.

(3) Splitting Dataset into Train & Test set.

※ We apply feature scaling after splitting data.

↓ because we only use train set for ML model while test set represents real world data. ~~& shouldn't be altered. It just needs to be fed to model & get op.~~

- Also when applying feature scaling we use mean & standard deviat that might leak some info to train set which isn't supposed to happen. The ML model will get into on test set & may fit itself for it.
- We scale test later after splitting & scaling train set.
For this will use sklearn module that has model_selection class with train_test_split fn.

- We split into 4. 2 feature matrices & 2 depend variables.

expects first feature mat

x_train, x_test, y_train, y_test = train_test_split(x, y

next expects dep. var

~2 more args ~ train_size = 0.8 or test_size = 0.2, random_state = 1)

int type seed for random value.

(6) Feature Scaling (Not necessary for all models).

works well all the time.

2 techniques (1) standardization (with mean(μ) & deviation(σ)).
↳ (+3, -3).

(2) Normalization (with x_{min} & x_{max}).
↳ (0, 1)

Works/recommended if we have normal distribution among most of our features.
(recommended for specific situations).

- For this step, we use same scikit module.
from sklearn.preprocessing import StandardScaler
↑ class.

sc = StandardScaler() ← automatically applies standardisation.

- Dummy variables → variable columns after encoding categorical data.

- We dont need to apply standardisation to dummy variables since they are already b/w (-3, 3) or (0, 1).

① First we fit our scaler on training sets.

so x_train[:, 3:] = sc.fit_transform(x_train[:, 3:])

↑ everything after & with 3rd column

② Next we fit x_test.

- we only apply transform method here since this data is like new data received IRL by model.
- The features of test set need to be scaled by the same scales used on training set.
- we donot calculate a new fit. we got that from fit_transform earlier

x_test[:, 3:] = sc.transform(x_test[:, 3:])

SUMMARY:

① All Libraries for preprocessing

- import pandas as pd [General]
- " numpy as np
- from sklearn.impute import SimpleImputer (class) (for Missing values)
- " sklearn.compose " ColumnTransformer (class) (for encoding categorical data)
- " sklearn.preprocessing " OneHotEncoder (class)
- " " " LabelEncoder (class) (for encoding label/dependent data)
- " sklearn.model_selection " train_test_split (function) (for splitting)
- " sklearn.preprocessing " StandardScaler (class) (for feature Scaling)

② To import dataset (and assign dep. & indep. variable).

```
dataset = pd.read_csv('./loadfilename')
X = dataset.iloc[:, :-1]
y = "  "  [:, 1]
```

③ Handle: Missing data

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(X[:, 1:3])
X[:, 1:3] = impurer.transform(X[:, 1:3])
```
→ first fit to compute
→ & then transform to replace

④ Encoding Categorical Data.

① Indep. variable.

```
colTr = ColumnTransformer(transformers=[('encoding', OneHotEncoder(),
                                         [0])], remainder='passthrough')
X = np.array(colTr.fit_transform(X))
```

② Dep. variable.

```
lbEnc = LabelEncoder()
y = lbEnc.fit_transform(y)
```

⑤ Split Data

```
X_tr, X_tst, y_tr, y_tst = train_test_split(X, y, test_size=0.2,
                                            random_state=0)
```

⑥ Scaling (only when necessary)

```
sclr = StandardScaler()
X_train[:, 3:] = sclr.fit_transform(X_train[:, 3:])
y_train[:, 3:] = sclr.transform(X_test[:, 3:])
```