Simple Linear Regression

- Statistical approach for modelling the relationship between a predictor variable X and response variable Y.

$X \xrightarrow{\text{linear relationship}} Y$ → we use X, Y to predict a quantitative output

Simple Linear Regression $\xrightarrow{\text{simple approach}}$ Supervised learning

- SLR is fundamental starting point for all regression methods.

- It is relationship b/w response and predictor

Mathematically, linear relationship:

$Y = \beta_0 + \beta_1 x + e$; where

- Y → o/p variable (also called response, target or dependent variable).

e.g. house prices

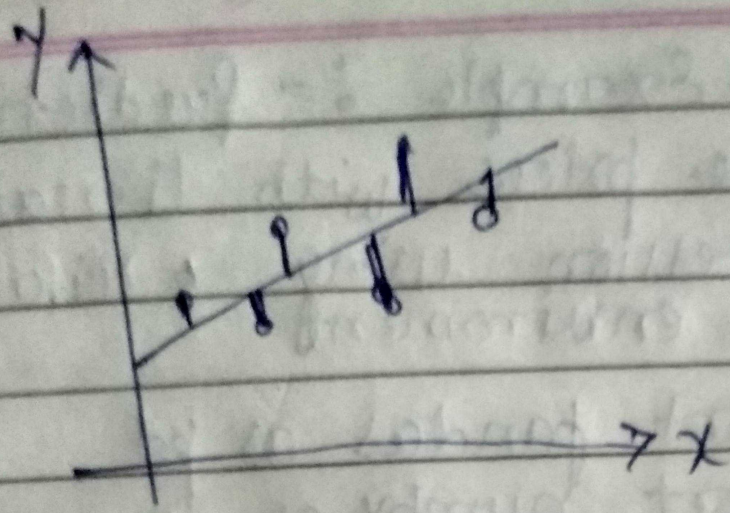
- x is the i/p variable (also called feature, explanatory or independent variable) e.g. size of house in squared meters.

- β_0 is the intercept (the value of y when x=0)

- β_1 is coefficient for x and slope of the regression line. ("the average increase in y associated with a one-unit increases in x")
- e is error term

→ Implementing linear regression → the algo. finds the line of best fit using the model coefficients β_0 and β_1, such that it is as close as possible to the actual points (minimise the sum of squared distances between each data point and data line.

Once we find β_0 and β_1, we can use the model to predict response.



- Black dots → observed values of x and y (actual data)
- the dark line → line of best fit → is the line that minimised the sum of squared errors.
- the blue line → are the errors (or residuals) — the vertical distances b/w the observed values and the ~~line of~~ line of best fit.
- The slope of dark line is β_1
- β_0 is the intercept (the value of y when $x = 0$)

An Example :- Predicting house prices with linear regression using scikit-learn.

Setting the environment

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
% matplotlib inline

from sklearn.linear_model import
                    LinearRegression
from sklearn.model_selection import
    train_test_split, crossval_score

from sklearn.metrics import mean
    _squared_error
```

Read house prices data:

```
houses = pd.read_csv("Kc_house_data
                        .csv")
houses.head()
```

Top 5 rows of the houses data set.

Data Dictionary

v. Data Cleansing and exploratory analysis

#Check for nulls in the data
houses.isnull().sum()

Output:
id 0
date 0
}

This dataset doesn't contain any null values

#Check for any correlation between variables
corr = houses.corr()
sns.heatmap(corr)
# sqft_living, grade, sqft_above and
#d sqft_living15 seems to have a high influence in price

→ Building a linear regression model

1. Prepare the data: define predictor and responsive variables

Create x and y

feature_cols = ['sqft_living']

x = house[feature_cols] # predictor

y = houses.price # response

2. Split data into train and test

The train/test split consists in randomly making 2 subsets of the data:

the training set, used to fit our learning algorithms so it learns how to predict,

and the test which we use to get get an idea of how the model would perform with new data.

```
# split data into train and test
x_train, y_test, y_train, y_test
    = train_test_split(
    x, y, test_size = 0.2)
# the test set will be 20% of
    the whole data set
```

3. Fit the model on the training set.

```
# instantiate, fit
linreg = LinearRegression()
linreg.fit(x_train, y_train)
```

4. Print the coefficients

```
print linreg.intercept
print linreg.coef-
-46773.6549892
[282.29917574] # for an increase
of 1 square meter in house size,
# the house price will go up by.
~$282, on average.
```

The intercept (β_0) is the value of y when $x = 0$. In this case it would be the price of a house when the sqft_living is 0.

(It doesn't always make sense to interpret the intercept.)

The coefficient of β_1 is the change in y

Teacher's Signature .....

divided by change in x (i.e the derivative, the slope of the line of best fit).
An increase of 1 square meter in house size is associated with a price increase of $ ~~35823~~ 282.3, on average.
Note that association doesn't always imply causation.

5. Predict the price of 1000 sqft living house using our model:
# manually
price = -6773.6549892 + 1000 * 282.29917574
# using the model
linreg.predict(1000)
array([238175.93397947])

6 Compute the Root Mean Squared Error (RMSE), which is a commonly used metric to evaluate regression models, on the test set:

rmse = mean_squared_error(y_test, linearg
.predict(X_test))
np.sqrt(rmse)
259163.~~4880~~4839816292 2 #not great

~~Lines~~
Linreg.score(X_test, y_test)
0.5543314276486042 1

We got a root mean squared error of $259,163.48 when predicting a price for a house, which is really high. This is kind of expected

- Using only one features in our model, ~~and it could be greatly~~ is

The R squared coefficient: 55%.

→ model is only able to explain 55% of the variability in house price.

→ multiple predictors (or features) we call it Multiple linear regression.

(ii) The inference call (.predict(1)) call requires 4 features for test sample in the form of a numpy array.

2. wrapping the inference logic into a flask web service

→ We have the trained model file,
→ we are ready to query the model to get a class label for a test sample.
→ The inference ⟶ as simple as ⟶ calling a predict() function on trained model with the test data.
→ However, we would like to build the inference as a web-service.
We use flask.
Flask → powerful Python microwebservice framework ⟶ allows to build ~~rest API~~ REST API based web-services quickly with minimum configuration hassle.

⟶ Code

→ Code :-

a. I. Define a simple function to load the trained model file.

```
model = None
def load_model():
    global model
    # model variable refers to the
      global variable
    with open('iris_trained_model.pkl',
      'rb') as f:
        model = pickle.load(f)
```

Here, define a global variable → 'model'
and populate 'model' with load_model() function.

b. Next, we instantiate a Flask object called 'app'.

```
app = Flask(__name__)
```