Concept

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression.

To give an overview, ML models can be classified on the basis of the task performed and the nature of the output:

Regression: Output is a continuous variable.
Classification: Output is a categorical variable.
Clustering: No notion of output.

Regression & Classification fall under supervised learning while Clustering fall under unsupervised learning

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).

Regression helps investment and financial managers to value assets and understand the relationships between variables, such as commodity prices and the stocks of businesses dealing in those commodities.

Linear Regression (LR)

By the term itself, you can guess that the model functions along a straight line, where y can be calculated from a linear combination of the input variables x. Linear Regression hypothesis function can be formulated as:

$$h_\Theta(x) = \Theta_0 x_0 + \Theta_1 x_1 + \Theta_2 x_2 + \ldots + \Theta_n x_n$$

Θ here stores the coefficients/weights of the input features **x** and is of the exact same dimensionality as **x**. Note that to add support for a constant term in our model, we prefix the vector **x** with **1**.

With single input and output variable, the method is called simple linear regression while with multiple inputs/features, it's called multiple linear regression.

$$\text{SSE} = \sum_i^n (actual\ y^i \ - predicted\ y'^i)^2$$

$$\text{MSE} = \frac{SSE}{n}$$

To proceed, we need to find the parameters/coefficients for that best fit line. There are various techniques to arrive at the geometric equation for the line such as least absolute deviations (minimizing the sum of absolute values of residuals) and the Theil–Sen estimator (which finds a line whose slope is the median of the slopes determined by pairs of sample points), however, statisticians typically use the *Ordinary Least Squares method (OLS)*. OLS is nothing but a method to minimize the distances between the line and the actual outputs. If you want to calculate the regression line by hand, it uses a slightly scary formula to find the slope '**θ1**' and line intercept '**θ0**'.

$$\theta_1 = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{m}(x_i - \bar{x})^2}$$

$$\theta_0 = \bar{y} - \theta_1\bar{x}$$

x = independent variables
$\bar{x}$ = average of independent variables
y = dependent variables
$\bar{y}$ = average of dependent variables

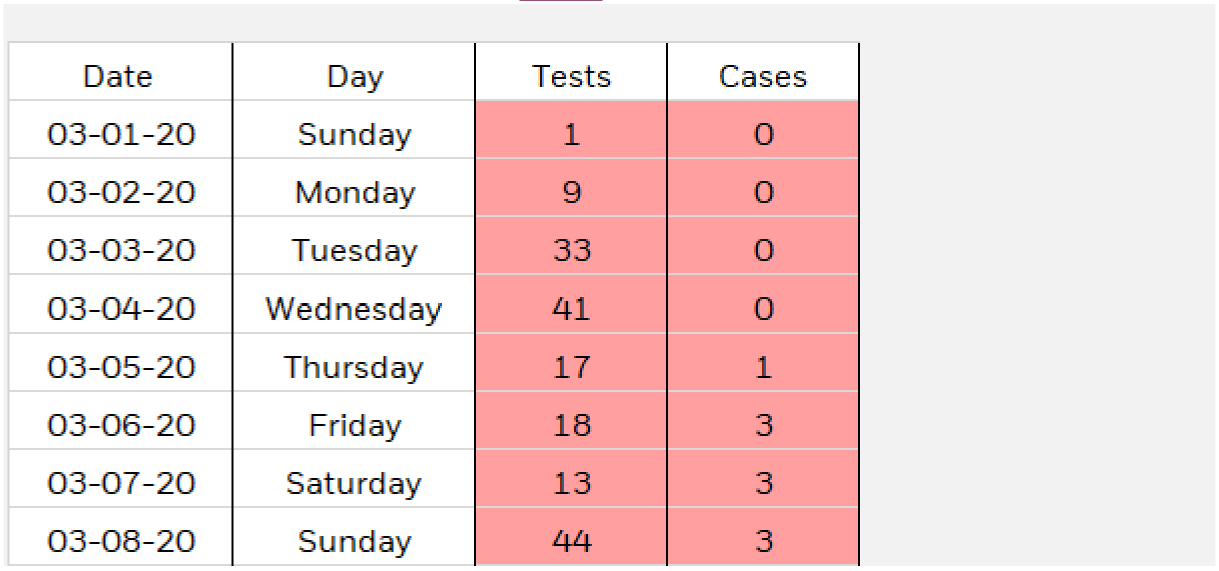
Finding intercept and slope using OLS (Image by Author)

There are 2 approaches/solutions that uses the Least Squares method to implement a linear regression model:

- Closed-form solution — Normal Equation
- An optimization algorithm (Gradient Descent, Newton's Method, etc.)

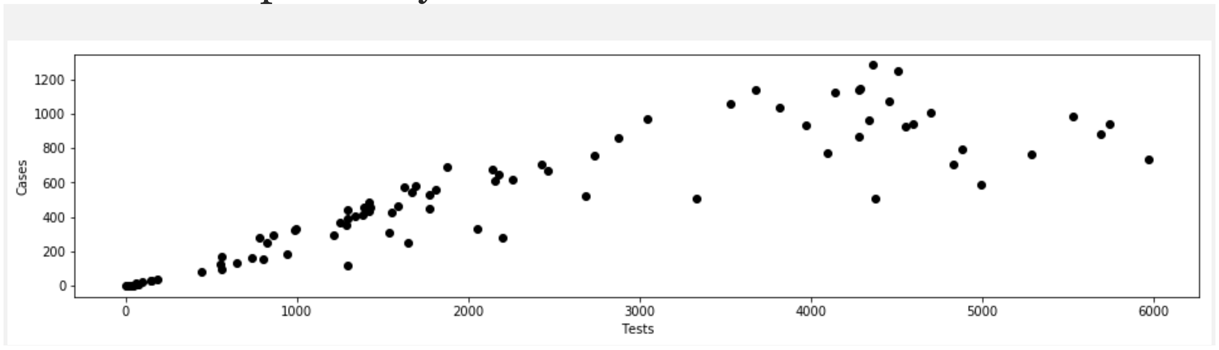
Linear Regression Model for COVID-19

Let us first walk through Chicago's COVID-19 dataset. You can download the dataset from here.



Date	Day	Tests	Cases
03-01-20	Sunday	1	0
03-02-20	Monday	9	0
03-03-20	Tuesday	33	0
03-04-20	Wednesday	41	0
03-05-20	Thursday	17	1
03-06-20	Friday	18	3
03-07-20	Saturday	13	3
03-08-20	Sunday	44	3

80% of the COVID datasets available on the web are in a time series format displaying the counts of cases on a daily basis. So, with just death and test counts, I could only visualize whether the peak has reached or if it is still increasing and so on. But to forecast, I wanted to have features on which the case count can have some dependency.

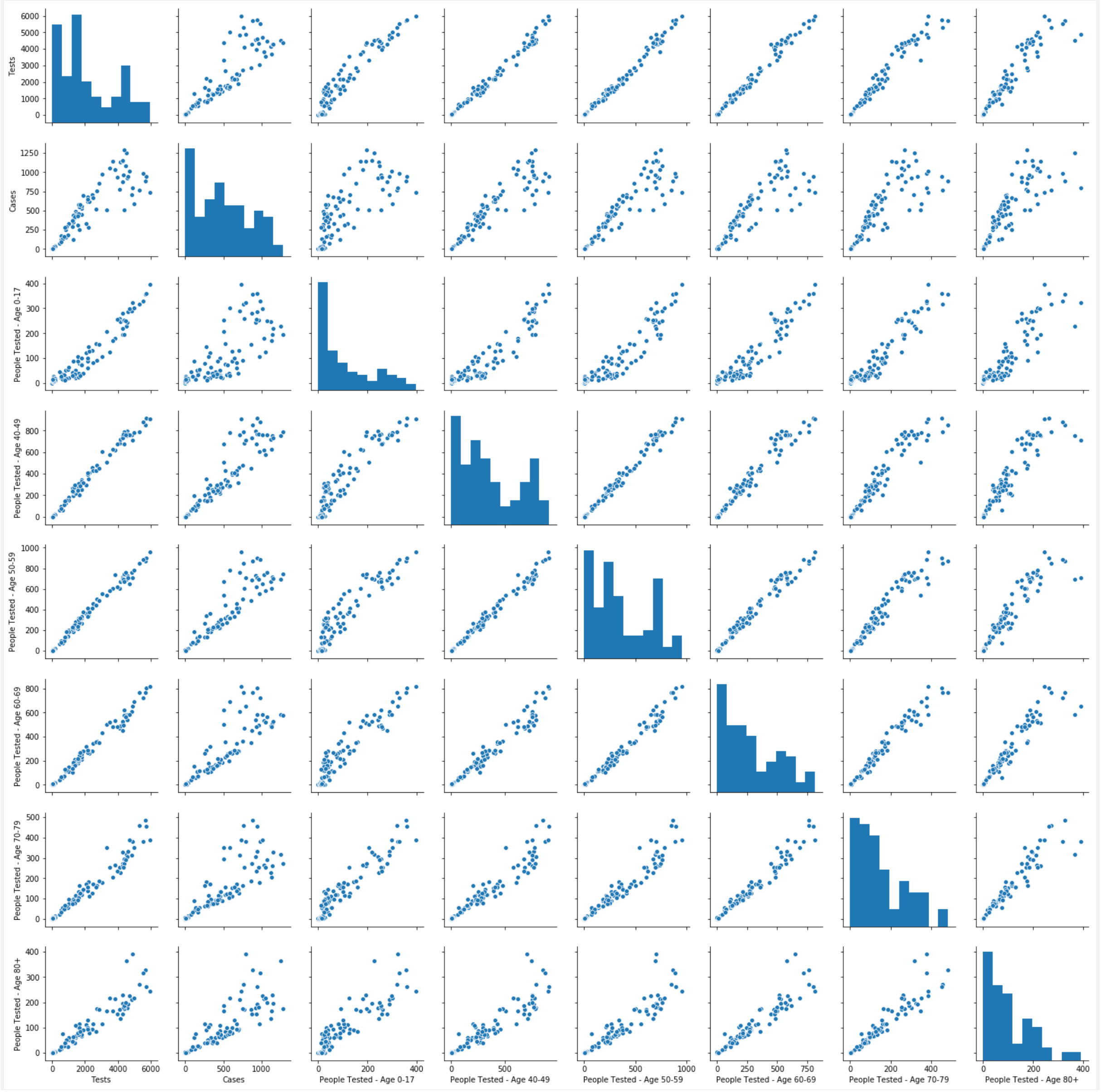


As you can see, the **independent variable is Tests** (x) here and **dependent is Cases (y)**. We have a few more features in the dataset, giving the count of people who are older than 30 or younger and those who are Latin, etc. I personally couldn't deduce what we could analyse with such features. You can try giving it a shot and let me know if we can infer something.

```
#COVID-19

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

#importing CSV dataset using PANDAS
data = pd.read_csv("COVID-19_Daily_Testing.csv")
data.head()

###################### Data
Cleaning#############################

data['Cases'] = data['Cases'].str.replace(',', '')
data['Tests'] = data['Tests'].str.replace(',', '')

data['Cases'] = pd.to_numeric(data['Cases'])
data['Tests'] = pd.to_numeric(data['Tests'])


#To get a concise summary
print(data.info())
```

Often, in real-world datasets, there are a lot of features, and it is difficult to check which pair of features are showing a good correlation. Also, to check if there is any multilinearity problem involved. The below code basically plots pairwise relationships in a dataset such that each variable will be shared both in the x and y-axis.

```
data_numeric = data.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(20, 10))
sns.pairplot(data_numeric)
plt.show()
```



Mostly all of them show a linear pattern that should not happen! This gives rise to the famous multilinearity problem.

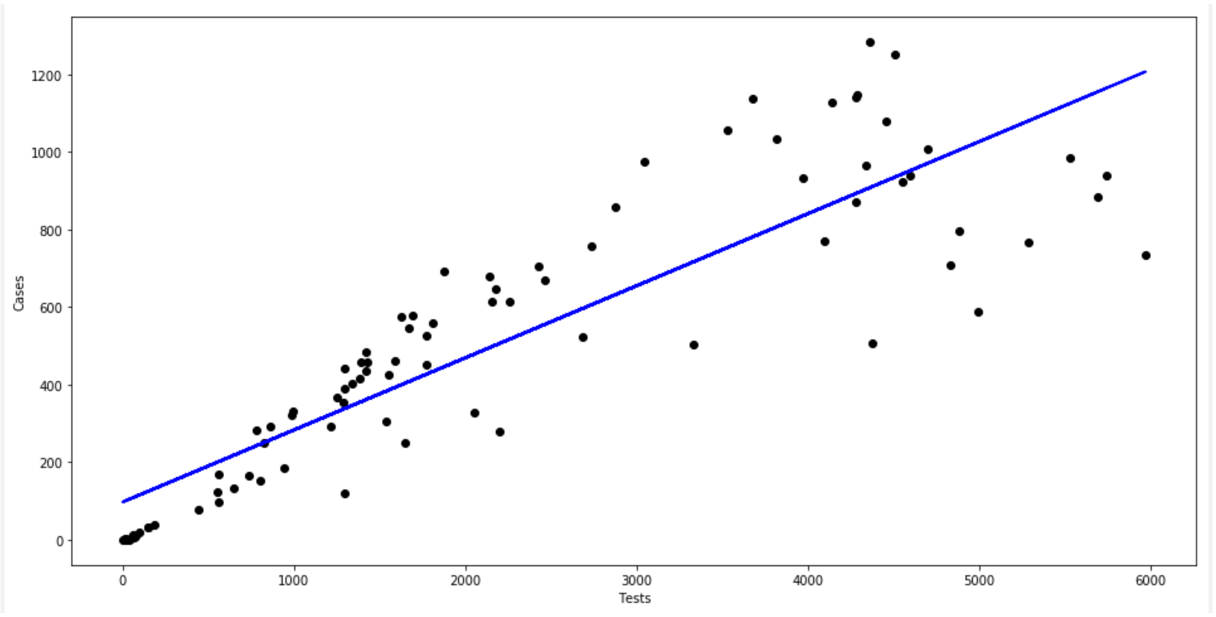
- **Scaling the dataset**
  The feature values might have different scales. So, it is always preferred to convert them on the same scale for better analysis and prediction

```
X = data['Tests'].values.reshape(-1,1)
y = data['Cases'].values.reshape(-1,1)
```

- **Applying Linear Regression**

```
reg = LinearRegression()
reg.fit(X, y)
predictions = reg.predict(X)
print("The linear model is: Y = {:.5} +
{:.5}X".format(reg.intercept_[0],
reg.coef_[0][0]))plt.figure(figsize=(16, 8))
plt.scatter(
    X,
    y,
    c='black'
)
plt.plot(
    X,
    predictions,
    c='blue',
    linewidth=2
)
plt.xlabel("Tests")
plt.ylabel("Cases")
plt.show()
```

We get this linear line **Y = 97.777 + 0.18572X**

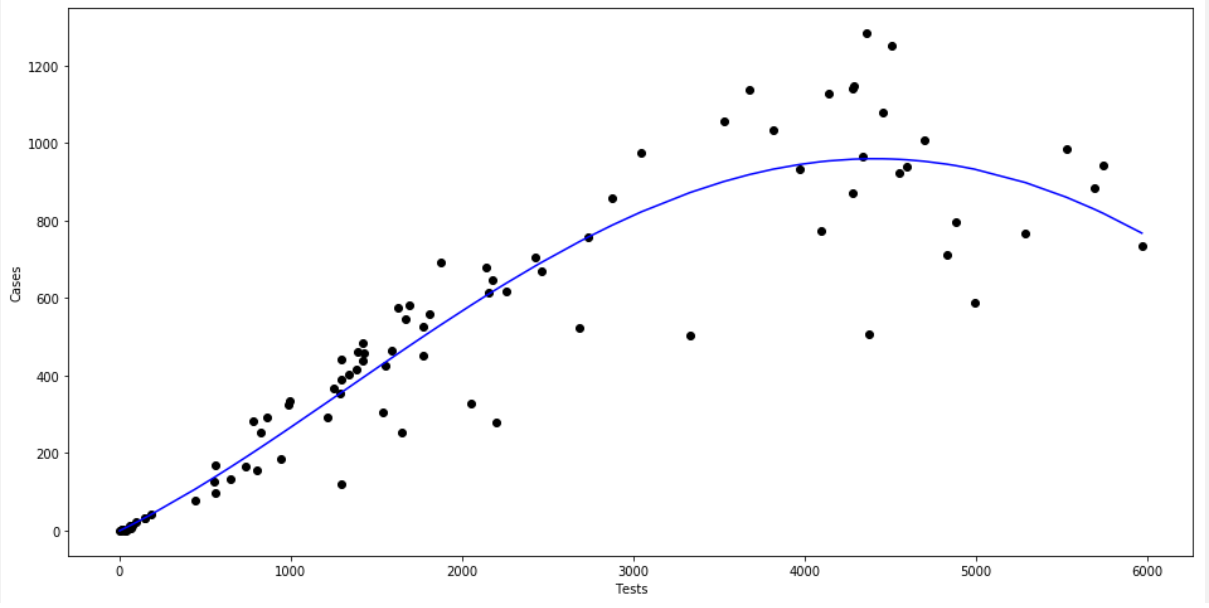


The Root Mean Square Error for Linear Regression =171.8

Polynomial Regression

Special Case of Linear Regression where we try to fit an n degree polynomial equation to the data.

```
poly = PolynomialFeatures(degree =4)
X_poly = poly.fit_transform(X)

poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
pred = lin2.predict(X_poly)
new_X, new_y = zip(*sorted(zip(X, pred)))plt.figure(figsize=(16,
8))
plt.scatter(
    X,
    y,
    c='black'
)
plt.plot(
    new_X, new_y,
    c='blue'
)
plt.xlabel("Tests")
plt.ylabel("Cases")
plt.show()
```



The Root Mean Square Error for Polynomial Regression=> 131.08.

FULL CODE