Stock Prediction

"The field of study that gives computers the ability to learn without being explicitly programmed" is what Arthur Samuel described as Machine Learning.

Machine Learning or rather **ML** has found its applications in various fields in the past few years, some of which include Virtual Personal Assistants, Online Customer Support, Product Recommendations etc.

A brief discussion on the types of Machine Learning

(Feel free to skip this part if you are already familiar with it)

ML can primarily be divided into two types:

- Supervised Learning
- Unsupervised Learning

Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. So, Supervised Learning is the Machine Learning task, of learning a function that maps an input to an output based on example input-output pairs. Supervised Learning Problems can further be categorized into **Classification** and **Regression** problems.

Classification

Classification problems involve prediction of results in a discrete output. In other words we map the input variables into discrete categories. For example, given a picture of a student, we have to predict whether He/She is of High school, College, Graduate age.

Regression

In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. Given data about the size of houses on the real estate market, trying to predict their price. Price as a function of size is a continuous output, so this is an example of a regression problem.

Unsupervised Learning

Unsupervised learning, on the other hand, allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. Unsupervised learning problems can be further categorized into **clustering** and **non clustering** problems.

Clustering

A Clustering problem is where we want to discover the inherent patterns / groupings in the data, such as grouping customers by purchasing behaviour. Another example could we grouping news of similar kind from a collection of news.

Non-Clustering

Identification of individual voices and music from a mesh of sounds at a cocktail party is an example of Non-Clustering Learning.

You can refer to the following link to further enhance your understanding

Difference betwen supervised and unsupervised learning algorithms

So What exactly are we going to make?

In these 10 days of Code we will be using ML, taming the Stock market. We'll create an application which will predict Stock Prices of a company using Supervised Machine Learning.

We will use libraries like `numpy`, `pandas`, `matplotlib`, `sklearn` and a few others. You may have heard about these libraries or you may be encountering their names for the first time and it may seem like jargon but don't worry, we will walkthrough all of them and make you familiar with them

Editor / IDE Setup

We would prefer doing all the coding on an online notebook like `google collab`, `kaggle` etc. However if you are comfortable in working on the project on your local machine and by installing all the required dependencies manually, you are free to do so. We will be using `google collab` for live coding and other demonstration purposes.

How to use google collab

NumPy

Let's start with `NumPy`, Shall we? **NumPy**, which for Numerical Python is an Open Source python library used for working with arrays. It consists of multidimensional array objects and also has functions for working in domain of Linear Algebra, fourier transform and matrices.

So Why use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster that traditional Python lists.

The NumPy module does not come pre-installed with the python distribution and we have to install it seperately. It is often used with libraries such as scipy and matplotlib.

Machine Learning data is universerally represented as arrays and in Python, data is almost universerally represented as NumPy arrays. To get started working on real data we need to know how to manipute and access the data correctly in NumPy arrays.

You can refer to the following tutorial which will demonstrate simple operations on our data such as :

- Converting list data to NumPy arrays
- Accessing data using pythonic indexing and slicing
- Resizing of the data

NumPy Arrays

After going through the above tutorial, you now should be familiar with performing basic operations on Arrays.

You can also refer to the following resources to know further about NumPy

- w3schools
- NumPy documentation

So, lets continue and move on to our next topic **Pandas**

Pandas

Pandas is a very popular open-source Python Library which provides high-performance data manipulation and analysis tools using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze. We will discuss about these steps in the later part of the guide.

Following are some of the key features of Pandas

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

You can also refer to the following resources to know further about Pandas

- tutorialspoint
- Pandas Documentation

Matplotlib

Matplotlib is an open source plotting library which creates static , animated and interactive visualizations in python. It is largely used for the 2-dimensional plotting of arrays. It is a multi platform data visualization library which has been built upon NumPy arrays. One of its remarkable feature is that it lets the user interact with large chunk of data in very easily representable visuals. Matplotlib consists of several plots like line bar, pie chart, histogram, etc. Matplotlib comes with large number of plots ,which helps in determining trends and building correlation such as the line bar, scatter bar, histogram etc.

A Short Tutorial

We have discussed in breif about some of the important libraries which are being used for data analysis. And So now, we would provide a very beginner friendly guide which will demonstrate how to use these libraries and will make you equipped so that you are able to tackle the actual project on your own.

Also, we will have to install these libraries into our system as they do not come pre-installed with python.

```
pip install numpy pandas matplotlib
```

You can create a new python file and follow along.

```
# importing the libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
```

We are going to use the `iris flowers` dataset. This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone.The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species. We will load the iris data from **csv** file URL.

If you don't know what a csv file is, a comma-separated values (csv) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

```
# Load Dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
```

Now, lets take a look at the data we have loaded. Some of the ways to look at the data are as follows:

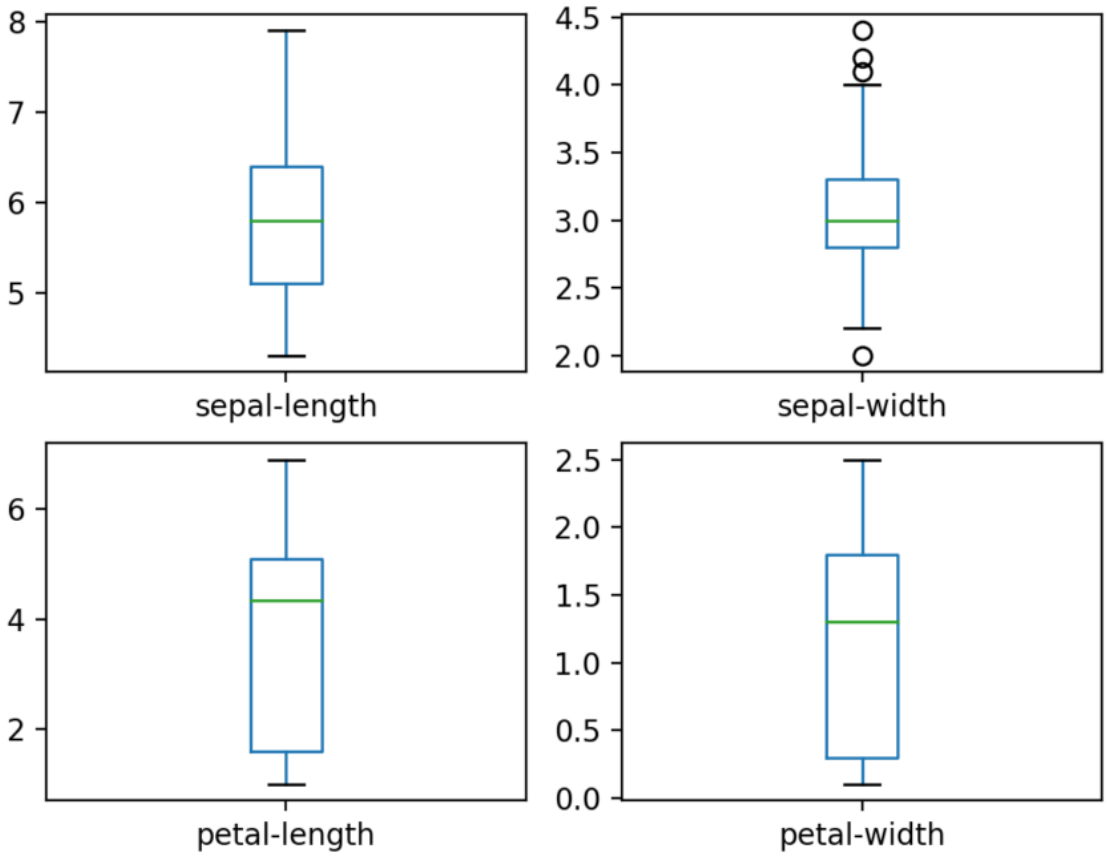
```
# Dimensions of the dataset
print(dataset.shape)

# Peeking into the dataset
print(dataset.head(10)) # prints the first 10 rows of the dataset.

# Statistical Summary
print(dataset.describe())
# prints the summary of each column head(attribute)
# This helps in finding maximum, minimum, count, and even some percentiles.
```

Now we get a basic understanding about our data. Let's dive into the data visualization part.

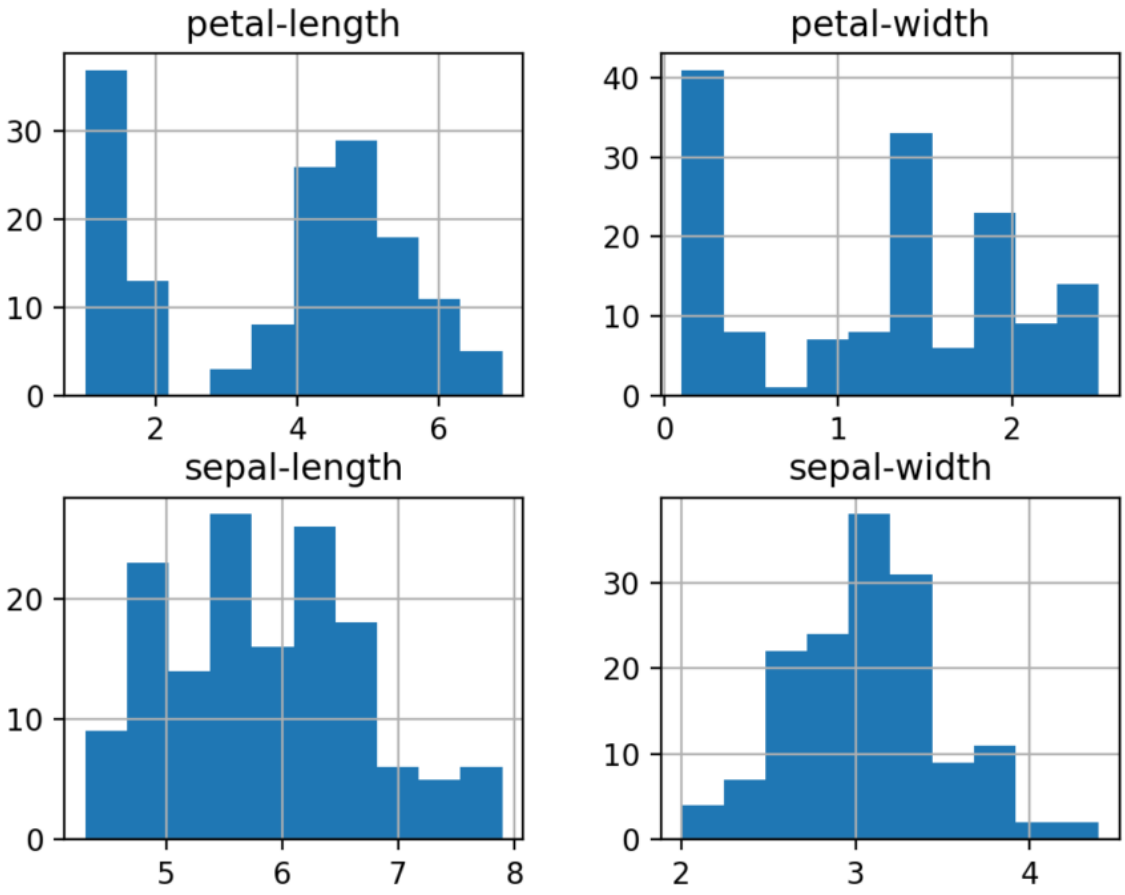
We generally study two types of plots:

- **Univariate plots** to undersatnd each attribute (box and whisker plots, Histograms, etc)
- **Multivariate plots** to understand the relationship between varoius attributes (Scatter plots)

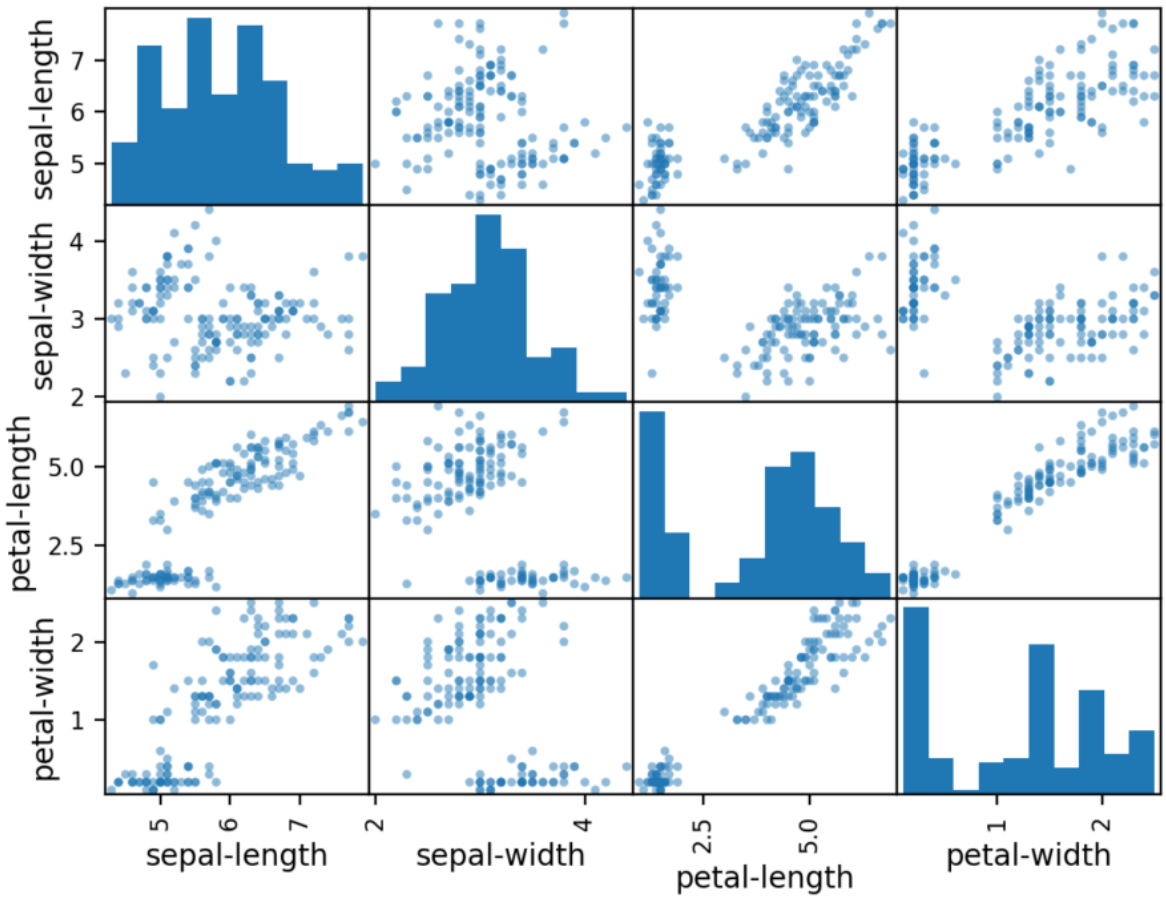
```
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
```




```
# histrograms
dataset.hist()
pyplot.show()
```




```
#scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```



So now you must have an idea of what these libraries help us to do and you might have also got an intuition as to where you can use them in you projects. Although we have only discussed the topics in very brief and there a whole lot of things that these libraries offer and what we can do with them but this should be a really good checkpoint we have reached.

Steps in Machine Learning

While performing any Machine Learning Task, we generally follow the following steps:

1. **Collecting the data** This is the most obvious step. If we want to work on a ML Project we first need data. Be it the raw data from excel, access, text files, or data in the form of images, video etc., this step forms the foundation of the future learning.

2. **Preparing the data** Bad data always leads to bad insights that leads to problems. Our prediction results depend on the quality of the data used. One needs to spend time determining the quality of data and then taking steps for fixing issues such as missing data etc.

3. **Training the model** This step involves choosing the appropriate algorithm and representation of data in the form of the model. In layman terms, model representation is a process to represent our real-life problem statement into a mathematical model for the computer to understand. The *cleaned data* is split into three parts – Training, Validation and Testing - proportionately depending on the scenario. The training part is then given to the model to learn the relationship / function.

4. **Evaluating the Model** Quite often, we don't train just one model but many. So, to compare the performance of the different models, we evaluate all these models on the validation data. As it has not been seen by any of the models, validation data helps us evaluate the real-world performance of models.

5. **Improving the Performance** Often, the performance of the model is not satisfactory at first and hence we need to revisit earlier choices we made in deciding data representations and model parameters. We may choose to use different variables (features) or even collect some more data. We might need to change the whole architecture to get the better performance in the worst case.

6. **Reporting the Performance** Once we are satisfied by the performance of the model on the validation set, we evaluate our chosen model on the testing data set and this provides us with a fair idea of the performance of our model on real-world data that it has not seen before.

Now coming to our project, as we are dealing with the stock market and trying to predict stock prices the most important thing is **being able to Read Stocks**

How to Read Stocks ?

Reading stock charts, or stock quotes, is a crucial skill in being able to understand how a stock is performing, what is happening in the broader market and how that stock is projected to perform.

Stocks have **quote pages** or **charts**, which give both basic and more detailed information about the stock, its performance and the company on the whole. So, the next question that comes up is what makes up a stock chart?

Stock Charts

A Stock Chart is a set of information on a particular company's stock that generally shows information about price changes, current trading price, historical highs and lows, dividends, trading volume and other company financial information.

Also we would like to familiarise you some some basic terminologies of the stock market

Ticker Symbol

The ticker symbol is the symbol that is used on the stock exchange to delineate a given stock. For example, Apple's ticker is (AAPL) while Snapchat's ticker is (SNAP).

Open Price

The open price is simply the price at which the stock opened on any given day

Close Price

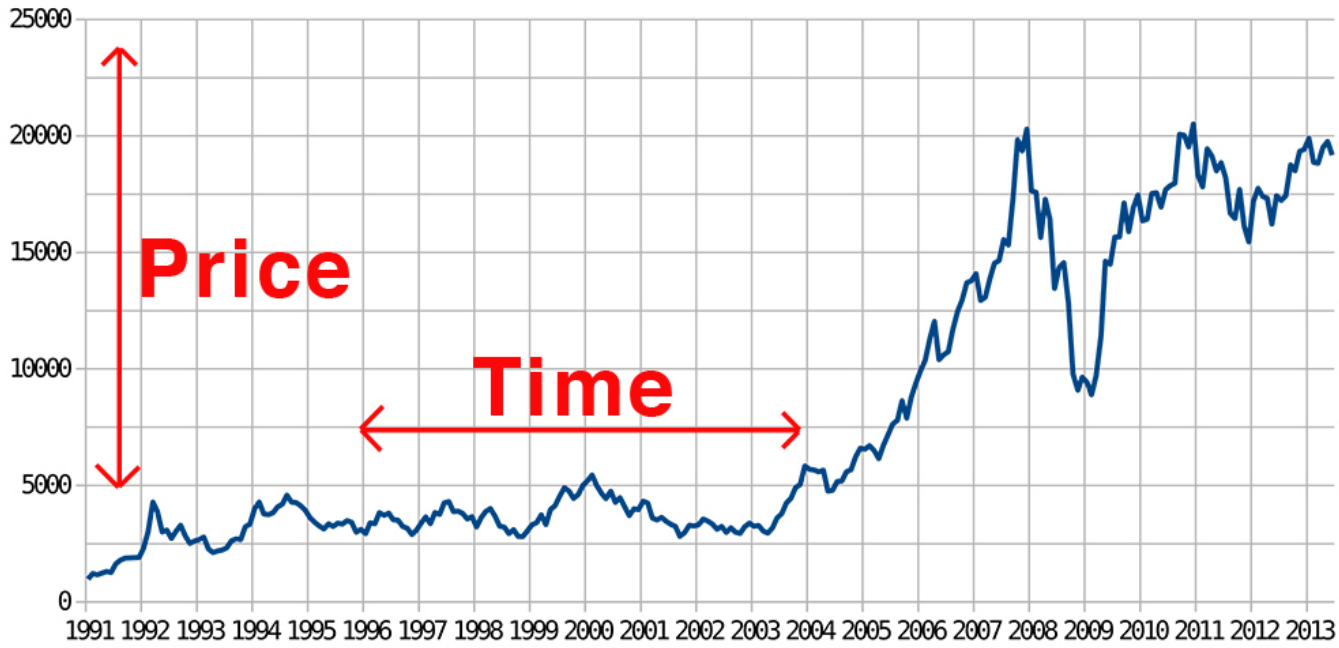
The close price is perhaps more significant than the open price for most stocks. The close is the price at which the stock stopped trading during normal trading hours (after-hours trading can impact the stock price as well). If a stock closes above the previous close, it is considered an upward movement for the stock. Vice versa, if a stock's close price is below the previous day's close, the stock is showing a downward movement.

How to Read a Stock Chart ?

A Stock Chart is a little different than the basic information on the stock. Stock charts include plot lines which represent the price movements of the given stock. While the price lines are generally represented in a line or a mountain chart form, the price movements over a given period (generally six months to one year) is represented by thin lines.

1. The Price and Time Axes

Every stock chart has two axes - the price axis and the time axis. The horizontal (or bottom) axis shows the time period selected for the stock chart. The vertical (or side) axis shows the price of the stock. These two axes help plot the trend lines that represent the stock's price over time, and are the framework for the whole stock chart.



2. The Trend Line

This should be pretty obvious, but a good bit of the information one can glean from a stock chart can be found in the trend line. Depending on the type of chart one is looking at, one can choose different chart styles including the traditional line, mountain, bar, candlestick and other chart styles. The traditional **Line charts** simply track the price movements of a stock using the last price of that stock.

3. Trading Volume

In addition to just the trend of the stock's prices, the stock's trading volume is another key factor to look at when reading a stock chart.

The volume is generally indicated on the bottom of the stock chart in green and red bars (or sometimes blue or purple bars). The key thing to look out for when examining trading volume is spikes in trading volume, which can indicate the strength of a trend - whether it is high trading volume down or up. If a stock's price drops and the trading volume is high, it might mean that there is strength to the downward trend on the stock as opposed to a momentary blip (and vice versa if the price moves up)

An Example Stock Chart

So, this was little bit of knowledge about the Stock Market and how to read the Stock charts.

Now its time to get your hands dirty and begin setting up the project

Initializing our project

To work with Stock Market prices we first need to get reliable financial data. So where do we get our data from? `yfinance` solves our problem by offering a reliable, threaded, and Pythonic way to download historical market data from Yahoo! finance. And this library is installed pretty much the same way as we have installed other libraries. So go ahead and install it.

Also to download the stock prices data of a particular company using the `yfiance` library you will need to know their **Ticker Symbol**

All stock ticker symbols

yfinance GitHub Repository

Use the `yfinance` library to download the dataframe. The dataframe which we get contains daily data about the stock. The downloaded dataframe gives us lot of information including Opening Price, Closing Price, Volume, etc. But we are interested in the opening prices with their corresponding dates.

Also it would convinient to convert the dates to their corresponding time-stamps. And finally we will be having a dataframe which will contain our opening prices and time-stamps.

We are getting the day-wise and this much data is not sufficiently enough to train our model properly. So we use an `api` to get the stock prices of the previous day for every minute. Following is the link to the api endpoint

Previous day Stock Prices

So now you get the data. You can use the `requests` and `json` modules to use the data and accordingly append it to your dataframe.

Also to further improve the performance of our results and make even more accurate predictions we also add the current day's stock prices using the follwing api endpoint

Today's Stock Prices

After appending the current day's prices we have sufficent number of records to train our models.

Create a Validation Dataset

We need to know that the model we created is good. We are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be.

We will split the loaded dataset into two, 80% of which we will use to train, evaluate and select among our models, and 20% that we will hold back as a validation dataset.

Following is a sample code

```
# Split-out validation dataset
dataset = prices_dataframe.values
X = ...
y = ...
# X and y are obtained by array slicing
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)
```

The function `train_test_split()` comes from the `scikit-learn` library.

scikit-learn (also known as sklearn) is a free software machine learning library for the Python. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data.

Building our Models

We don’t know which algorithms would be good on this project or what configurations to use.

And So, we are testing with 6 different alogrithms:

- Linear Regression (LR)
- Lasso (LASSO)
- Elastic Net (EN)
- KNN (K-Nearest Neighnors)
- CART (Classification and Regression Trees)
- SVR (Support Vector Regression)

Let us briefly discuss about these algorithms

Linear Regression

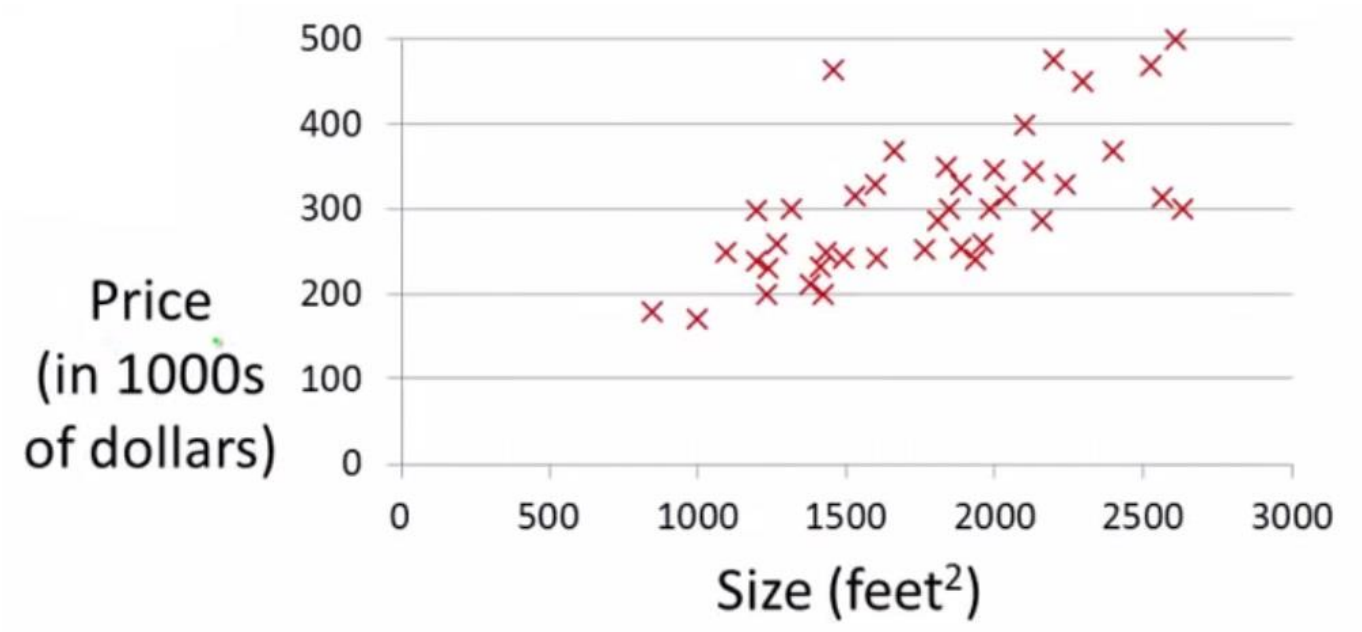
Linear regression is a supervised learning algorithm and tries to model the relationship between a continuous target variable and one or more independent variables by fitting a linear equation to the data. For a linear regression to be a good choice, there needs to be a linear relation between independent variable(s) and target variable. There are many tools to explore the relationship among variables such as scatter plots and correlation matrix.

Let us understand how Linear Regression works by an example of **Housing Price Prediction**

Let us consider that we have **100 houses** in a city of **different prices** and just for keeping things simple we assume that the price of house depends only on **the size**. Now we need to understand the relationship between the size and the price of houses.

We first collect the data of the size and prices of all the houses in the colony. Once we have the data, we need to Train the model.

Training the model : We need to choose the model and put it into training. The following picture depicts how a real world data might look like.



By seeing the above graph, at first you may think that it is impossible to fit a direct mathematical relationship. And, there is no direct model on which we can train this data. We split our data into training, validation and testing. A rule of thumb is to use 80% of the data for training, 10% for validation and 10% for testing. This might vary according to the problem one is working on.

We need to choose some model on which we can train this data. We try to fit a linear model (i.e. A linear linear relationship between X and Y) on it. A line will fit overall trend of the data appropriately. Graphically, in two dimensions, this results in a line of best fit, although if we have more features or parameters we would refer it as a plane of best fit and so on.

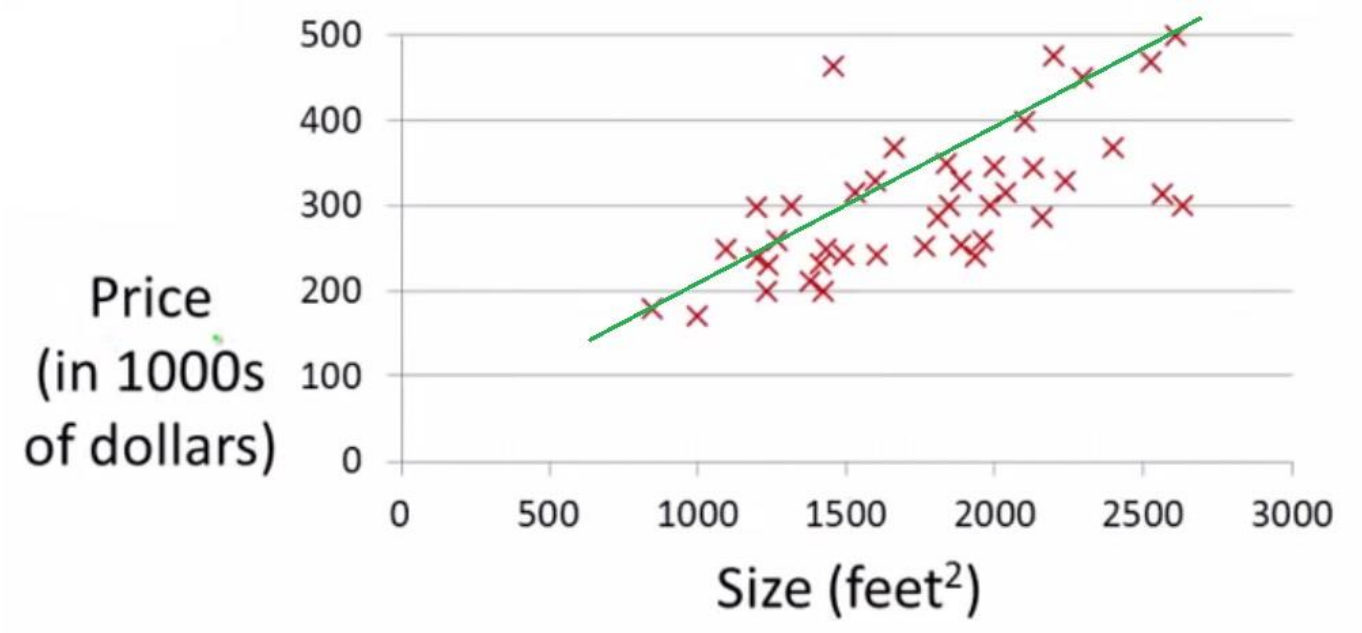
Mathematically, we can model the problem as

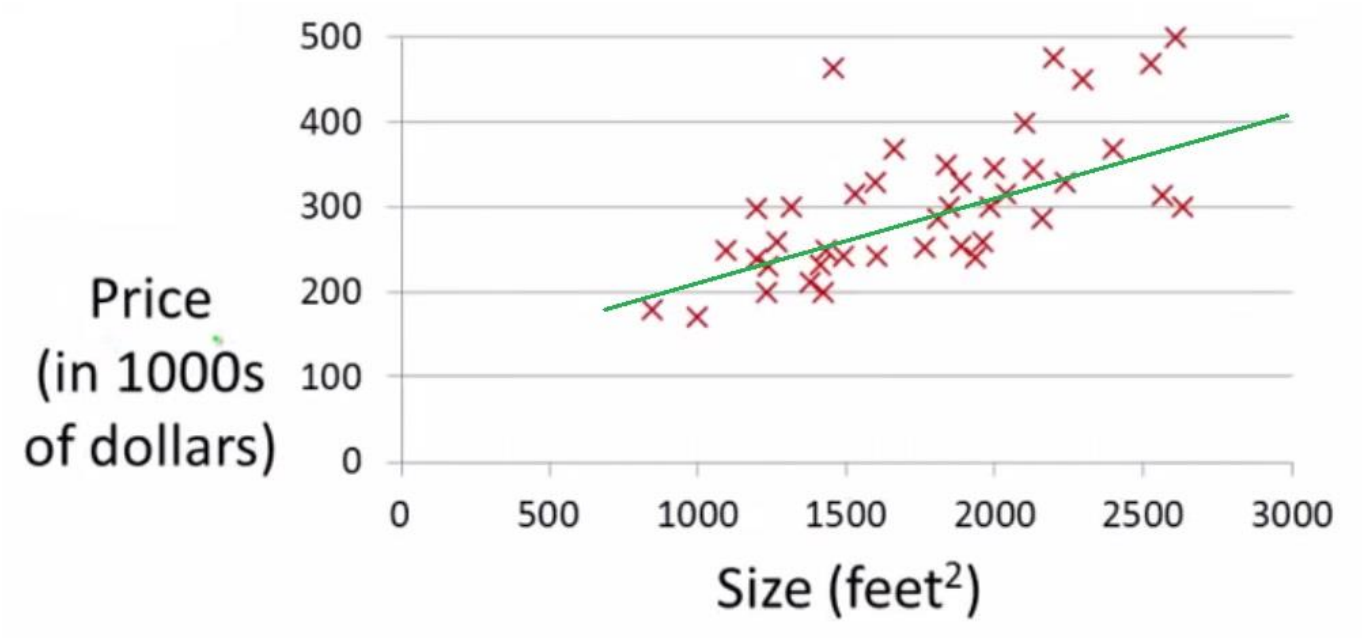
$$\mathbf{y = \theta_0 + \theta_1 x}$$

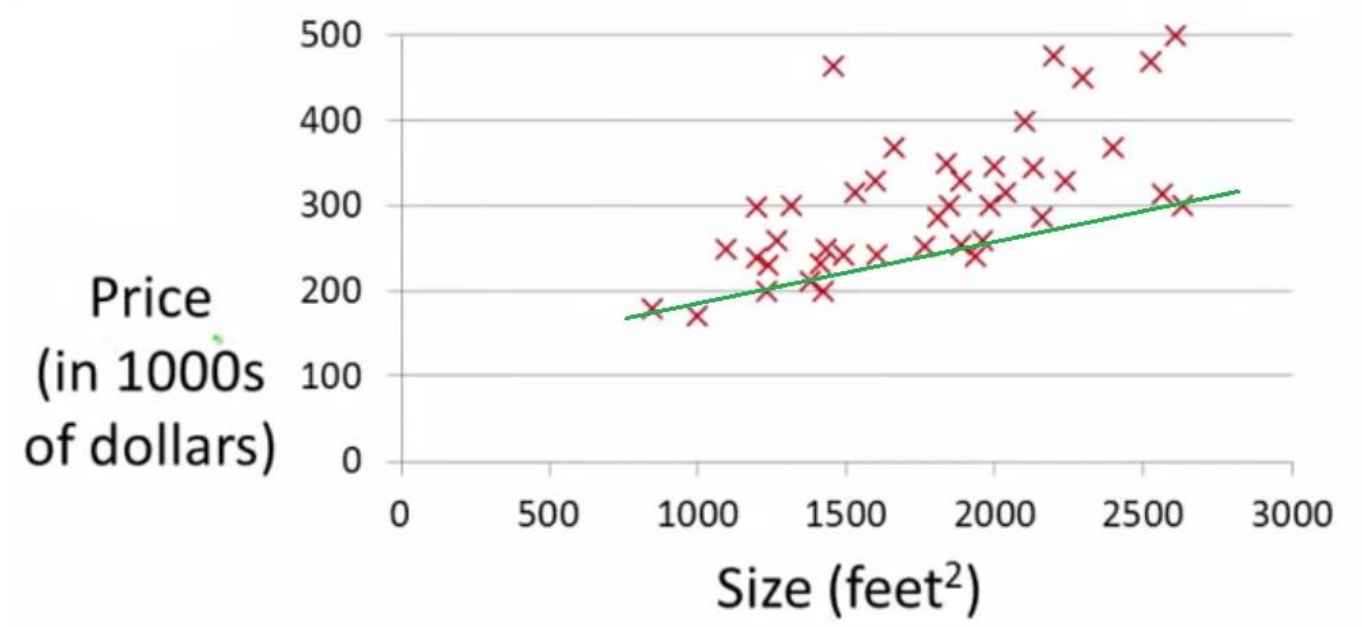
where 'y' is the price (in 1000$) and 'x' is the size (in feet^2) and θ_0 and θ_1 are arbitrary constants and we will find out their values.

Our model will be a function that predicts *y* for a given *x* and our goal is to learn the model parameters (θ_0 & θ_1 in our case) But what are the best model parameters?

Because of the noise in the data, there can be many solutions possible for this problem based on different values of θ_0 & θ_1. Let's look at a few of them.







But how do we decide which of these models should br picked? How to learn which one will be the best model parameters (θ_0 & θ_1)?

The best model parameters are those which gives most accurate results. The line which covers most of the trend in data will be most accurate. Alternatively, the best model parameters are those which gives least error. The line which gives least inaccurate results will be best line. We thus choose the line which minimizes error in model's predictions.

The task of finding the model parameters reduces to finding those parameters that minimize error, i.e. we have to make the model as accurate as possible. But how do we measure error?

To measure error, we define a error function commonly known as **cost function** or **loss function** which measures how inaccurate/accurate the model's predictions are.

Mathematically, we look at the difference between each real output ($\hat{y}$) and our model's prediction (y). We square these differences to avoid negative numbers and then add them up and take the average. This is a measure of how well our data fits the line and is commonly known as the **mean squared error**.

You can refer to the following video to get a better visualisation of linear regression

Linear Regression Analysis

K-nearest neighbors (kNN)

K-nearest neighbors (KNN) algorithm, also is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. KNN is a **lazy learning algorithm** because it does not have a specialized training phase and uses all the data for training while classification.KNN is also a **non-parametric learning algorithm** because it doesn't assume anything about the underlying data.

Step-by-Step working of KNN algorithm

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data, we have to follow the below procedure :-

1) Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is the Euclidean Method.

2) Now, based on the distance value, sort them in ascending order.

3) Next, it will choose the top K rows from the sorted array.

4) Now, it will assign a class to the test point based on most frequent class of these rows.

Let us understand how kNN works by an example of **T-shirt Size Prediction** Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have. Data including height, weight and T-shirt size information is shown below -

Height(in cms)	Weight(in kgs)	T-Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L

Height(in cms)	Weight(in kgs)	T-Shirt Size
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

Calculate Similarity based on distance function

There are many distance functions but Euclidean is the most commonly used measure. It is mainly used when data is continuous.

Euclidean :

$$d(x,y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x,y) = \sum_{i=1}^{m}|x_i - y_i|$$

The idea to use distance measure is to find the distance (similarity) between new sample and training cases and then finds the k-closest customers to new customer in terms of height and weight. New customer named 'Avinash' has height 161cm and weight 61kg.

Euclidean distance between first observation and new observation (Avinash) is as follows - =SQRT((161-158)^2+(61-58)^2)

Similarly, we will calculate distance of all the training cases with new case and calculates the rank in terms of distance. The smallest distance value will be ranked 1 and considered as nearest neighbor.

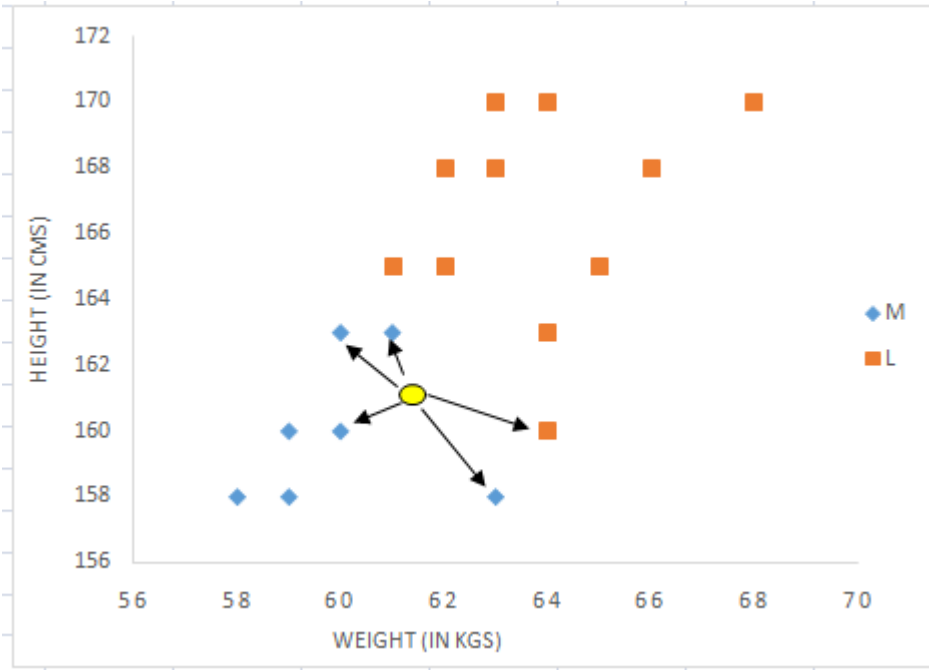
Finding K-Neighbours

Let k be 5. Then the algorithm searches for the 5 customers closest to Avinash, i.e. most similar to Avinash in terms of attributes, and see what categories those 5 customers were in. If 4 of them had ‘Medium T shirt sizes’ and 1 had ‘Large T shirt size’ then your best guess for Avinash is ‘Medium T shirt. See the calculation shown in the snapshot below -

=SQRT((A21-A6)^2+(B21-B6)^2)

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	**M**	2.2	**3**
6	160	60	**M**	1.4	**1**
7	163	60	**M**	2.2	**3**
8	163	61	**M**	2.0	**2**
9	160	64	**L**	3.2	**5**
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	**161**	**61**			

In the graph below, binary dependent variable (T-shirt size) is displayed in blue and orange color. 'Medium T-shirt size' is in blue color and 'Large T-shirt size' in orange color. New customer information is exhibited in yellow circle. Four blue highlighted data points and one orange highlighted data point are close to yellow circle. so the prediction for the new case is blue highlighted data point which is Medium T-shirt size.



You can refer to the following video to get a better visualisation of the kNN Algorithm.

KNN Algorithm

Lasso

lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

Elastic-Net

The Elastic-Net is a regularised regression method that linearly combines both penalties i.e. L1 and L2 of the Lasso and Ridge regression methods. It is useful when there are multiple correlated features. The difference between Lasso and Elastic-Net lies in the fact that Lasso is likely to pick one of these features at random while elastic-net is likely to pick both at once.

```
# imports
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
```

```
# Spot-Check Algorithms
models = []
models.append((' LR ', LinearRegression()))
models.append((' LASSO ', Lasso()))
models.append((' EN ', ElasticNet()))
models.append((' KNN ', KNeighborsRegressor()))
models.append((' CART ', DecisionTreeRegressor()))
models.append((' SVR ', SVR()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds, random_state=seed, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring=scoring)
    # print(cv_results)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

The output of the above code gives us the accuracy estimations for each of our algorithms. We need to compare the models to each other and select the most accurate.

Once we are able to choose which results in the best accuracy, all we have to do is to

- Define the model
- Fit data into our model
- Make predictions

Plot your predictions alongwith the actual data and the two plots will nearly overlap.