Session – 6

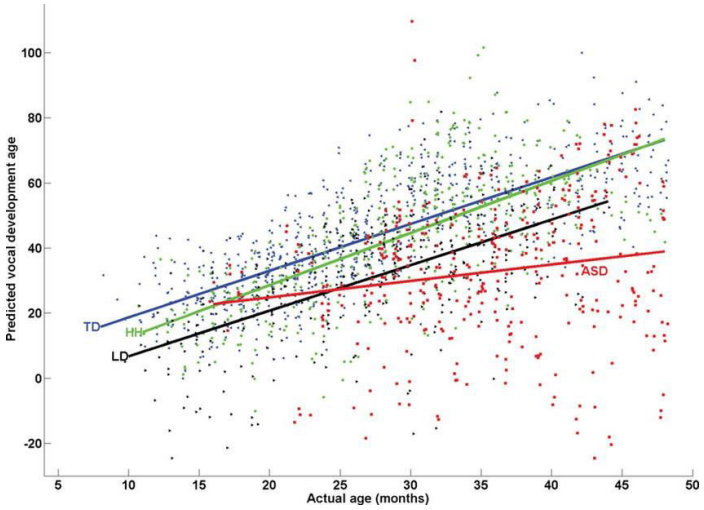
Multiple Linear Regression

Linear Regression, one of the most popular and discussed models, is certainly the gateway to go deeper into Machine Learning (ML). Such a simplistic, straightforward approach to modeling is worth learning as one of your first steps into ML.

A statistical technique that is used to predict the outcome of a variable based on the value of two or more variables

What is Multiple Linear Regression?

Multiple linear regression refers to a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is sometimes known simply as multiple regression, and it is an extension of linear regression. The variable that we want to predict is known as the dependent variable, while the variables we use to predict the value of the dependent variable are known as independent or explanatory variables.



Why Multiple Linear Regression?

“ To predict the outcome from multiple input variables. Duh!”. But, is that it? Well, hold that thought.

Consider this, suppose you have to estimate the price of a certain house you want to buy. You know the floor area, the age of the house, its distance from your workplace, the crime rate of the place, etc.

Now, some of these factors will affect the price of the house positively. For example more the area, the more the price. On the other hand, factors like distance from the workplace, and the crime rate can influence your estimate of the house negatively (unless you are a rich criminal with interest in Machine Learning looking for a hideout, yeah I don't think so).

Disadvantages of Simple Linear Regression → Running separate simple linear regressions will lead to different outcomes when we are interested in just one. Besides that, there may be an input variable that is itself correlated with or dependent on some other predictor. This can cause wrong predictions and unsatisfactory results.

Assumptions of Multiple Linear Regression

Multiple linear regression is based on the following assumptions:

1. A linear relationship between the dependent and independent variables

The first assumption of multiple linear regression is that there is a linear relationship between the dependent variable and each of the independent variables. The best way to check the linear relationships is to create scatterplots and then visually inspect the scatterplots for linearity. If the relationship displayed in the scatterplot is not linear, then the analyst will need to run a non-linear regression or transform the data using statistical software, such as SPSS.

2. The independent variables are not highly correlated with each other

The data should not show multicollinearity, which occurs when the independent variables (explanatory variables) are highly correlated. When independent variables show multicollinearity, there will be problems figuring out the specific variable that contributes to the variance in the dependent variable. The best method to test for the assumption is the Variance Inflation Factor method.

3. The variance of the residuals is constant

Multiple linear regression assumes that the amount of error in the residuals is similar at each point of the linear model. This scenario is known as homoscedasticity. When analyzing the data, the analyst should plot the standardized residuals against the predicted values to determine if the points are distributed fairly across all the values of independent variables. To test the assumption, the data can be plotted on a scatterplot or by using statistical software to produce a scatterplot that includes the entire model.

4. Independence of observation

The model assumes that the observations should be independent of one another. Simply put, the model assumes that the values of residuals are independent. To test for this assumption, we use the Durbin Watson statistic.

The test will show values from 0 to 4, where a value of 0 to 2 shows positive autocorrelation, and values from 2 to 4 show negative autocorrelation. The mid-point, i.e., a value of 2, shows that there is no autocorrelation.

5. Multivariate normality

Multivariate normality occurs when residuals are normally distributed. To test this assumption, look at how the values of residuals are distributed. It can also be tested using two main methods, i.e., a histogram with a superimposed normal curve or the Normal Probability Plot method.

Multiple Linear Regression Formula

$$yi = \beta0 + \beta1xi1 + \beta2xi2 + ... + \beta pxip + \epsilon$$

Where:

- **yi** is the dependent or predicted variable
- **β0** is the y-intercept, i.e., the value of y when both xi and x2 are 0.

- **β1** and **β2** are the regression coefficients representing the change in y relative to a one-unit change in **xi1** and **xi2**, respectively.
- **βp** is the slope coefficient for each independent variable
- **ϵ** is the model's random error (residual) term.

Understanding Multiple Linear Regression

Simple linear regression enables statisticians to predict the value of one variable using the available information about another variable. Linear regression attempts to establish the relationship between the two variables along a straight line.

Multiple regression is a type of regression where the dependent variable shows a **linear** relationship with two or more independent variables. It can also be **non-linear**, where the dependent and independent variables do not follow a straight line.

Both linear and non-linear regression track a particular response using two or more variables graphically. However, non-linear regression is usually difficult to execute since it is created from assumptions derived from trial and error.

Identification and collection of regression dataset

Within multiple types of regression models, it is important to choose the best suited technique based on type of independent and dependent variables, dimensionality in the data and other essential characteristics of the data. Below are the key factors that you should practice to select the right regression model:

1. Data exploration is an inevitable part of building predictive model. It should be you first step before selecting the right model like identify the relationship and impact of variables
2. To compare the goodness of fit for different models, we can analyse different metrics like statistical significance of parameters, R-square, Adjusted r-square, AIC, BIC and error term. Another one is the Mallow's Cp criterion. This essentially checks for possible bias in your model, by comparing the model with all possible submodels (or a careful selection of them).

3. Cross-validation is the best way to evaluate models used for prediction. Here you divide your data set into two group (train and validate). A simple mean squared difference between the observed and predicted values give you a measure for the prediction accuracy.
4. If your data set has multiple confounding variables, you should not choose automatic model selection method because you do not want to put these in a model at the same time.
5. It'll also depend on your objective. It can occur that a less powerful model is easy to implement as compared to a highly statistically significant model.
6. Regression regularization methods(Lasso, Ridge and ElasticNet) works well in case of high dimensionality and multicollinearity among the variables in the data set.

Perform data exploration, preprocessing and splitting on datasets like

1. Boston housing price from scikit learn datasets

```
# Load libraries
import numpy as np
import pylab as pl
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor

################################
### ADD EXTRA LIBRARIES HERE ###
################################
from sklearn.metrics import mean_squared_error,median_absolute_error,r2_score,mean_absolute_error
from sklearn import grid_search
from sklearn.cross_validation import train_test_split

def load_data():
    """Load the Boston dataset."""

    boston = datasets.load_boston()
    return boston
```

```
def explore_city_data(city_data):
    """Calculate the Boston housing statistics."""

    # Get the labels and features from the housing data
    housing_prices = city_data.target
    housing_features = city_data.data

    ###################################
    ### Step 1. YOUR CODE GOES HERE ###
    ###################################

    # Please calculate the following values using the Numpy library
    # Size of data (number of houses)?
    # Number of features?
    # Minimum price?
    # Maximum price?
    # Calculate mean price?
    # Calculate median price?
    # Calculate standard deviation?
    number_of_houses = housing_features.shape[0]
    number_of_features = housing_features.shape[1]
    max_price = np.max(housing_prices)
    min_price = np.min(housing_prices)
    mean_price = np.mean(housing_prices)
    median_price = np.median(housing_prices)
    standard_deviation = np.std(housing_prices)

    print "number of houses:",number_of_houses
    print "number of features:",number_of_features
    print "max price of house:",max_price
    print "min price of house:",min_price
    print "mean price of house:",mean_price
    print "median price of house:",median_price
    print "standard deviation for prices of house:",standard_deviation
```

```
def split_data(city_data):
    """Randomly shuffle the sample set. Divide it into 70 percent training and 30 percent testing data."""

    # Get the features and labels from the Boston housing data
    X, y = city_data.data, city_data.target

    ###################################
    ### Step 3. YOUR CODE GOES HERE ###
    ###################################
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.30, train_size=0.70, random_state=42)
    return X_train, y_train, X_test, y_test
```

2. Cricket match result - past data

Importing the dataset

```
import pandas as pd
dataset = pd.read_csv('/content/ipl.csv')
X = dataset.iloc[:,[7,8,9,12,13]].values #Input features
y = dataset.iloc[:, 14].values #Label
```

I have used 'ipl.csv' datafile here for predicting scores in ODI cricket. One can use 't20.csv' or 'odi.c respectively.

```
X
```

```
array([[1.00e+00, 0.00e+00, 1.00e-01, 0.00e+00, 0.00e+00],
       [1.00e+00, 0.00e+00, 2.00e-01, 0.00e+00, 0.00e+00],
       [2.00e+00, 0.00e+00, 2.00e-01, 0.00e+00, 0.00e+00],
       ...,
       [1.28e+02, 7.00e+00, 1.94e+01, 4.70e+01, 1.20e+01],
       [1.29e+02, 7.00e+00, 1.95e+01, 4.70e+01, 1.30e+01],
       [1.29e+02, 8.00e+00, 1.96e+01, 4.70e+01, 1.30e+01]])
```

```
y
```

```
array([222, 222, 222, ..., 129, 129, 129])
```

Splitting data into training and testing set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

We will train our model on 75 percent of the dataset and test the model on remaining dataset.

Feature Scaling the data

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Feature scaling is a very important part of machine learning. You can read more about it here

3. Performance of a cricket player - past data

```
import numpy as np
import pandas as pd
import re
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/cricinfo-statsguru-data/Men ODI Player Innings Stats - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men T20I Team Match Results - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men ODI Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men ODI Team Batting Stats.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Team Batting Stats.csv
/kaggle/input/cricinfo-statsguru-data/Women ODI Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Team Match Results - 19th Century.csv
/kaggle/input/cricinfo-statsguru-data/IPL Player Stats - 2016 till 2019.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Team Match Results - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Player Innings Stats - 19th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Player Innings Stats - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Women Test Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Men ODI Team Match Results - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Women ODI Player Innings Stats - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men T20I Team Bowling Stats.csv
/kaggle/input/cricinfo-statsguru-data/Men ODI Team Bowling Stats.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Team Match Results - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men T20I Team Batting Stats.csv
/kaggle/input/cricinfo-statsguru-data/Men T20I Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Women T20I Player Innings Stats - 21st Century.csv
/kaggle/input/cricinfo-statsguru-data/Women Test Player Innings Stats - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men ODI Team Match Results - 20th Century.csv
/kaggle/input/cricinfo-statsguru-data/Men Test Team Bowling Stats.csv
/kaggle/input/project/personal_male.csv
```

```
dataset=pd.read_csv('/kaggle/input/cricinfo-statsguru-data/Men ODI Player Innings Stats - 21st Century.csv')
```

```
dataset.columns
```

```
Index(['Innings Player', 'Innings Runs Scored', 'Innings Runs Scored Num',
       'Innings Minutes Batted', 'Innings Batted Flag', 'Innings Not Out Flag',
       'Innings Balls Faced', 'Innings Boundary Fours',
       'Innings Boundary Sixes', 'Innings Batting Strike Rate',
       'Innings Number', 'Opposition', 'Ground', 'Innings Date', 'Country',
       '50's', '100's', 'Innings Runs Scored Buckets', 'Innings Overs Bowled',
       'Innings Bowled Flag', 'Innings Maidens Bowled',
       'Innings Runs Conceded', 'Innings Wickets Taken', '4 Wickets',
```

```
batsman=dataset[dataset['Innings Overs Bowled'].isnull()]
```

```
#removing unnecessary columns
drop=['Innings Overs Bowled',
       'Innings Bowled Flag', 'Innings Maidens Bowled',
       'Innings Runs Conceded', 'Innings Wickets Taken', '4 Wickets',
       '5 Wickets', '10 Wickets', 'Innings Wickets Taken Buckets',
       'Innings Economy Rate','Innings Runs Scored Num', 'Innings Minutes Batted', 'Innings Batted Flag'
     ,'Innings Not Out Flag']
batsman=batsman.drop(drop, axis=1)
```

```
batsman['Innings_Runs_Score']=0
batsman=batsman[(batsman['Innings Runs Scored']!='DNB') & (batsman['Innings Runs Scored']!='TDNB')]
```

```
#writing regular expressions to extract runs scored
runs = r'([0-9]*)'
index_2=batsman.columns.get_loc('Innings Runs Scored')
index_runs=batsman.columns.get_loc('Innings_Runs_Score')
for row in range(0,len(batsman)):
    run=re.search(runs,batsman.iat[row,index_2]).group()
    if run!='':
        batsman.iat[row,index_runs]=int(run)
```

```
#import standard data sci libs
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
/Users/aditya/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the fu
nctions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Reading in the .csv datasets:

```
df_2019 = pd.read_csv('https://raw.githubusercontent.com/adityarc19/IPL-analysis/main/data/2019.csv')
df_2019
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	DA Warner	Sunrisers Hyderabad	410	1.414	0.224	$1,953,130	$2,742,662
1	2	AD Russell	Kolkata Knight Riders	307	1.060	0.242	$1,328,130	$2,200,658
2	3	MS Dhoni	Chennai Super Kings	305	1.053	0.176	$2,343,750	$2,189,940
3	4	Imran Tahir	Chennai Super Kings	295	1.018	0.200	$156,250	$2,136,352
4	5	KL Rahul	Kings XI Punjab	288	0.994	0.217	$1,718,750	$2,099,606
...	...	...	...	...	...	...	...	...
156	157	DS Kulkarni	Rajasthan Royals	-155	-0.534	0.044	$117,190	$-239,892
157	158	B Kumar	Sunrisers Hyderabad	-158	-0.544	0.095	$1,328,130	$-255,203
158	159	Kuldeep Yadav	Kolkata Knight Riders	-159	-0.549	0.023	$906,250	$-262,858
159	160	K Gowtham	Rajasthan Royals	-201	-0.695	0.016	$968,750	$-486,397
160	161	M Prasidh Krishna	Kolkata Knight Riders	-276	-0.952	0.040	$15,620	$-879,886

161 rows × 8 columns

```
df_2020 = pd.read_csv('https://raw.githubusercontent.com/adityarc19/IPL-analysis/main/data/2020.csv')
df_2020
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	JJ Bumrah	Mumbai Indians	379	1.281	0.238	$1,093,750	$2,448,655
1	2	KL Rahul	Kings XI Punjab	330	1.113	0.194	$1,718,750	$2,204,319
2	3	K Rabada	Delhi Capitals	320	1.082	0.232	$656,250	$2,159,233
3	4	Ishan Kishan	Mumbai Indians	261	0.881	0.157	$968,750	$1,866,902
4	5	S Dhawan	Delhi Capitals	249	0.842	0.168	$812,500	$1,810,181
...	...	...	...	...	...	...	...	...
145	146	AR Patel	Delhi Capitals	-180	-0.607	0.081	$714,300	$-297,218
146	147	NA Saini	Royal Challengers Bangalore	-191	-0.646	0.061	$468,750	$-353,939
147	148	S Gopal	Rajasthan Royals	-215	-0.726	0.067	$31,250	$-470,290
148	149	KH Pandya	Mumbai Indians	-251	-0.848	0.066	$1,375,000	$-647,724
149	150	SP Narine	Kolkata Knight Riders	-292	-0.988	0.051	$1,953,130	$-851,338

150 rows × 8 columns

```
df_2018 = pd.read_csv('/Users/aditya/Downloads/archive/2018.csv')
df_2018
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	KS Williamson	Sunrisers Hyderabad	351	1.221	0.214	$468,750	$2,397,239
1	2	KL Rahul	Kings XI Punjab	328	1.141	0.193	$1,718,750	$2,280,549
2	3	AT Rayudu	Chennai Super Kings	327	1.137	0.198	$343,750	$2,274,714
3	4	RR Pant	Delhi Daredevils	322	1.122	0.201	$2,343,750	$2,252,835
4	5	MS Dhoni	Chennai Super Kings	290	1.011	0.167	$2,343,750	$2,090,927
...	...	...	...	...	...	...	...	...
144	145	Shivam Mavi	Kolkata Knight Riders	-190	-0.662	0.038	$468,750	$-349,363
145	146	AR Patel	Kings XI Punjab	-191	-0.665	0.039	$1,953,130	$-353,739
146	147	JD Unadkat	Rajasthan Royals	-204	-0.712	0.079	$1,796,880	$-422,295
147	148	BA Stokes	Rajasthan Royals	-207	-0.720	0.078	$1,953,130	$-433,964

```
df_2017 = pd.read_csv('/Users/aditya/Downloads/archive/2017.csv')
df_2017
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	DA Warner	Sunrisers Hyderabad	351	1.274	0.211	$916,000	$1,783,802
1	2	JD Unadkat	Rising Pune Supergiants	316	1.146	0.207	$45,000	$1,648,517
2	3	B Kumar	Sunrisers Hyderabad	304	1.104	0.228	$708,000	$1,604,126
3	4	HM Amla	Kings XI Punjab	236	0.854	0.151	NaN	$1,339,898
4	5	AJ Tye	Gujarat Lions	205	0.742	0.132	$75,000	$1,221,524
...	...	...	...	...	...	...	...	...
155	156	MM Sharma	Kings XI Punjab	-171	-0.619	0.093	$975,000	$-216,936
156	157	S Aravind	Royal Challengers Bangalore	-183	-0.665	0.049	NaN	$-265,554
157	158	SR Watson	Royal Challengers Bangalore	-196	-0.712	0.050	$1,425,000	$-315,229
158	159	I Sharma	Sunrisers Hyderabad	-217	-0.786	0.013	$570,000	$-393,441
159	160	SP Narine	Kolkata Knight Riders	-223	-0.807	0.121	$700,000	$-415,636

160 rows × 8 columns

```
df_2016 = pd.read_csv('/Users/aditya/Downloads/archive/2016.csv')
df_2016
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	V Kohli	Royal Challengers Bangalore	615	2.041	0.365	$2,500,000	$2,594,886
1	2	DA Warner	Sunrisers Hyderabad	414	1.375	0.237	$916,000	$1,890,050
2	3	AB de Villiers	Royal Challengers Bangalore	307	1.019	0.189	$1,583,333	$1,513,292
3	4	YS Chahal	Royal Challengers Bangalore	251	0.833	0.155	$16,000	$1,316,446
4	5	G Gambhir	Kolkata Knight Riders	241	0.799	0.147	$1,666,667	$1,280,463
...	...	...	...	...	...	...	...	...
153	154	VR Aaron	Royal Challengers Bangalore	-151	-0.502	0.021	$333,000	$-96,399
154	155	GJ Maxwell	Kings XI Punjab	-159	-0.528	0.033	$1,000,000	$-123,916
155	156	P Kumar	Gujarat Lions	-214	-0.712	0.076	$525,000	$-318,645
156	157	HH Pandya	Mumbai Indians	-218	-0.722	0.028	$16,667	$-329,228

Combining both years' datasets :

```
df = pd.concat([df_2008, df_2009, df_2010, df_2011, df_2012, df_2013, df_2014, df_2015, df_2016, df_2017,
                df_2018, df_2019, df_2020], axis=0)
df
```

	Rank	Player	Team	RAA	Wins	EFscore	Salary	Value
0	1	SE Marsh	Kings XI Punjab	377	1.355	0.231	NaN	$1,270,355
1	2	SR Watson	Rajasthan Royals	347	1.250	0.297	$125,000	$1,209,660
2	3	Sohail Tanvir	Rajasthan Royals	294	1.059	0.194	NaN	$1,099,252
3	4	G Gambhir	Delhi Daredevils	216	0.776	0.160	$725,000	$935,664
4	5	GC Smith	Rajasthan Royals	202	0.725	0.139	$250,000	$906,183
...	...	...	...	...	...	...	...	...
145	146	AR Patel	Delhi Capitals	-180	-0.607	0.081	$714,300	$-297,218
146	147	NA Saini	Royal Challengers Bangalore	-191	-0.646	0.061	$468,750	$-353,939
147	148	S Gopal	Rajasthan Royals	-215	-0.726	0.067	$31,250	$-470,290
148	149	KH Pandya	Mumbai Indians	-251	-0.848	0.066	$1,375,000	$-647,724
149	150	SP Narine	Kolkata Knight Riders	-292	-0.988	0.051	$1,953,130	$-851,338

2164 rows × 8 columns

```
df.drop(['Rank'],axis=1,inplace=True)
```

```
df.to_csv('df',index=False)
```

```
df=pd.read_csv('df')
df
```

	Player	Team	RAA	Wins	EFscore	Salary	Value
0	SE Marsh	Kings XI Punjab	377	1.355	0.231	NaN	$1,270,355
1	SR Watson	Rajasthan Royals	347	1.250	0.297	$125,000	$1,209,660
2	Sohail Tanvir	Rajasthan Royals	294	1.059	0.194	NaN	$1,099,252
3	G Gambhir	Delhi Daredevils	216	0.776	0.160	$725,000	$935,664
4	GC Smith	Rajasthan Royals	202	0.725	0.139	$250,000	$906,183
...	...	...	...	...	...	...	...
2159	AR Patel	Delhi Capitals	-180	-0.607	0.081	$714,300	$-297,218
2160	NA Saini	Royal Challengers Bangalore	-191	-0.646	0.061	$468,750	$-353,939
2161	S Gopal	Rajasthan Royals	-215	-0.726	0.067	$31,250	$-470,290
2162	KH Pandya	Mumbai Indians	-251	-0.848	0.066	$1,375,000	$-647,724
2163	SP Narine	Kolkata Knight Riders	-292	-0.988	0.051	$1,953,130	$-851,338

2164 rows × 7 columns

```
#Removing the dollar sign and commas from salary and value columns
df['Salary'] = df['Salary'].str.replace(',', '').str.replace('$', '').astype(float)
df['Value'] = df['Value'].str.replace(',', '').str.replace('$', '').astype(float)
df
```

	Player	Team	RAA	Wins	EFscore	Salary	Value
0	SE Marsh	Kings XI Punjab	377	1.355	0.231	NaN	1270355.0
1	SR Watson	Rajasthan Royals	347	1.250	0.297	125000.0	1209660.0
2	Sohail Tanvir	Rajasthan Royals	294	1.059	0.194	NaN	1099252.0
3	G Gambhir	Delhi Daredevils	216	0.776	0.160	725000.0	935664.0
4	GC Smith	Rajasthan Royals	202	0.725	0.139	250000.0	906183.0
...	...	...	...	...	...	...	...
2159	AR Patel	Delhi Capitals	-180	-0.607	0.081	714300.0	-297218.0
2160	NA Saini	Royal Challengers Bangalore	-191	-0.646	0.061	468750.0	-353939.0
2161	S Gopal	Rajasthan Royals	-215	-0.726	0.067	31250.0	-470290.0

4. Crop yield - past data

```
import numpy as np
import pandas as pd
```

```
df_yield = pd.read_csv('yield.csv')
df_yield.shape
```

```
(56717, 12)
```

```
df_yield.head()
```

	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year Code	Year	Unit	Value
0	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1961	1961	hg/ha	14000
1	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1962	1962	hg/ha	14000
2	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1963	1963	hg/ha	14260
3	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1964	1964	hg/ha	14257
4	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1965	1965	hg/ha	14400

```
df_yield.tail()
```

	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year Code	Year	Unit	Value
56712	QC	Crops	181	Zimbabwe	5419	Yield	15	Wheat	2012	2012	hg/ha	24420
56713	QC	Crops	181	Zimbabwe	5419	Yield	15	Wheat	2013	2013	hg/ha	22888
56714	QC	Crops	181	Zimbabwe	5419	Yield	15	Wheat	2014	2014	hg/ha	21357
56715	QC	Crops	181	Zimbabwe	5419	Yield	15	Wheat	2015	2015	hg/ha	19826
56716	QC	Crops	181	Zimbabwe	5419	Yield	15	Wheat	2016	2016	hg/ha	18294

```
# rename columns.
df_yield = df_yield.rename(index=str, columns={"Value": "hg/ha_yield"})
df_yield.head()
```

	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year Code	Year	Unit	hg/ha_yield
0	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1961	1961	hg/ha	14000
1	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1962	1962	hg/ha	14000
2	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1963	1963	hg/ha	14260
3	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1964	1964	hg/ha	14257
4	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1965	1965	hg/ha	14400

```
# drop unwanted columns.
df_yield = df_yield.drop(['Year Code','Element Code','Element','Year Code','Area Code','Domain Code','Domain','Unit','Item Code'], axis=1)
df_yield.head()
```

	Area	Item	Year	hg/ha_yield
0	Afghanistan	Maize	1961	14000
1	Afghanistan	Maize	1962	14000
2	Afghanistan	Maize	1963	14260
3	Afghanistan	Maize	1964	14257
4	Afghanistan	Maize	1965	14400

```
df_yield.describe()
```

	Year	hg/ha_yield
count	56717.000000	56717.000000
mean	1989.669570	62094.660084
std	16.133198	67835.932856
min	1961.000000	0.000000
25%	1976.000000	15680.000000
50%	1991.000000	36744.000000
75%	2004.000000	86213.000000
max	2016.000000	1000000.000000

Climate Data : Rainfall

The climatic factors include rainfall and temperature. They are abiotic components, including pesticides and soil, of the environmental factors that influence plant grow and development.

Rainfall has a dramatic effect on agriculture. For this project rain fall per year information was gathered from World Data Bank.

```
df_rain = pd.read_csv('rainfall.csv')
df_rain.head()
```

	Area	Year	average_rain_fall_mm_per_year
0	Afghanistan	1985	327
1	Afghanistan	1986	327
2	Afghanistan	1987	327
3	Afghanistan	1989	327
4	Afghanistan	1990	327

```
df_rain = df_rain.rename(index=str, columns={" Area": 'Area'})
```

Making sure that names of columns are unified across all dataframes is important for merging after cleaning afterwards.

```
# check data types
df_rain.info()
```

```
Index: 6727 entries, 0 to 6726
Data columns (total 3 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   Area                          6727 non-null   object
 1   Year                          6727 non-null   int64 
 2   average_rain_fall_mm_per_year  5953 non-null   object
dtypes: int64(1), object(2)
memory usage: 210.2+ KB
```

Next, droping any empty rows from dataset and merge yield dataframe with rain dataframe by year and area columns

Part Two: Data Exploration

yield_df is the final obtained dataframe;

36…
```
yield_df.groupby('Item').count()
```

36…

Item	Area	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
Cassava	2045	2045	2045	2045	2045	2045
Maize	4121	4121	4121	4121	4121	4121
Plantains and others	556	556	556	556	556	556
Potatoes	4276	4276	4276	4276	4276	4276
Rice, paddy	3388	3388	3388	3388	3388	3388
Sorghum	3039	3039	3039	3039	3039	3039
Soybeans	3223	3223	3223	3223	3223	3223
Sweet potatoes	2890	2890	2890	2890	2890	2890
Wheat	3857	3857	3857	3857	3857	3857
Yams	847	847	847	847	847	847

37…
```
yield_df.describe()
```

37…

	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
count	28242.000000	28242.000000	28242.000000	28242.000000	28242.000000
mean	2001.544296	77053.332094	1149.055908	37076.909344	20.542627
std	7.051905	84956.612897	709.812134	59958.784665	6.312051
min	1990.000000	50.000000	51.000000	0.040000	1.300000
25%	1995.000000	19919.250000	593.000000	1702.000000	16.702500
50%	2001.000000	38295.000000	1083.000000	17529.440000	21.510000
75%	2008.000000	104676.750000	1668.000000	48687.880000	26.000000
max	2013.000000	501412.000000	3240.000000	367778.000000	30.650000

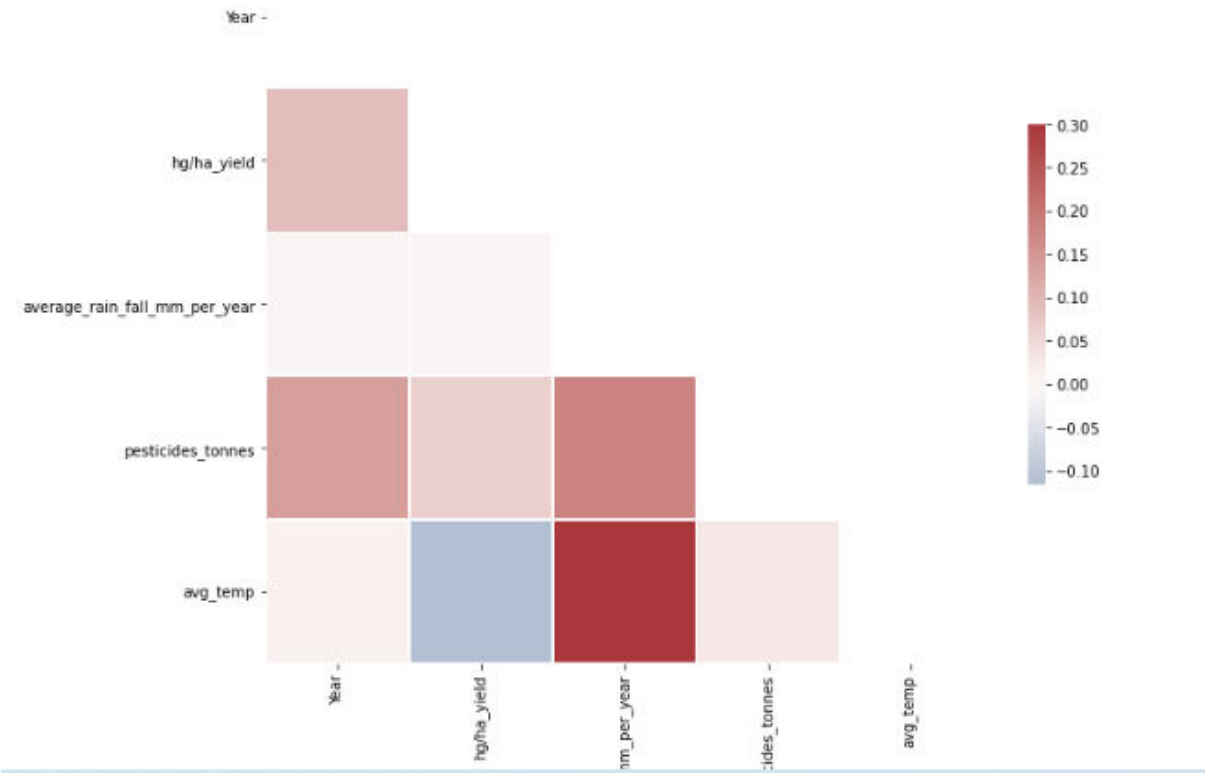
```
correlation_data=yield_df.select_dtypes(include=[np.number]).corr()

mask = np.zeros_like(correlation_data, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.palette="vlag"

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_data, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5});
```



Part Three: Data Preprocessing

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the d collected in raw format which is not feasible for the analysis.

```
yield_df.head()
```

	Area	Item	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
0	Albania	Maize	1990	36613	1485.0	121.0	16.37
1	Albania	Potatoes	1990	66667	1485.0	121.0	16.37
2	Albania	Rice, paddy	1990	23333	1485.0	121.0	16.37
3	Albania	Sorghum	1990	12500	1485.0	121.0	16.37
4	Albania	Soybeans	1990	7000	1485.0	121.0	16.37

```
features = features.drop(['Year'], axis=1)
```

```
features.info()
```

```
Int64Index: 28242 entries, 0 to 28241
Columns: 114 entries, average_rain_fall_mm_per_year to Item_Yams
dtypes: float32(1), float64(2), uint8(111)
memory usage: 3.7 MB
```

```
features.head()
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
features=scaler.fit_transform(features)
```

After dropping year column in addition to scaling all values in features, the resulting array will look something like this :

```
features
```

```
from sklearn.model_selection import train_test_split
train_data, test_data, train_labels, test_labels = train_test_split(features, label, test_size=0.3, random_state=42)
```

```
#write final df to csv file
yield_df.to_csv('yield_df.csv')
```

```
from sklearn.model_selection import train_test_split
train_data, test_data, train_labels, test_labels = train_test_split(features, label, test_size=0.3, random_state=42)
```