FISH + GDP

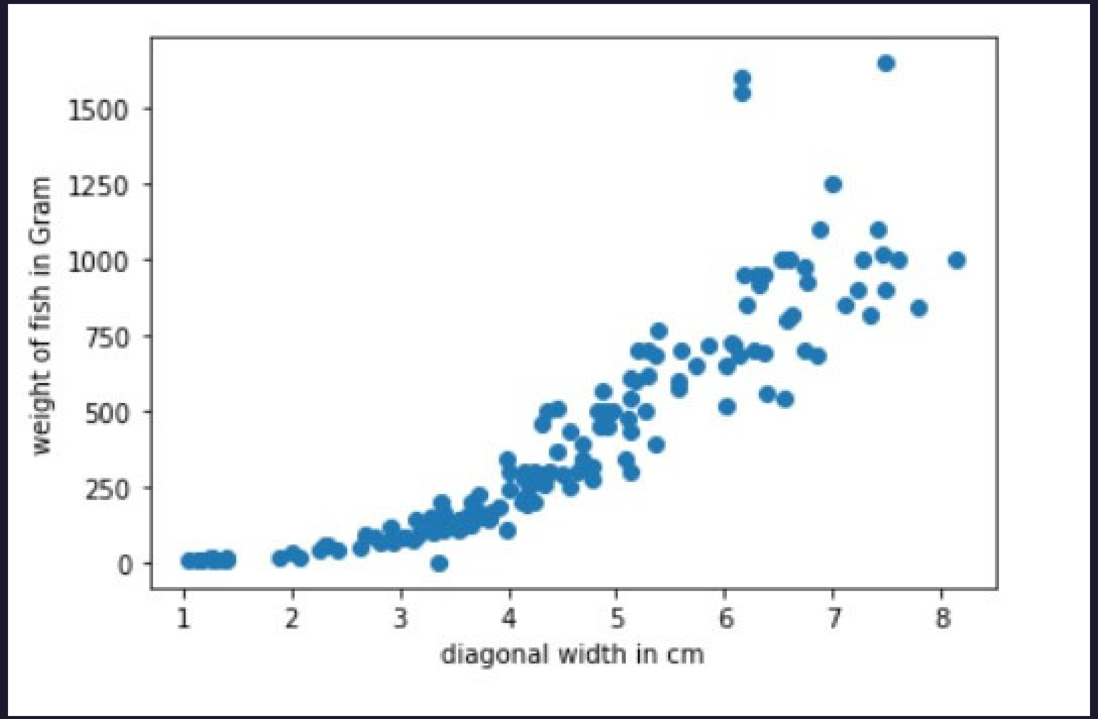
POLYNOMIAL REGRESSION

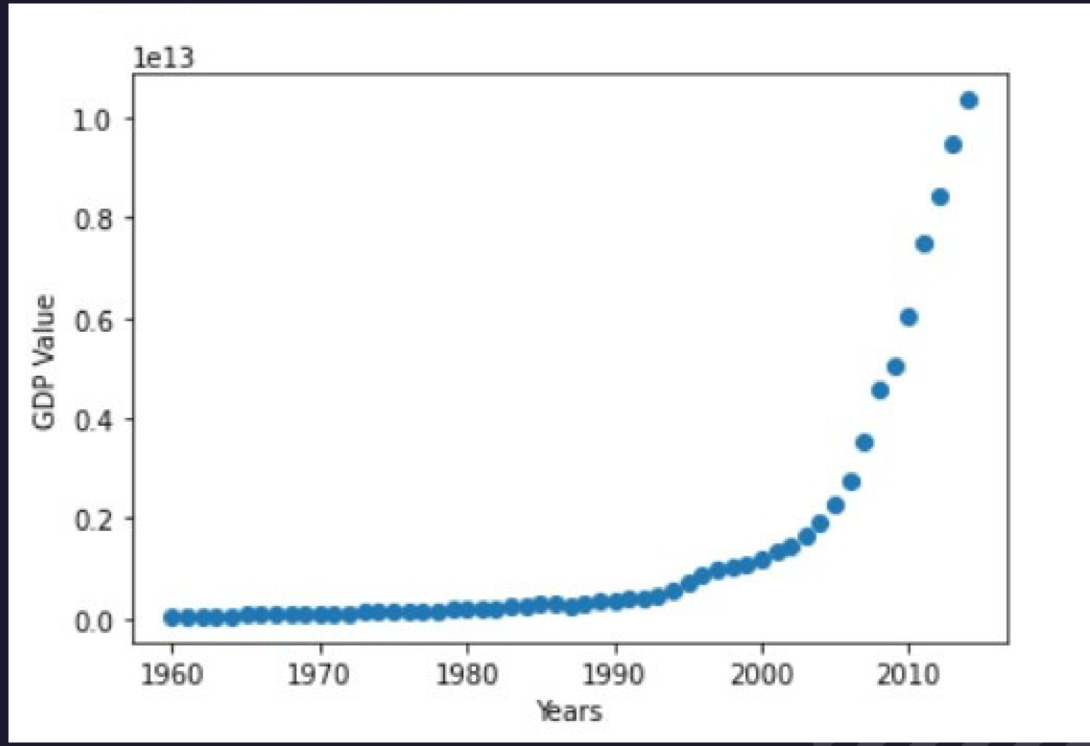
GROUP - 12

DEFINITION

Nonlinear regression is a form of regression analysis in which data is fit to a model and then expressed as a mathematical function. Simple linear regression relates two variables (X and Y) with a straight line ($y = mx + b$), while nonlinear regression relates the two variables in a nonlinear (curved) relationship.[1]

EXAMPLES





`numpy` : NumPy stands for numeric Python, a python package for the computation and processing of the multi-dimensional and single-dimensional array elements.

`pandas` : Pandas provide high-performance data manipulation in Python.

`matplotlib` : Matplotlib is a library used for data visualization. It is mainly used for basic plotting. Visualization using Matplotlib generally consists of bars, pies, lines, scatter plots, and so on.

`seaborn` : Seaborn is a library used for making statistical graphics of the dataset. It provides a variety of visualization patterns. It uses fewer syntax and has easily interesting default themes. It is used to summarize data in visualizations and show the data's distribution.

DATASET

Fish

GDP

Fish Dataset

```
In [4]: # Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

```
In [5]: Fish = pd.read_csv(r"C:\Users\ajeet\OneDrive\Desktop\Fish.csv")
```

IMPORTING LIBRARIES

```
In [6]: Fish.head()
```

Out[6]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

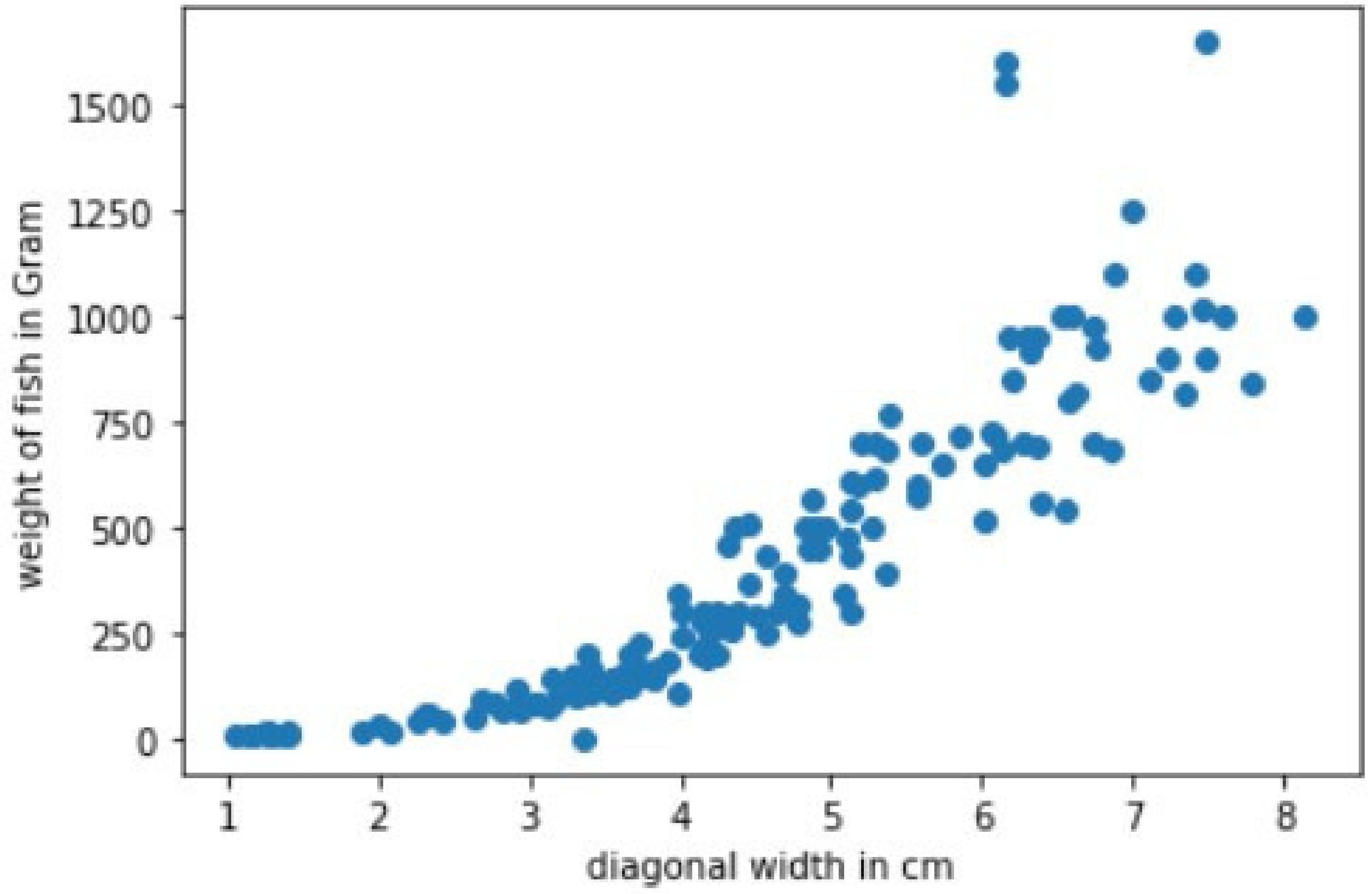
DISPLAYING TOP 5 ENTRIES

In [49]:

```
y = Fish.Weight.values.reshape(-1,1)
x = Fish.Width.values.reshape(-1,1)

plt.scatter(x,y)
plt.ylabel("weight of fish in Gram")
plt.xlabel("diagonal width in cm")
```

Out[49]:

```
Text(0.5, 0, 'diagonal width in cm')
```

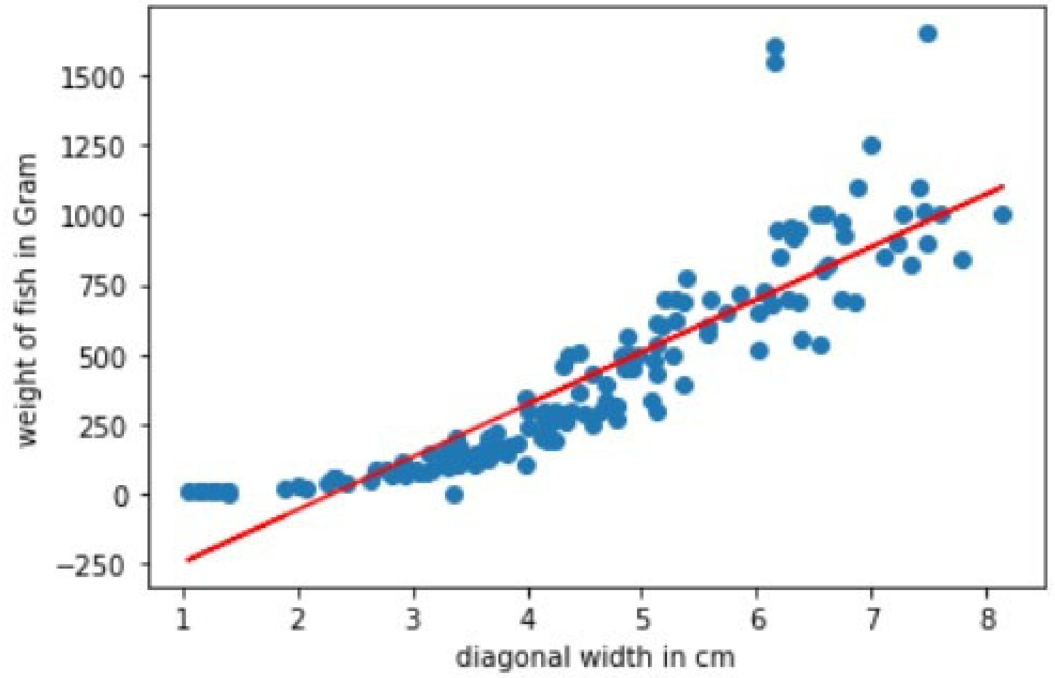


SCATTER PLOT

WIDTH V/S WEIGHT

SCATTER PLOT WITH LINEAR LINE

In [11]:
```
plt.scatter(x,y)
plt.ylabel("weight of fish in Gram")
plt.xlabel("diagonal width in cm")

plt.plot(x,y_predicted, color="red", label="linear")
plt.show()
print("Predict weight of fish in 800 Gram: ", lr.predict([[800]]))
```




```
Predict weight of fish in 800 Gram:  [[150165.58496564]]
```

MSE AND R SQUARE VALUE

In [12]:
```
# model evaluation
mse = mean_squared_error(y, y_predicted)

rmse = np.sqrt(mean_squared_error(y, y_predicted))
r2 = r2_score(y, y_predicted)

# printing values

print('MSE of Linear model', mse)

print('R2 score of Linear model: ', r2)
```

```
MSE of Linear model 27264.800366377916
R2 score of Linear model:  0.7858939611400793
```

In [55]:
```
poly_features = PolynomialFeatures(degree = 11, include_bias = False)
x_poly = poly_features.fit_transform(x)
x[3]
```

Out[55]:
```
array([4.4555])
```

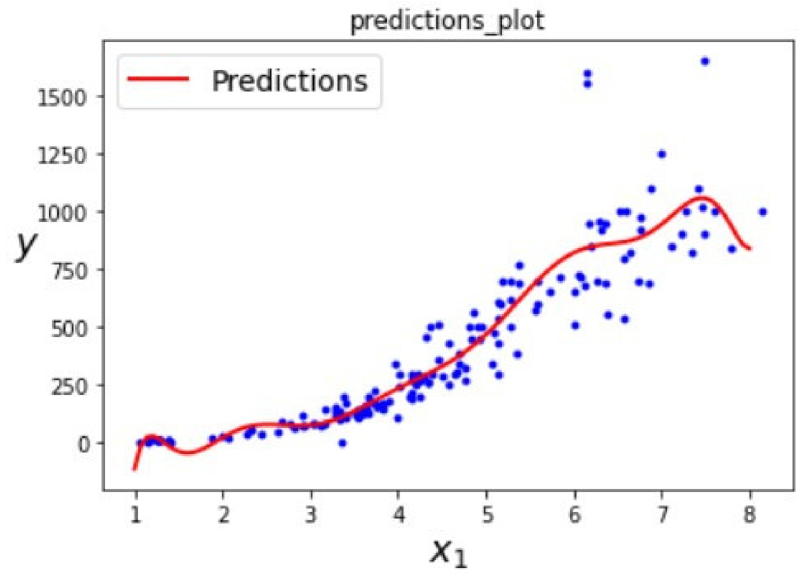
R square

THE VALUE OF R-SQUARED STAYS BETWEEN 0 AND 100%:

- 0% CORRESPONDS TO A MODEL THAT DOES NOT EXPLAIN THE VARIABILITY OF THE RESPONSE DATA AROUND ITS MEAN. THE MEAN OF THE DEPENDENT VARIABLE HELPS TO PREDICT THE DEPENDENT VARIABLE AND ALSO THE REGRESSION MODEL.
- ON THE OTHER HAND, 100% CORRESPONDS TO A MODEL THAT EXPLAINS THE VARIABILITY OF THE RESPONSE VARIABLE AROUND ITS MEAN.

IF YOUR VALUE OF R2 IS LARGE, YOU HAVE A BETTER CHANCE OF YOUR REGRESSION MODEL FITTING THE OBSERVATIONS.

```python
In [58]: x_new = np.linspace(1, 8, 100).reshape(100, 1)
x_new_poly = poly_features.transform(x_new)
y_new = lin_reg.predict(x_new_poly)
plt.plot(x, y, "b.")
plt.plot(x_new, y_new, "r-", linewidth = 2, label ="Predictions")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.legend(loc ="upper left", fontsize = 14)

plt.title("predictions_plot")
plt.show()
```



FITTING A POLYNOMIAL CURVE OF HIGHER DEGREE

OVERFITTING

MSE REDUCED TO 17840 FROM 27264

```python
In [59]: y_deg2 = lin_reg.predict(x_poly)
# model evaluation
mse_deg2 = mean_squared_error(y, y_deg2)

r2_deg2 = r2_score(y, y_deg2)

# printing values

print('MSE of Polyregression model', mse_deg2)

print('R2 score of Linear model: ', r2_deg2)
```

```
MSE of Polyregression model 17840.391759513375
R2 score of Linear model:  0.8599023077370866
```

GDP Dataset

```
In [5]: gdp.head()
```

Out[5]:

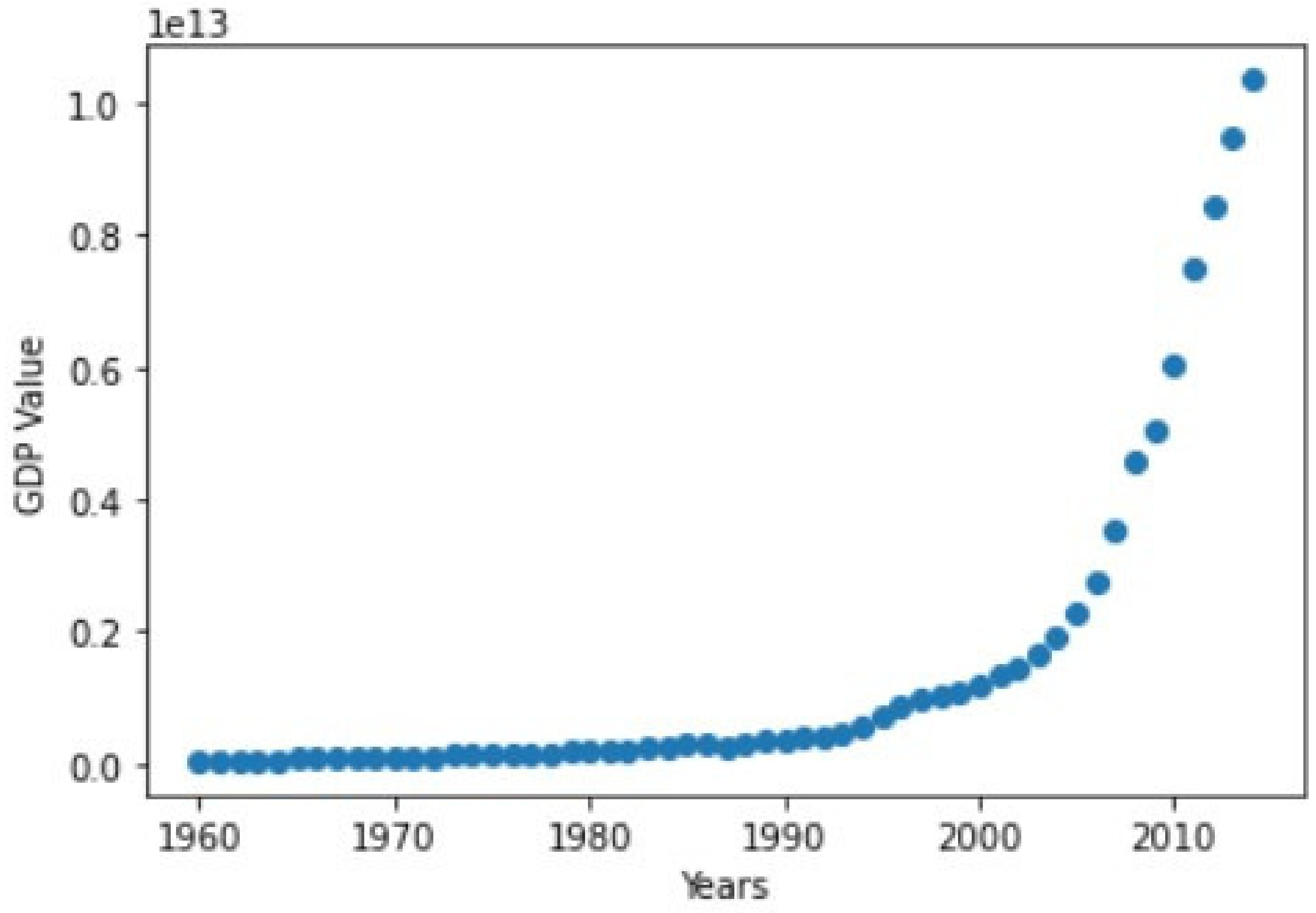
	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

GCD

DISPLAYING TOP 5 ENTRIES

```
In [44]: x = gdp.Year.values.reshape(-1,1)
         y = gdp.Value.values.reshape(-1,1)

         plt.scatter(x,y)
         plt.ylabel("GDP Value")
         plt.xlabel("Years")
```

```
Out[44]: Text(0.5, 0, 'Years')
```



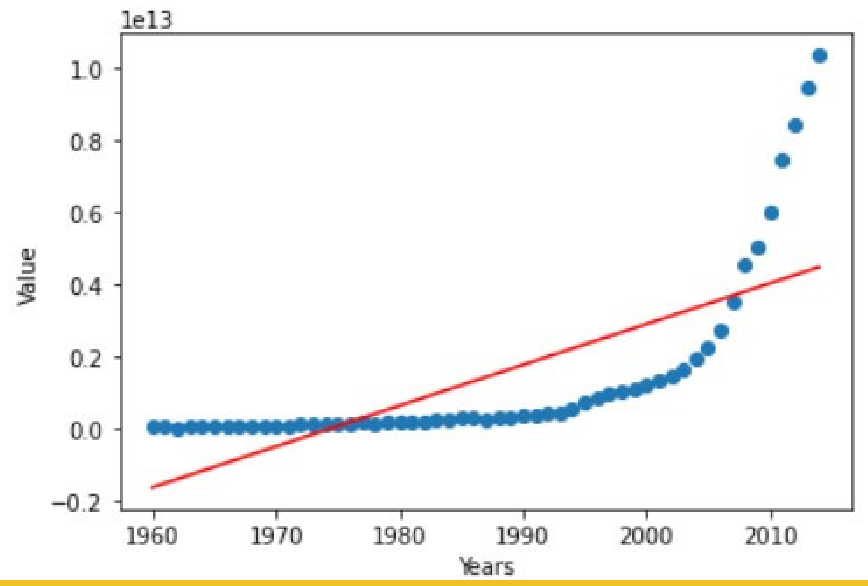
SCATTER PLOT

YEARS V/S VALUE

SCATTER PLOT WITH LINEAR LINE

```
In [8]: lr = LinearRegression()
        lr.fit(x,y)

        print('Slope of the line is', lr.coef_)
        print('Intercept value is', lr.intercept_)
        # Predict
        y_predicted = lr.predict(x)

Slope of the line is [[1.12959699e+11]]
Intercept value is [-2.23013881e+14]
```

```
In [45]: plt.scatter(x,y)
         plt.ylabel("Value")
         plt.xlabel("Years")

         plt.plot(x,y_predicted, color="red", label="linear")
         plt.show()
```

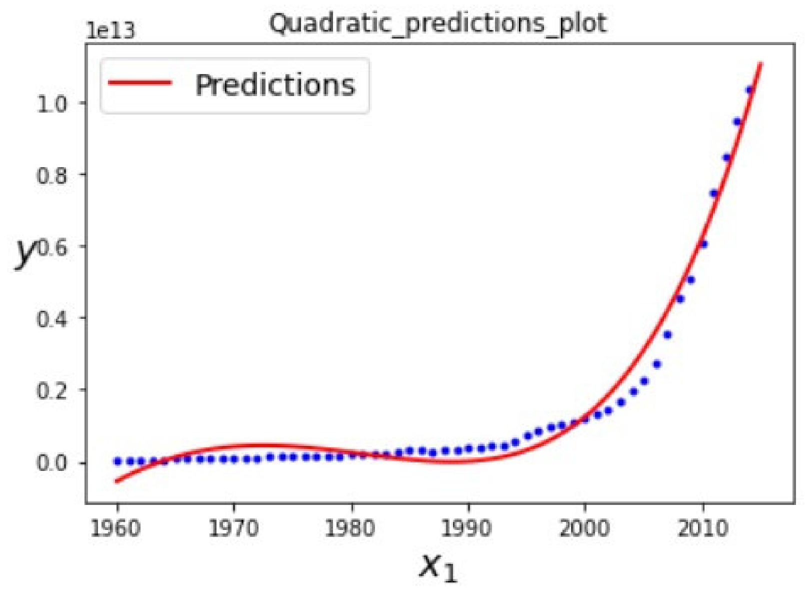


MSE AND R SQUARE VALUE

```
In [10]: # model evaluation
         mse = mean_squared_error(y, y_predicted)

         rmse = np.sqrt(mean_squared_error(y, y_predicted))
         r2 = r2_score(y, y_predicted)

         # printing values

         print('MSE of Linear model', mse)

         print('R2 score of Linear model: ', r2)

MSE of Linear model 2.921285914150742e+24
R2 score of Linear model:  0.5239708235574754
```

```
In [40]: poly_features = PolynomialFeatures(degree = 30, include_bias = False)
         x_poly = poly_features.fit_transform(x)
         x[3]

Out[40]: array([1963], dtype=int64)
```

```
In [42]: x_new = np.linspace(1960, 2015, 100).reshape(100, 1)
x_new_poly = poly_features.transform(x_new)
y_new = lin_reg.predict(x_new_poly)
plt.plot(x, y, "b.")
plt.plot(x_new, y_new, "r-", linewidth = 2, label ="Predictions")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.legend(loc ="upper left", fontsize = 14)

plt.title("Quadratic_predictions_plot")
plt.show()
```



FITTING A POLYNOMIAL CURVE OF HIGHER DEGREE

MSE REDUCED TO 1.3 FROM 2.9

```
In [43]: y_deg2 = lin_reg.predict(x_poly)
# model evaluation
mse_deg2 = mean_squared_error(y, y_deg2)

r2_deg2 = r2_score(y, y_deg2)

# printing values

print('MSE of Polyregression model', mse_deg2)

print('R2 score of Linear model: ', r2_deg2)

MSE of Polyregression model 1.339652981592359e+23
R2 score of Linear model:  0.9781700961738429
```

REMOVING OVERFITTING BY REGULARIZATION

What is Regularization

In regression analysis, the features are estimated using coefficients while modelling. Also, if the estimates can be restricted, or shrinked or regularized towards zero, then the impact of insignificant features might be reduced and would prevent models from high variance with a stable fit.

Regularization is the most used technique to penalize complex models in machine learning, it is deployed for reducing overfitting (or, contracting generalization errors) by putting network weights small. Also, it enhances the performance of models for new inputs.

BIBLIOGRAPHY

https://www.kaggle.com/code/mathchi/study-polynomial-regression

https://www.geeksforgeeks.org/polynomial-regression-for-non-linear-data-ml/

https://www.quora.com/Where-can-I-download-some-nonlinear-data-sets

https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning?fbclid=IwAR1pkL-RKR0xkKkdXtC1Tjcqm7CRX-FTD64U-nwNZYm_qnP2HEhhUdx0wW8

Bibliography

THANKS
FOR WATCHING

GROUP - 12

Ajeet Kushwaha

TRAINING MY DATA

```
In [21]: X_train1, X_test1, Y_train1, Y_test1 = train_test_split(x_poly, y, test_size=0.2, random_state= 42)

In [22]: y_pred1= LinearRegression()
         y_pred1.fit(X_train1,Y_train1)

Out[22]: LinearRegression()

In [23]: print(" Intercept value of Model is " ,y_pred1.intercept_)
         print("Co-efficient Value of Log Model is : ", y_pred1.coef_)

          Intercept value of Model is  [-4.03998111e+14]
         Co-efficient Value of Log Model is :  [[ 1.66165175e-082  2.94607169e-082  8.46552930e-088 -4.19047275e-163
            6.41610288e-150  1.40749267e-146  2.99102100e-143  6.20193638e-140
            1.26043637e-136  2.51800070e-133  4.95385748e-130  9.60924344e-127
            1.83895002e-123  3.47278127e-120  6.47051022e-117  1.18882656e-113
            2.15183584e-110  3.83177388e-107  6.69926057e-104  1.14680497e-100
            1.91476134e-097  3.10119224e-094  4.83347578e-091  7.16124729e-088
            9.88574133e-085  1.22627026e-081  1.26575093e-078  8.69825386e-076
           -8.48173375e-079  2.07007604e-082]]

In [24]: plt.scatter(X_test1, Y_test1,  color='gray')
         plt.scatter(X_train1, Y_train1,  color='red')
         plt.show()
```