# Topic: Simple Linear Regression

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Anandakrishnan k v ;;;;; Batch ID:** 19042021

**Topic: Text Mining and NLP**

**Grading Guidelines:**

**An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered for evaluation.**

**2. Assignments submitted after the deadline will affect your grades.**

**Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ans** | **Date** |  |  | **Ans** | **Date** |
| Correct | On time | A | 100 |  |  |
| 80% & above | On time | B | 85 | Correct | Late |
| 50% & above | On time | C | 75 | 80% & above | Late |
| 50% & below | On time | D | 65 | 50% & above | Late |
|  |  | E | 55 | 50% & below |  |
| Copied/No Submission |  | F | 45 |  |  |

* **Grade A: (>= 90):** When all assignments are submitted on or before the given deadline.
* **Grade B: (>= 80 and < 90):** 
  + When assignments are submitted on time but less than 80% of problems are completed.

(OR)

* + All assignments are submitted after the deadline.
* **Grade C: (>= 70 and < 80):** 
  + When assignments are submitted on time but less than 50% of the problems are completed.

(OR)

* + Less than 80% of problems in the assignments are submitted after the deadline.
* **Grade D: (>= 60 and < 70):**
  + Assignments submitted after the deadline and with 50% or less problems.
* **Grade E: (>= 50 and < 60):** 
  + Less than 30% of problems in the assignments are submitted after the deadline.

(OR)

* + Less than 30% of problems in the assignments are submitted before the deadline.
* **Grade F: (< 50):** No submission (or) malpractice.

**Hints:**

1. **Business Problem**

A certain food-based company conducted a survey with the help of a fitness company to find the relationship between a person’s weight gain and the number of calories they consumed in order to come up with diet plans for these individuals. Build a Simple Linear Regression model with calories consumed as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

* 1. **What is the business objective**

find the relationship between a person’s weight gain and the number of calories they consumed in order to come up with diet plans for individuals.

* 1. **Are there any constraints**

**Maximize:-** The accuracy of the prediction model

**Minimize:**- Th e complexity of the model

**Python Code:-**

# Importing necessary libraries

import pandas as pd # deals with data frame

import numpy as np # deals with numerical values

from sklearn.preprocessing import LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import linear\_model

df = pd.read\_csv("C:/Users/user/Downloads/New folder (3)/Datasets\_SLR/calories\_consumed.csv")

# Exploratory data analysis:

# 1. Measures of central tendency

# 2. Measures of dispersion

# 3. Third moment business decision

# 4. Fourth moment business decision

# 5. Probability distributions of variables

# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)

#changing column names

df.rename({'Weight gained (grams)':'wg' ,'Calories Consumed':'cc' }, axis=1, inplace =True)

#Data Cleaning

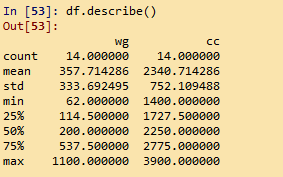
###### Null value Treatment ########

df.isna().sum() ## no null values

###### Summary of the data set ####

df.columns

df.describe()



#Graphical Representation

import matplotlib.pyplot as plt # mostly used for visualization purposes

plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['cc'], x = np.arange(1, 15, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

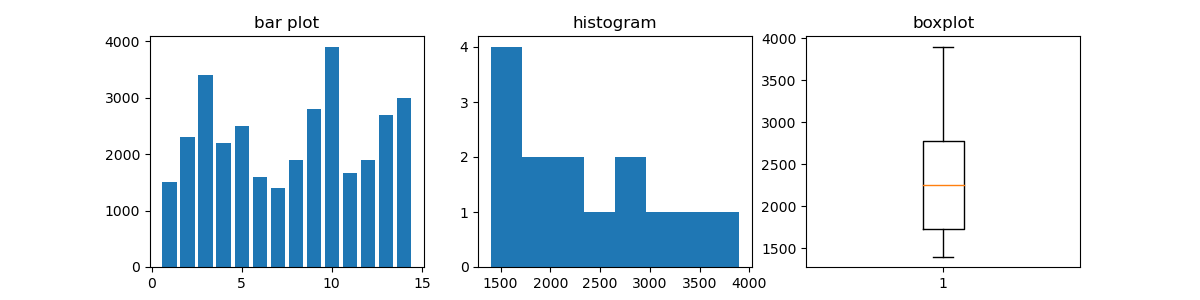
plt.hist(df['cc'],bins = 8) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

plt.boxplot(df['cc']) #boxplot

plt.title('boxplot')



plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['wg'], x = np.arange(1, 15, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

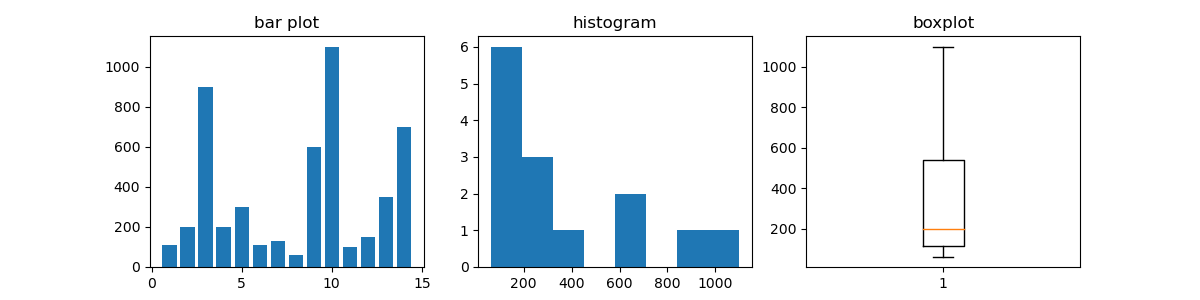
plt.hist(df['wg'],bins = 8) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

plt.boxplot(df['wg']) #boxplot

plt.title('boxplot')

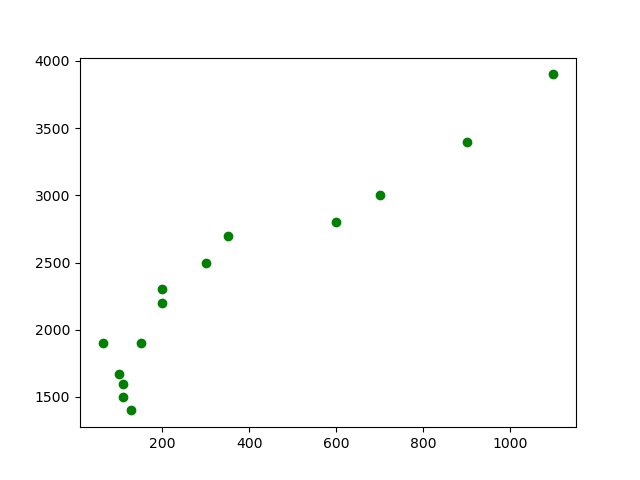


# Scatter plot

plt.scatter(x = df['wg'], y = df['cc'], color = 'green')

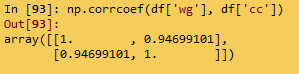
# correlation

np.corrcoef(df['wg'], df['cc'])



# correlation

np.corrcoef(df['wg'], df['cc'])



# Covariance

# NumPy does not have a function to calculate the covariance between two variables directly.

# Function for calculating a covariance matrix called cov()

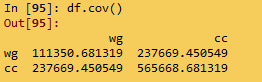
# By default, the cov() function will calculate the unbiased or sample covariance between the provided random variables.

cov\_output = np.cov(df['wg'], df['cc'])[0, 1]

cov\_output

C:\Users\user\Documents\paint.png

df.cov()



# Import library

import statsmodels.formula.api as smf

# Simple Linear Regression

model = smf.ols('cc ~ wg', data = df).fit()

model.summary()

pred1 = model.predict(pd.DataFrame(df['wg']))

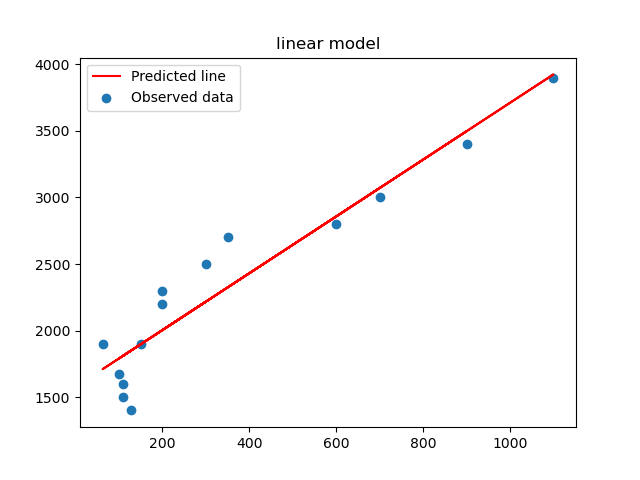
# Regression Line

plt.scatter(df['wg'], df['cc'])

plt.plot(df['wg'], pred1, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res1 = df.cc - pred1

res\_sqr1 = res1 \* res1

mse1 = np.mean(res\_sqr1)

rmse1 = np.sqrt(mse1)

rmse1

C:\Users\user\Documents\paint.png

######### Model building on Transformed Data

# Log Transformation

# x = log(wg); y = cc

plt.scatter(x = np.log(df['wg']), y = df['cc'], color = 'brown')

np.corrcoef(np.log(df['wg']), df['cc']) #correlation

model2 = smf.ols('cc ~ np.log(wg)', data = df).fit()

model2.summary()

pred2 = model2.predict(pd.DataFrame(df['wg']))

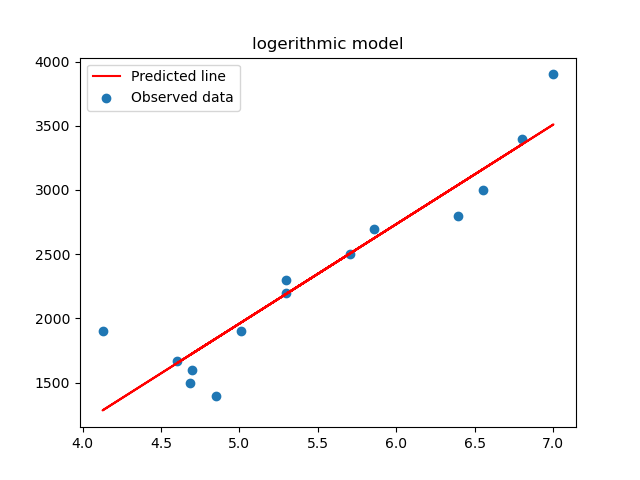
# Regression Line

plt.scatter(np.log(df['wg']), df['cc'])

plt.plot(np.log(df['wg']), pred2, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res2 = df['cc'] - pred2

res\_sqr2 = res2 \* res2

mse2 = np.mean(res\_sqr2)

rmse2 = np.sqrt(mse2)

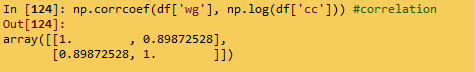
rmse2

#### Exponential transformation

# x = wg; y = log(cc)

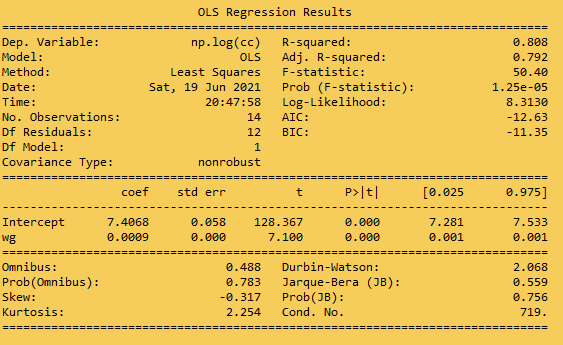
plt.scatter(x = df['wg'], y = np.log(df['cc']), color = 'orange')

np.corrcoef(df['wg'], np.log(df['cc'])) #correlation



model3 = smf.ols('np.log(cc) ~ wg', data = df).fit()

model3.summary()



pred3 = model3.predict(pd.DataFrame(df['wg']))

pred3\_at = np.exp(pred3)

pred3\_at

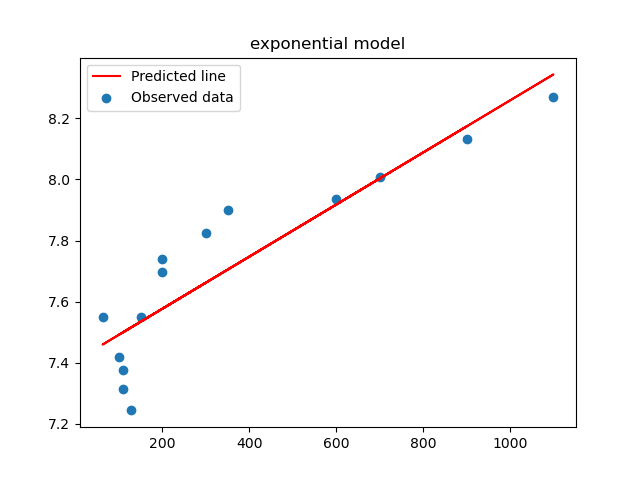
# Regression Line

plt.scatter(df['wg'], np.log(df['cc']))

plt.plot(df['wg'], pred3, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res3 = df.cc - pred3\_at

res\_sqr3 = res3 \* res3

mse3 = np.mean(res\_sqr3)

rmse3 = np.sqrt(mse3)

rmse3

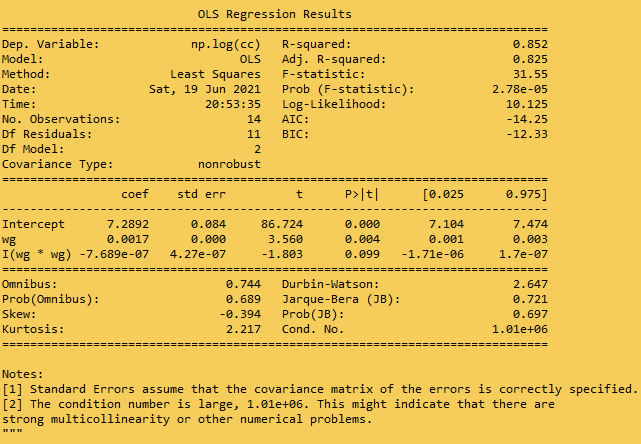
C:\Users\user\Documents\paint.png

#### Polynomial transformation

# x = wg; x^2 = wg\*wg; y = log(cc)

model4 = smf.ols('np.log(cc) ~ wg + I(wg\*wg)', data = df).fit()

model4.summary()



pred4 = model4.predict(pd.DataFrame(df))

pred4\_at = np.exp(pred4)

pred4\_at

# Regression line

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X = df.iloc[:, 0:1].values

X\_poly = poly\_reg.fit\_transform(X)

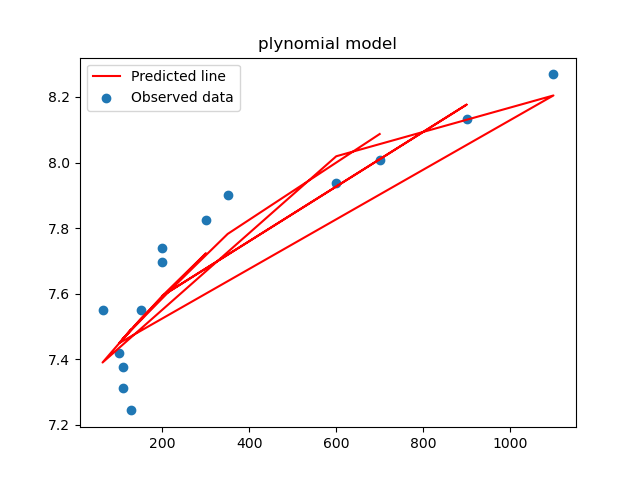
# y = df.iloc[:, 1].values

plt.scatter(df['wg'], np.log(df['cc']))

plt.plot(X, pred4, color = 'red')

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res4 = df.cc - pred4\_at

res\_sqr4 = res4 \* res4

mse4 = np.mean(res\_sqr4)

rmse4 = np.sqrt(mse4)

rmse4

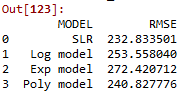
C:\Users\user\Documents\paint.png

# Choose the best model using RMSE

data = {"MODEL":pd.Series(["SLR", "Log model", "Exp model", "Poly model"]), "RMSE":pd.Series([rmse1, rmse2, rmse3, rmse4])}

table\_rmse = pd.DataFrame(data)

table\_rmse



###################

# The best model is linear model itself

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size = 0.2,random\_state =42)

finalmodel = smf.ols('cc ~ wg', data = train).fit()

finalmodel.summary()

# Predict on test data

test\_pred = finalmodel.predict(pd.DataFrame(test))

test\_pred

# Model Evaluation on Test data

test\_res = test.cc - test\_pred

test\_sqrs = test\_res \* test\_res

test\_mse = np.mean(test\_sqrs)

test\_rmse = np.sqrt(test\_mse)

test\_rmse

C:\Users\user\Documents\Figure_1.png

# Prediction on train data

train\_pred = finalmodel.predict(pd.DataFrame(train))

train\_pred

# Model Evaluation on train data

train\_res = train.cc - train\_pred

train\_sqrs = train\_res \* train\_res

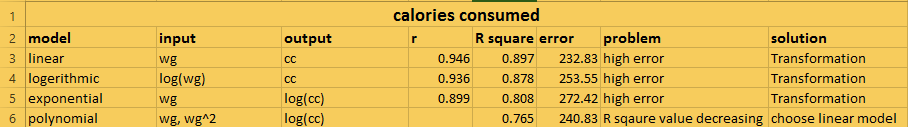
train\_mse = np.mean(train\_sqrs)

train\_rmse = np.sqrt(train\_mse)

train\_rmse

C:\Users\user\Documents\Figure_1.png

**Summary:-**

****

Linearl model showing better performance compare to all other model by showing higher “R square” value ( more number of true accurate predictions) and lower “error” value.

**Business advantage:-**

By using this model we can predict the relation between caloriess of food consumed and the corresponding weight gain. This will helps the individuals to predict how much calories of food make a person gain a perticular weight. According to that can plan a diet plan by knowing the calories of each food items.

In business level it will helps the client to determine the ingradients amount can add to a perticular food product by knowing calories content of each ingradient. That will helps to earn the positive feedback and reviews from the customer for the product and leads to to improving the business.

**Business Problem**

A logistics company recorded the time taken for delivery and the time taken for the sorting of the items for delivery. Build a Simple Linear Regression model to find the relationship between delivery time and sorting time with delivery time as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

**What is the business objective:-**

Build a Simple Linear Regression model to find the relationship between delivery time and sorting time, thereby ensure on time delivery.

**Are there any constraints:-**

**Maximize:-** strength of on time delivery

**Maximize:-** customer satisfaction

**Maximize:-** The accuracy of the prediction model

**Minimize:**- Th e complexity of the model

**Python Code:-**

# Importing necessary libraries

import pandas as pd # deals with data frame

import numpy as np # deals with numerical values

from sklearn.preprocessing import LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import linear\_model

df = pd.read\_csv("C:/Users/user/Downloads/New folder (3)/Datasets\_SLR/delivery\_time.csv")

# Exploratory data analysis:

# 1. Measures of central tendency

# 2. Measures of dispersion

# 3. Third moment business decision

# 4. Fourth moment business decision

# 5. Probability distributions of variables

# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)

#changing column names

df.rename({'Delivery Time':'dt' ,'Sorting Time':'st' }, axis=1, inplace =True)

#Data Cleaning

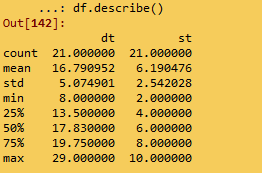
###### Null value Treatment ########

df.isna().sum() ## no null values

###### Summary of the data set ####

df.columns

df.describe()



#Graphical Representation

import matplotlib.pyplot as plt # mostly used for visualization purposes

plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['dt'], x = np.arange(1, 22, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

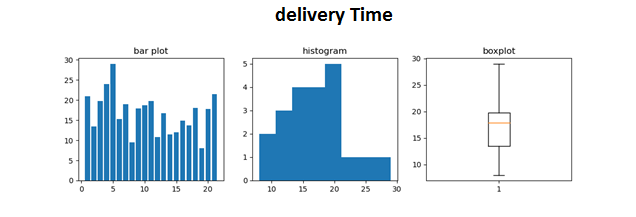
plt.hist(df['dt'],bins = 8) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

plt.boxplot(df['dt']) #boxplot

plt.title('boxplot')



plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['st'], x = np.arange(1, 22, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

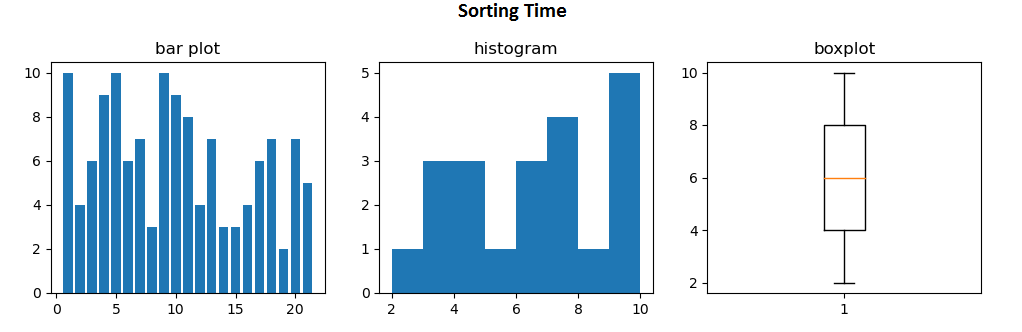
plt.hist(df['st'],bins = 8) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

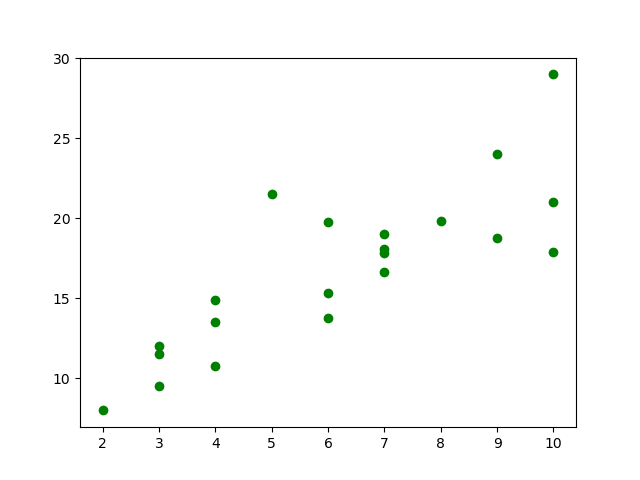
plt.boxplot(df['st']) #boxplot

plt.title('boxplot')



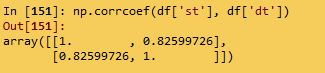
# Scatter plot

plt.scatter(x = df['st'], y = df['dt'], color = 'green')



# correlation

np.corrcoef(df['st'], df['dt'])



# Covariance

# NumPy does not have a function to calculate the covariance between two variables directly.

# Function for calculating a covariance matrix called cov()

# By default, the cov() function will calculate the unbiased or sample covariance between the provided random variables.

cov\_output = np.cov(df['st'], df['dt'])[0, 1]

cov\_output

df.cov()

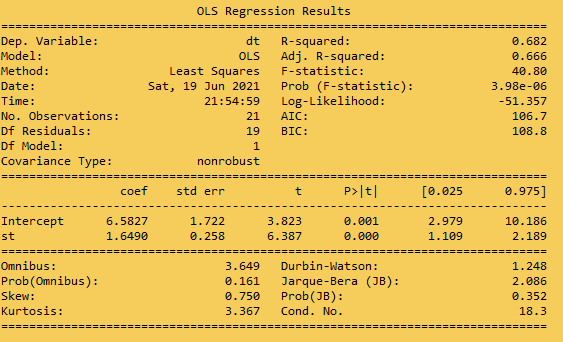
# Import library

import statsmodels.formula.api as smf

# Simple Linear Regression

model = smf.ols('dt ~ st', data = df).fit()

model.summary()



pred1 = model.predict(pd.DataFrame(df['st']))

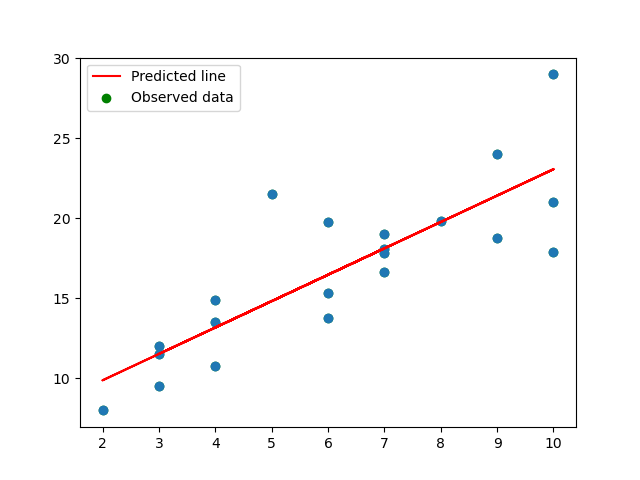
# Regression Line

plt.scatter(df['st'], df['dt'])

plt.plot(df['st'], pred1, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res1 = df.dt - pred1

res\_sqr1 = res1 \* res1

mse1 = np.mean(res\_sqr1)

rmse1 = np.sqrt(mse1)

rmse1

C:\Users\user\Documents\paint.png

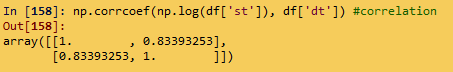
######### Model building on Transformed Data

# Log Transformation

# x = log(st); y = dt

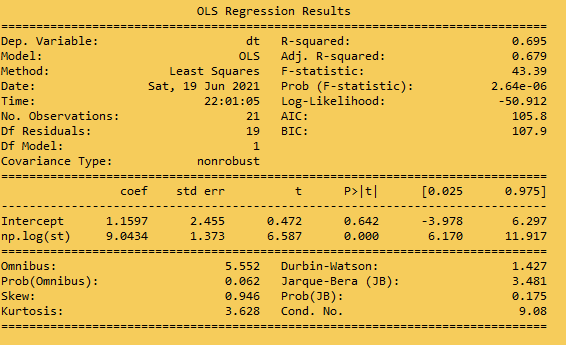
plt.scatter(x = np.log(df['st']), y = df['dt'], color = 'brown')

np.corrcoef(np.log(df['st']), df['dt']) #correlation



model2 = smf.ols('dt ~ np.log(st)', data = df).fit()

model2.summary()



pred2 = model2.predict(pd.DataFrame(df['st']))

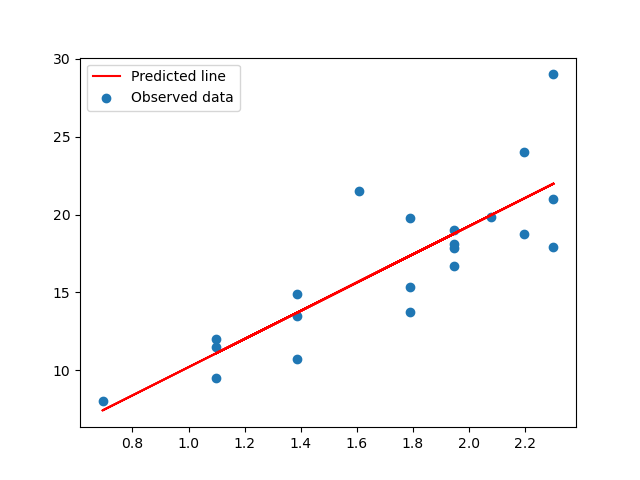
# Regression Line

plt.scatter(np.log(df['st']), df['dt'])

plt.plot(np.log(df['st']), pred2, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res2 = df['dt'] - pred2

res\_sqr2 = res2 \* res2

mse2 = np.mean(res\_sqr2)

rmse2 = np.sqrt(mse2)

rmse2

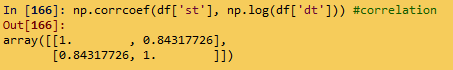
C:\Users\user\Documents\paint.png

#### Exponential transformation

# x = st; y = log(dt)

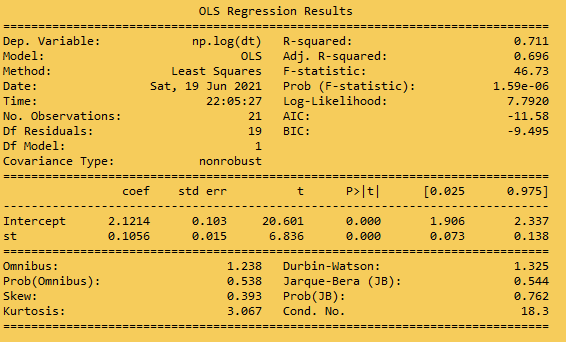
plt.scatter(x = df['st'], y = np.log(df['dt']), color = 'orange')

np.corrcoef(df['st'], np.log(df['dt'])) #correlation



model3 = smf.ols('np.log(dt) ~ st', data = df).fit()

model3.summary()



pred3 = model3.predict(pd.DataFrame(df['st']))

pred3\_at = np.exp(pred3)

pred3\_at

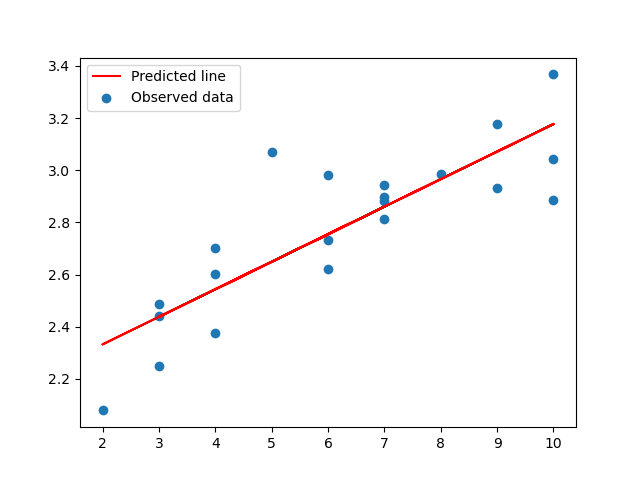
# Regression Line

plt.scatter(df['st'], np.log(df['dt']))

plt.plot(df['st'], pred3, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res3 = df.dt - pred3\_at

res\_sqr3 = res3 \* res3

mse3 = np.mean(res\_sqr3)

rmse3 = np.sqrt(mse3)

rmse3

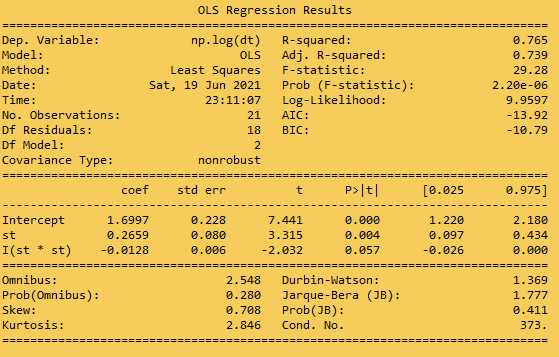
C:\Users\user\Documents\paint.png

#### Polynomial transformation

# x = st; x^2 = st\*st; y = log(dt)

model4 = smf.ols('np.log(dt) ~ st + I(st\*st)', data = df).fit()

model4.summary()



pred4 = model4.predict(pd.DataFrame(df))

pred4\_at = np.exp(pred4)

pred4\_at

# Regression line

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X = df.iloc[:, 1: ].values

X\_poly = poly\_reg.fit\_transform(X)

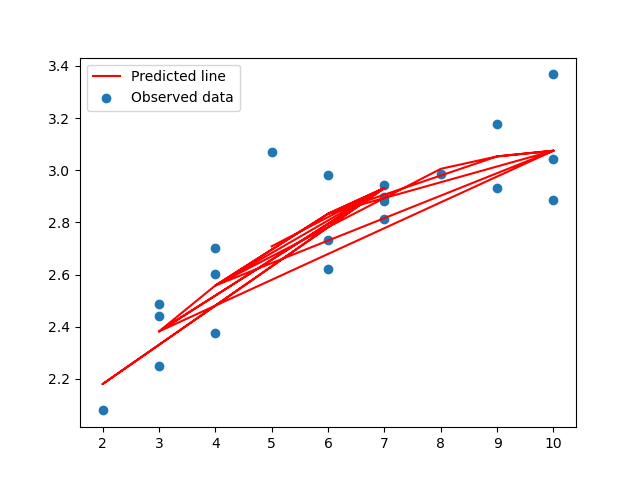
# y = df.iloc[:, 1].values

plt.scatter(df['st'], np.log(df['dt']))

plt.plot(X, pred4, color = 'red')

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res4 = df.dt - pred4\_at

res\_sqr4 = res4 \* res4

mse4 = np.mean(res\_sqr4)

rmse4 = np.sqrt(mse4)

rmse4

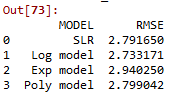
C:\Users\user\Documents\paint.png

# Choose the best model using RMSE

data = {"MODEL":pd.Series(["SLR", "Log model", "Exp model", "Poly model"]), "RMSE":pd.Series([rmse1, rmse2, rmse3, rmse4])}

table\_rmse = pd.DataFrame(data)

table\_rmse



###################

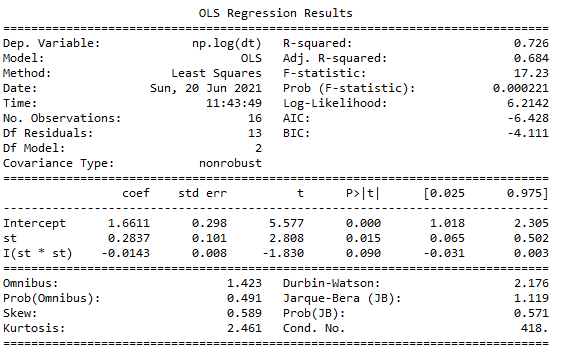
# The best model

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size = 0.2)

finalmodel = smf.ols('np.log(dt) ~ st + I(st\*st)', data = train).fit()

finalmodel.summary()



# Predict on test data

test\_pred = finalmodel.predict(pd.DataFrame(test))

pred\_test\_dt = np.exp(test\_pred)

pred\_test\_dt

# Model Evaluation on Test data

test\_res = test.dt - pred\_test\_dt

test\_sqrs = test\_res \* test\_res

test\_mse = np.mean(test\_sqrs)

test\_rmse = np.sqrt(test\_mse)

test\_rmse

C:\Users\user\Documents\paint.png

# Prediction on train data

train\_pred = finalmodel.predict(pd.DataFrame(train))

pred\_train\_dt = np.exp(train\_pred)

pred\_train\_dt

# Model Evaluation on train data

train\_res = train.dt - pred\_train\_dt

train\_sqrs = train\_res \* train\_res

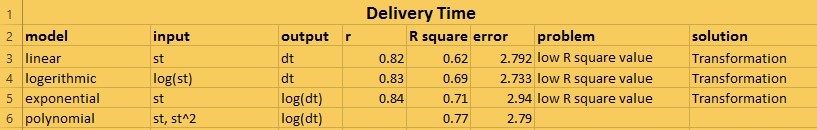
train\_mse = np.mean(train\_sqrs)

train\_rmse = np.sqrt(train\_mse)

train\_rmse

C:\Users\user\Documents\paint.png

**Summary:-**

****

Polynomial model showing better performance compare to all other model by showing higher “R square” value ( more number of true accurate predictions) and lower “error” value.

**Business advantage:-**

By using this model we can predict the corresponding delivery time by knowing about the sorting time of the products hence ensure on time delivery of the products thereby can earn customer satisfaction.

**Business Problem:-**

A certain organization wants an early estimate of their employee churn out rate. So the HR department gathered the data regarding the employee’s salary hike and the churn out rate in a financial year. The analytics team will have to perform an analysis and predict an estimate of employee churn based on the salary hike. Build a Simple Linear Regression model with churn out rate as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

**What is the business objective:-**

Perform an analysis and predict an estimate of employee churn based on the salary hike. And thereby minimize churn out rate

**Are there any constraints:-**

**Minimize:**- salary hike

**Maximize:-** employees satisfaction

**Minimize:**- churn out rate

**Maximize:-** The accuracy of the prediction model

**Minimize:**- The complexity of the model

**Python Code:**-

# Importing necessary libraries

import pandas as pd # deals with data frame

import numpy as np # deals with numerical values

from sklearn.preprocessing import LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import linear\_model

df = pd.read\_csv("C:/Users/user/Downloads/New folder (3)/Datasets\_SLR/emp\_data.csv")

# Exploratory data analysis:

# 1. Measures of central tendency

# 2. Measures of dispersion

# 3. Third moment business decision

# 4. Fourth moment business decision

# 5. Probability distributions of variables

# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)

#changing column names

df.rename({'Salary\_hike':'sh' ,'Churn\_out\_rate':'co' }, axis=1, inplace =True)

#Data Cleaning

###### Null value Treatment ########

df.isna().sum() ## no null values

###### Summary of the data set ####

df.columns

df.describe()

#Graphical Representation

import matplotlib.pyplot as plt # mostly used for visualization purposes

plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['co'], x = np.arange(1, 11, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

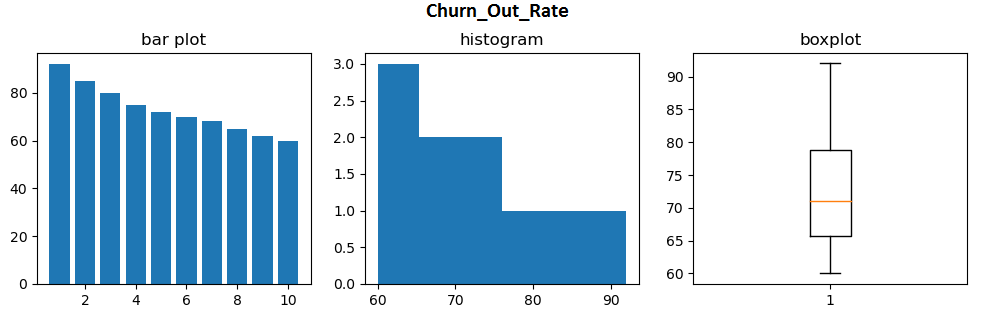
plt.hist(df['co'],bins = 6) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

plt.boxplot(df['co']) #boxplot

plt.title('boxplot')



plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['sh'], x = np.arange(1, 11, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

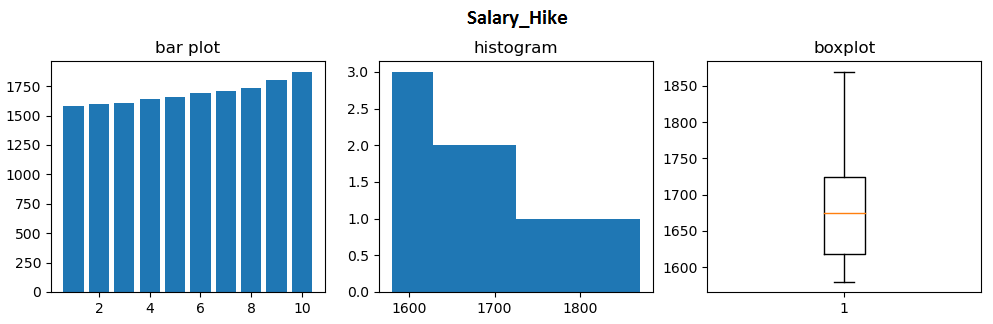
plt.hist(df['sh'],bins = 6) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

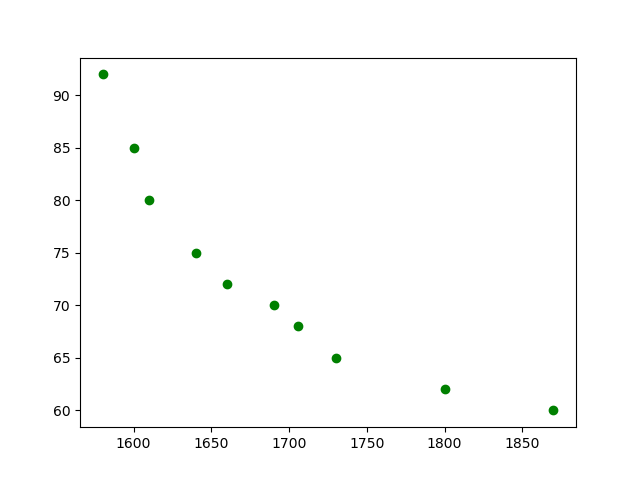
plt.boxplot(df['sh']) #boxplot

plt.title('boxplot')



# Scatter plot

plt.scatter(x = df['sh'], y = df['co'], color = 'green')



# correlation

np.corrcoef(df['sh'], df['co'])

# Covariance

# NumPy does not have a function to calculate the covariance between two variables directly.

# Function for calculating a covariance matrix called cov()

# By default, the cov() function will calculate the unbiased or sample covariance between the provided random variables.

cov\_output = np.cov(df['sh'], df['co'])[0, 1]

cov\_output

df.cov()

# Import library

import statsmodels.formula.api as smf

# Simple Linear Regression

model = smf.ols('co ~ sh', data = df).fit()

model.summary()

pred1 = model.predict(pd.DataFrame(df['sh']))

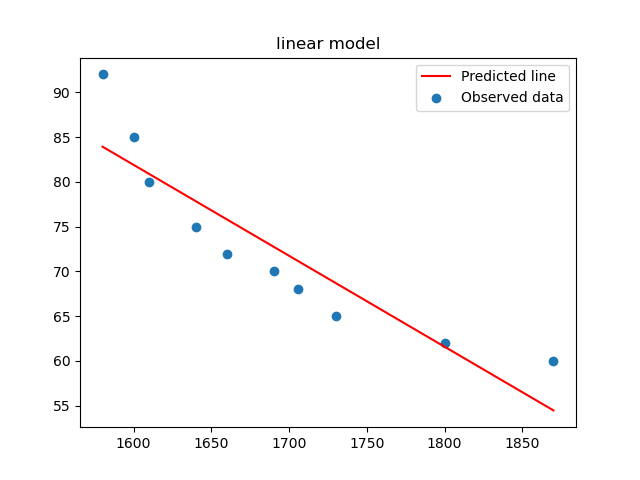
# Regression Line

plt.scatter(df['sh'], df['co'])

plt.plot(df['sh'], pred1, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res1 = df.co - pred1

res\_sqr1 = res1 \* res1

mse1 = np.mean(res\_sqr1)

rmse1 = np.sqrt(mse1)

rmse1

######### Model building on Transformed Data

# Log Transformation

# x = log(sh); y = co

plt.scatter(x = np.log(df['sh']), y = df['co'], color = 'brown')

np.corrcoef(np.log(df['sh']), df['co']) #correlation

model2 = smf.ols('co ~ np.log(sh)', data = df).fit()

model2.summary()

pred2 = model2.predict(pd.DataFrame(df['sh']))

# Regression Line

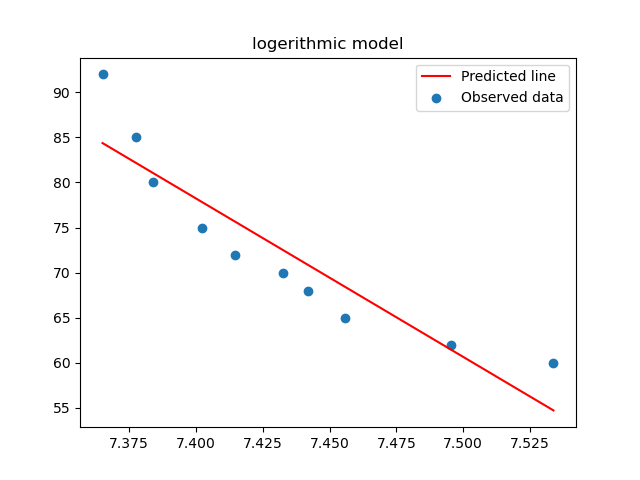
plt.scatter(np.log(df['sh']), df['co'])

plt.plot(np.log(df['sh']), pred2, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.title('logerithmic model')

plt.show()



# Error calculation

res2 = df['co'] - pred2

res\_sqr2 = res2 \* res2

mse2 = np.mean(res\_sqr2)

rmse2 = np.sqrt(mse2)

rmse2

#### Exponential transformation

# x = sh; y = log(co)

plt.scatter(x = df['sh'], y = np.log(df['co']), color = 'orange')

np.corrcoef(df['sh'], np.log(df['co'])) #correlation

model3 = smf.ols('np.log(co) ~ sh', data = df).fit()

model3.summary()

pred3 = model3.predict(pd.DataFrame(df['sh']))

pred3\_at = np.exp(pred3)

pred3\_at

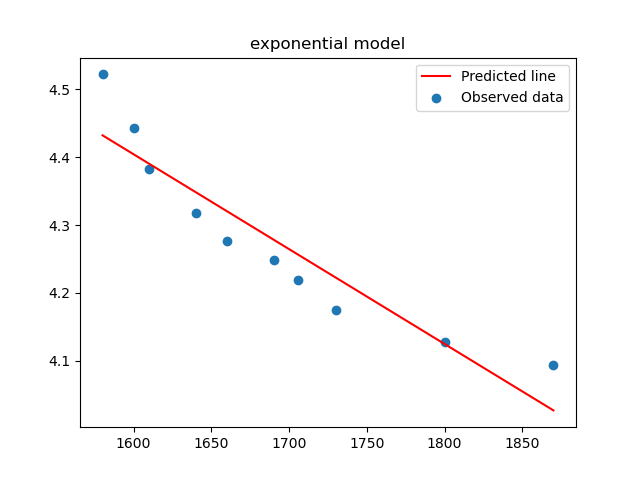
# Regression Line

plt.scatter(df['sh'], np.log(df['co']))

plt.plot(df['sh'], pred3, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res3 = df.co - pred3\_at

res\_sqr3 = res3 \* res3

mse3 = np.mean(res\_sqr3)

rmse3 = np.sqrt(mse3)

rmse3

#### Polynomial transformation

# x = sh; x^2 = sh\*sh; y = log(co)

model4 = smf.ols('np.log(co) ~ sh + I(sh\*sh)', data = df).fit()

model4.summary()

pred4 = model4.predict(pd.DataFrame(df))

pred4\_at = np.exp(pred4)

pred4\_at

# Regression line

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X = df.iloc[:, 0:1].values

X\_poly = poly\_reg.fit\_transform(X)

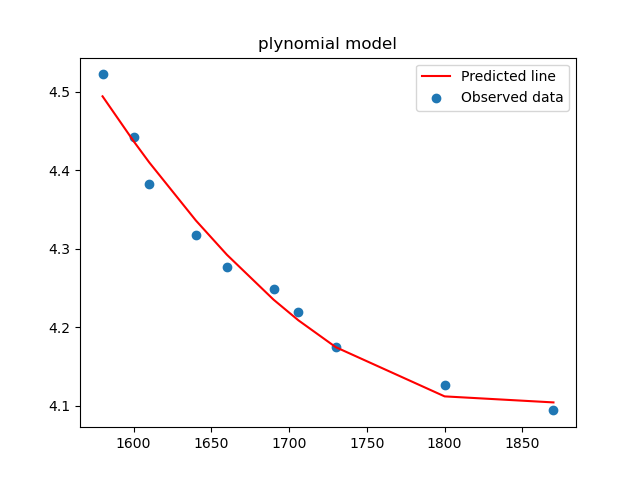
# y = df.iloc[:, 1].values

plt.scatter(df['sh'], np.log(df['co']))

plt.plot(X, pred4, color = 'red')

plt.legend(['Predicted line', 'Observed data'])

plt.show()



# Error calculation

res4 = df.co - pred4\_at

res\_sqr4 = res4 \* res4

mse4 = np.mean(res\_sqr4)

rmse4 = np.sqrt(mse4)

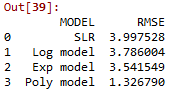
rmse4

# Choose the best model using RMSE

data = {"MODEL":pd.Series(["SLR", "Log model", "Exp model", "Poly model"]), "RMSE":pd.Series([rmse1, rmse2, rmse3, rmse4])}

table\_rmse = pd.DataFrame(data)

table\_rmse



###################

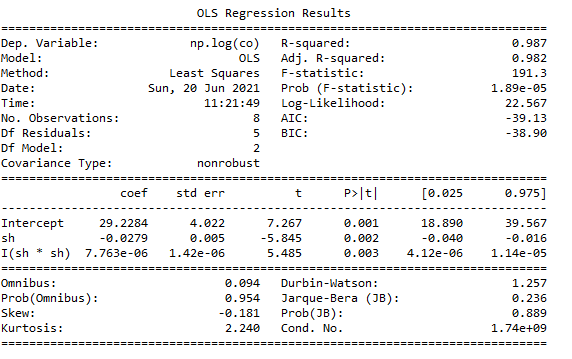
# The best model is polynomial model

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size = 0.2)

finalmodel = smf.ols('np.log(co) ~ sh + I(sh\*sh)', data = train).fit()

finalmodel.summary()



# Predict on test data

test\_pred = finalmodel.predict(pd.DataFrame(test))

pred\_test\_co = np.exp(test\_pred)

pred\_test\_co

# Model Evaluation on Test data

test\_res = test.co - pred\_test\_co

test\_sqrs = test\_res \* test\_res

test\_mse = np.mean(test\_sqrs)

test\_rmse = np.sqrt(test\_mse)

test\_rmse

C:\Users\user\Documents\paint.png

# Prediction on train data

train\_pred = finalmodel.predict(pd.DataFrame(train))

pred\_train\_co = np.exp(train\_pred)

pred\_train\_co

# Model Evaluation on train data

train\_res = train.co - pred\_train\_co

train\_sqrs = train\_res \* train\_res

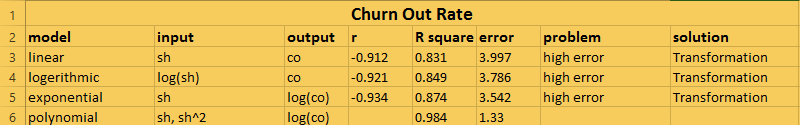
train\_mse = np.mean(train\_sqrs)

train\_rmse = np.sqrt(train\_mse)

train\_rmse

C:\Users\user\Documents\paint.png

**Summary:-**

****

Polynomial model showing better performance compare to all other model by showing higher “R square” value ( more number of accurate predictions) and “low” error value.

**Business advantage:-**

By using this model we can find the minimum salary hike that’s make employees happy and stay on with the company and thereby prevent the churn out rate.

**Business Problem:-**

## The head of HR of a certain organization wants to automate their salary hike estimation. The organization consulted an analytics service provider and asked them to build a basic prediction model by providing them with a dataset that contains the data about the number of years of experience and the salary hike given accordingly. Build a Simple Linear Regression model with salary as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

**What is the business objective**

## Buid a model to automate the salary hike estimation of employees according to their year of experience

**Are there any constraints**

**Maximize:-** emplyees satisfaction

**Minimize:-** salary hike

**Minimize:**- churn out rate

**Maximize:-** The accuracy of the prediction model

**Minimize:**- The complexity of the model

**Python Code:-**

# Importing necessary libraries

import pandas as pd # deals with data frame

import numpy as np # deals with numerical values

from sklearn.preprocessing import LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import linear\_model

df = pd.read\_csv("C:/Users/user/Downloads/New folder (3)/Datasets\_SLR/Salary\_Data.csv")

# Exploratory data analysis:

# 1. Measures of central tendency

# 2. Measures of dispersion

# 3. Third moment business decision

# 4. Fourth moment business decision

# 5. Probability distributions of variables

# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot, Bar plot, etc.)

#changing column names

df.rename({'YearsExperience':'sh' ,'Salary':'co' }, axis=1, inplace =True)

#Data Cleaning

###### Null value Treatment ########

df.isna().sum() ## no null values

###### Summary of the data set ####

df.columns

df.describe()

#Graphical Representation

import matplotlib.pyplot as plt # mostly used for visualization purposes

plt.figure(figsize= (12,3))

plt.subplot(1,3,1)

plt.bar(height = df['co'], x = np.arange(0, 30,1 ))

plt.title('bar plot')

plt.subplot(1,3,2)

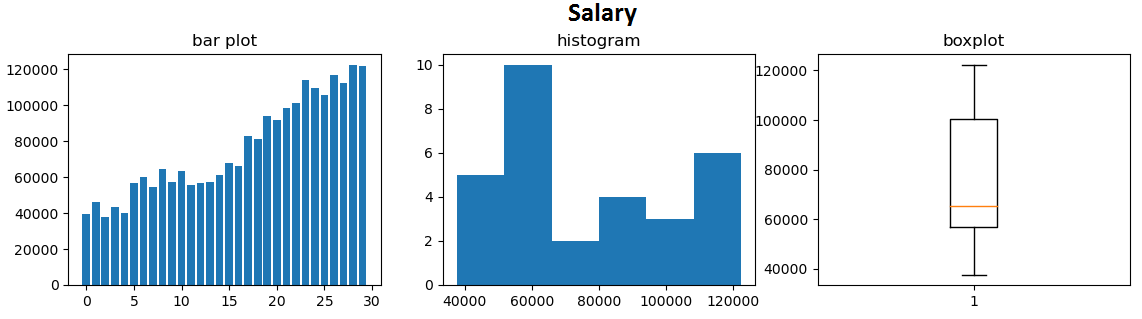
plt.hist(df['co'],bins = 6) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

plt.boxplot(df['co']) #boxplot

plt.title('boxplot')



plt.figure(figsize= (14,3))

plt.subplot(1,3,1)

plt.bar(height = df['sh'], x = np.arange(1, 30, 1))

plt.title('bar plot')

plt.subplot(1,3,2)

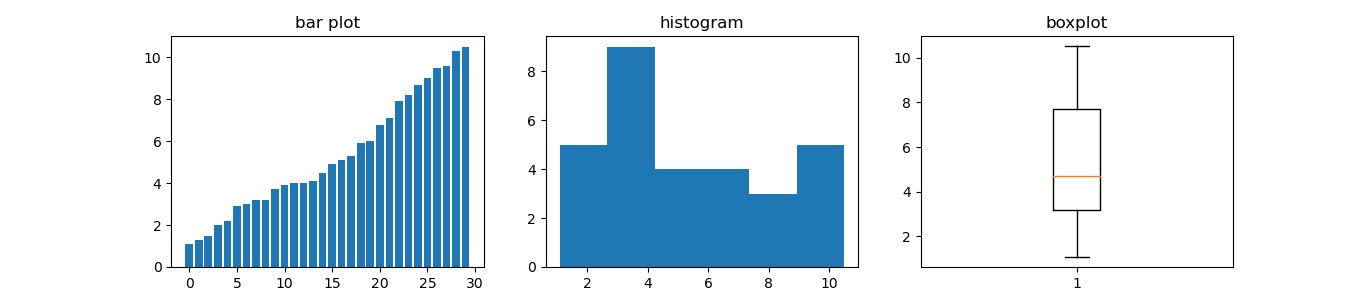
plt.hist(df['sh'],bins = 6) #histogram

plt.title('histogram')

plt.subplot(1,3,3)

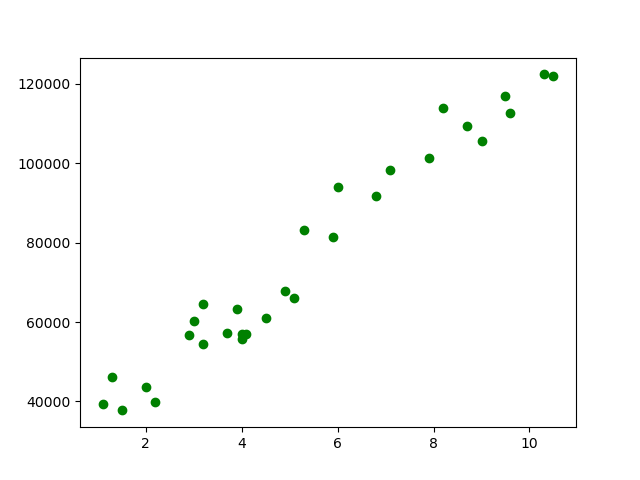
plt.boxplot(df['sh']) #boxplot

plt.title('boxplot')



# Scatter plot

plt.scatter(x = df['sh'], y = df['co'], color = 'green')



# correlation

np.corrcoef(df['sh'], df['co'])

# Covariance

# NumPy does not have a function to calculate the covariance between two variables directly.

# Function for calculating a covariance matrix called cov()

# By default, the cov() function will calculate the unbiased or sample covariance between the provided random variables.

cov\_output = np.cov(df['sh'], df['co'])[0, 1]

cov\_output

df.cov()

# Import library

import statsmodels.formula.api as smf

# Simple Linear Regression

model = smf.ols('co ~ sh', data = df).fit()

model.summary()

pred1 = model.predict(pd.DataFrame(df['sh']))

# Regression Line

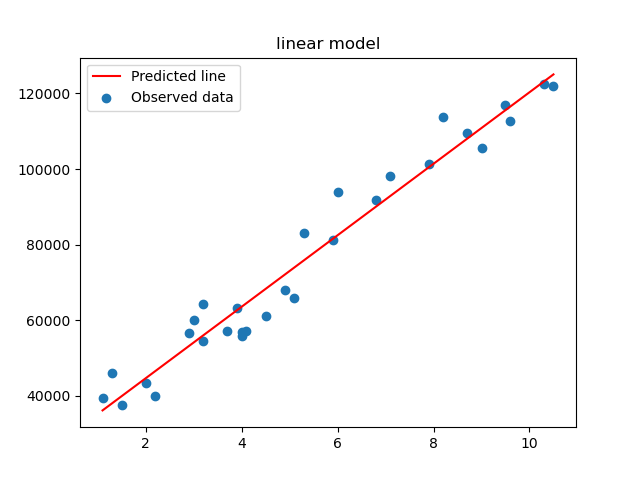
plt.scatter(df['sh'], df['co'])

plt.plot(df['sh'], pred1, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.title('linear model')

plt.show()



# Error calculation

res1 = df.co - pred1

res\_sqr1 = res1 \* res1

mse1 = np.mean(res\_sqr1)

rmse1 = np.sqrt(mse1)

rmse1

######### Model building on Transformed Data

# Log Transformation

# x = log(sh); y = co

plt.scatter(x = np.log(df['sh']), y = df['co'], color = 'brown')

np.corrcoef(np.log(df['sh']), df['co']) #correlation

model2 = smf.ols('co ~ np.log(sh)', data = df).fit()

model2.summary()

pred2 = model2.predict(pd.DataFrame(df['sh']))

# Regression Line

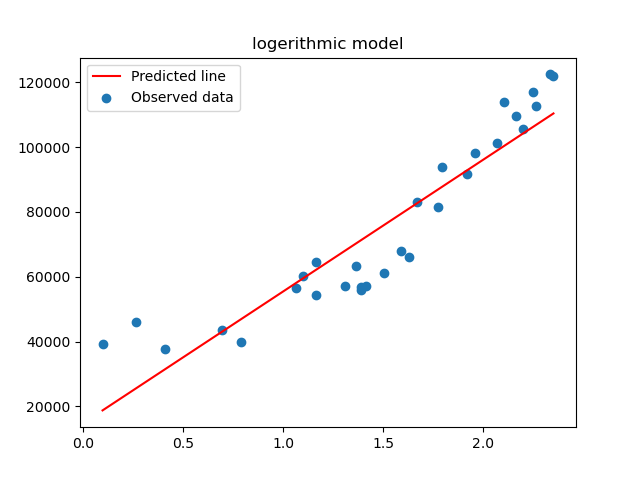
plt.scatter(np.log(df['sh']), df['co'])

plt.plot(np.log(df['sh']), pred2, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.title('logerithmic model')

plt.show()



# Error calculation

res2 = df['co'] - pred2

res\_sqr2 = res2 \* res2

mse2 = np.mean(res\_sqr2)

rmse2 = np.sqrt(mse2)

rmse2

#### Exponential transformation

# x = sh; y = log(co)

plt.scatter(x = df['sh'], y = np.log(df['co']), color = 'orange')

np.corrcoef(df['sh'], np.log(df['co'])) #correlation

model3 = smf.ols('np.log(co) ~ sh', data = df).fit()

model3.summary()

pred3 = model3.predict(pd.DataFrame(df['sh']))

pred3\_at = np.exp(pred3)

pred3\_at

# Regression Line

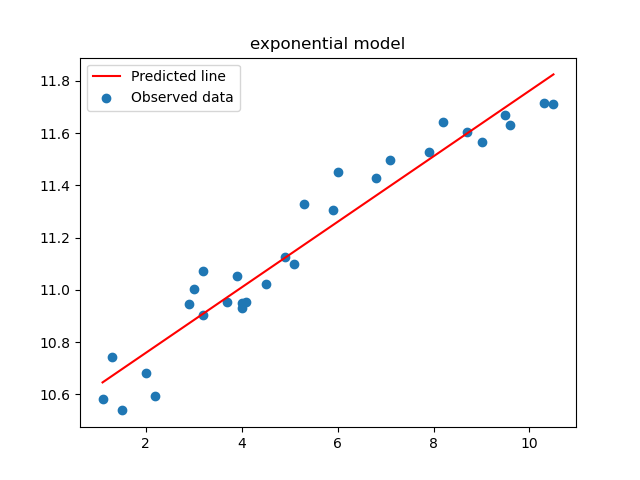
plt.scatter(df['sh'], np.log(df['co']))

plt.plot(df['sh'], pred3, "r")

plt.legend(['Predicted line', 'Observed data'])

plt.title("exponential model")

plt.show()



# Error calculation

res3 = df.co - pred3\_at

res\_sqr3 = res3 \* res3

mse3 = np.mean(res\_sqr3)

rmse3 = np.sqrt(mse3)

rmse3

#### Polynomial transformation

# x = sh; x^2 = sh\*sh; y = log(co)

model4 = smf.ols('np.log(co) ~ sh + I(sh\*sh)', data = df).fit()

model4.summary()

pred4 = model4.predict(pd.DataFrame(df))

pred4\_at = np.exp(pred4)

pred4\_at

# Regression line

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X = df.iloc[:, 0:1].values

X\_poly = poly\_reg.fit\_transform(X)

# y = df.iloc[:, 1].values

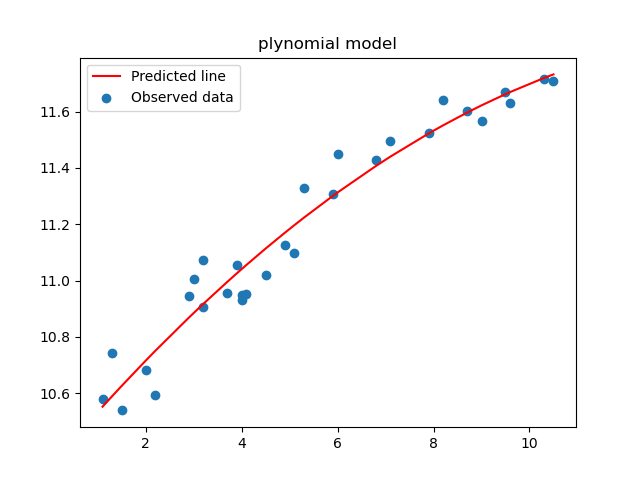
plt.scatter(df['sh'], np.log(df['co']))

plt.plot(X, pred4, color = 'red')

plt.legend(['Predicted line', 'Observed data'])

plt.title("plynomial model")

plt.show()



# Error calculation

res4 = df.co - pred4\_at

res\_sqr4 = res4 \* res4

mse4 = np.mean(res\_sqr4)

rmse4 = np.sqrt(mse4)

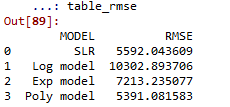
rmse4

# Choose the best model using RMSE

data = {"MODEL":pd.Series(["SLR", "Log model", "Exp model", "Poly model"]), "RMSE":pd.Series([rmse1, rmse2, rmse3, rmse4])}

table\_rmse = pd.DataFrame(data)

table\_rmse



###################

# The best model

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size = 0.2)

finalmodel = smf.ols('np.log(co) ~ sh + I(sh\*sh)', data = train).fit()

finalmodel.summary()

# Predict on test data

test\_pred = finalmodel.predict(pd.DataFrame(test))

pred\_test\_co = np.exp(test\_pred)

pred\_test\_co

# Model Evaluation on Test data

test\_res = test.co - pred\_test\_co

test\_sqrs = test\_res \* test\_res

test\_mse = np.mean(test\_sqrs)

test\_rmse = np.sqrt(test\_mse)

test\_rmse

C:\Users\user\Documents\Figure_1.png

# Prediction on train data

train\_pred = finalmodel.predict(pd.DataFrame(train))

pred\_train\_co = np.exp(train\_pred)

pred\_train\_co

# Model Evaluation on train data

train\_res = train.co - pred\_train\_co

train\_sqrs = train\_res \* train\_res

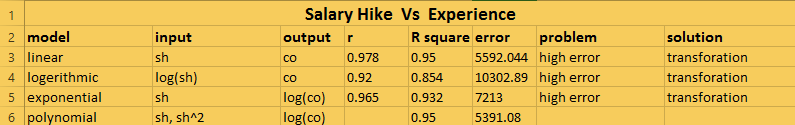
train\_mse = np.mean(train\_sqrs)

train\_rmse = np.sqrt(train\_mse)

train\_rmse

C:\Users\user\Documents\Figure_1.png

**Summary:-**



Polynomial model showing better performance compare to all other model by showing higher “R square value” ( more number of accurate predictions) and lower “error “ value.

**Business advantage:-**

By using this model we can find the minimum salary hike according to year of experience of employess that’s make them happy and stay on with the company and thereby prevent the churn out rate.

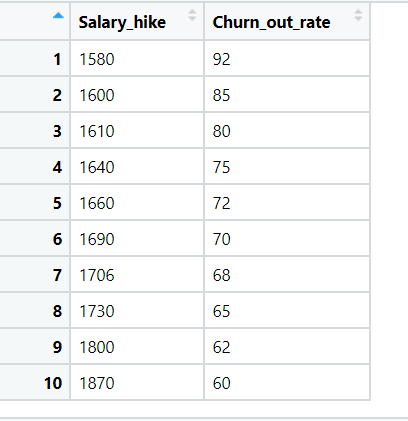
**Problem Statement: -**

A logistics company recorded the time taken for delivery and the time taken for the sorting of the items for delivery. Build a Simple Linear Regression model to find the relationship between delivery time and sorting time with delivery time as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.



**Problem Statement: -**

A certain organization wants an early estimate of their employee churn out rate. So the HR department gathered the data regarding the employee’s salary hike and the churn out rate in a financial year. The analytics team will have to perform an analysis and predict an estimate of employee churn based on the salary hike. Build a Simple Linear Regression model with churn out rate as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.



**Problem Statement: -**

## The head of HR of a certain organization wants to automate their salary hike estimation. The organization consulted an analytics service provider and asked them to build a basic prediction model by providing them with a dataset that contains the data about the number of years of experience and the salary hike given accordingly. Build a Simple Linear Regression model with salary as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.



## **Problem Statement: -**

## A certain university wants to understand the relationship between students’ SAT scores and their GPA. Build a Simple Linear Regression model with GPA as the target variable and record the RMSE and correlation coefficient values for different models.

