File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

SIMPLE LINEAR REGRESSION MODEL

In [1]:
```
#IMPORT LIBRARIES AND PACKAGES
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error

#LOAD THE DATASET
sal_df=pd.read_excel(r'/Users/megha/Python/Data/Salary_SLR.xlsx')
```

In [2]:
```
#TOP 5 COLUMNS
sal_df.head(5)
```

Out[2]:

	Percentage in Grade 10	Salary
0	62.00	270000
1	76.33	200000
2	72.00	240000
3	60.00	250000
4	61.00	180000

In [ ]:
```
#DATA INFO
sal_df.info()
```

In [ ]:
```
#CORRELATION heatmap
```

7:36

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [2]:
```
#TOP 5 COLUMNS
sal_df.head(5)
```

Out[2]:

	Percentage in Grade 10	Salary
0	62.00	270000
1	76.33	200000
2	72.00	240000
3	60.00	250000
4	61.00	180000

In [3]:
```
#DATA INFO
sal_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 2 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Percentage in Grade 10  50 non-null     float64
 1   Salary                 50 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 928.0 bytes
```

In [ ]:
```
#CORRELATION heatmap
viz=sns.heatmap(sal_df[['Percentage in Grade 10','Salary']].corr(),annot=True)
```

1. The correlation value lies between −1.0 and 1.0. The sign indicates whether it is positive or negative correlation.
2. −1.0 indicates a perfect negative correlation, whereas +1.0 indicates perfect positive correlation.

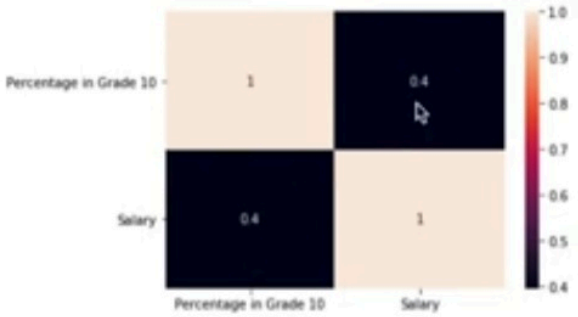
In [ ]:
```
# IDENTIFY THE FEATURE X AND OUTCOME VARIABLE Y IN THE DATAFRAME FOR BUILDING THE MODEL
x=sm.add_constant(sal_df['Percentage in Grade 10'])
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 2 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Percentage in Grade 10  50 non-null     float64
 1   Salary                  50 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 928.0 bytes
```

In [4]:
```
#CORRELATION heatmap
viz=sns.heatmap(sal_df[['Percentage in Grade 10','Salary']].corr(),annot=True)
```

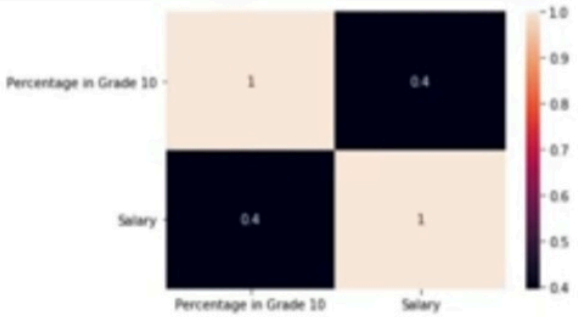


1. The correlation value lies between −1.0 and 1.0. The sign indicates whether it is positive or negative correlation.
2. −1.0 indicates a perfect negative correlation, whereas +1.0 indicates perfect positive correlation.

In [ ]:
```
# IDENTIFY THE FEATURE X AND OUTCOME VARIABLE Y IN THE DATAFRAME FOR BUILDING THE MODEL
x=sm.add_constant(sal_df['Percentage in Grade 10'])
y=sal_df['Salary']
print(x.shape)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown



1. The correlation value lies between −1.0 and 1.0. The sign indicates whether it is positive or negative correlation.
2. −1.0 indicates a perfect negative correlation, whereas +1.0 indicates perfect positive correlation.

In [ ]:
```
# IDENTIFY THE FEATURE X AND OUTCOME VARIABLE Y IN THE DATAFRAME FOR BUILDING THE MODEL
x=sm.add_constant(sal_df['Percentage in Grade 10'])
y=sal_df['Salary']
print(x.shape)
print(y.shape)
x
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [5]:
```
# IDENTIFY THE FEATURE X AND OUTCOME VARIABLE Y IN THE DATAFRAME FOR BUILDING THE MODEL
x=sm.add_constant(sal_df['Percentage in Grade 10'])
y=sal_df['Salary']
print(x.shape)
print(y.shape)
x
```

```
(50, 2)
(50,)
```

Out[5]:

	const	Percentage in Grade 10
0	1.0	62.00
1	1.0	76.33
2	1.0	72.00
3	1.0	60.00
4	1.0	61.00
5	1.0	55.00
6	1.0	70.00
7	1.0	68.00
8	1.0	82.80
9	1.0	59.00
10	1.0	58.00
11	1.0	60.00
12	1.0	66.00
13	1.0	83.00
14	1.0	68.00

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [5]:
```
# IDENTIFY THE FEATURE X AND OUTCOME VARIABLE Y IN THE DATAFRAME FOR BUILDING THE MODEL
x=sm.add_constant(sal_df['Percentage in Grade 10'])
y=sal_df['Salary']
print(x.shape)
print(y.shape)
x
```

```
(50, 2)
(50,)
```

Out[5]:

	const	Percentage in Grade 10
0	1.0	62.00
1	1.0	76.33
2	1.0	72.00
3	1.0	60.00
4	1.0	61.00
5	1.0	55.00
6	1.0	70.00
7	1.0	68.00
8	1.0	82.80
9	1.0	59.00
10	1.0	58.00
11	1.0	60.00
12	1.0	66.00
13	1.0	83.00
14	1.0	68.00
15	1.0	37.33

In [6]:
```
#SPLIT THE DATA INTO TRAIN AND TEST
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.80,test_size=0.2,random_state=100)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(40, 2)
(10, 2)
(40,)
(10,)
```

In [ ]:
```
#FITTING THE MODEL
salary_lm=sm.OLS(y_train,x_train).fit()
#or
#from sklearn.linear_model import LinearRegression
#lm=LinearRegression(fit_intercept=True)
#salary_lm=lm.fit(x_train,y_train)
```

In [ ]:
```
#lm.coef_
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

component

In [6]:
```
#SPLIT THE DATA INTO TRAIN AND TEST
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.80,test_size=0.2,random_state=100)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(40, 2)
(10, 2)
(40,)
(10,)
```

In [10]:
```
#FITTING THE MODEL
salary_lm=sm.OLS(y_train,x_train).fit()
#or
#from sklearn.linear_model import LinearRegression
#lm=LinearRegression(fit_intercept=True)
#salary_lm=lm.fit(x_train,y_train)
```

In [9]:
```
#lm.coef_
#lm.intercept_
```

Out[9]: 30587.2856515233

In [ ]:
```
#PRINT THE PARAMETERS AND INTERPRET THEM
print(salary_lm.params)
```

In [ ]:
```
#SUMMARY
salary_lm.summary2()
```

In [ ]:
```
#PREDICT VALUES TEST DATASET
y_pred_test=salary_lm.predict(x_test)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Out[9]: 30587.2856515233

In [11]:
```
#PRINT THE PARAMETERS AND INTERPRET THEM
print(salary_lm.params)
```

```
const                      30587.285652
Percentage in Grade 10      3560.587383
dtype: float64
```

In [13]:
```
#SUMMARY
salary_lm.summary2()
```

Out[13]:

Model:	OLS	Adj. R-squared:	0.190
Dependent Variable:	Salary	AIC:	1008.8680
Date:	2020-11-02 22:16	BIC:	1012.2458
No. Observations:	40	Log-Likelihood:	-502.43
Df Model:	1	F-statistic:	10.16
Df Residuals:	38	Prob (F-statistic):	0.00287
R-squared:	0.211	Scale:	5.0121e+09

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	30587.2857	71869.4497	0.4256	0.6728	-114904.8089	176079.3802
Percentage in Grade 10	3560.5874	1116.9258	3.1878	0.0029	1299.4892	5821.6855

Omnibus:	2.048	Durbin-Watson:	2.611
Prob(Omnibus):	0.359	Jarque-Bera (JB):	1.724
Skew:	0.369	Prob(JB):	0.422
Kurtosis:	2.300	Condition No.:	413

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
(40,)
(10,)
```

In [10]:
```
#FITTING THE MODEL
salary_lm=sm.OLS(y_train,x_train).fit()
#or
#from sklearn.linear_model import LinearRegression
#lm=LinearRegression(fit_intercept=True)
#salary_lm=lm.fit(x_train,y_train)
```

In [9]:
```
#lm.coef_
#lm.intercept_
```

Out[9]:
```
30587.2856515233
```

In [11]:
```
#PRINT THE PARAMETERS AND INTERPRET THEM
print(salary_lm.params)
```

```
const                      30587.285652
Percentage in Grade 10      3560.587383
dtype: float64
```

In [ ]:
```
#SUMMARY
salary_lm.summary2()
```

In [ ]:
```
#PREDICT VALUES TEST DATASET
y_pred_test=salary_lm.predict(x_test)
y_pred_train=salary_lm.predict(x_train)
```

MEASURING ACCURACY OF PREDICTIONS

In [ ]:
```
#R sqare for train data
np.abs(r2_score(y_train,y_pred_train))
```

In [ ]:
```
# Calculating Root Mean Squared Error
```

SUBSCRIBE

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Prob(Omnibus):	0.359	Jarque-Bera (JB):	1.724
Skew:	0.369	Prob(JB):	0.422
Kurtosis:	2.300	Condition No.:	413

In [ ]:
```
#PREDICT VALUES TEST DATASET
y_pred_test=salary_lm.predict(x_test)
y_pred_train=salary_lm.predict(x_train)
```

MEASURING ACCURACY OF PREDICTIONS

In [ ]:
```
#R sqare for train data
np.abs(r2_score(y_train,y_pred_train))
```

In [ ]:
```
# Calculating Root Mean Squared Error
np.sqrt(mean_squared_error(y_train,y_pred_train))
```

In [ ]:
```
#scatter plot for train data
plt.figure(figsize=(15,10))
plt.scatter(y_train,y_pred_train,c='green')
plt.plot([y_train.min(),y_train.max()],[y_train.min(),y_train.max()],'k--',c='blue',lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

In [ ]:
```
pred_y_df=pd.DataFrame({'grade 10 perc':x_train['Percentage in Grade 10'],'original sal':y_train,'pred_y':y_pred_tra
pred_y_df[0:20]
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	30587.2857	71869.4497	0.4256	0.6728	-114904.8089	176079.3802
Percentage in Grade 10	3560.5874	1116.9258	3.1878	0.0029	1299.4892	5821.6855

Omnibus:	2.048	Durbin-Watson:	2.611
Prob(Omnibus):	0.359	Jarque-Bera (JB):	1.724
Skew:	0.369	Prob(JB):	0.422
Kurtosis:	2.300	Condition No.:	413

In [27]:
```
#PREDICT VALUES TEST DATASET
y_pred_test=salary_lm.predict(x_test)
y_pred_train=salary_lm.predict(x_train)
```

In [30]:
```
#R sqare for train data
np.abs(r2_score(y_test,y_pred_test))
```

Out[30]: 0.15664584974230378

MEASURING ACCURACY OF PREDICTIONS

In [31]:
```
# Calculating Root Mean Squared Error
np.sqrt(mean_squared_error(y_train,y_pred_train))
```

Out[31]: 69003.44815808642

In [ ]:
```
#scatter plot for train data
plt.figure(figsize=(15,10))
plt.scatter(y_train,y_pred_train,c='green')
plt.plot([y_train.min(),y_train.max()],[y_train.min(),y_train.max()],'k--',c='blue',lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [27]:
```
#PREDICT VALUES TEST DATASET
y_pred_test=salary_lm.predict(x_test)
y_pred_train=salary_lm.predict(x_train)
```

In [30]:
```
#R sqare for train data
np.abs(r2_score(y_test,y_pred_test))
```

Out[30]: 0.15664584974230378

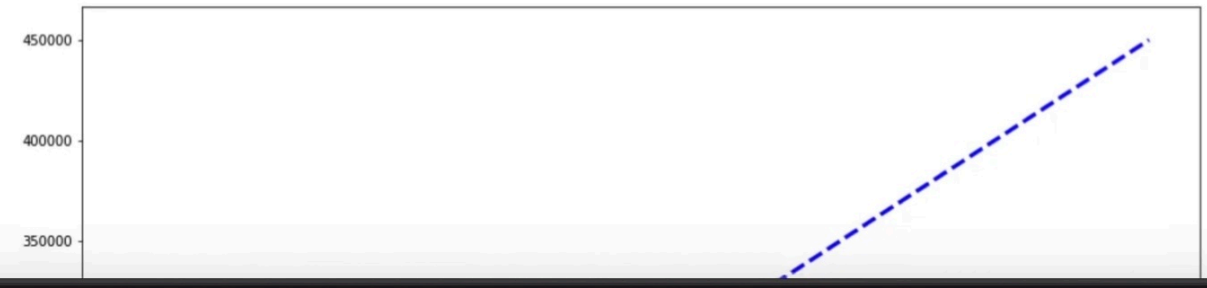
MEASURING ACCURACY OF PREDICTIONS

In [32]:
```
# Calculating Root Mean Squared Error
np.sqrt(mean_squared_error(y_test,y_pred_test))
```

Out[32]: 73458.04348346894

In [33]:
```
#scatter plot for train data
plt.figure(figsize=(15,10))
plt.scatter(y_train,y_pred_train,c='green')
plt.plot([y_train.min(),y_train.max()],[y_train.min(),y_train.max()],'k--',c='blue',lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

Out[33]: Text(0, 0.5, 'Predicted')

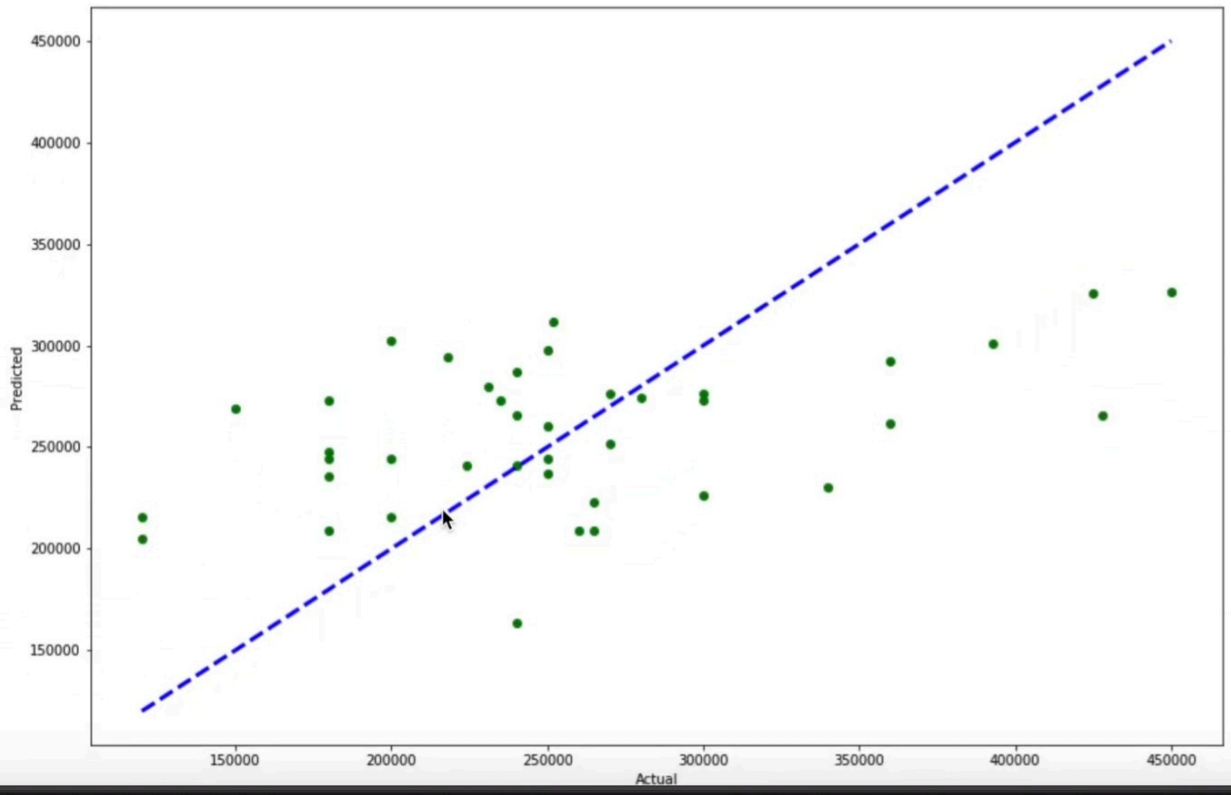


File Edit View Insert Cell Kernel Widgets Help Trusted Python

Run Code

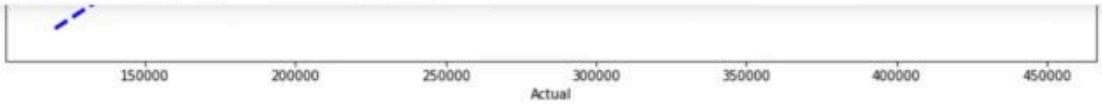
```
plt.ylabel('Predicted')
```

Out[33]: Text(0, 0.5, 'Predicted')



File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code



In [35]:
```
pred_y_df=pd.DataFrame({'grade 10 perc':x_train['Percentage in Grade 10'],'original sal':y_train,'pred_y':y_pred_tra
pred_y_df[0:20]
```

Out[35]:

	grade 10 perc	original sal	pred_y	difference
0	62.00	270000	251343.703389	18656.296611
11	60.00	180000	244222.528623	-64222.528623
18	70.00	231000	279828.402452	-48828.402452
45	57.58	180000	235605.907157	-55605.907157
38	54.00	265000	222859.004326	42140.995674
25	64.60	250000	260601.230584	-10601.230584
26	50.00	180000	208616.654795	-28616.654795
35	56.00	340000	229980.179092	110019.820908
17	68.40	280000	274131.462639	5868.537361
7	68.00	235000	272707.227686	-37707.227686
47	69.00	270000	276267.815069	-6267.815069
31	60.00	200000	244222.528623	-44222.528623
32	55.00	300000	226419.591709	73580.408291
19	59.00	224000	240661.941240	-16661.941240
21	50.00	260000	208616.654795	51383.345205
13	83.00	450000	326116.038429	123883.961571
1	76.33	200000	302366.920585	-102366.920585