

IN LAB

LAB TASK 1 :

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

from sklearn.metrics import classification_report, confusion_matrix

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

import numpy as np

from sklearn import linear_model

from sklearn import preprocessing

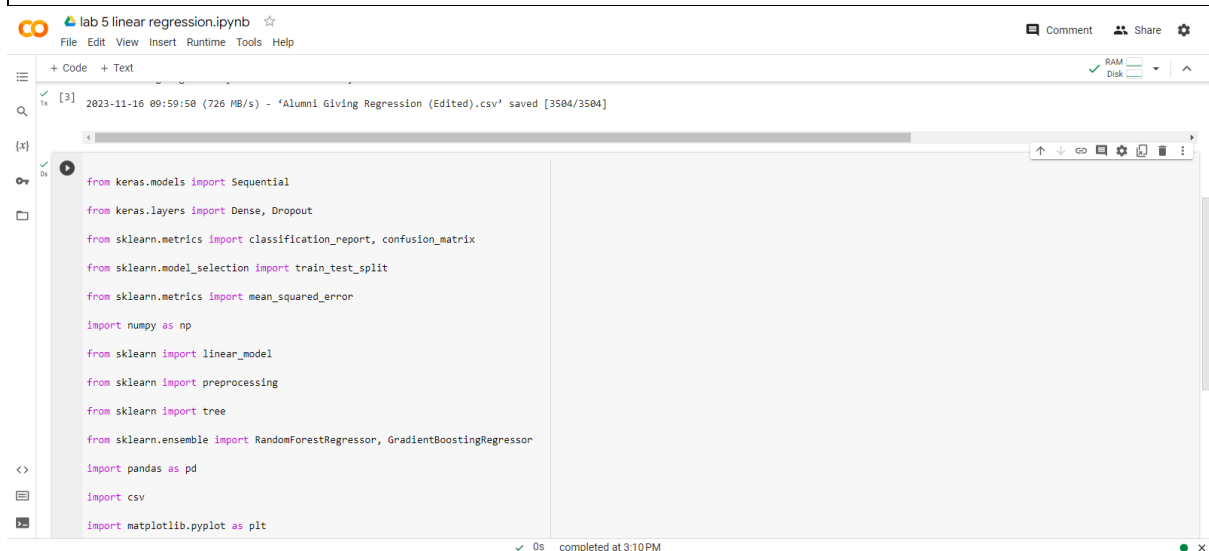
from sklearn import tree

from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor

import pandas as pd

import csv

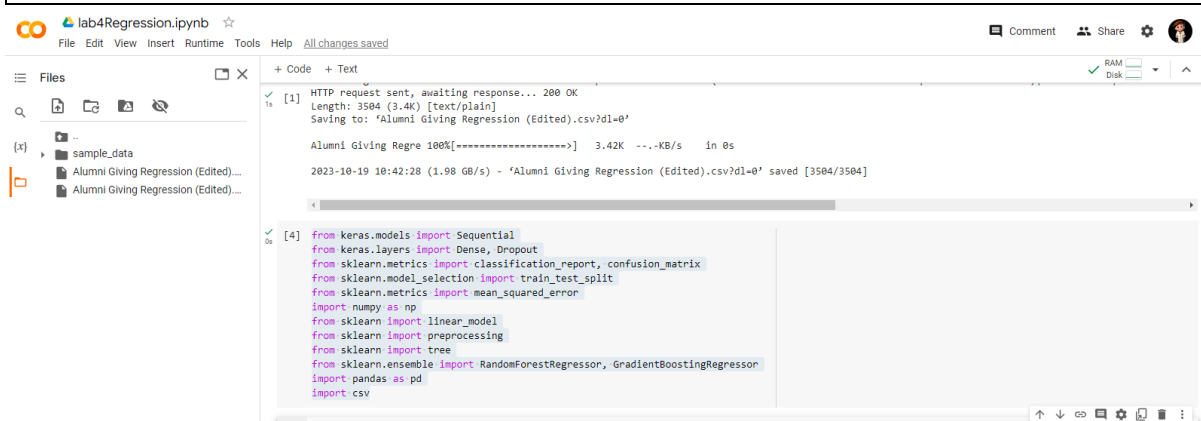
import matplotlib.pyplot as plt
```



LAB TASK 2:

Importing important file into the google collab using python commands.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
import pandas as pd
import csv
```



LAB TASK 3:

Displaying some rows and columns of data which are at the starting of the file.

```
np.random.seed(7)
df = pd.read_csv("Alumni Giving Regression (Edited) (1).csv",
delimiter = ",")
dd_df_1 = df.head()

print(dd_df_1)
```

The Jupyter Notebook interface shows the following code and output:

```
[15]: import numpy as np

from sklearn import linear_model

from sklearn import preprocessing

from sklearn import tree

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

import pandas as pd

import csv

import matplotlib.pyplot as plt
```

```
np.random.seed(7)
df = pd.read_csv("Alumni Giving Regression (Edited) (1).csv", delimiter=",")
dd_df_1 = df.head()
print(dd_df_1)
```

	A	B	C	D	E	F
0	24	0.42	0.16	0.59	0.81	0.08
1	19	0.49	0.04	0.37	0.69	0.11
2	18	0.24	0.17	0.66	0.87	0.31
3	8	0.74	0.00	0.81	0.88	0.11
4	8	0.95	0.00	0.86	0.92	0.28

```
dd_df_1.describe()
```

LAB TASK 4:

This command will give us the whole analysis of the file with some parameters like count, mean, minimum, maximum etc

```
dd_df_1.describe()
```

The Jupyter Notebook interface shows the output of the `dd_df_1.describe()` command:

```
dd_df_1.describe()
```

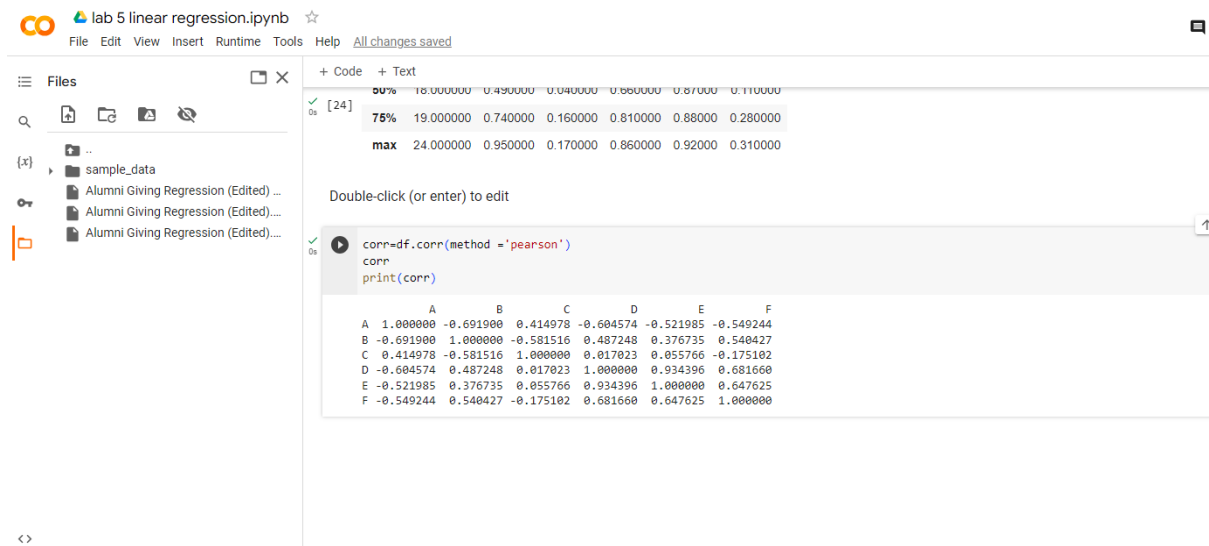
	A	B	C	D	E	F
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	15.400000	0.568000	0.074000	0.658000	0.834000	0.178000
std	7.127412	0.278873	0.084735	0.194602	0.08961	0.108028
min	8.000000	0.240000	0.000000	0.370000	0.690000	0.080000
25%	8.000000	0.420000	0.000000	0.590000	0.810000	0.110000
50%	18.000000	0.490000	0.040000	0.660000	0.870000	0.110000
75%	19.000000	0.740000	0.160000	0.810000	0.880000	0.280000
max	24.000000	0.950000	0.170000	0.860000	0.920000	0.310000

Double-click (or enter) to edit

LAB TASK 5:

Here we calculate the Pearson correlation coefficients between the columns (variables) in a DataFrame and store the resulting correlation matrix in the variable corr.

```
corr=df.corr(method='pearson')
corr
print(corr)
```



LAB TASK 6:

These lines of code prepare data for regression modeling by selecting the independent features (predictors) and the target variable

```
Y_POSITION = 5
model_1_features = [i for i in range(0,Y_POSITION)]
X = df.iloc[:,model_1_features]
Y = df.iloc[:,Y_POSITION]
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.20, random_state=2020)
```

The screenshot shows a Jupyter Notebook environment. On the left, a file explorer displays a directory structure with 'sample_data' and three files named 'Alumni Giving Regression (Edited) ...'. The main workspace is divided into two panes. The top pane shows a data table with 6 columns (A-F) and 6 rows of numerical data. The bottom pane contains the following Python code:

```
Y_POSITION = 5
model_1_features = [i for i in range(0,Y_POSITION)]
X = df.iloc[:,model_1_features]
Y = df.iloc[:,Y_POSITION]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=2020)
```

LAB TASK 6:

This is the main model. In this code, a linear regression model is trained using the training data and evaluated on both the training and testing datasets. It calculates and reports the Root Mean Squared Error (RMSE), a measure of the model's predictive performance. Additionally, it extracts and displays the coefficients of the features, indicating their influence on the target variable in the linear regression model.

```
modell = linear_model.LinearRegression()
modell.fit(X_train, y_train)
y_pred_train1 = modell.predict(X_train)
print("Regression")
print("=====")
RMSE_train1 = mean_squared_error(y_train,y_pred_train1)
print("Regression Train set: RMSE {}".format(RMSE_train1))
print("=====")
y_pred1 = modell.predict(X_test)
RMSE_test1 = mean_squared_error(y_test, y_pred1)
print("Regression Test set: RMSE {}".format(RMSE_test1))
print("=====")

coef_dict = {}
for coef, feat in zip(modell.coef_, model_1_features):
    coef_dict[df.columns[feat]] = coef
print(coef_dict)
```



```

lab 5 linear regression.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
[26] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=2020)

model1 = linear_model.LinearRegression()
model1.fit(X_train, y_train)
y_pred_train1 = model1.predict(X_train)
print("Regression")
print("=====")
RMSE_train1 = mean_squared_error(y_train, y_pred_train1)
print("Regression Train set: RMSE {}".format(RMSE_train1))
print("=====")
y_pred1 = model1.predict(X_test)
RMSE_test1 = mean_squared_error(y_test, y_pred1)
print("Regression Test set: RMSE {}".format(RMSE_test1))
print("=====")

coef_dict = {}
for coef, feat in zip(model1.coef_, model_1.features):
    coef_dict[df.columns[feat]] = coef
print(coef_dict)

Regression
=====
Regression Train set: RMSE 0.002761693322289229
=====
Regression Test set: RMSE 0.004269824026356377
=====
{'A': -0.0009337757382416938, 'B': 0.16012156898162943, 'C': -0.044160015425349614, 'D': 0.15217907817100407, 'E': 0.17539950794101047}

```

Lab task 7

```

x_values = np.arange(len(y_test))

plt.scatter(x_values, y_test, color='red', label='Actual')

#Scatter plot of predicted values (y_pred) in green

plt.scatter(x_values, y_pred1, color='green', label='Predicted')

plt.xlabel('Index or Sequence of Values')

plt.ylabel('Values')

plt.title("Actual vs Predicted Values")

plt.legend()

plt.show()

```

