# In-Lab

## Task 1:

## Linear Regression:

First, we load the necessary libraries which we require in this given section in our case we use the linear regression model because we want to predict the discrete values rather than the true or false for that prediction, we will use the logistic regression.

```
# lab task 1
# Importing libraries needed
# Note that keras is generally used for deep learning as well
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
```

## Task 2:

Load and show the data set in the given below section:

```
#load and show the data set
```

```
# lab task 2 np.random.seed(7) df =pd.read_csv("Alumni
Giving Regression (Edited).csv" , delimiter=",") dd_df_1
=df.head()
print(dd_df_1)
```

## Output:

| | 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 |

## Task 3:

In this section, we will show the statistics of the dataset and it will show the following parameters as given below:

- count: Number of non-null values.
- Mean: Average value.
- std: Standard deviation.
- min: Minimum value.
- 25%: First quartile (25th percentile).
- 50% (median): Median value (50th percentile).
- 75%: Third quartile (75th percentile). • max: Maximum value.

```
# It will show the statistics of the data set
df.describe()
```

## Output:

| | | | | | | |
|---|---|---|---|---|---|---|
| count | 123.000000 | 123.000000 | 123.000000 | 123.000000 | 123.000000 | 123.000000 |
| mean | 17.772358 | 0.403659 | 0.136260 | 0.645203 | 0.841138 | 0.141789 |
| std | 4.517385 | 0.133897 | 0.060101 | 0.169794 | 0.083942 | 0.080674 |
| min | 6.000000 | 0.140000 | 0.000000 | 0.260000 | 0.580000 | 0.020000 |
| 25% | 16.000000 | 0.320000 | 0.095000 | 0.505000 | 0.780000 | 0.080000 |
| 50% | 18.000000 | 0.380000 | 0.130000 | 0.640000 | 0.840000 | 0.130000 |
| 75% | 20.000000 | 0.460000 | 0.180000 | 0.785000 | 0.910000 | 0.170000 |
| max | 31.000000 | 0.950000 | 0.310000 | 0.960000 | 0.980000 | 0.410000 |

## Step 4:

In this section we will calculate the correlation as given below:

```
# In lab task 4  # correlation calculation
corr=df.corr(method ='pearson') corr
print(corr)
```

## Output:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 1.000000 | -0.691900 | 0.414978 | -0.604574 | -0.521985 | -0.549244 |
| B | -0.691900 | 1.000000 | -0.581516 | 0.487248 | 0.376735 | 0.540427 |
| C | 0.414978 | -0.581516 | 1.000000 | 0.017023 | 0.055766 | -0.175102 |
| D | -0.604574 | 0.487248 | 0.017023 | 1.000000 | 0.934396 | 0.681660 |
| E | -0.521985 | 0.376735 | 0.055766 | 0.934396 | 1.000000 | 0.647625 |
| F | -0.549244 | 0.540427 | -0.175102 | 0.681660 | 0.647625 | 1.000000 |

## Task 5:

The code establishes a target variable's column position as 5, creates a list of feature indices up to the target column, selects features and the target variable from a data frame accordingly, and then splits the data into training and testing sets. The training set comprises features for training, while the testing set holds features for testing, with a 20% proportion allocated for testing, and a random seed set for reproducibility.

```
# In lab task 5
# Define a constant Y_POSITION with a value of 5, which represents the
column position for the target variable (Y).
Y_POSITION = 5

# Create a list model_1_features containing column indices from 0 up to
(Y_POSITION - 1).
# These indices represent the features used for model 1.
model_1_features = [i for i in range(0, Y_POSITION)]

# Select the feature columns (X) and the target variable (Y) from the
DataFrame (df) using the specified column indices.
X = df.iloc[:, model_1_features]  # X contains the feature columns Y
= df.iloc[:, Y_POSITION]        # Y contains the target variable

# Split the data into training and testing sets using train_test_split.
# X_train: Features for training, X_test: Features for testing
# y_train: Target variable for training, y_test: Target variable for testing #
The test_size parameter specifies the proportion of the data to be used for
testing (in this case, 20%).
# The random_state parameter is set to 2020 for reproducibility of the random
split.
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,
random_state=2020)
```

## Task 6:

Concisely, this code creates a Linear Regression model (model1), trains it on the training data (X_train, y_train), makes predictions on the training data, calculates the Root Mean Squared Error (RMSE) to evaluate its performance, and prints the RMSE value for the training set.

```
# In lab task 6
# Create a Linear Regression model instance named model1. model1
= linear_model.LinearRegression()

# Train the model1 on the training data.
model1.fit(X_train, y_train)

# Use the trained model1 to make predictions on the training data.
y_pred_train1 = model1.predict(X_train)

# Print a header for the output. print("Regression")
print("================================")

# Calculate the Root Mean Squared Error (RMSE) for the predictions on the
training data.
RMSE_train1 = mean_squared_error(y_train, y_pred_train1)

# Print the RMSE value for the training set. print("Regression
Train set: RMSE {}".format(RMSE_train1))
```

## Output:

```
Regression
================================
Regression Train set: RMSE 0.002761693322289229
```

## Task 7:

Briefly, this code first prints separators for visual distinction. It then utilizes the trained Linear Regression model (model1) to make predictions on the testing data and calculates the Root Mean Squared Error (RMSE) for assessing its performance on the testing set. After printing the RMSE value, it adds another separator to separate the RMSE result from the coefficient analysis. In the subsequent part, it constructs a dictionary to store the model's coefficients and associates them with their corresponding feature names. Finally, it prints this dictionary, showing the relationship between feature names and their respective coefficients.

```
# Task 7
# Print a separator to distinguish between different sections of the output.
print("==================================")

# Use the trained model1 to make predictions on the testing data (X_test).
y_pred1 = model1.predict(X_test)

# Calculate the Root Mean Squared Error (RMSE) for the predictions on the
testing data.
RMSE_test1 = mean_squared_error(y_test, y_pred1)

# Print the RMSE value for the testing set.
print("Regression Test set: RMSE {}".format(RMSE_test1))

# Print another separator to separate the RMSE result from the coefficient
analysis.
print("==================================")

# Create an empty dictionary to store the coefficients and their
corresponding feature names. coef_dict = {}

# Iterate through the coefficients and corresponding feature indices.
# Map the coefficients to the feature names from the DataFrame's columns. for
coef, feat in zip(model1.coef_, model_1_features):
    coef_dict[df.columns[feat]] = coef

# Print the dictionary that maps feature names to their coefficients.
print(coef_dict)
```

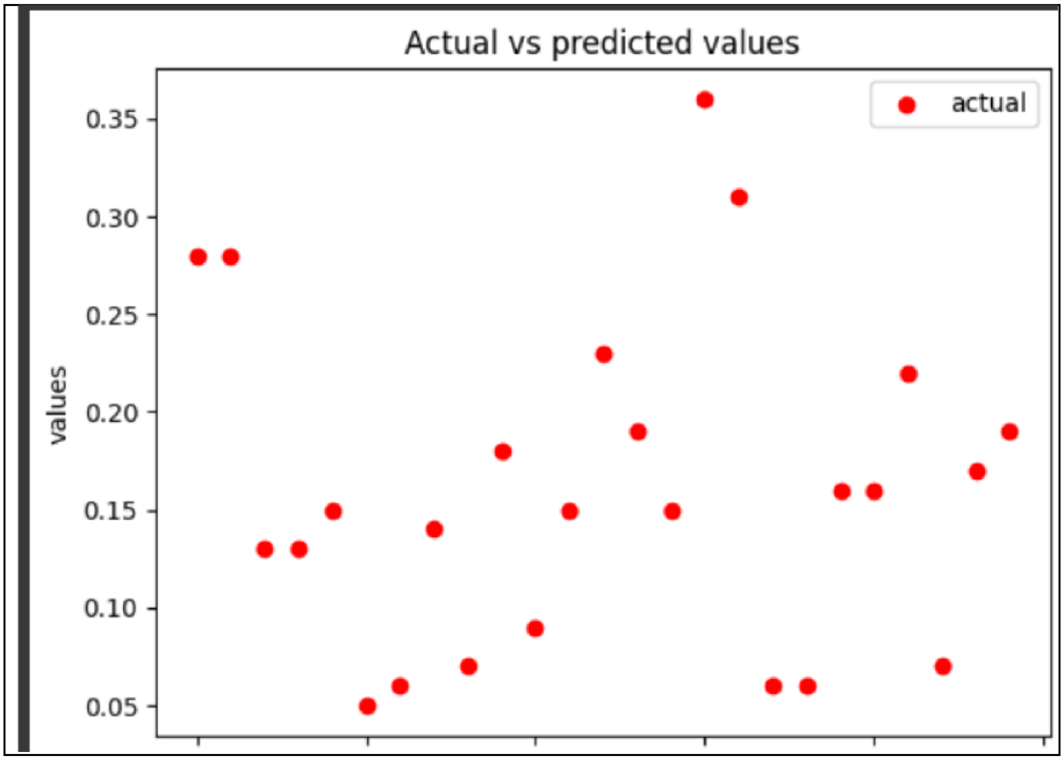
## Output:

```
================================
Regression Test set: RMSE 0.004209824026356377
================================
{'A': -0.0009337757382416938, 'B': 0.16012156890162943, 'C': -0.044160015425349614, 'D
```

## Task 8:

```
#Task 8
x_values = np.arange(len(y_test))
plt.scatter(x_values,y_test,color='red',label='actual')
plt.xlabel("Index or sequence of values")
plt.ylabel("values") plt.title("Actual vs predicted
values") plt.legend()
plt.show()
```

## Output:


Actual vs predicted values
actual
values
0.35
0.30
0.25
0.20
0.15
0.10
0.05