lab:-3

Step:-1 import pandas as pd.

import csv.

fuel = pd.read_csv (fuel_data = csv'')

fuel

Step:-2

fuel.tail()

fuel.shape

df=pd.read_csv("fuel_data.csv'')

df

df.info()

df = df.columns

df.

Step:-3 df.isnull()

Step:-4 import pandas as pd.

import matplotlib.pyplot as plt.

import seaborn as sns.

import numpy as np.

Lab3. Fuel Amount Prediction using Linear Regression

Objectives

In this lab, you will build a Linear Regression model to predict the amount to be spent on fuel (ie., diesel) for your car for a road trip.

Learning Outcomes

After completing this lab, you will

- Acquire data and prepare dataset
- Split dataset for training and testing
- Build LR model, perform training, testing and compute MSE and R2 error values
- Apply normalization methods for feature scaling
- Compare performance with KNN Regressor and SGD Regressor

Business Use Case

Say you're planning a road trip to **Goa** with two of your best friends. You start off in **Trichy** and you know it's going to be a ~20h drive. While your friends are in charge of the party operations you're in charge of the all the logistics involved. You have to plan every detail: the schedule, when to stop and where, make sure you get there on time.

So, what's the first thing you do?. You get yourself a blank sheet of paper and start planning!. First item on your checklist? Budget! It's a 20h — approximately 800 kilometers — fun ride, so a total of 20h on the road. The follow-up question: **How much money should I allocate for diesel?**

This is a very important question. You don't want to stop in the middle of the highway and possibly walk a few miles just because you ran out of diesel ! How much money should you allocate for diesel ?.

You approach this problem with a science-oriented mindset, thinking that there must be a way to estimate the amount of money needed, based on the distance you're traveling.

First, you look at some data. You've been laboriously tracking your car's efficiency for the last year — because who doesn't! — so somewhere in your computer, you have stored in an Excel sheet (***fuel_data.csv***) , as shown below.

drivenKM	fuelAmount
390	3600
403	3705
396.5	3471
383.5	3250.5
321.1	3263.7
391.3	3445.2
386.1	3679
371.8	3744.5
404.3	3809
392.2	3905
386.43	3874
395.2	3910
381	4020.7
372	3622
397	3450.5
407	4179
372.4	3454.2

sns.relplot(x="drivenKM", y="fuel Amount",
data=fuel)

Step:5

data1 = ["drivenKM"]

X = fuel[data1]

data2 = ["fuel Amount"]

y = fuel Amount.

Step:6

print(X)

X.dtypes

print(y)

y.dtypes.

Step:7

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split
(X, y, train_size = 0.8, test_size = 0.2)

X_train, X_test, X_train, y_test.

X_train.shape

X_test.shape

y_train.shape

y_test.shape

375.6	3883.8
399	4235.9

Step1. [Prepare your dataset]. Create ***fuel_data.csv*** file as shown above.

Step2. [Import dataset]. Using Pandas, import ***"fuel_data.csv"*** file and print properties such as head(), shape, columns, type and info.

Step3. [Preprocessing]. Check for missing values (Use isnull() method)

Step4. [Visualize Relationships]. Plot ***relplot*** between "**drivenKM**" and "**fuelAmount**".

Step5. [Prepare X matrix and y vector]. Extract "**drivenKM**" column and store into new dataframe **X**. Similarly, extract "**fuelAmount**" and store into **y**.

Step6. [Examine X and y]. Print X, y, type of X and type of y.

Step7. [Split dataset]. Split dataset into 4 parts using ***train_test_split()*** method, such as *X_train, X_test, y_train* and *y_test*. Use **20%** for test size. Later you can play around with this test size. Print the shape of all 4 parts.

Part-I. Linear Regression Baseline Model

Step8. [Build Model]. Create Linear Regression model and train with ***fit()*** using X_train and y_train values.

Step9. [Predict price for 800 KM]. If I need to travel 800 KM, how much do I need to spend on Diesel?. Are you getting this ouput, **array([6905.64571567]). ?**

Step10. [Predict on entire dataset]. Now, perform prediction using entire **X_test** and store result as **y_pred**.

Step11. [Print Mean Squared Error and R2 Error]. Are you getting output **"MSE: 46181.0"**. Also, print values of model parameters: ***coef_*** and ***intercept_*** values.

Part-II. Linear Regression with Scaling using StandardScaler

Step12. [Normalize X_train and X_test values]. Use **StandardScaler**, scale X_train using **fit_transform()** method and X_test using **transform()** method.

Step13. [Build LR model]. Create a new LR model, fit on *scaled X_train* and predict on *scaled X_test*.

Step14. [Print Mean Squared Error and R2 Error]. What is the output?. MSE reduced or not?. Why?.

Step15. [Plot scatter plot]. Display Scatter Plot between actual y (aka ground truth) vs predicted y values. That is, between ***y_test*** and ***y_pred***.

Part-III. Linear Regression with Scaling using MinMaxScaler and Comparison with KNeighborsRegressor and SGDRegressor

Step16. [Repeat with MinmaxScaler]. Repeat scaling using **MinMaxScaler**, LR model creation, fit, predict and error computation steps.

Step17. [Compare KNN Regressor]. Repeat the above steps for **KNeighborsRegressor** model and compare MSE of LR with KNN Regressor.

Step 8:

```
from sklearn.linear_model.import linear
                                Register Regression
model = linear.Regression()
model.fit(x_train, dy_train)
```

Step 9:

```
n = [['800']]
m = model.predict(n)
m.
```

Step 10:

```
y_pred = model.predict(x_test)
y_pred.
```

Step 11:

```
from.sklearn.metrices.import mean_
                        Squared_Error.
from sklearn.metrices.import r2_score
mse_ln = mean_squared_error(y_test, y_pred)
mse_ln
r2 - score(y_test, y_pred)
```

Step18. [Compare SGD Regressor]. Repeat the above steps for **SGDRegressor** model and compare MSE of LR with SGD Regressor.

Step19. [Select best model]. Tabulate MSE values of LR, KNNR and SGDR and select the model with the lowest MSE.

```
model.coef_
model.intercept_
```

Step:12

```
from sklearn.preprocessing import standard scaler.
scaler = StandardScaler()
SS3 = scaler.fit_transform(x_train)
Print(SS3)
SS5 = scaler.transform(x_test)
print(SS5)
```

Step:13:

```
Model 1 = linear Regression()
model.1.fit(SS3, y_train)
S1_S_pred = model 1.predict(SS5)
S1_y_pred
```

Step:14:

```
mean_squared_error(y_test, S1_y_pred)
r2_score(y_test, S1_y_pred)
```

Step:15:

```
plt.scatter(y_test, y_pred)
```

Step: 16

```
from sklearn.preprocessing import minmax_scalar

mm_scalar = minmaxscalar()

mm_ss = mm_scalar.fit_transform(x_train)

mm_ss

mm_sst = mm_scalar.transform(x_test)

mm_sst

model2 = linear Regression()

model2.fit(mm_ss, y_train)

mms_y_pred = model2.predict(mm_sst)

mms_y_pred

mean_squared_error(y_test, mms_y_pred)

r2_score(y_test, mms_y_pred)
```

Step 17:

```
from sklearn.neighbours import KNeighborsRegressor

m_neig = kNeighbours Regressor(n_neighbours = 5)

m_neig.fit(x, y)

m1_y_pred = m_neig.predict(x)

m1_y_pred

mse = mean_squared_error(y, m1_y_pred)

mse

rg_score(y, m1_y_pred
```

NOTES

Step 15

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
r = make_pipeline(standardScaler(), SGDRegressor(
        (max_iter=1000, tol=1e-3))
r.fit(x,y)
re_y_pred = r.predict(x)
re_y_pred
mse3 = mean_squared_error(y, re_y_pred)
mse3
r2_score(y, re_y_pred)
```

Step 16

```
print("SR Model", mse_1[1])
print("KNR model", mse)
print("SGDR model", mse3)
```