Importing Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Dataset

```
df = pd.read_excel("/content/fake_reg.xlsx")
df.head()
```

	price	feature1	feature2
0	461.527929	999.787558	999.766096
1	548.130011	998.861615	1001.042403
2	410.297162	1000.070267	998.844015
3	540.382220	999.952251	1000.440940
4	546.024553	1000.446011	1000.338531

Understanding Dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   price     1000 non-null   float64
 1   feature1  1000 non-null   float64
                               float64
```

Saving...

```
df.describe()
```

	price	feature1	feature2
count	1000.000000	1000.000000	1000.000000
mean	498.673029	1000.014171	999.979847
std	93.785431	0.974018	0.948330
min	223.346793	997.058347	996.995651
25%	433.025732	999.332068	999.316106
50%	502.382117	1000.009915	1000.002243
75%	564.921588	1000.637580	1000.645380
max	774.407854	1003.207934	1002.666308

Standardization

```
from sklearn.preprocessing import StandardScaler
SS = StandardScaler()
X = df.iloc[:,1:]
X = SS.fit_transform(X)
X
```

```
array([[-0.23277507, -0.22551032],
       [-1.18389305,  1.12100994],
       [ 0.05762089, -1.19831824],
       ...,
       [ 1.47655837, -1.19452978],
       [ 0.77742953, -1.49494963],
       [-0.80318737,  1.55251439]])
```

```
Y = df.price
Y
```

```
0      461.527929
1      548.130011
2      410.297162
3      540.382220
4      546.024553
          ...
995    476.526078
996    457.313186
997    456.720992
998    403.315576
999    599.367093
Name: price, Length: 1000, dtype: float64
```

Splitting the data

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=1)
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

Applying ANN Modelling

```
# step1 :- initialize the Model
ann = Sequential()

# step2 :- Add Layers into model
ann.add( Dense(units = 100, activation = "relu") )
ann.add( Dense(units = 100, activation = "relu") )

ann.add( Dense(units = 1) )

# step3 :- Establihing connection
ann.compile(optimizer='adam', loss = 'mse')

= 30, epochs = 200)

# step5 :- Predict the model
Y_pred = ann.predict(X_test)
```

Saving...

```
Epoch 170/200
24/24 [==============================] - 0s 7ms/step - loss: 24.7773
Epoch 171/200
24/24 [==============================] - 0s 3ms/step - loss: 24.3693
Epoch 172/200
24/24 [==============================] - 0s 3ms/step - loss: 24.2754
Epoch 173/200
24/24 [==============================] - 0s 5ms/step - loss: 24.0348
Epoch 174/200
24/24 [==============================] - 0s 4ms/step - loss: 23.9500
Epoch 175/200
24/24 [==============================] - 0s 5ms/step - loss: 24.2327
Epoch 176/200
24/24 [==============================] - 0s 3ms/step - loss: 24.6234
Epoch 177/200
24/24 [==============================] - 0s 2ms/step - loss: 25.2070
Epoch 178/200
24/24 [==============================] - 0s 2ms/step - loss: 23.9292
```

Evaluation of ANN

```
from sklearn.metrics import r2_score
print(f"R2 --> {r2_score(Y_test,Y_pred)}")
```

```
R2 --> 0.9964141263556662
```

```
from sklearn.metrics import mean_absolute_error,mean_squared_error
print(f"MAE ---> {mean_absolute_error(Y_test,Y_pred)}")
print(f"MSE ---> {mean_squared_error(Y_test,Y_pred)}")
print(f"RMSE --> {np.sqrt(mean_squared_error(Y_test,Y_pred))}")
```

```
MAE ---> 4.336487363515324
MSE ---> 29.77126846202769
RMSE --> 5.456305385700812
```

Applying LinearRegression Model

Saving...

```
earRegression

Y_pred = LR.predict(X_test)
```

```
from sklearn.metrics import r2_score
print(f"R2 --> {r2_score(Y_test,Y_pred)}")
```

```
R2 --> 0.9966246601516173
```

Evaluation

```
from sklearn.metrics import mean_absolute_error,mean_squared_error
print(f"MAE ---> {mean_absolute_error(Y_test,Y_pred)}")
print(f"MSE ---> {mean_squared_error(Y_test,Y_pred)}")
print(f"RMSE --> {np.sqrt(mean_squared_error(Y_test,Y_pred))}")
```

```
MAE ---> 4.2611615519749
MSE ---> 28.023337893002925
RMSE --> 5.293707386416719
```

```
df.ndim
```

```
2
```

Saving...