TASK #1: UNDERSTAND THE PROBLEM STATEMENT

- In this project, we will build a regression model to predict the chance of admission into a particular university based on the student's profile.

- INPUTS (FEATURES):
  - GRE Scores (out of 340)
  - TOEFL Scores (out of 120)
  - University Rating (out of 5)
  - Statement of Purpose (SOP)
  - Letter of Recommendation (LOR) Strength (out of 5)
  - Undergraduate GPA (out of 10)
  - Research Experience (either 0 or 1)

- OUTPUTS:
  - Chance of admission (ranging from 0 to 1)

- Data Source: https://www.kaggle.com/mohansacharya/graduate-admissions
- Photo Credit: https://www.pexels.com/photo/accomplishment-ceremony-education-graduation-267885/

TASK #2: IMPORT LIBRARIES AND DATASET

In [34]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import PIL
!pip3 install --upgrade pip
#from jupyterthemes import jtplot
#jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
```

```
Requirement already satisfied: pip in c:\users\manthenahanvith\anaconda3\lib\site-packages (22.3)
```

In [38]:

```
# read the csv file
admission_df = pd.read_csv(r"C:\Users\MANTHENAHANVITH\Jedi\Downloads\Graduate_Admission_Prediction-master\Graduat
admission_df
```

Out[38]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 9 columns

In [39]:

```
admission_df.head()
```

Out[39]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

In [40]:

```
# Let's drop the serial no.
admission_df.drop('Serial No.', axis=1, inplace=True)
admission_df.head()
```

Out[40]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

TASK #3: PERFORM EXPLORATORY DATA ANALYSIS

In [41]:

```
# checking the null values
admission_df.isnull().sum()
```

Out[41]:

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [42]:

```
# Check the dataframe information
admission_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          500 non-null    int64  
 1   TOEFL Score        500 non-null    int64  
 2   University Rating  500 non-null    int64  
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64  
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

In [43]:

```
# Statistical summary of the dataframe
admission_df.describe()
```

Out[43]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000

25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [44]:

```
# Grouping by University ranking
df_university = admission_df.groupby(by = 'University Rating').mean()
df_university
```

Out[44]:

University Rating	GRE Score	TOEFL Score	SOP	LOR	CGPA	Research	Chance of Admit
1	304.911765	100.205882	1.941176	2.426471	7.798529	0.294118	0.562059
2	309.134921	103.444444	2.682540	2.956349	8.177778	0.293651	0.626111
3	315.030864	106.314815	3.308642	3.401235	8.500123	0.537037	0.702901
4	323.304762	110.961905	4.000000	3.947619	8.936667	0.780952	0.801619
5	327.890411	113.438356	4.479452	4.404110	9.278082	0.876712	0.888082

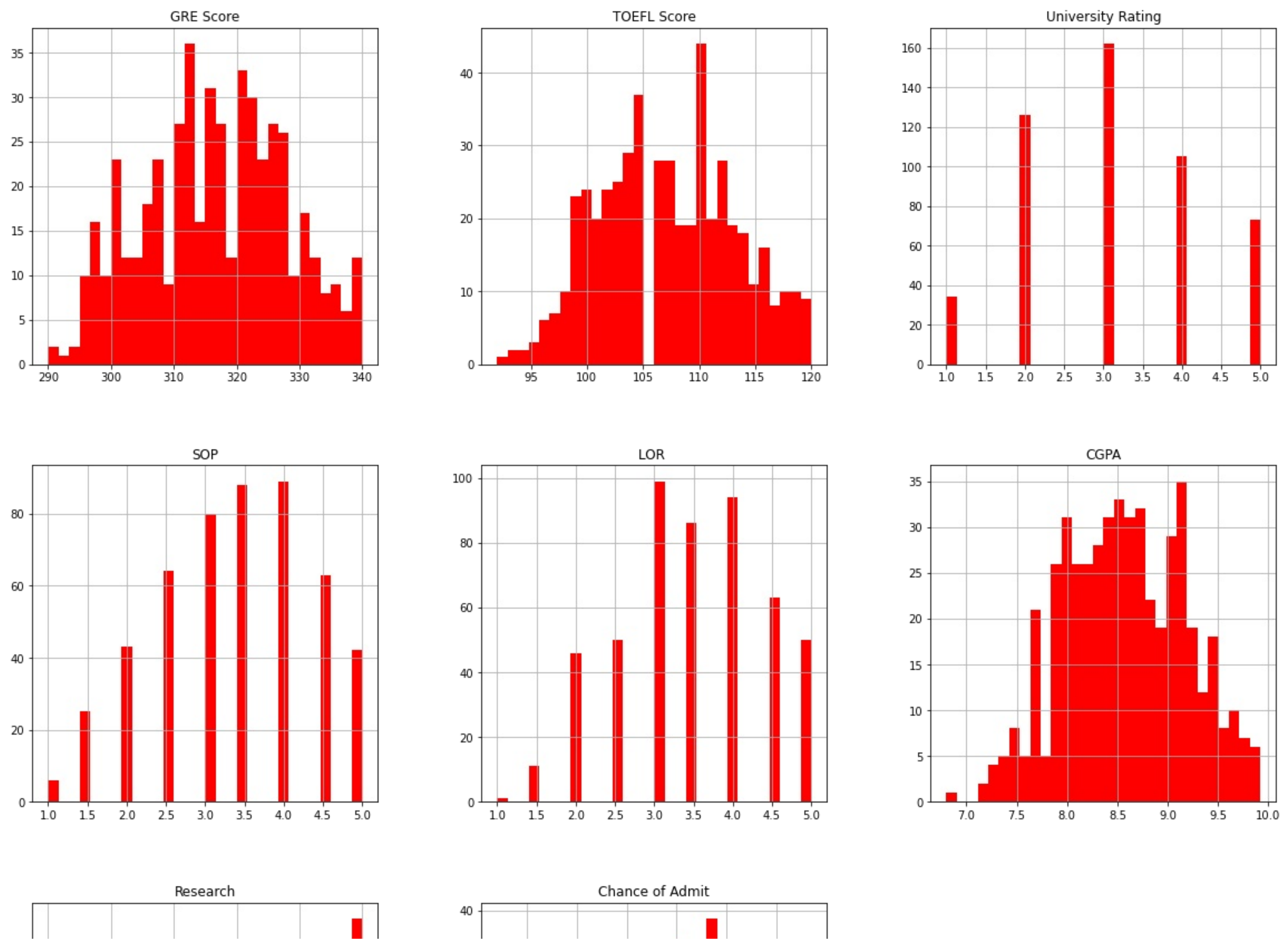
TASK #4: PERFORM DATA VISUALIZATION

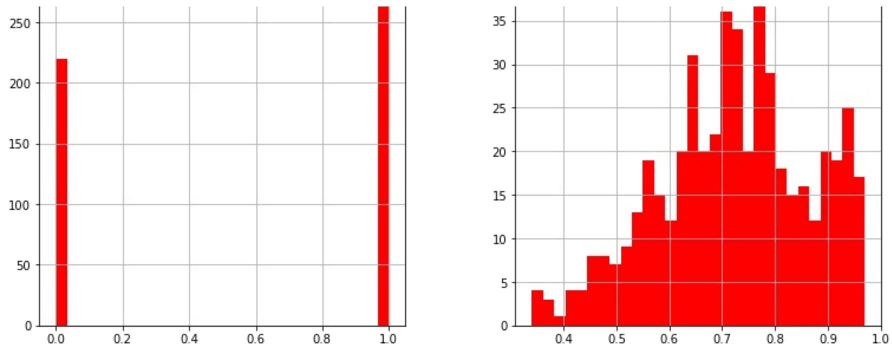
In [45]:

```
admission_df.hist(bins = 30, figsize = (20, 20), color='r')
```

Out[45]:

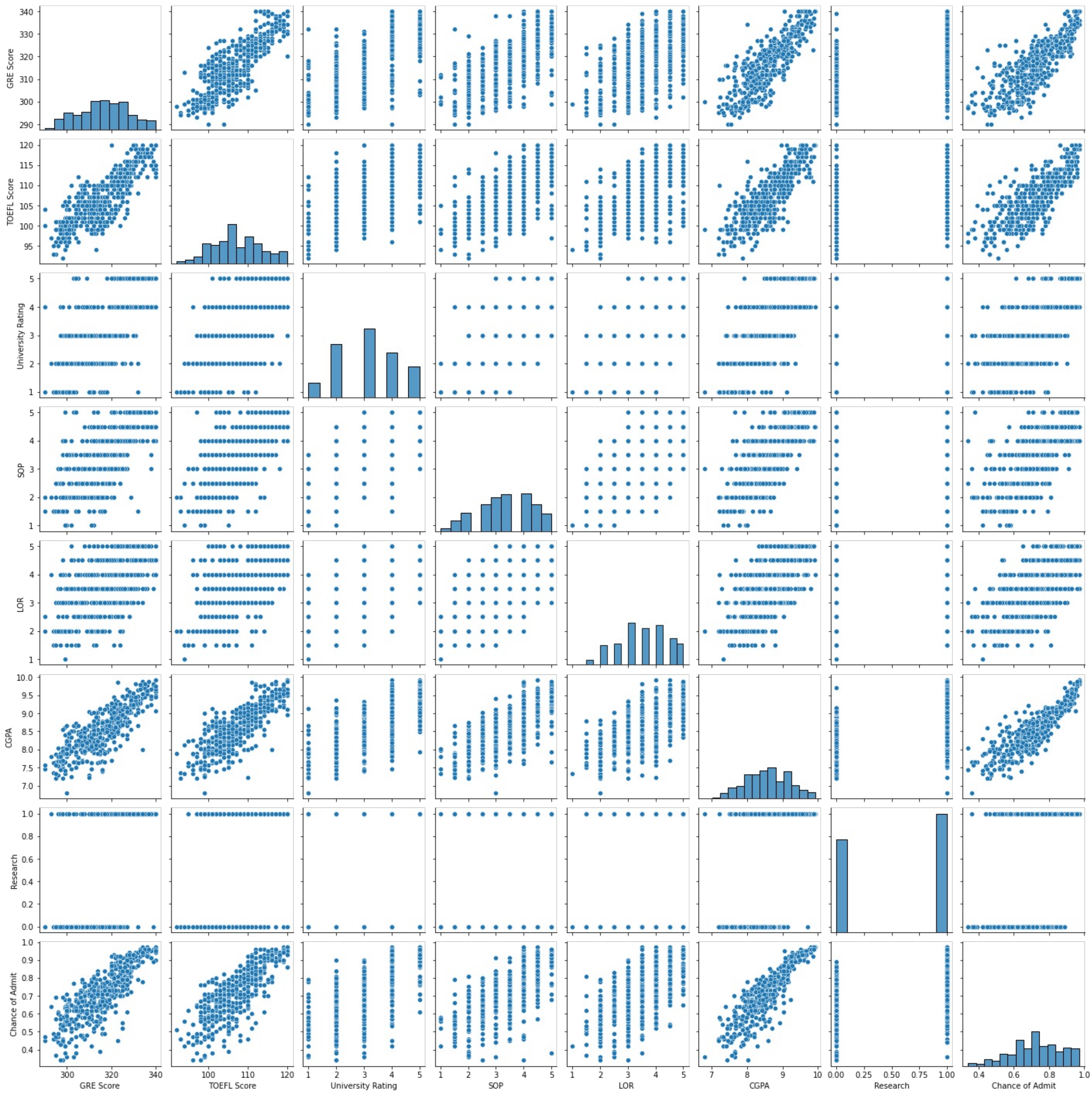
```
array([[<AxesSubplot:title={'center':'GRE Score'}>,
        <AxesSubplot:title={'center':'TOEFL Score'}>,
        <AxesSubplot:title={'center':'University Rating'}>],
       [<AxesSubplot:title={'center':'SOP'}>,
        <AxesSubplot:title={'center':'LOR '}>,
        <AxesSubplot:title={'center':'CGPA'}>],
       [<AxesSubplot:title={'center':'Research'}>,
        <AxesSubplot:title={'center':'Chance of Admit'}>, <AxesSubplot:>]],
      dtype=object)
```



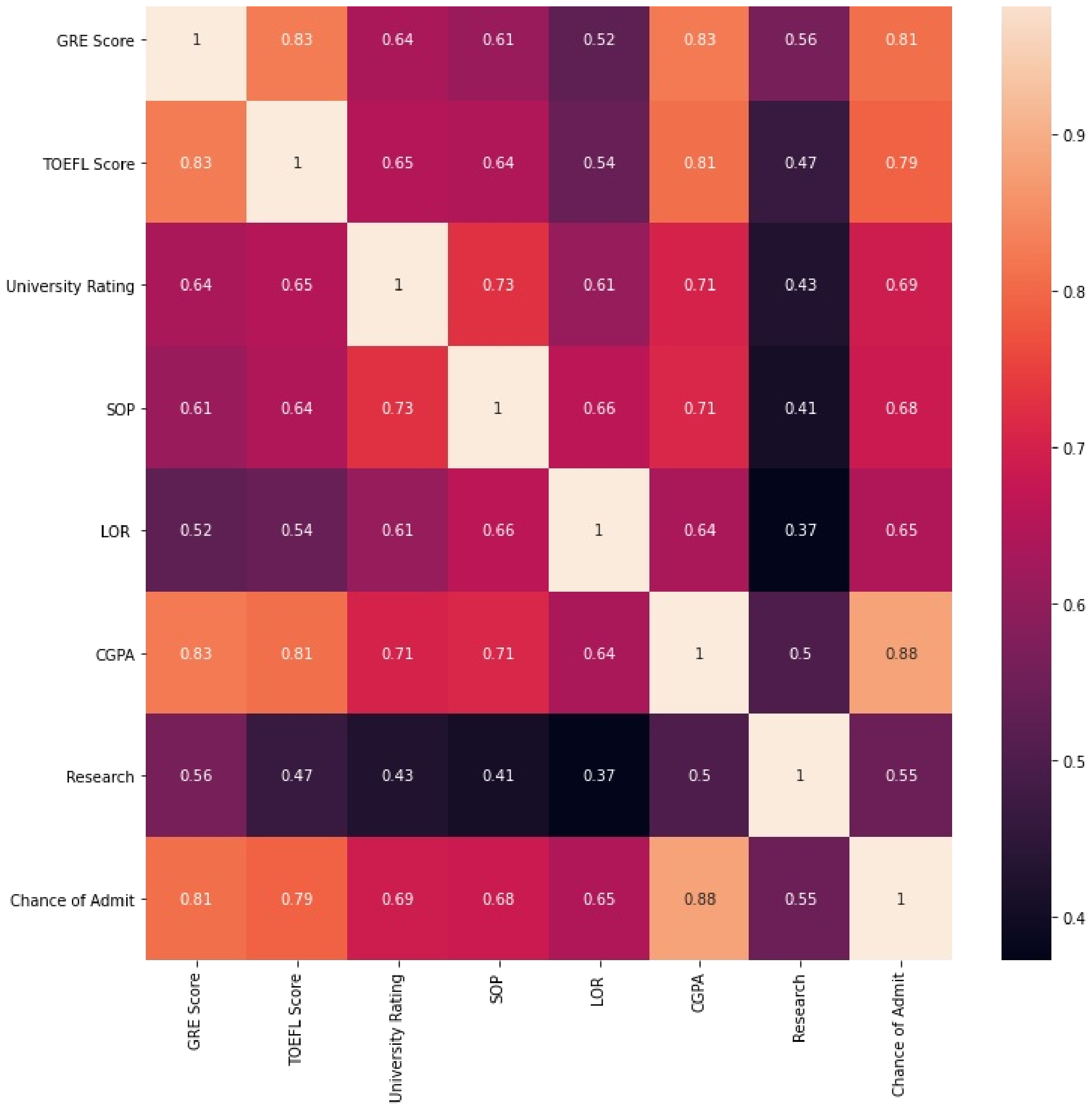


In [46]:
```
sns.pairplot(admission_df)
```

Out[46]: <seaborn.axisgrid.PairGrid at 0x23facab2d30>



In [47]:
```
corr_matrix = admission_df.corr()
plt.figure(figsize=(12,12,))
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



In [ ]:

In [ ]:

TASK #5: CREATE TRAINING AND TESTING DATASET

In [48]:
```
admission_df.columns
```
Out[48]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

In [49]:
```
X = admission_df.drop(columns=['Chance of Admit'])
```

In [50]:
```
y = admission_df['Chance of Admit']
```

In [51]:
```
X.shape
```
Out[51]:
```
(500, 7)
```

In [52]:
```
y.shape
```
Out[52]:
```
(500,)
```

In [53]:

```
X = np.array(X)
y = np.array(y)
```

In [54]:

```
y = y.reshape(-1,1)
```

In [55]:

```
y.shape
```

Out[55]: (500, 1)

In [56]:

```
# scaling the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X = scaler_x.fit_transform(X)
```

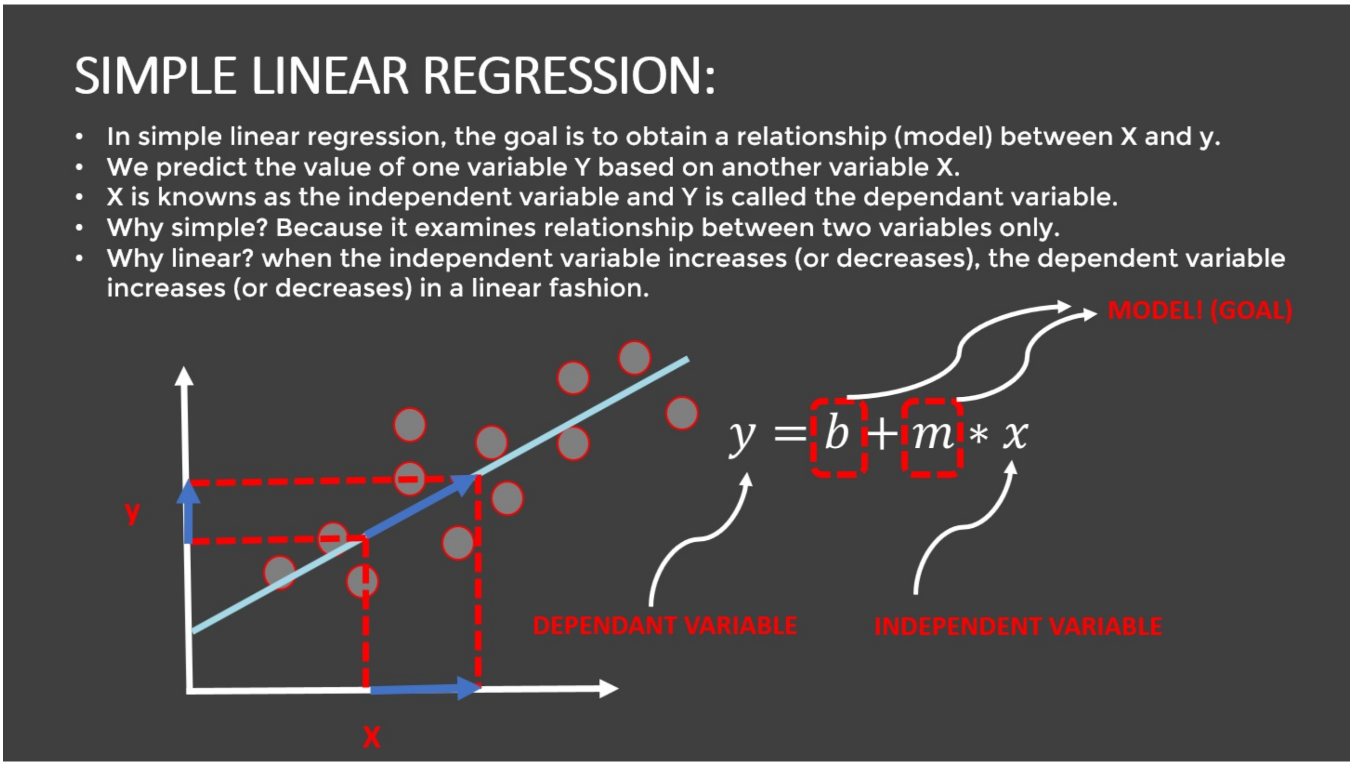
In [57]:

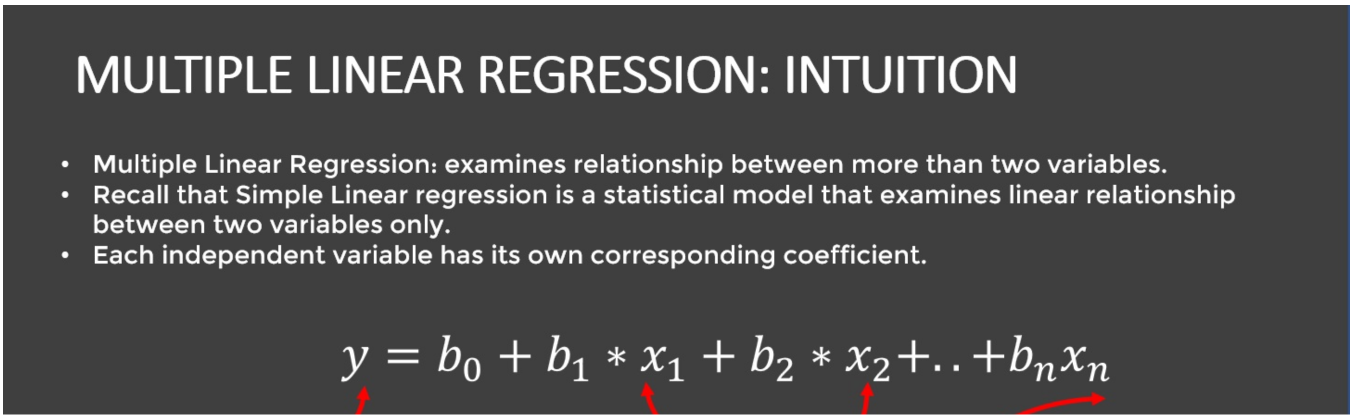
```
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)
```

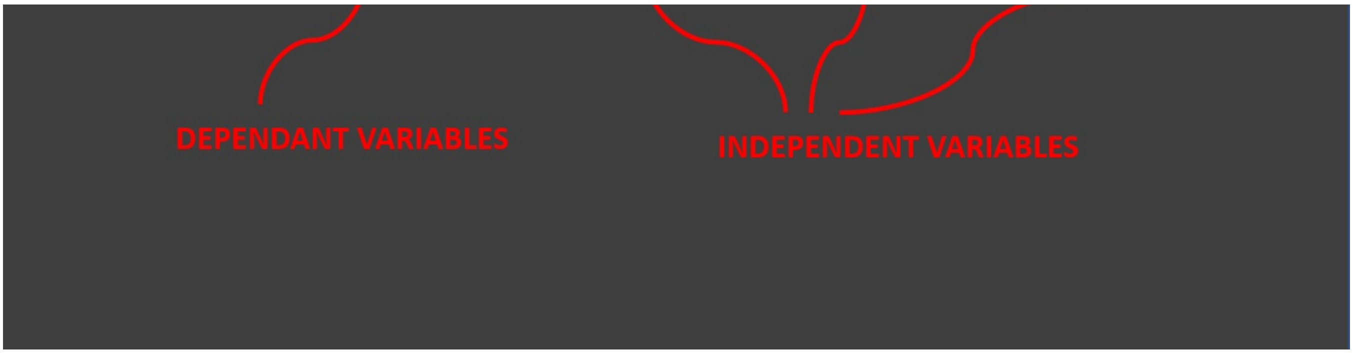
In [58]:

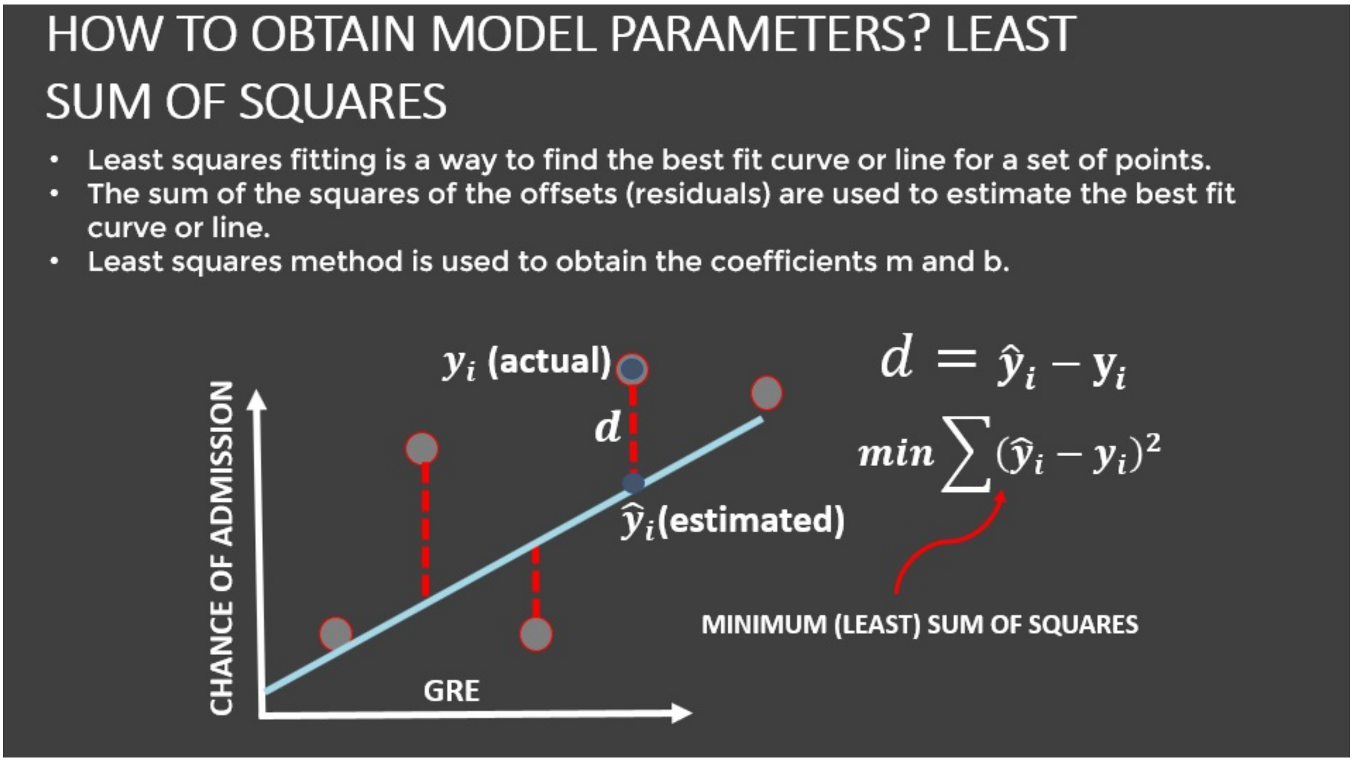
```
# spliting the data in to test and train sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
```

TASK #6: TRAIN AND EVALUATE A LINEAR REGRESSION MODEL









In [59]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score
```

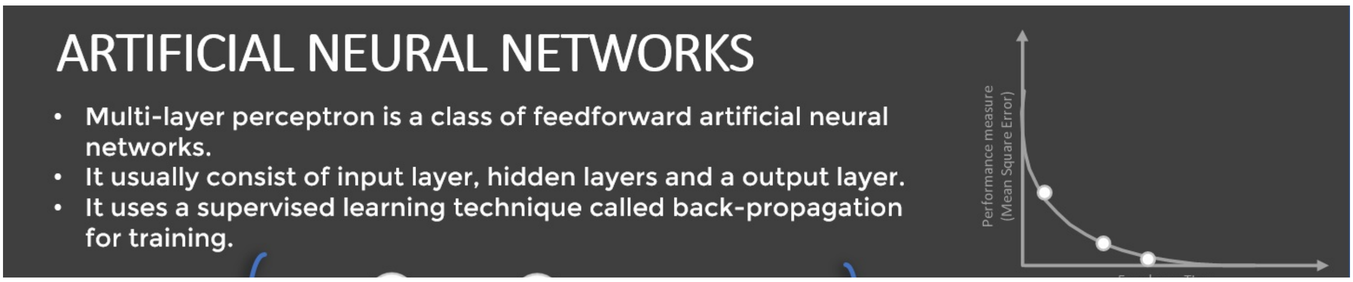
In [60]:
```
linear_regression_model = LinearRegression()
linear_regression_model.fit(X_train, y_train)
```

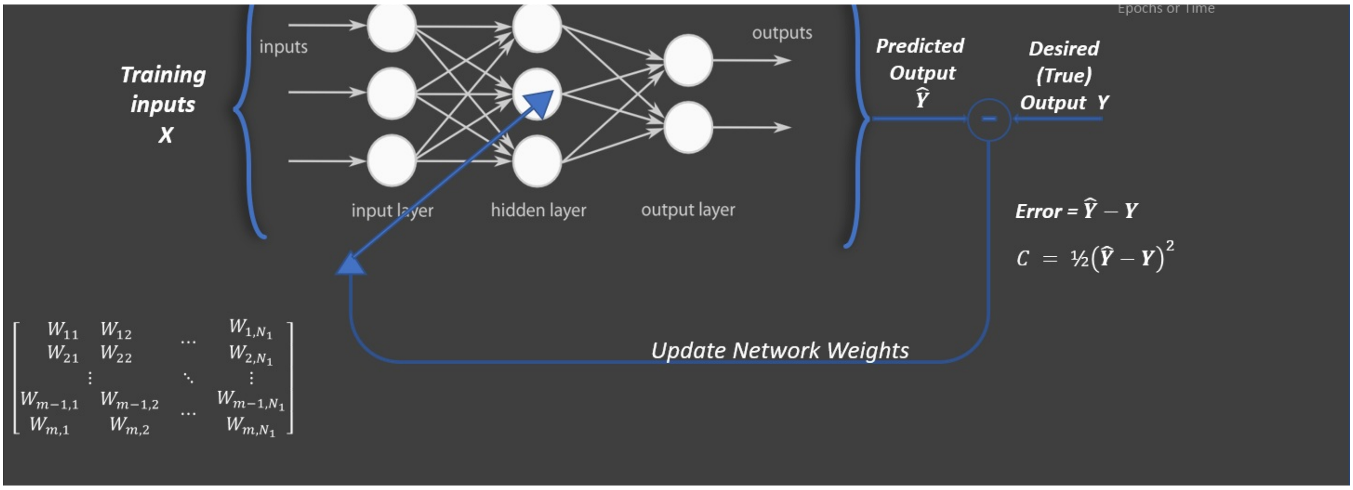
Out[60]:
```
▾ LinearRegression
LinearRegression()
```

In [61]:
```
accuracy_LinearRegression = linear_regression_model.score(X_test, y_test)
accuracy_LinearRegression
```

Out[61]: 0.8233341459249034

TASK #7: TRAIN AND EVALUATE AN ARTIFICIAL NEURAL NETWORK





In [62]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
```

In [63]:

```
ANN_model = keras.Sequential()
ANN_model.add(Dense(50, input_dim = 7))
ANN_model.add(Activation('relu'))
ANN_model.add(Dense(150))
ANN_model.add(Activation('relu'))
ANN_model.add(Dropout(0.5))
ANN_model.add(Dense(150))
ANN_model.add(Activation('relu'))
ANN_model.add(Dropout(0.5))
ANN_model.add(Dense(50))
ANN_model.add(Activation('linear'))
ANN_model.add(Dense(1))
ANN_model.compile(loss = 'mse', optimizer = 'adam')
ANN_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 50)                400

 activation_8 (Activation)   (None, 50)                0

 dense_11 (Dense)            (None, 150)               7650

 activation_9 (Activation)   (None, 150)               0

 dropout_4 (Dropout)         (None, 150)               0

 dense_12 (Dense)            (None, 150)               22650

 activation_10 (Activation)  (None, 150)               0

 dropout_5 (Dropout)         (None, 150)               0

 dense_13 (Dense)            (None, 50)                7550

 activation_11 (Activation)  (None, 50)                0

 dense_14 (Dense)            (None, 1)                 51

=================================================================
Total params: 38,301
Trainable params: 38,301
Non-trainable params: 0
_________________________________________________________________
```

In [64]:

```
ANN_model.compile(optimizer='Adam', loss='mean_squared_error')
```

In [65]:

```
epochs_hist = ANN_model.fit(X_train, y_train, epochs = 100, batch_size = 20, validation_split = 0.2)
```

```
Epoch 1/100
17/17 [==============================] - 1s 15ms/step - loss: 0.6791 - val_loss: 0.1616
```

```
Epoch 2/100
17/17 [==============================] - 0s 4ms/step - loss: 0.4165 - val_loss: 0.1843
Epoch 3/100
17/17 [==============================] - 0s 6ms/step - loss: 0.4046 - val_loss: 0.1017
Epoch 4/100
17/17 [==============================] - 0s 4ms/step - loss: 0.3407 - val_loss: 0.1031
Epoch 5/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2999 - val_loss: 0.1218
Epoch 6/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2925 - val_loss: 0.1083
Epoch 7/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2780 - val_loss: 0.1028
Epoch 8/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2749 - val_loss: 0.0948
Epoch 9/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2844 - val_loss: 0.1031
Epoch 10/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2652 - val_loss: 0.1301
Epoch 11/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2464 - val_loss: 0.1086
Epoch 12/100
17/17 [==============================] - 0s 6ms/step - loss: 0.2490 - val_loss: 0.1194
Epoch 13/100
17/17 [==============================] - 0s 7ms/step - loss: 0.2610 - val_loss: 0.0814
Epoch 14/100
17/17 [==============================] - 0s 6ms/step - loss: 0.2499 - val_loss: 0.1140
Epoch 15/100
17/17 [==============================] - 0s 7ms/step - loss: 0.2342 - val_loss: 0.0910
Epoch 16/100
17/17 [==============================] - 0s 6ms/step - loss: 0.2457 - val_loss: 0.1326
Epoch 17/100
17/17 [==============================] - 0s 7ms/step - loss: 0.2528 - val_loss: 0.0801
Epoch 18/100
17/17 [==============================] - 0s 7ms/step - loss: 0.2515 - val_loss: 0.1571
Epoch 19/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2666 - val_loss: 0.1167
Epoch 20/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2336 - val_loss: 0.0900
Epoch 21/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2363 - val_loss: 0.0835
Epoch 22/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2269 - val_loss: 0.1172
Epoch 23/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2391 - val_loss: 0.1399
Epoch 24/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2166 - val_loss: 0.1034
Epoch 25/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2095 - val_loss: 0.1098
Epoch 26/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2101 - val_loss: 0.0912
Epoch 27/100
17/17 [==============================] - 0s 6ms/step - loss: 0.2164 - val_loss: 0.0837
Epoch 28/100
17/17 [==============================] - 0s 3ms/step - loss: 0.2273 - val_loss: 0.0922
Epoch 29/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1977 - val_loss: 0.1357
Epoch 30/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2159 - val_loss: 0.0898
Epoch 31/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2120 - val_loss: 0.1081
Epoch 32/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2255 - val_loss: 0.0989
Epoch 33/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1965 - val_loss: 0.1037
Epoch 34/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2352 - val_loss: 0.0827
Epoch 35/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2123 - val_loss: 0.1229
Epoch 36/100
17/17 [==============================] - 0s 5ms/step - loss: 0.2045 - val_loss: 0.0830
Epoch 37/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1946 - val_loss: 0.0862
Epoch 38/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1910 - val_loss: 0.1034
Epoch 39/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1942 - val_loss: 0.0861
Epoch 40/100
17/17 [==============================] - 0s 3ms/step - loss: 0.1921 - val_loss: 0.1228
Epoch 41/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2297 - val_loss: 0.1080
Epoch 42/100
17/17 [==============================] - 0s 4ms/step - loss: 0.2071 - val_loss: 0.1053
Epoch 43/100
```

```
17/17 [==============================] - 0s 4ms/step - loss: 0.1911 - val_loss: 0.0929
Epoch 44/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1983 - val_loss: 0.0901
Epoch 45/100
17/17 [==============================] - 0s 6ms/step - loss: 0.1865 - val_loss: 0.1098
Epoch 46/100
17/17 [==============================] - 0s 7ms/step - loss: 0.1974 - val_loss: 0.1072
Epoch 47/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1820 - val_loss: 0.1070
Epoch 48/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1757 - val_loss: 0.0855
Epoch 49/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1766 - val_loss: 0.0851
Epoch 50/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1714 - val_loss: 0.0832
Epoch 51/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1902 - val_loss: 0.0871
Epoch 52/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1744 - val_loss: 0.1225
Epoch 53/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1956 - val_loss: 0.1095
Epoch 54/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1726 - val_loss: 0.1003
Epoch 55/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1692 - val_loss: 0.1038
Epoch 56/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1647 - val_loss: 0.0953
Epoch 57/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1650 - val_loss: 0.1047
Epoch 58/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1701 - val_loss: 0.1234
Epoch 59/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1532 - val_loss: 0.1248
Epoch 60/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1733 - val_loss: 0.1034
Epoch 61/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1591 - val_loss: 0.0978
Epoch 62/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1683 - val_loss: 0.0979
Epoch 63/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1521 - val_loss: 0.1224
Epoch 64/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1576 - val_loss: 0.1146
Epoch 65/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1590 - val_loss: 0.1405
Epoch 66/100
17/17 [==============================] - 0s 3ms/step - loss: 0.1729 - val_loss: 0.0999
Epoch 67/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1616 - val_loss: 0.1128
Epoch 68/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1821 - val_loss: 0.1199
Epoch 69/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1605 - val_loss: 0.1420
Epoch 70/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1767 - val_loss: 0.1032
Epoch 71/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1371 - val_loss: 0.1081
Epoch 72/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1393 - val_loss: 0.1277
Epoch 73/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1735 - val_loss: 0.1044
Epoch 74/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1374 - val_loss: 0.1475
Epoch 75/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1544 - val_loss: 0.1094
Epoch 76/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1583 - val_loss: 0.1107
Epoch 77/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1357 - val_loss: 0.1130
Epoch 78/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1520 - val_loss: 0.1323
Epoch 79/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1389 - val_loss: 0.1029
Epoch 80/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1505 - val_loss: 0.0963
Epoch 81/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1333 - val_loss: 0.1170
Epoch 82/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1224 - val_loss: 0.1366
Epoch 83/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1526 - val_loss: 0.1257
Epoch 84/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1244 - val_loss: 0.1085
```

```
Epoch 85/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1413 - val_loss: 0.1151
Epoch 86/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1410 - val_loss: 0.1277
Epoch 87/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1333 - val_loss: 0.1214
Epoch 88/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1471 - val_loss: 0.1264
Epoch 89/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1480 - val_loss: 0.1292
Epoch 90/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1300 - val_loss: 0.1138
Epoch 91/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1325 - val_loss: 0.1195
Epoch 92/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1193 - val_loss: 0.1223
Epoch 93/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1309 - val_loss: 0.1306
Epoch 94/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1338 - val_loss: 0.1132
Epoch 95/100
17/17 [==============================] - 0s 5ms/step - loss: 0.1330 - val_loss: 0.1281
Epoch 96/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1243 - val_loss: 0.1216
Epoch 97/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1157 - val_loss: 0.1263
Epoch 98/100
17/17 [==============================] - 0s 3ms/step - loss: 0.1408 - val_loss: 0.1331
Epoch 99/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1302 - val_loss: 0.1118
Epoch 100/100
17/17 [==============================] - 0s 4ms/step - loss: 0.1204 - val_loss: 0.1202
```

In [66]:

```
result = ANN_model.evaluate(X_test, y_test)
accuracy_ANN = 1 - result
print("Accuracy : {}".format(accuracy_ANN))
```

```
3/3 [==============================] - 0s 2ms/step - loss: 0.2327
Accuracy : 0.7672826200723648
```

In [67]:

```
epochs_hist.history.keys()
```

Out[67]:

```
dict_keys(['loss', 'val_loss'])
```

In [68]:

```
plt.plot(epochs_hist.history['loss'])
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.legend(['Training Loss'])
```

Out[68]:

```
<matplotlib.legend.Legend at 0x23fb30a7190>
```

TASK #8: TRAIN AND EVALUATE A DECISION TREE AND

RANDOM FOREST MODELS

In [69]:

```
# Decision tree builds regression or classification models in the form of a tree structure.
# Decision tree breaks down a dataset into smaller subsets while at the same time an associated decision tree is
# The final result is a tree with decision nodes and leaf nodes.
# Great resource: https://www.saedsayad.com/decision_tree_reg.htm

from sklearn.tree import DecisionTreeRegressor
decisionTree_model = DecisionTreeRegressor()
decisionTree_model.fit(X_train, y_train)
```

Out[69]:

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [70]:

```
accuracy_decisionTree = decisionTree_model.score(X_test, y_test)
accuracy_decisionTree
```

Out[70]: 0.5523895843075302

In [71]:

```
# Many decision Trees make up a random forest model which is an ensemble model.
# Predictions made by each decision tree are averaged to get the prediction of random forest model.
# A random forest regressor fits a number of classifying decision trees on various sub-samples of the dataset and
```

In [72]:

```
from sklearn.ensemble import RandomForestRegressor
randomForest_model = RandomForestRegressor(n_estimators=100, max_depth=10)
randomForest_model.fit(X_train, y_train)
```

```
<ipython-input-72-b46cdefa77dc>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples,), for example using ravel().
  randomForest_model.fit(X_train, y_train)
```

Out[72]:

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=10)
```

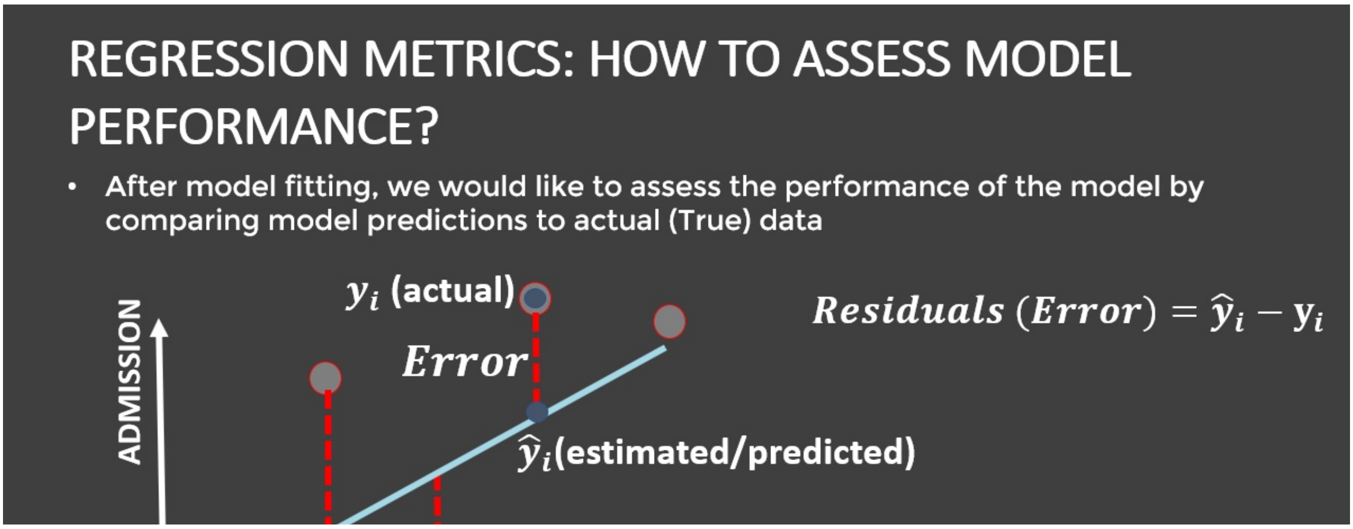
In [73]:

```
accuracy_randomforest = randomForest_model.score(X_test, y_test)
accuracy_randomforest
```

Out[73]: 0.7924314770705428

TASK #9: UNDERSTAND VARIOUS REGRESSION KPIs

In [ ]:



REGRESSION METRICS: MEAN ABSOLUTE ERROR (MAE)

- Mean Absolute Error (MAE) is obtained by calculating the absolute difference between the model predictions and the true (actual) values
- MAE is a measure of the **average magnitude of error** generated by the regression model
- The mean absolute error (MAE) is calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- MAE is calculated by following these steps:
  1. Calculate the residual of every data point
  2. Calculate the absolute value (to get rid of the sign)
  3. Calculate the average of all residuals
- If MAE is zero, this indicates that the model predictions are perfect.

REGRESSION METRICS: MEAN SQUARE ERROR (MSE)

- Mean Square Error (MSE) is very similar to the Mean Absolute Error (MAE) but instead of using absolute values, squares of the difference between the model predictions and the training dataset (true values) is being calculated.
- MSE values are generally **larger** compared to the MAE since the **residuals are being squared.**
- In case of data outliers, MSE will become much larger compared to MAE
- In MSE, error increases in a **quadratic fashion** while the error increases in **proportional fashion in MAE**
- In MSE, since the error is being squared, any predicting error is being heavily penalized
- The MSE is calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- MSE is calculated by following these steps:
  1. Calculate the residual for every data point
  2. Calculate the squared value of the residuals
  3. Calculate the average of results from step #2

REGRESSION METRICS: ROOT MEAN SQUARE ERROR (RMSE)

- Root Mean Square Error (RMSE) represents the **standard deviation of the residuals** (i.e.: differences between the model predictions and the true values (training data)).
- RMSE can be **easily interpreted** compared to MSE because RMSE units match the units of the output.
- RMSE provides an estimate of how large the residuals are being dispersed.

- The RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2}$$

- RMSE is calculated by following these steps:
  1. Calculate the residual for every data point
  2. Calculate the squared value of the residuals
  3. Calculate the average of the squared residuals
  4. Obtain the square root of the result

REGRESSION METRICS: R SQUARE (R^2)-COEFFICIENT OF DETERMINATION

- R-square or the coefficient of determination represents the proportion of variance (of y) that has been explained by the independent variables in the model.
- If $R^2 = 80$, this means that 80% of the increase in university admission is due to GRE score (assuming a simple linear regression model).

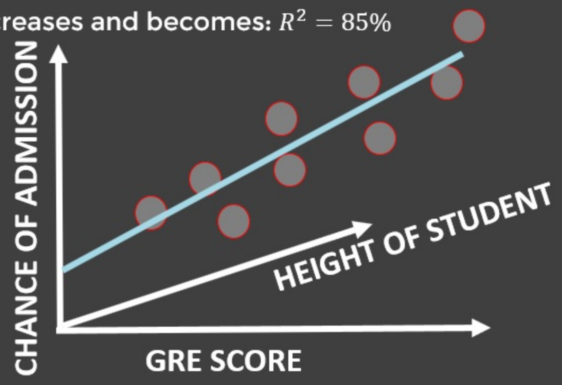
REGRESSION METRICS: R SQUARE (R^2)-COEFFICIENT OF DETERMINATION

- R-square or the coefficient of determination represents the proportion of variance (y) that has been explained by the independent variables (X) in the model.
- It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.
- Best possible score is 1.0
- A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2}{\sum_{i=1}^{n}\left(y_i - \overline{y}\right)^2}$$

REGRESSION METRICS: ADJUSTED R SQUARE (R^2)-

- If $R^2 = 80$, this means that 80% of the increase in chance of university admission is due to increase in GRE score.
- Let's add another 'useless' independent variable, let's say the "height of the student" to the Z-axis.
- Now R^2 increases and becomes: $R^2 = 85\%$



REGRESSION METRICS: ADJUSTED R SQUARE (R^2)-

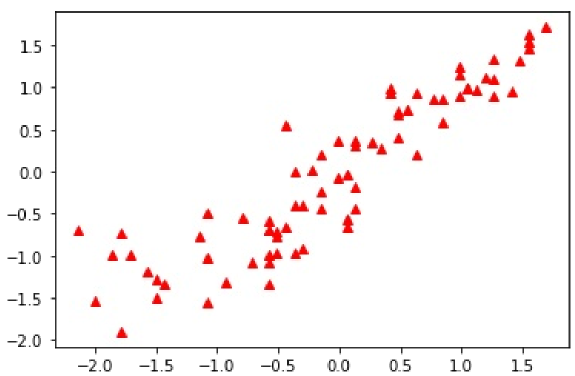
- One limitation of R^2 is that it increases by adding independent variables to the model which is misleading since some added variables might be useless with minimal significance.
- Adjusted R^2 overcomes this issue by adding a penalty if we make an attempt to add independent variable that does not improve the model.
- Adjusted R^2 is a modified version of the R^2 and takes into account the number of predictors in the model.
- If useless predictors are added to the model, Adjusted R^2 will decrease
- If useful predictors are added to the model, Adjusted R^2 will increase
- K is the number of independent variables and n is the number of samples

$$R^2_{adjusted} = 1 - [\frac{(1 - R^2)(n - 1)}{n - k - 1}]$$

TASK #10: CALCULATE REGRESSION MODEL KPIs

In [74]:
```
y_pred = linear_regression_model.predict(X_test)
plt.plot(y_test, y_pred, '^', color='r')
```

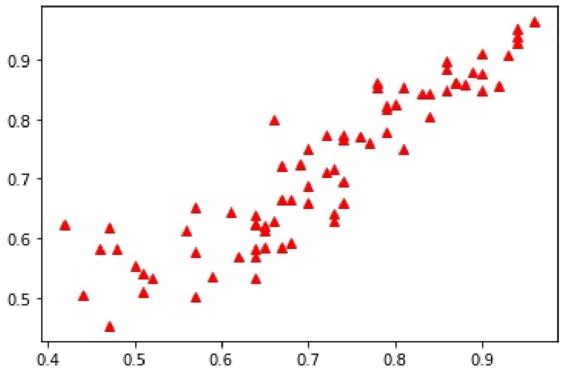
Out[74]: [<matplotlib.lines.Line2D at 0x23fb335b580>]



In [75]:
```
y_predict_orig = scaler_y.inverse_transform(y_pred)
y_test_orig = scaler_y.inverse_transform(y_test)
```

In [76]:
```
plt.plot(y_test_orig, y_predict_orig, '^', color='r')
```

Out[76]: [<matplotlib.lines.Line2D at 0x23fb33b1ee0>]



In [77]:
```
k = X_test.shape[1]
n = len(X_test)
n
```

Out[77]: 75

In [78]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test_orig, y_predict_orig)),'.3f'))
MSE = mean_squared_error(y_test_orig, y_predict_orig)
MAE = mean_absolute_error(y_test_orig, y_predict_orig)
r2 = r2_score(y_test_orig, y_predict_orig)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 0.058
MSE = 0.0033716764620205304
MAE = 0.04344986895616674
R2 = 0.8233341459249033
Adjusted R2 = 0.8048765193797439
```

EXCELLENT JOB! YOU SHOULD BE PROUD OF YOUR NEWLY ACQUIRED SKILLS

In [79]:
```
import pickle
```

In [80]:
```
s = np.array([320, 110, 1, 5, 5, 9, 1])
print(s.shape)
s = s.reshape(1,-1)
print(s.shape)
```

```
(7,)
(1, 7)
```

In [81]:
```
pickle.dump(linear_regression_model, open('linear_regression_model_sc.pkl', 'wb'))
```

In [82]:
```
model = pickle.load(open('linear_regression_model_sc.pkl', 'rb'))
print(model.predict(s))
```

```
[[60.29857917]]
```

```
[[80.29837917]]
```

In [84]:

```
# https://graduate-admission-prediction0.herokuapp.com/predict
```

In [ ]:

```

```

Loading [MathJax]/extensions/Safe.js