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/

In [3]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
```

In [4]:

```
admission_df = pd.read_csv('admission_predict.csv')
```

In [5]:

```
admission_df.head()
```

Out[5]:

	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 [7]:

```
admission_df
```

Out[7]:

	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
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

EDA

In [8]:

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

Out[8]:

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

In [9]:

```
# 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 [10]:

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

Out[10]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	5(
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	

In [11]:

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

Out[11]:

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

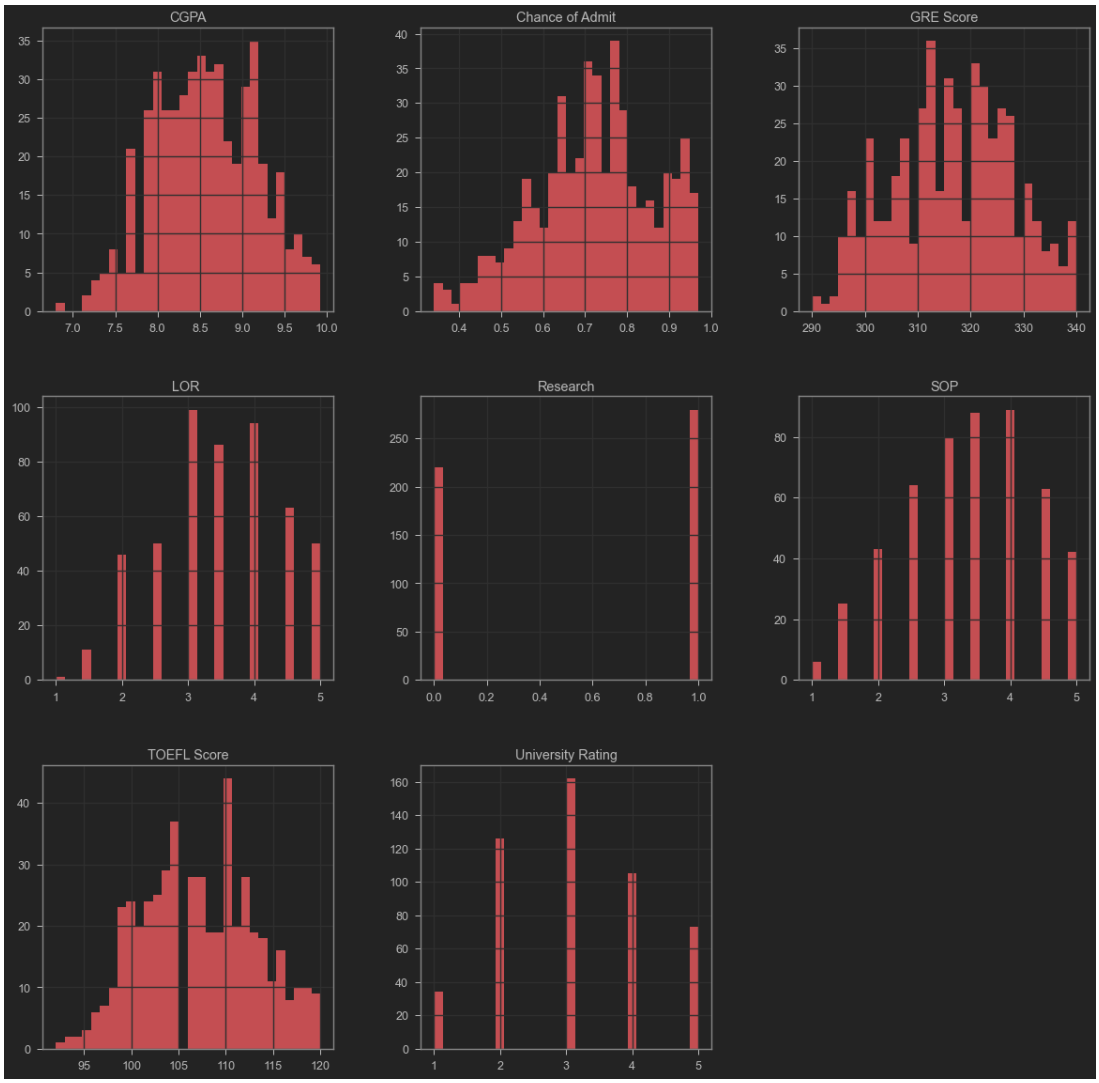
Data Visualization

In [12]:

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

Out[12]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002362
526CBA8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
59C8BA8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
59FDE10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5A3A0B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5A68320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5A99588>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5AC97F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5AF9A20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002362
5AF9A90>]],
      dtype=object)
```

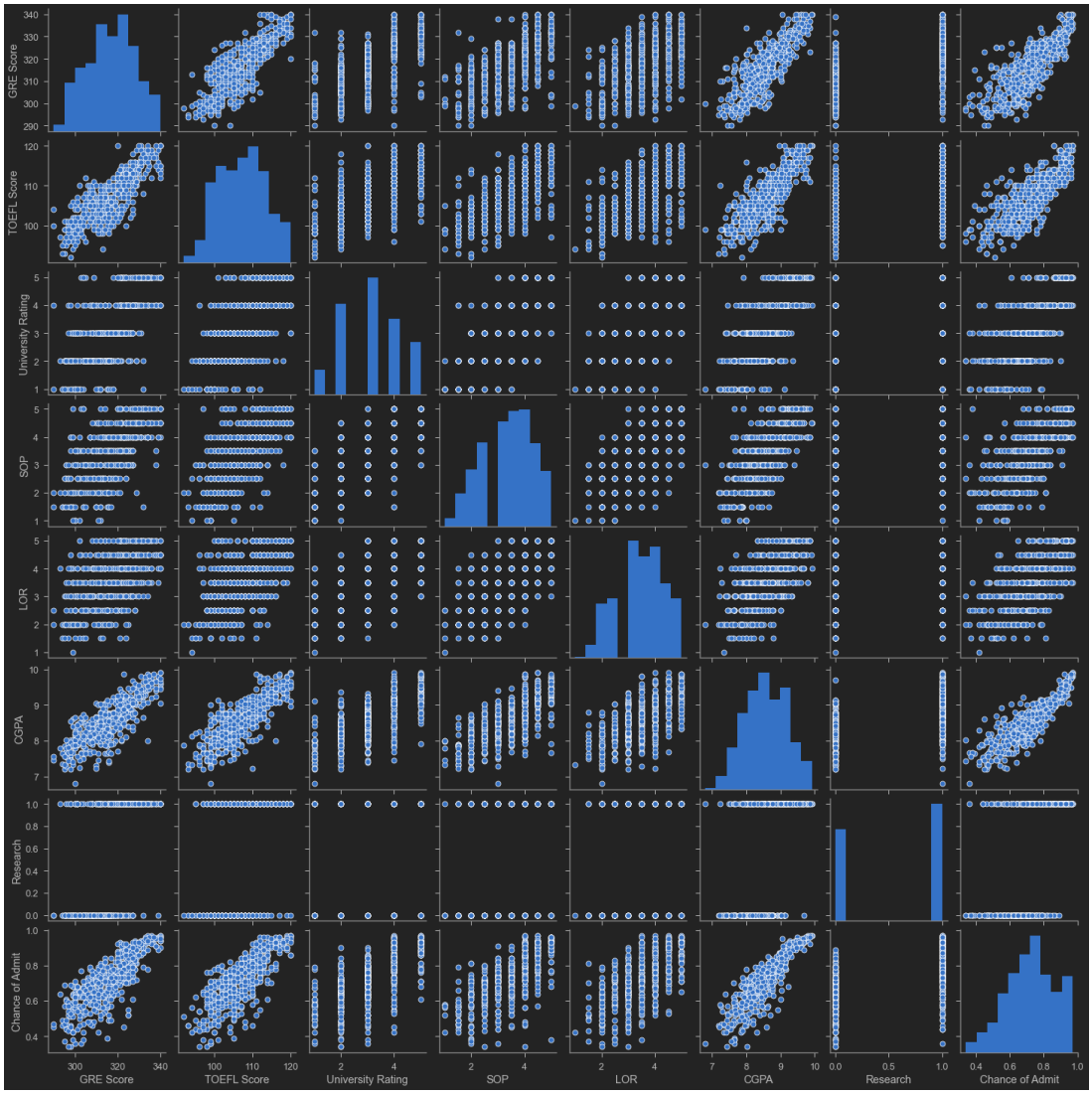


In [13]:

```
sns.pairplot(admission_df)
```

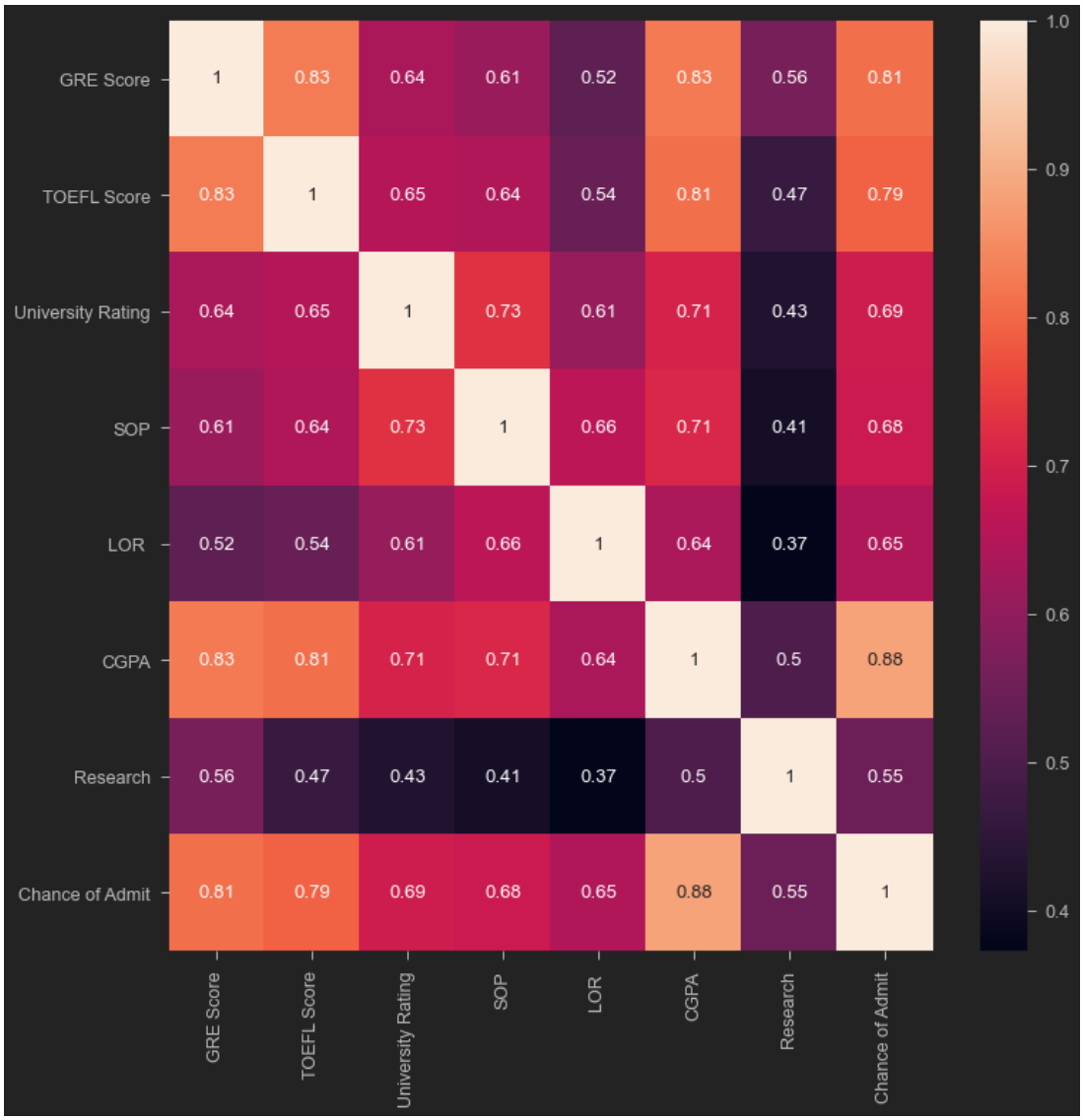
Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x236267b4198>
```



In [14]:

```
corr_matrix = admission_df.corr()
plt.figure(figsize = (12, 12))
sns.heatmap(corr_matrix, annot = True)
plt.show()
```



Create training and test set

In [15]:

```
admission_df.columns
```

Out[15]:

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR
', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

In [16]:

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

In [17]:

```
y = admission_df['Chance of Admit']
```

In [18]:

```
X.shape
```

Out[18]:

```
(500, 7)
```

In [19]:

```
y.shape
```

Out[19]:

```
(500,)
```

In [20]:

```
y
```

Out[20]:

```
0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
       ...
495    0.87
496    0.96
497    0.93
498    0.73
499    0.84
Name: Chance of Admit, Length: 500, dtype: float64
```

In [21]:

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

In [22]:

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

Out[22]:

```
(500, 1)
```

In [23]:

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

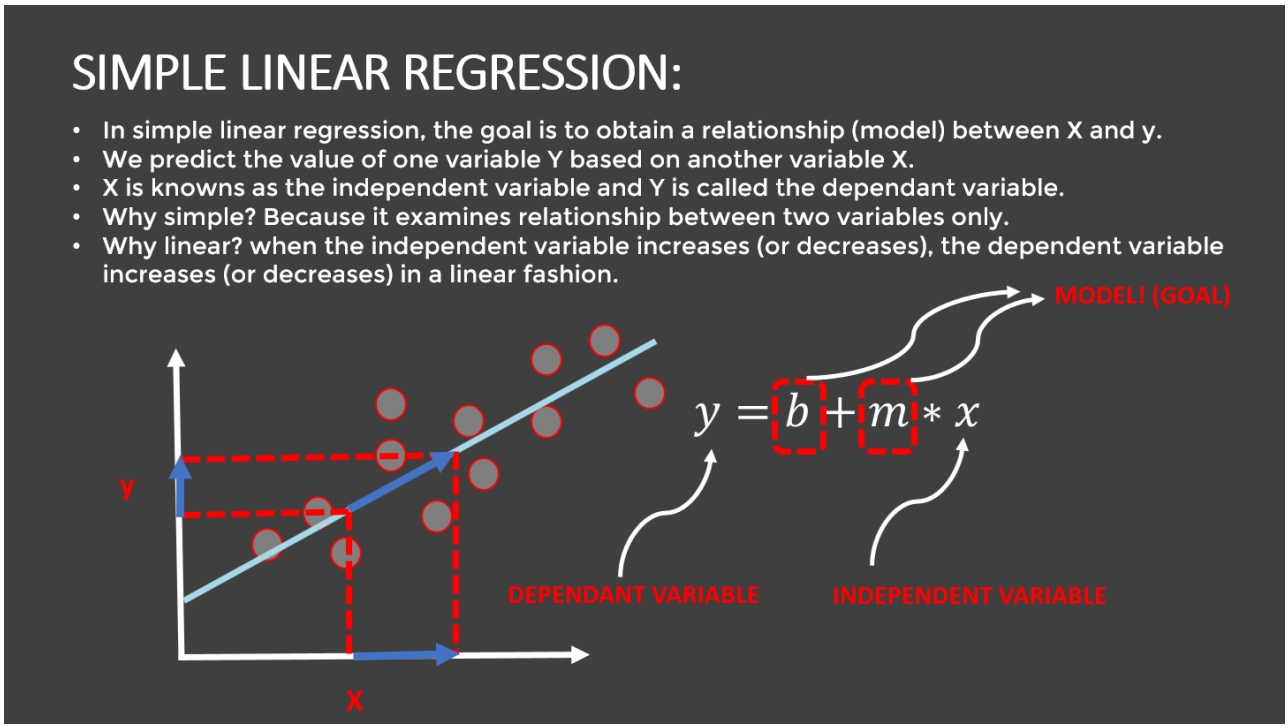
In [26]:

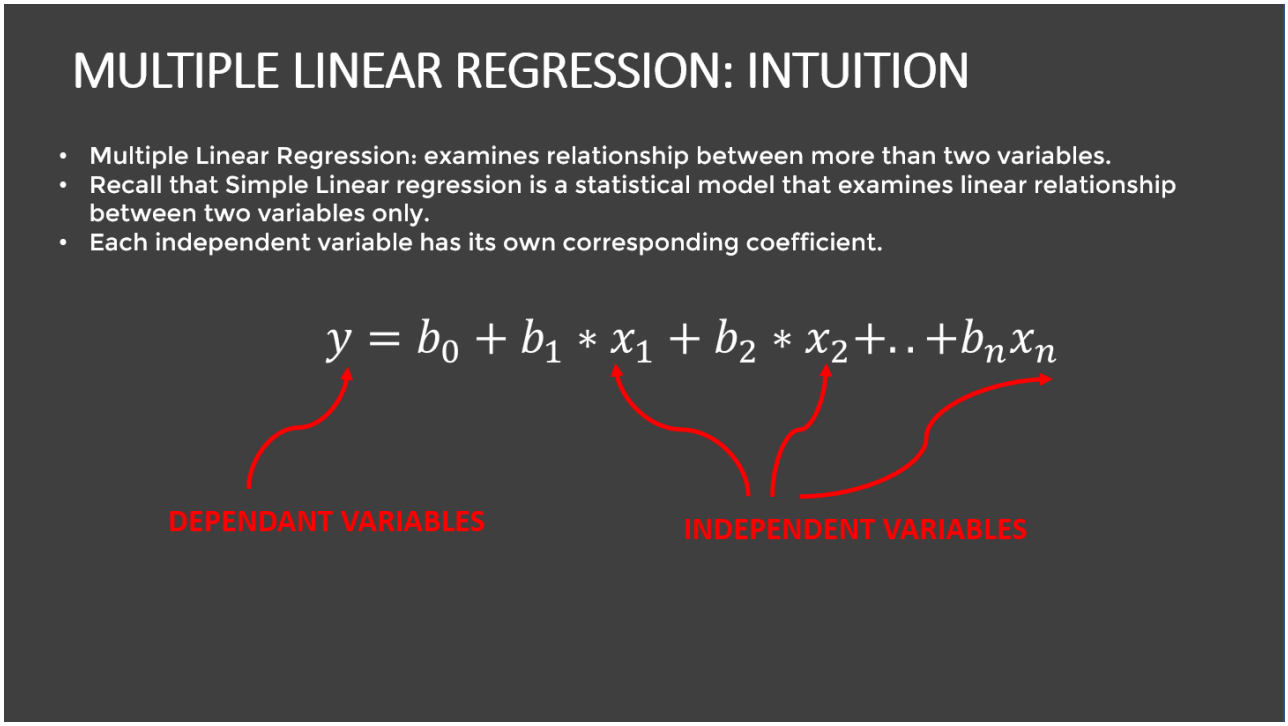
```
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y)
```

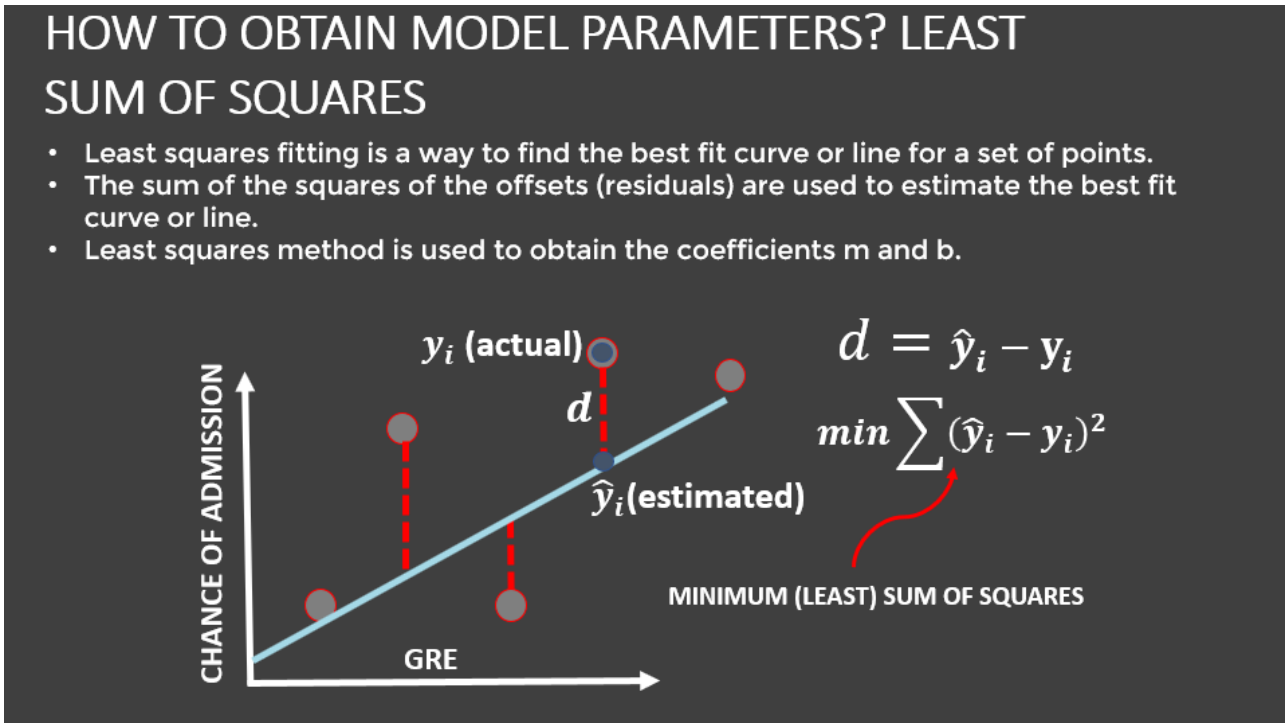
In [27]:

```
# 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)
```

Train and evaluate Linear Regression Model







In [28]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score
```

In [29]:

```
LinearRegression_model = LinearRegression()
LinearRegression_model.fit(X_train, y_train)
```

Out[29]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, norma
lize=False)
```

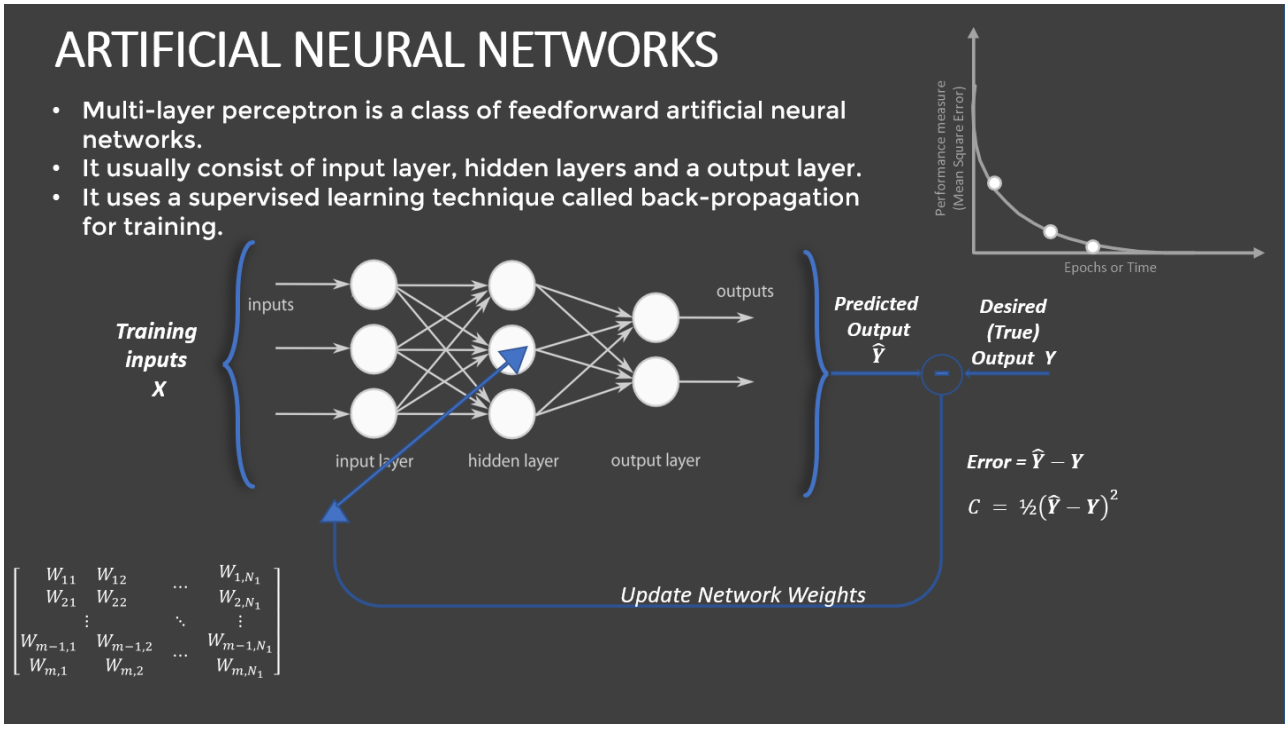
In [30]:

```
accuracy_LinearRegression = LinearRegression_model.score(X_test, y_test)
accuracy_LinearRegression
```

Out[30]:

```
0.8089302650690123
```

Train and evaluate an artificial neural network model



In [31]:

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

In [32]:

```
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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                400
_________________________________________________________________
activation (Activation)      (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 150)               7650
_________________________________________________________________
activation_1 (Activation)    (None, 150)               0
_________________________________________________________________
dropout (Dropout)            (None, 150)               0
_________________________________________________________________
dense_2 (Dense)              (None, 150)               22650
_________________________________________________________________
activation_2 (Activation)    (None, 150)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                7550
_________________________________________________________________
activation_3 (Activation)    (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 38,301
Trainable params: 38,301
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

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

In [34]:

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

```
Train on 340 samples, validate on 85 samples
Epoch 1/100
340/340 [==============================] - 2s 6ms/sample - loss: 0.5
949 - val_loss: 0.2767
Epoch 2/100
340/340 [==============================] - 0s 259us/sample - loss:
0.3637 - val_loss: 0.2851
Epoch 3/100
340/340 [==============================] - 0s 276us/sample - loss:
0.3006 - val_loss: 0.2739
Epoch 4/100
340/340 [==============================] - 0s 278us/sample - loss:
0.2393 - val_loss: 0.2622
Epoch 5/100
340/340 [==============================] - 0s 258us/sample - loss:
0.2531 - val_loss: 0.2609
Epoch 6/100
340/340 [==============================] - 0s 234us/sample - loss:
0.2396 - val_loss: 0.2988
Epoch 7/100
340/340 [==============================] - 0s 270us/sample - loss:
0.2414 - val_loss: 0.2630
Epoch 8/100
340/340 [==============================] - 0s 257us/sample - loss:
0.2318 - val_loss: 0.2987
Epoch 9/100
340/340 [==============================] - 0s 239us/sample - loss:
0.2291 - val_loss: 0.2952
Epoch 10/100
340/340 [==============================] - 0s 268us/sample - loss:
0.2195 - val_loss: 0.2715
Epoch 11/100
340/340 [==============================] - 0s 297us/sample - loss:
0.2082 - val_loss: 0.3081
Epoch 12/100
340/340 [==============================] - 0s 299us/sample - loss:
0.1952 - val_loss: 0.2604
Epoch 13/100
340/340 [==============================] - 0s 481us/sample - loss:
0.2025 - val_loss: 0.3335
Epoch 14/100
340/340 [==============================] - 0s 451us/sample - loss:
0.1996 - val_loss: 0.2792
Epoch 15/100
340/340 [==============================] - 0s 488us/sample - loss:
0.1861 - val_loss: 0.2831
Epoch 16/100
340/340 [==============================] - 0s 417us/sample - loss:
0.2214 - val_loss: 0.2805
Epoch 17/100
340/340 [==============================] - 0s 441us/sample - loss:
0.1975 - val_loss: 0.2851
Epoch 18/100
340/340 [==============================] - 0s 440us/sample - loss:
0.1837 - val_loss: 0.2881
Epoch 19/100
340/340 [==============================] - 0s 441us/sample - loss:
0.2022 - val_loss: 0.2811
Epoch 20/100
340/340 [==============================] - 0s 496us/sample - loss:
0.2062 - val_loss: 0.3085
```

```
Epoch 21/100
340/340 [==============================] - 0s 488us/sample - loss:
0.1966 - val_loss: 0.2770
Epoch 22/100
340/340 [==============================] - 0s 442us/sample - loss:
0.1962 - val_loss: 0.2672
Epoch 23/100
340/340 [==============================] - 0s 440us/sample - loss:
0.2018 - val_loss: 0.2628
Epoch 24/100
340/340 [==============================] - 0s 444us/sample - loss:
0.1843 - val_loss: 0.2528
Epoch 25/100
340/340 [==============================] - 0s 438us/sample - loss:
0.1886 - val_loss: 0.2946
Epoch 26/100
340/340 [==============================] - 0s 537us/sample - loss:
0.1643 - val_loss: 0.2342
Epoch 27/100
340/340 [==============================] - 0s 438us/sample - loss:
0.1744 - val_loss: 0.2988
Epoch 28/100
340/340 [==============================] - 0s 494us/sample - loss:
0.1744 - val_loss: 0.2421
Epoch 29/100
340/340 [==============================] - 0s 494us/sample - loss:
0.1668 - val_loss: 0.2828
Epoch 30/100
340/340 [==============================] - 0s 469us/sample - loss:
0.1792 - val_loss: 0.2849
Epoch 31/100
340/340 [==============================] - 0s 479us/sample - loss:
0.1851 - val_loss: 0.2745
Epoch 32/100
340/340 [==============================] - 0s 470us/sample - loss:
0.1811 - val_loss: 0.3283
Epoch 33/100
340/340 [==============================] - 0s 440us/sample - loss:
0.1924 - val_loss: 0.2746
Epoch 34/100
340/340 [==============================] - 0s 474us/sample - loss:
0.1885 - val_loss: 0.2565
Epoch 35/100
340/340 [==============================] - 0s 494us/sample - loss:
0.1759 - val_loss: 0.3290
Epoch 36/100
340/340 [==============================] - 0s 518us/sample - loss:
0.1685 - val_loss: 0.2659
Epoch 37/100
340/340 [==============================] - 0s 419us/sample - loss:
0.1849 - val_loss: 0.2874
Epoch 38/100
340/340 [==============================] - 0s 489us/sample - loss:
0.1436 - val_loss: 0.2998
Epoch 39/100
340/340 [==============================] - 0s 440us/sample - loss:
0.1582 - val_loss: 0.2864
Epoch 40/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1544 - val_loss: 0.2583
Epoch 41/100
```

```
340/340 [==============================] - 0s 513us/sample - loss:
0.1643 - val_loss: 0.3014
Epoch 42/100
340/340 [==============================] - 0s 508us/sample - loss:
0.1666 - val_loss: 0.3053
Epoch 43/100
340/340 [==============================] - 0s 481us/sample - loss:
0.1490 - val_loss: 0.2580
Epoch 44/100
340/340 [==============================] - 0s 467us/sample - loss:
0.1846 - val_loss: 0.2798
Epoch 45/100
340/340 [==============================] - 0s 471us/sample - loss:
0.1396 - val_loss: 0.2952
Epoch 46/100
340/340 [==============================] - 0s 463us/sample - loss:
0.1503 - val_loss: 0.2700
Epoch 47/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1614 - val_loss: 0.3170
Epoch 48/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1542 - val_loss: 0.2670
Epoch 49/100
340/340 [==============================] - 0s 439us/sample - loss:
0.1432 - val_loss: 0.2889
Epoch 50/100
340/340 [==============================] - 0s 493us/sample - loss:
0.1542 - val_loss: 0.2873
Epoch 51/100
340/340 [==============================] - 0s 464us/sample - loss:
0.1486 - val_loss: 0.2734
Epoch 52/100
340/340 [==============================] - 0s 471us/sample - loss:
0.1495 - val_loss: 0.3032
Epoch 53/100
340/340 [==============================] - 0s 457us/sample - loss:
0.1348 - val_loss: 0.2544
Epoch 54/100
340/340 [==============================] - 0s 437us/sample - loss:
0.1384 - val_loss: 0.3126
Epoch 55/100
340/340 [==============================] - 0s 486us/sample - loss:
0.1500 - val_loss: 0.2594
Epoch 56/100
340/340 [==============================] - 0s 486us/sample - loss:
0.1324 - val_loss: 0.2842
Epoch 57/100
340/340 [==============================] - 0s 463us/sample - loss:
0.1425 - val_loss: 0.2625
Epoch 58/100
340/340 [==============================] - 0s 439us/sample - loss:
0.1251 - val_loss: 0.2707
Epoch 59/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1395 - val_loss: 0.2871
Epoch 60/100
340/340 [==============================] - 0s 439us/sample - loss:
0.1387 - val_loss: 0.2705
Epoch 61/100
340/340 [==============================] - 0s 419us/sample - loss:
```

```
0.1348 - val_loss: 0.2810
Epoch 62/100
340/340 [==============================] - 0s 453us/sample - loss:
0.1461 - val_loss: 0.2794
Epoch 63/100
340/340 [==============================] - 0s 516us/sample - loss:
0.1419 - val_loss: 0.2633
Epoch 64/100
340/340 [==============================] - 0s 496us/sample - loss:
0.1277 - val_loss: 0.2719
Epoch 65/100
340/340 [==============================] - 0s 448us/sample - loss:
0.1405 - val_loss: 0.3147
Epoch 66/100
340/340 [==============================] - 0s 470us/sample - loss:
0.1314 - val_loss: 0.2470
Epoch 67/100
340/340 [==============================] - 0s 451us/sample - loss:
0.1345 - val_loss: 0.3079
Epoch 68/100
340/340 [==============================] - 0s 480us/sample - loss:
0.1424 - val_loss: 0.3008
Epoch 69/100
340/340 [==============================] - 0s 503us/sample - loss:
0.1409 - val_loss: 0.2869
Epoch 70/100
340/340 [==============================] - 0s 464us/sample - loss:
0.1400 - val_loss: 0.3150
Epoch 71/100
340/340 [==============================] - 0s 446us/sample - loss:
0.1369 - val_loss: 0.2534
Epoch 72/100
340/340 [==============================] - 0s 462us/sample - loss:
0.1369 - val_loss: 0.2829
Epoch 73/100
340/340 [==============================] - 0s 466us/sample - loss:
0.1249 - val_loss: 0.2908
Epoch 74/100
340/340 [==============================] - 0s 491us/sample - loss:
0.1346 - val_loss: 0.2634
Epoch 75/100
340/340 [==============================] - 0s 451us/sample - loss:
0.1206 - val_loss: 0.2673
Epoch 76/100
340/340 [==============================] - 0s 454us/sample - loss:
0.1101 - val_loss: 0.2787
Epoch 77/100
340/340 [==============================] - 0s 520us/sample - loss:
0.1018 - val_loss: 0.2802
Epoch 78/100
340/340 [==============================] - 0s 479us/sample - loss:
0.1263 - val_loss: 0.2958
Epoch 79/100
340/340 [==============================] - 0s 310us/sample - loss:
0.1174 - val_loss: 0.2650
Epoch 80/100
340/340 [==============================] - 0s 340us/sample - loss:
0.1148 - val_loss: 0.2619
Epoch 81/100
340/340 [==============================] - 0s 431us/sample - loss:
0.1104 - val_loss: 0.2847
```

```
Epoch 82/100
340/340 [==============================] - 0s 270us/sample - loss:
0.1171 - val_loss: 0.2739
Epoch 83/100
340/340 [==============================] - ETA: 0s - loss: 0.107 - 0
s 288us/sample - loss: 0.1063 - val_loss: 0.2947
Epoch 84/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1133 - val_loss: 0.2729
Epoch 85/100
340/340 [==============================] - 0s 502us/sample - loss:
0.1168 - val_loss: 0.2806
Epoch 86/100
340/340 [==============================] - 0s 492us/sample - loss:
0.1169 - val_loss: 0.2902
Epoch 87/100
340/340 [==============================] - 0s 497us/sample - loss:
0.1195 - val_loss: 0.2954
Epoch 88/100
340/340 [==============================] - 0s 503us/sample - loss:
0.1324 - val_loss: 0.3036
Epoch 89/100
340/340 [==============================] - 0s 421us/sample - loss:
0.1317 - val_loss: 0.2844
Epoch 90/100
340/340 [==============================] - 0s 355us/sample - loss:
0.1236 - val_loss: 0.2829
Epoch 91/100
340/340 [==============================] - 0s 325us/sample - loss:
0.0971 - val_loss: 0.2989
Epoch 92/100
340/340 [==============================] - 0s 441us/sample - loss:
0.1053 - val_loss: 0.2917
Epoch 93/100
340/340 [==============================] - 0s 468us/sample - loss:
0.1060 - val_loss: 0.2888
Epoch 94/100
340/340 [==============================] - 0s 489us/sample - loss:
0.1133 - val_loss: 0.2833
Epoch 95/100
340/340 [==============================] - 0s 465us/sample - loss:
0.1217 - val_loss: 0.2854
Epoch 96/100
340/340 [==============================] - 0s 476us/sample - loss:
0.1202 - val_loss: 0.2846
Epoch 97/100
340/340 [==============================] - 0s 335us/sample - loss:
0.1044 - val_loss: 0.2709
Epoch 98/100
340/340 [==============================] - 0s 304us/sample - loss:
0.1069 - val_loss: 0.2930
Epoch 99/100
340/340 [==============================] - 0s 290us/sample - loss:
0.1092 - val_loss: 0.2896
Epoch 100/100
340/340 [==============================] - 0s 293us/sample - loss:
0.1100 - val_loss: 0.2720
```

In [35]:

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

```
75/1 [===============================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
=====================================================================
============] - 0s 117us/sample - loss: 0.2184
Accuracy : 0.7226079026858012
```

In [36]:

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

Out[36]:

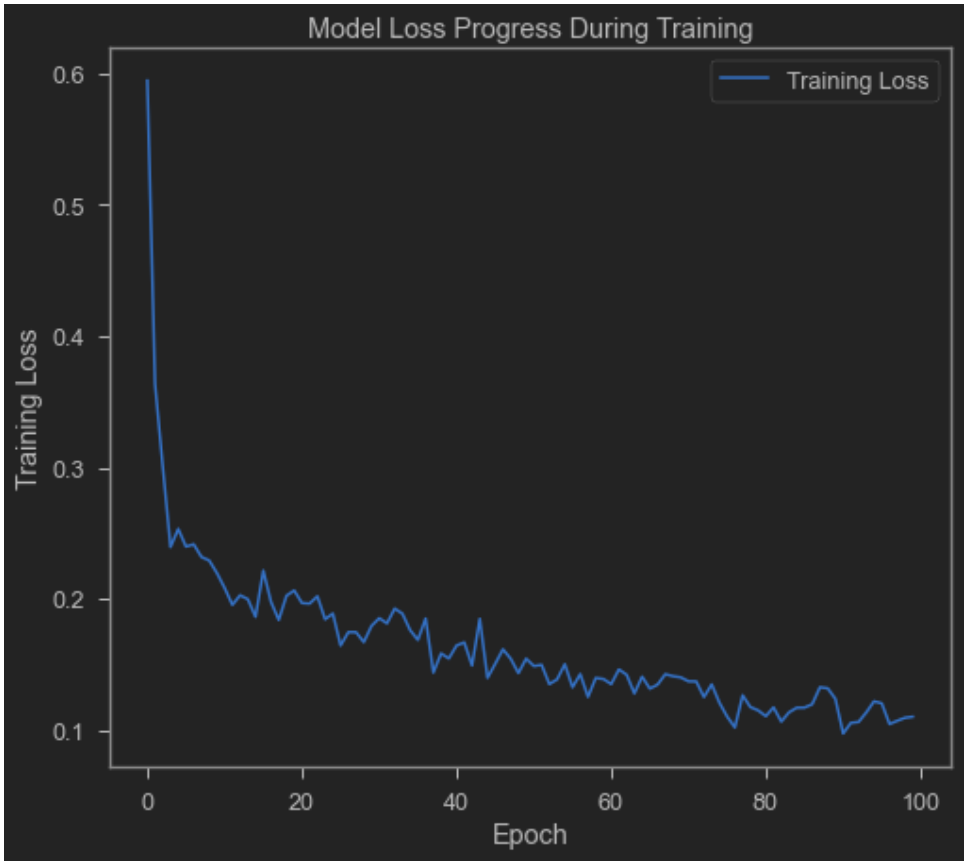
```
dict_keys(['loss', 'val_loss'])
```

In [37]:

```
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[37]:

```
<matplotlib.legend.Legend at 0x236329bcfd0>
```



Train and evaluate a Decision Tree and Random Forest model

In [38]:

```
# 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 tim
e an associated decision tree is incrementally developed.
# 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[38]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=
None,
                      max_leaf_nodes=None, min_impurity_decrease=0.
0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=
0.0,
                      presort=False, random_state=None, splitter='be
st')
```

In [39]:

```
accuracy_DecisionTree = DecisionTree_model.score(X_test, y_test)
accuracy_DecisionTree
```

Out[39]:

```
0.6533813654292446
```

In [40]:

```
# 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 r
andom forest model.
# A random forest regressor fits a number of classifying decision trees on vario
us sub-samples of the dataset and uses averaging to improve the predictive accur
acy and control over-fitting.
from sklearn.ensemble import RandomForestRegressor
RandomForest_model = RandomForestRegressor(n_estimators=100, max_depth = 10)
RandomForest_model.fit(X_train, y_train)
```

```
C:\Users\Riswin\Anaconda3\lib\site-packages\ipykernel_launcher.py:6:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for exam
ple using ravel().
```

Out[40]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=
None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10
0,
                      n_jobs=None, oob_score=False, random_state=Non
e,
                      verbose=0, warm_start=False)
```

In [41]:

```
accuracy_RandomForest = RandomForest_model.score(X_test, y_test)
accuracy_RandomForest
```

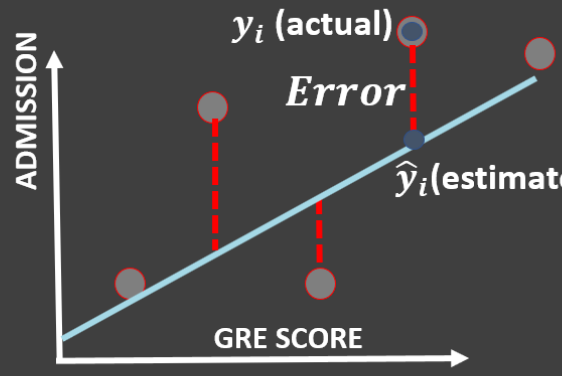
Out[41]:

```
0.7766102465703576
```

Understand various KPIs

REGRESSION METRICS: HOW TO ASSESS MODEL PERFORMANCE?

- After model fitting, we would like to assess the performance of the model by comparing model predictions to actual (True) data



$$Residuals\ (Error) = \hat{y}_i - y_i$$

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).

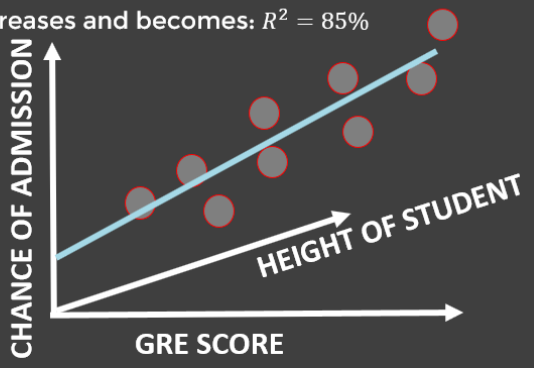
REGRESSION METRICS: R SQUARE (R^2)-COEFFICIENT OF DETERMINATION

- 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

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

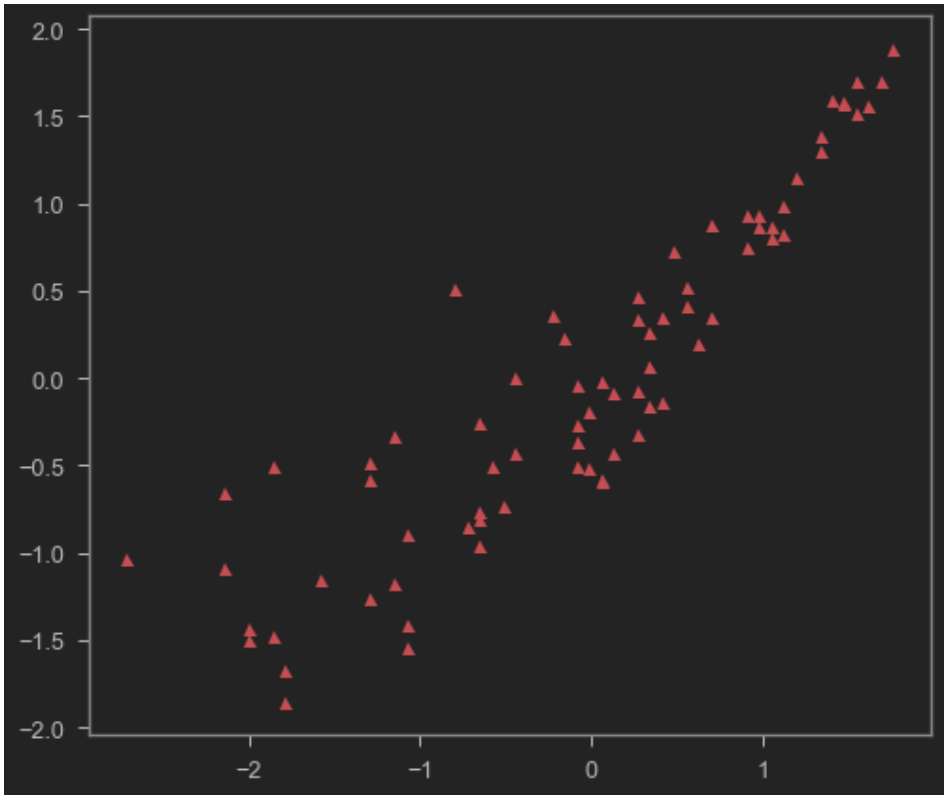
Calculate Regression model KPIs

In [42]:

```
y_predict = LinearRegression_model.predict(X_test)
plt.plot(y_test, y_predict, '^', color = 'r')
```

Out[42]:

```
[<matplotlib.lines.Line2D at 0x23632d230b8>]
```



In [43]:

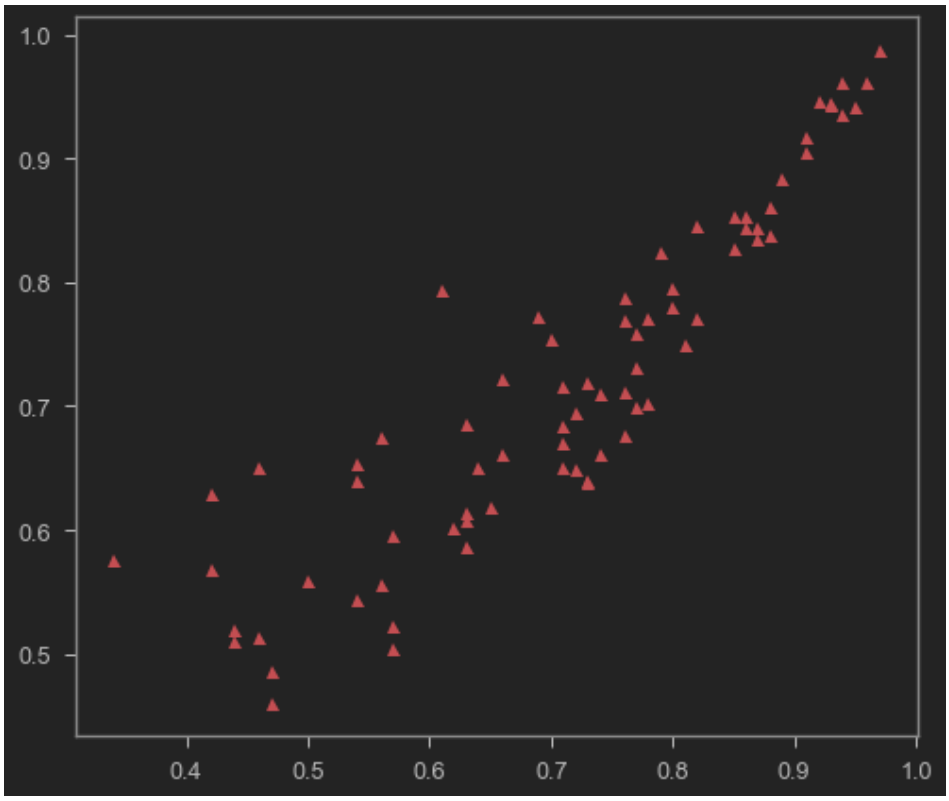
```
y_predict_orig = scaler_x.inverse_transform(y_predict)
y_test_orig = scaler_x.inverse_transform(y_test)
```

In [44]:

```
plt.plot(y_test_orig, y_predict_orig, '^', color = 'r')
```

Out[44]:

```
[<matplotlib.lines.Line2D at 0x23632eb26d8>]
```



In [45]:

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

Out[45]:

```
75
```

In [46]:

```
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)),'.3
f'))
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 = 0.068
MSE = 0.00461938343477846
MAE = 0.04696852870869254
R2 = 0.8089302650690124
Adjusted R2 = 0.788967755449357
```