IMPORTING LIBRARIES AND DATASET

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df_admission = pd.read_csv("Admission_Predict.csv")
```

In [3]:
```
df_admission.head()
```

Out[3]:

	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

Let's drop the serial no.

In [4]:
```
df_admission.drop("Serial No.", axis = 1, inplace = True)
```

EXPLORATORY DATA ANALYSIS

checking the null values

In [5]:
```
df_admission.isnull()
```

Out[5]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False
19	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False
22	False	False	False	False	False	False	False	False
23	False	False	False	False	False	False	False	False
24	False	False	False	False	False	False	False	False
25	False	False	False	False	False	False	False	False
26	False	False	False	False	False	False	False	False
27	False	False	False	False	False	False	False	False
28	False	False	False	False	False	False	False	False

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
29	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
470	False	False	False	False	False	False	False	False
471	False	False	False	False	False	False	False	False
472	False	False	False	False	False	False	False	False
473	False	False	False	False	False	False	False	False
474	False	False	False	False	False	False	False	False
475	False	False	False	False	False	False	False	False
476	False	False	False	False	False	False	False	False
477	False	False	False	False	False	False	False	False
478	False	False	False	False	False	False	False	False
479	False	False	False	False	False	False	False	False
480	False	False	False	False	False	False	False	False
481	False	False	False	False	False	False	False	False
482	False	False	False	False	False	False	False	False
483	False	False	False	False	False	False	False	False
484	False	False	False	False	False	False	False	False
485	False	False	False	False	False	False	False	False
486	False	False	False	False	False	False	False	False
487	False	False	False	False	False	False	False	False
488	False	False	False	False	False	False	False	False
489	False	False	False	False	False	False	False	False
490	False	False	False	False	False	False	False	False
491	False	False	False	False	False	False	False	False
492	False	False	False	False	False	False	False	False
493	False	False	False	False	False	False	False	False
494	False	False	False	False	False	False	False	False
495	False	False	False	False	False	False	False	False
496	False	False	False	False	False	False	False	False
497	False	False	False	False	False	False	False	False

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
498	False	False	False	False	False	False	False	False
499	False	False	False	False	False	False	False	False

500 rows × 8 columns

In [6]:
```
df_admission.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
GRE Score            500 non-null int64
TOEFL Score          500 non-null int64
University Rating    500 non-null int64
SOP                  500 non-null float64
LOR                  500 non-null float64
CGPA                 500 non-null float64
Research             500 non-null int64
Chance of Admit      500 non-null float64
dtypes: float64(4), int64(4)
memory usage: 31.3 KB
```

In [7]:
```
df_admission.describe()
```

Out[7]:

	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

Grouping by University Rating

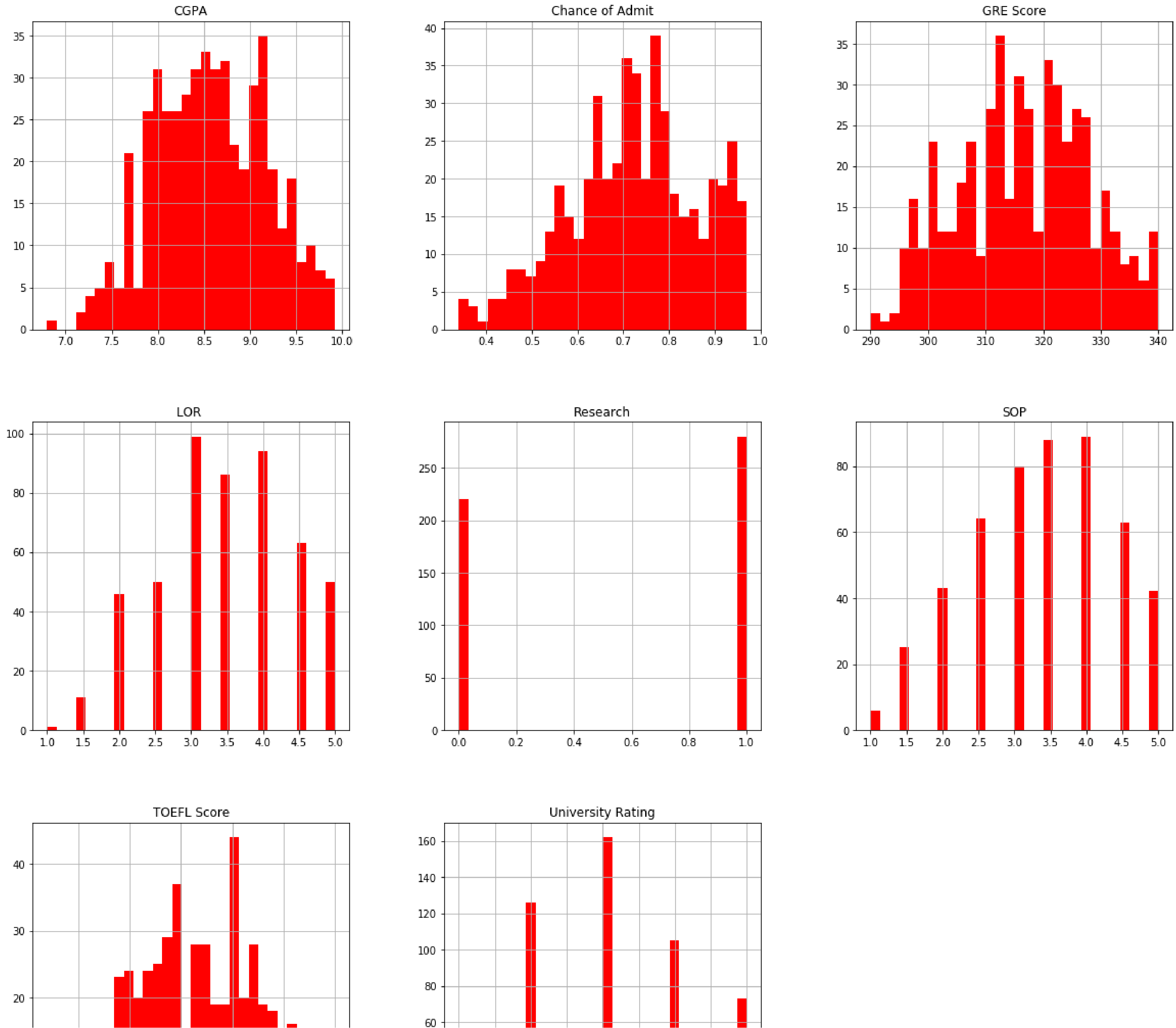
In [8]:
```
df_university_rating = df_admission.groupby("University Rating").mean()
df_university_rating
```

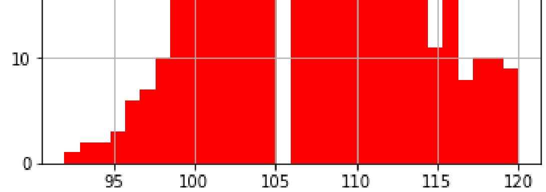
Out[8]:

	GRE Score	TOEFL Score	SOP	LOR	CGPA	Research	Chance of Admit
University Rating							
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

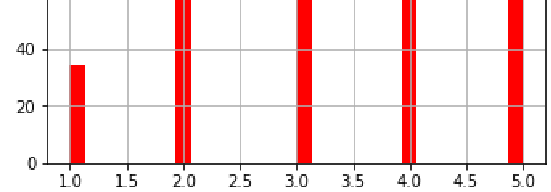
DATA VISUALIZATION

In [9]:
```
df_admission.hist(bins = 30, figsize = (20, 20), color = "r")
plt.show()
```


CGPA
7.0 7.5 8.0 8.5 9.0 9.5 10.0
0 5 10 15 20 25 30 35
Chance of Admit
0.4 0.5 0.6 0.7 0.8 0.9 1.0
0 5 10 15 20 25 30 35 40
GRE Score
290 300 310 320 330 340
0 5 10 15 20 25 30 35
LOR
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
0 20 40 60 80 100
Research
0.0 0.2 0.4 0.6 0.8 1.0
0 50 100 150 200 250
SOP
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
0 20 40 60 80
TOEFL Score
20 30 40
University Rating
60 80 100 120 140 160



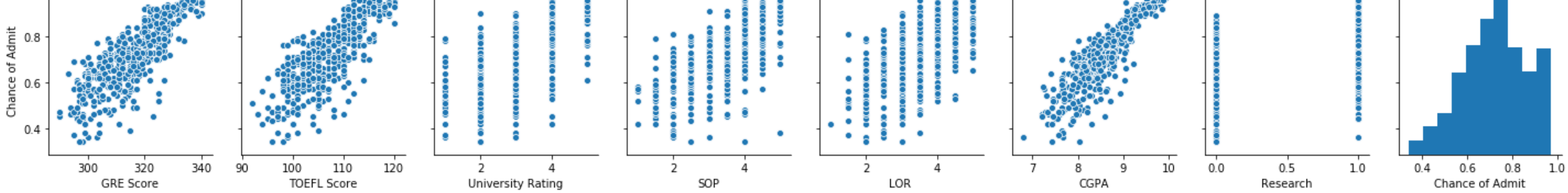
10
0
95
100
105
110
115
120



40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

In [10]:
```
sns.pairplot(df_admission)
plt.show()
```

GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research



In [11]:
```
corr_matrix = df_admission.corr()
plt.figure(figsize = (20, 8))
sns.heatmap(corr_matrix, annot = True)
plt.show()
```

TRAINING AND TESTING DATASET

In [12]:
```
df_admission.columns
```

Out[12]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

In [13]:
```
X = df_admission.drop("Chance of Admit", axis = 1)
```

In [14]:
```
y = df_admission["Chance of Admit"]
```

In [15]:
```
X.shape
```

Out[15]: (500, 7)

In [16]:
```
y.shape
```

Out[16]: (500,)

In [17]:
```
X = np.array(X)
y = np.array(y)
```

In [18]:
```
y = y.reshape(-1, 1)
y.shape
```

Out[18]: (500, 1)

Scaling the data before training the model

In [19]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X = scaler_x.fit_transform(X)
```

In [20]:
```
scaler_y = StandardScaler()
y = scaler_x.fit_transform(y)
```

Spliting the data in to test and train sets

In [21]:
```
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 A LINEAR REGRESSION MODEL

In [22]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score
```

In [23]:
```
LinearRegression_model = LinearRegression()
LinearRegression_model.fit(X_train, y_train)
```

Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [24]:
```
accuracy_LinearRegression = LinearRegression_model.score(X_test, y_test)
accuracy_LinearRegression
```

Out[24]: 0.8213768463774025

TRAIN AND EVALUATE AN ARTIFICIAL NEURAL NETWORK

In [25]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
```

In [26]:
```
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()
```

```
WARNING:tensorflow:From /srv/conda/envs/notebook/lib/python3.7/site-packages/tensorflow/python/ops/resource_variable_ops.py:435: coloc
ate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /srv/conda/envs/notebook/lib/python3.7/site-packages/tensorflow/python/keras/layers/core.py:143: calling dropo
ut (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /srv/conda/envs/notebook/lib/python3.7/site-packages/tensorflow/python/keras/utils/losses_utils.py:170: to_flo
at (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
_________________________________________________________________
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 [27]:

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

In [28]:

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

```
Train on 340 samples, validate on 85 samples
WARNING:tensorflow:From /srv/conda/envs/notebook/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from te
nsorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/100
340/340 [==============================] - 1s 2ms/sample - loss: 0.5515 - val_loss: 0.2389
Epoch 2/100
340/340 [==============================] - 0s 399us/sample - loss: 0.3696 - val_loss: 0.2463
Epoch 3/100
340/340 [==============================] - 0s 327us/sample - loss: 0.3117 - val_loss: 0.2517
Epoch 4/100
340/340 [==============================] - 0s 315us/sample - loss: 0.3086 - val_loss: 0.2418
Epoch 5/100
340/340 [==============================] - 0s 325us/sample - loss: 0.2820 - val_loss: 0.2401
Epoch 6/100
340/340 [==============================] - 0s 481us/sample - loss: 0.2351 - val_loss: 0.2278
Epoch 7/100
340/340 [==============================] - 0s 308us/sample - loss: 0.2419 - val_loss: 0.2168
Epoch 8/100
340/340 [==============================] - 0s 327us/sample - loss: 0.2457 - val_loss: 0.2526
Epoch 9/100
340/340 [==============================] - 0s 315us/sample - loss: 0.2355 - val_loss: 0.2179
Epoch 10/100
340/340 [==============================] - 0s 283us/sample - loss: 0.2198 - val_loss: 0.2238
Epoch 11/100
340/340 [==============================] - 0s 472us/sample - loss: 0.2420 - val_loss: 0.2491
Epoch 12/100
340/340 [==============================] - 0s 325us/sample - loss: 0.2247 - val_loss: 0.2198
Epoch 13/100
340/340 [==============================] - 0s 318us/sample - loss: 0.2043 - val_loss: 0.2210
Epoch 14/100
340/340 [==============================] - 0s 323us/sample - loss: 0.1979 - val_loss: 0.2402
Epoch 15/100
340/340 [==============================] - 0s 538us/sample - loss: 0.2124 - val_loss: 0.2350
Epoch 16/100
340/340 [==============================] - 0s 345us/sample - loss: 0.2112 - val_loss: 0.2450
Epoch 17/100
340/340 [==============================] - 0s 489us/sample - loss: 0.2002 - val_loss: 0.2357
Epoch 18/100
340/340 [==============================] - 0s 314us/sample - loss: 0.1943 - val_loss: 0.2426
Epoch 19/100
340/340 [==============================] - 0s 248us/sample - loss: 0.2159 - val_loss: 0.2606
Epoch 20/100
340/340 [==============================] - 0s 293us/sample - loss: 0.2025 - val_loss: 0.2484
Epoch 21/100
340/340 [==============================] - 0s 349us/sample - loss: 0.1880 - val_loss: 0.2717
Epoch 22/100
340/340 [==============================] - 0s 305us/sample - loss: 0.2092 - val_loss: 0.2397
Epoch 23/100
340/340 [==============================] - 0s 522us/sample - loss: 0.1772 - val_loss: 0.2641
Epoch 24/100
```

```
340/340 [==============================] - 0s 339us/sample - loss: 0.1724 - val_loss: 0.2454
Epoch 25/100
340/340 [==============================] - 0s 304us/sample - loss: 0.2048 - val_loss: 0.2915
Epoch 26/100
340/340 [==============================] - 0s 335us/sample - loss: 0.1947 - val_loss: 0.2554
Epoch 27/100
340/340 [==============================] - 0s 441us/sample - loss: 0.1815 - val_loss: 0.2648
Epoch 28/100
340/340 [==============================] - 0s 355us/sample - loss: 0.1632 - val_loss: 0.2223
Epoch 29/100
340/340 [==============================] - 0s 358us/sample - loss: 0.1927 - val_loss: 0.2516
Epoch 30/100
340/340 [==============================] - 0s 282us/sample - loss: 0.1849 - val_loss: 0.2526
Epoch 31/100
340/340 [==============================] - 0s 428us/sample - loss: 0.1742 - val_loss: 0.2397
Epoch 32/100
340/340 [==============================] - 0s 338us/sample - loss: 0.2013 - val_loss: 0.2725
Epoch 33/100
340/340 [==============================] - 0s 325us/sample - loss: 0.1623 - val_loss: 0.2401
Epoch 34/100
340/340 [==============================] - 0s 321us/sample - loss: 0.1816 - val_loss: 0.2346
Epoch 35/100
340/340 [==============================] - 0s 492us/sample - loss: 0.1673 - val_loss: 0.2557
Epoch 36/100
340/340 [==============================] - 0s 335us/sample - loss: 0.1646 - val_loss: 0.2433
Epoch 37/100
340/340 [==============================] - 0s 504us/sample - loss: 0.1599 - val_loss: 0.2596
Epoch 38/100
340/340 [==============================] - 0s 318us/sample - loss: 0.1543 - val_loss: 0.2645
Epoch 39/100
340/340 [==============================] - 0s 362us/sample - loss: 0.1337 - val_loss: 0.2833
Epoch 40/100
340/340 [==============================] - 0s 285us/sample - loss: 0.1580 - val_loss: 0.2596
Epoch 41/100
340/340 [==============================] - 0s 509us/sample - loss: 0.1436 - val_loss: 0.2727
Epoch 42/100
340/340 [==============================] - 0s 317us/sample - loss: 0.1543 - val_loss: 0.2580
Epoch 43/100
340/340 [==============================] - 0s 326us/sample - loss: 0.1364 - val_loss: 0.2683
Epoch 44/100
340/340 [==============================] - 0s 503us/sample - loss: 0.1568 - val_loss: 0.2691
Epoch 45/100
340/340 [==============================] - 0s 343us/sample - loss: 0.1475 - val_loss: 0.2649
Epoch 46/100
340/340 [==============================] - 0s 322us/sample - loss: 0.1664 - val_loss: 0.2835
Epoch 47/100
340/340 [==============================] - 0s 507us/sample - loss: 0.1546 - val_loss: 0.2568
Epoch 48/100
340/340 [==============================] - 0s 316us/sample - loss: 0.1511 - val_loss: 0.2627
Epoch 49/100
340/340 [==============================] - 0s 385us/sample - loss: 0.1540 - val_loss: 0.2710
Epoch 50/100
340/340 [==============================] - 0s 335us/sample - loss: 0.1340 - val_loss: 0.2645
```

```
Epoch 51/100
340/340 [==============================] - 0s 413us/sample - loss: 0.1440 - val_loss: 0.2591
Epoch 52/100
340/340 [==============================] - 0s 328us/sample - loss: 0.1272 - val_loss: 0.3024
Epoch 53/100
340/340 [==============================] - 0s 324us/sample - loss: 0.1370 - val_loss: 0.2755
Epoch 54/100
340/340 [==============================] - 0s 486us/sample - loss: 0.1519 - val_loss: 0.2869
Epoch 55/100
340/340 [==============================] - 0s 318us/sample - loss: 0.1403 - val_loss: 0.2546
Epoch 56/100
340/340 [==============================] - 0s 326us/sample - loss: 0.1498 - val_loss: 0.2789
Epoch 57/100
340/340 [==============================] - 0s 331us/sample - loss: 0.1504 - val_loss: 0.2819
Epoch 58/100
340/340 [==============================] - 0s 513us/sample - loss: 0.1453 - val_loss: 0.2739
Epoch 59/100
340/340 [==============================] - 0s 341us/sample - loss: 0.1356 - val_loss: 0.2641
Epoch 60/100
340/340 [==============================] - 0s 317us/sample - loss: 0.1466 - val_loss: 0.2697
Epoch 61/100
340/340 [==============================] - 0s 489us/sample - loss: 0.1270 - val_loss: 0.2687
Epoch 62/100
340/340 [==============================] - 0s 320us/sample - loss: 0.1521 - val_loss: 0.2842
Epoch 63/100
340/340 [==============================] - 0s 307us/sample - loss: 0.1253 - val_loss: 0.2758
Epoch 64/100
340/340 [==============================] - 0s 304us/sample - loss: 0.1248 - val_loss: 0.2766
Epoch 65/100
340/340 [==============================] - 0s 319us/sample - loss: 0.1318 - val_loss: 0.2956
Epoch 66/100
340/340 [==============================] - 0s 493us/sample - loss: 0.1311 - val_loss: 0.2689
Epoch 67/100
340/340 [==============================] - 0s 295us/sample - loss: 0.1228 - val_loss: 0.2959
Epoch 68/100
340/340 [==============================] - 0s 315us/sample - loss: 0.1206 - val_loss: 0.2925
Epoch 69/100
340/340 [==============================] - 0s 300us/sample - loss: 0.1254 - val_loss: 0.3065
Epoch 70/100
340/340 [==============================] - 0s 319us/sample - loss: 0.1391 - val_loss: 0.2888
Epoch 71/100
340/340 [==============================] - 0s 357us/sample - loss: 0.1346 - val_loss: 0.2660
Epoch 72/100
340/340 [==============================] - 0s 428us/sample - loss: 0.1128 - val_loss: 0.2715
Epoch 73/100
340/340 [==============================] - 0s 332us/sample - loss: 0.1058 - val_loss: 0.2796
Epoch 74/100
340/340 [==============================] - 0s 342us/sample - loss: 0.1204 - val_loss: 0.2804
Epoch 75/100
340/340 [==============================] - 0s 310us/sample - loss: 0.1235 - val_loss: 0.2953
Epoch 76/100
340/340 [==============================] - 0s 325us/sample - loss: 0.1198 - val_loss: 0.2651
Epoch 77/100
```

```
340/340 [==============================] - 0s 440us/sample - loss: 0.1105 - val_loss: 0.2743
Epoch 78/100
340/340 [==============================] - 0s 323us/sample - loss: 0.1229 - val_loss: 0.2889
Epoch 79/100
340/340 [==============================] - 0s 328us/sample - loss: 0.1103 - val_loss: 0.2920
Epoch 80/100
340/340 [==============================] - 0s 512us/sample - loss: 0.0951 - val_loss: 0.2774
Epoch 81/100
340/340 [==============================] - 0s 350us/sample - loss: 0.1085 - val_loss: 0.3101
Epoch 82/100
340/340 [==============================] - 0s 319us/sample - loss: 0.1327 - val_loss: 0.2941
Epoch 83/100
340/340 [==============================] - 0s 481us/sample - loss: 0.1233 - val_loss: 0.3051
Epoch 84/100
340/340 [==============================] - 0s 355us/sample - loss: 0.1132 - val_loss: 0.2862
Epoch 85/100
340/340 [==============================] - 0s 301us/sample - loss: 0.1110 - val_loss: 0.2706
Epoch 86/100
340/340 [==============================] - 0s 317us/sample - loss: 0.1132 - val_loss: 0.2796
Epoch 87/100
340/340 [==============================] - 0s 489us/sample - loss: 0.1083 - val_loss: 0.2850
Epoch 88/100
340/340 [==============================] - 0s 320us/sample - loss: 0.1012 - val_loss: 0.2781
Epoch 89/100
340/340 [==============================] - 0s 347us/sample - loss: 0.1039 - val_loss: 0.2842
Epoch 90/100
340/340 [==============================] - 0s 327us/sample - loss: 0.1075 - val_loss: 0.2824
Epoch 91/100
340/340 [==============================] - 0s 465us/sample - loss: 0.1106 - val_loss: 0.2709
Epoch 92/100
340/340 [==============================] - 0s 347us/sample - loss: 0.1071 - val_loss: 0.3062
Epoch 93/100
340/340 [==============================] - 0s 351us/sample - loss: 0.1118 - val_loss: 0.2802
Epoch 94/100
340/340 [==============================] - 0s 256us/sample - loss: 0.0931 - val_loss: 0.2703
Epoch 95/100
340/340 [==============================] - 0s 305us/sample - loss: 0.1031 - val_loss: 0.2651
Epoch 96/100
340/340 [==============================] - 0s 476us/sample - loss: 0.1010 - val_loss: 0.2887
Epoch 97/100
340/340 [==============================] - 0s 329us/sample - loss: 0.1050 - val_loss: 0.2942
Epoch 98/100
340/340 [==============================] - 0s 325us/sample - loss: 0.1008 - val_loss: 0.3051
Epoch 99/100
340/340 [==============================] - 0s 518us/sample - loss: 0.1028 - val_loss: 0.2800
Epoch 100/100
340/340 [==============================] - 0s 321us/sample - loss: 0.1034 - val_loss: 0.2972
```

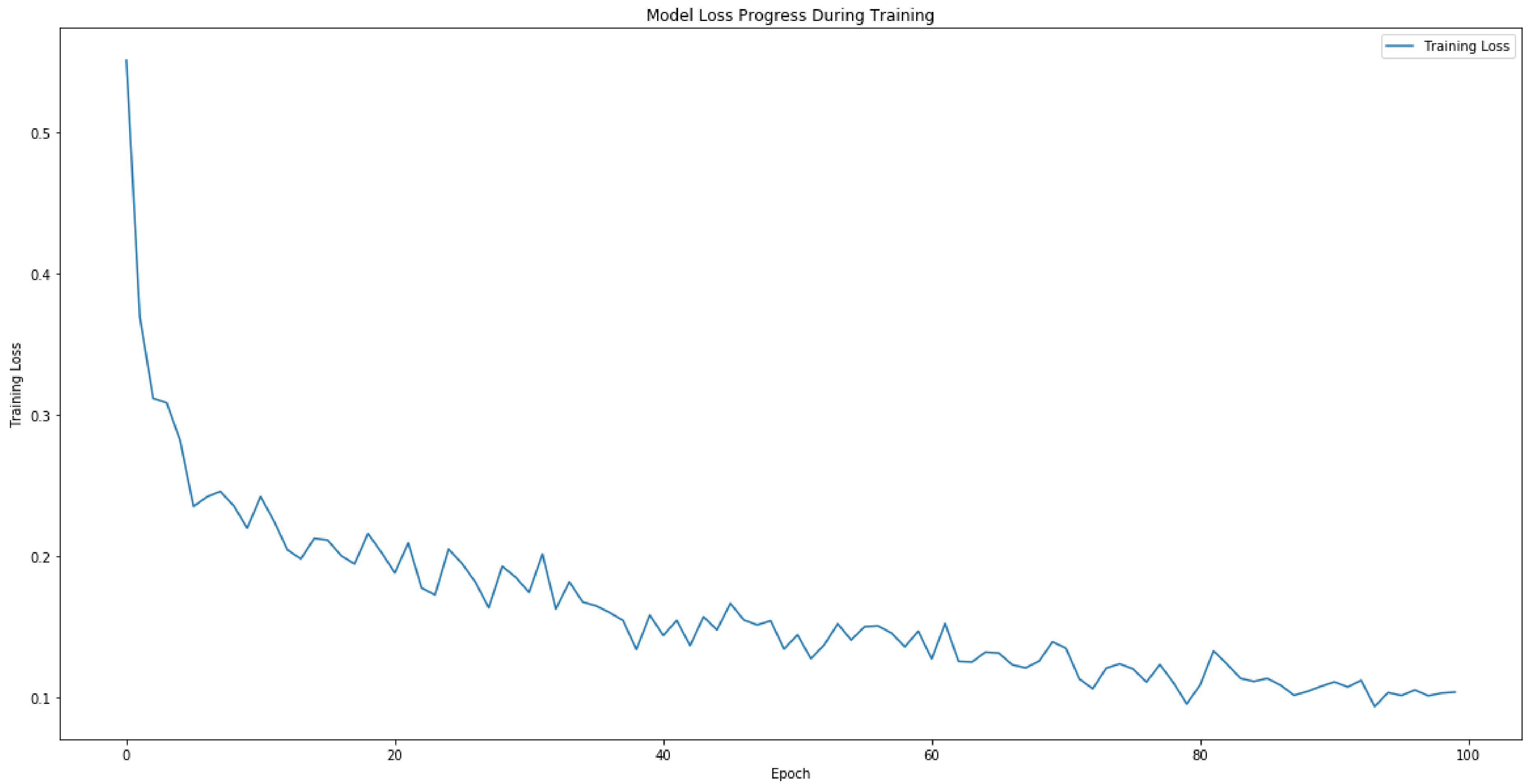
In [29]:
```
result = ANN_model.evaluate(X_test, y_test)
accuracy_ANN = 1 - result
print("Accuracy : {}".format(accuracy_ANN))
```

```
75/75 [==============================] - 0s 90us/sample - loss: 0.3057
Accuracy : 0.6943431389331818
```

In [30]:
```
epochs_hist.history.keys()
```

Out[30]:
```
dict_keys(['loss', 'val_loss'])
```

In [31]:
```
plt.figure(figsize = (20, 10))
plt.plot(epochs_hist.history["loss"])
plt.title("Model Loss Progress During Training")
plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.legend(["Training Loss"])
plt.show()
```



TRAIN AND EVALUATE A DECISION TREE AND RANDOM FOREST MODELS

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 incrementally developed. The final result is a tree with decision nodes and leaf nodes.

In [32]:
```
from sklearn.tree import DecisionTreeRegressor
DecisionTree_model = DecisionTreeRegressor()
DecisionTree_model.fit(X_train, y_train)
```

Out[32]:
```
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='best')
```

In [33]:
```
accuracy_DecisionTree = DecisionTree_model.score(X_test, y_test)
accuracy_DecisionTree
```

Out[33]:
```
0.6206917938492921
```

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 uses averaging to improve the predictive accuracy and control over-fitting.

In [34]:
```
from sklearn.ensemble import RandomForestRegressor
RandomForest_model = RandomForestRegressor(n_estimators = 100, max_depth = 10)
RandomForest_model.fit(X_train, y_train)
```

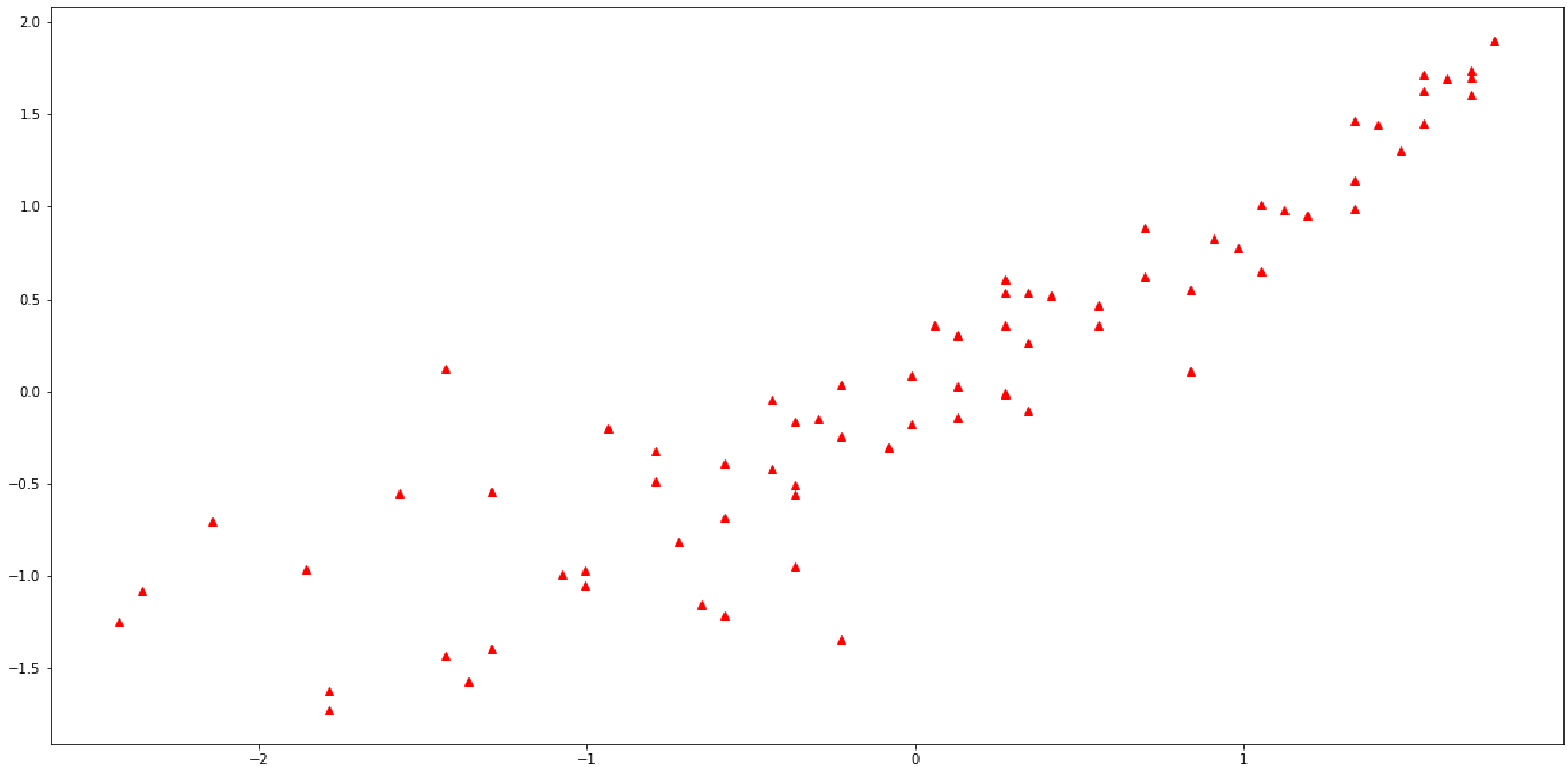
Out[34]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [35]:
```
accuracy_RandomForest = RandomForest_model.score(X_test, y_test)
accuracy_RandomForest
```

Out[35]:
```
0.7996469037033596
```

REGRESSION MODEL KPIs

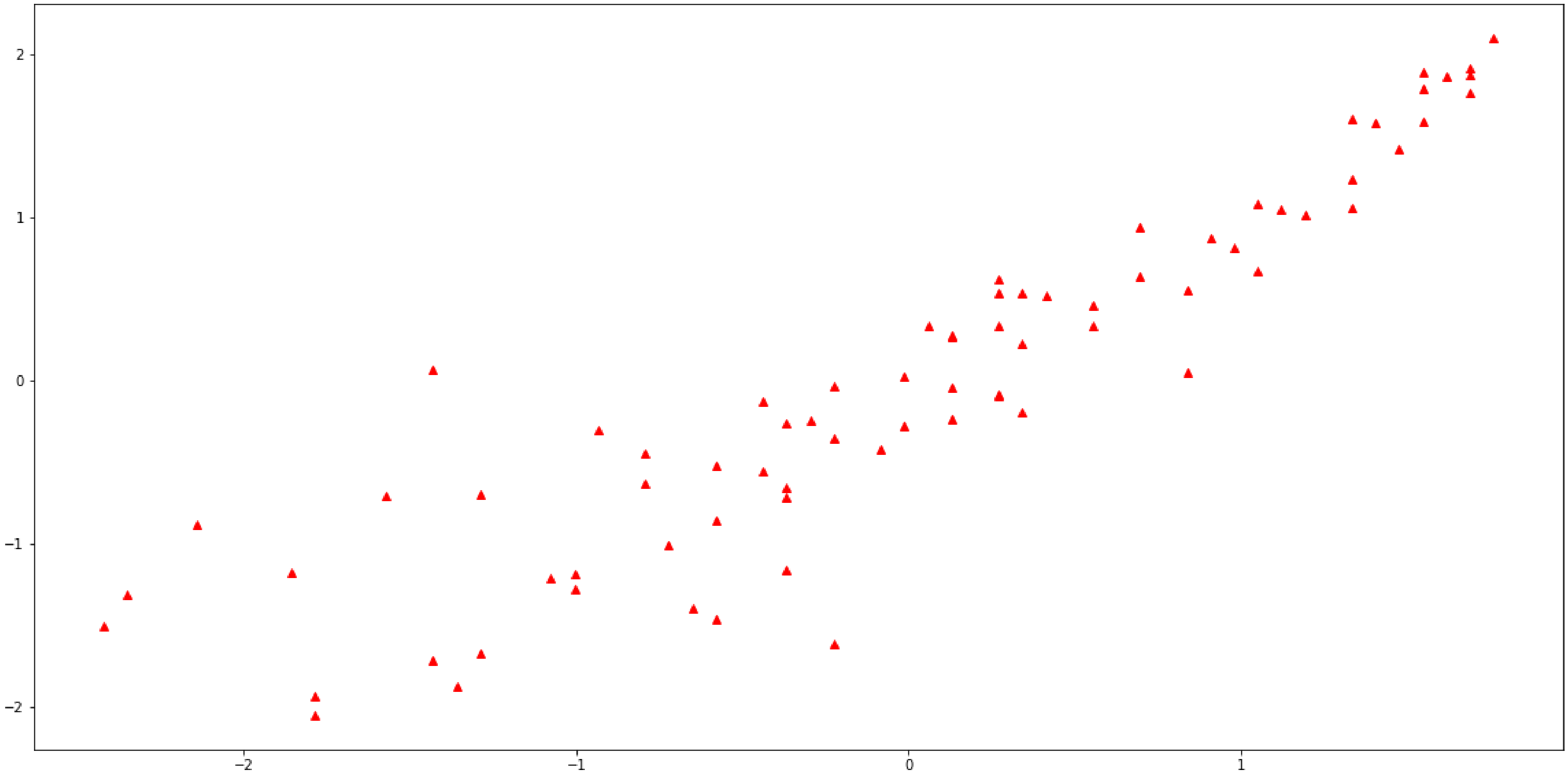
In [36]:
```
plt.figure(figsize = (20, 10))
y_predict = LinearRegression_model.predict(X_test)
plt.plot(y_test, y_predict, "^", color = "r")
plt.show()
```



In [37]:
```
y_predict_original1 = scaler_y.fit_transform(y_predict)
y_test_original1 = scaler_y.fit_transform(y_test)
```

In [38]:
```
y_predict_original = scaler_y.inverse_transform(y_predict_original1)
y_test_original = scaler_y.inverse_transform(y_test_original1)
```

In [39]:
```
plt.figure(figsize=(20, 10))
plt.plot(y_test_original, y_predict_original, "^", color = "r")
plt.show()
```



In [40]:
```
k = X_test.shape[1]
n = len(X_test)
n
```

Out[40]: 75

In [41]:
```
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_original, y_predict_original)),".3f"))
MSE = mean_squared_error(y_test_original, y_predict_original)
MAE = mean_absolute_error(y_test_original, y_predict_original)
r2 = r2_score(y_test_original, y_predict_original)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

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

```
RMSE = 0.461
MSE = 0.21295089052901162
MAE = 0.33882897338781265
R2 = 0.8176551953314685
Adjusted R2 = 0.7986042455899801
```