Software Development for A.I.

Title: **Health Insurance Premium Prediction using AWS Cloud**

Important Libraries

1. Numpy

- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2. Pandas

- Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series

3. Seaborn

- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

4. scikit-learn

- Scikit-learn is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem

5. TensorFlow

- TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

In [2]:
```
#Importing the required libraries.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
# Reading the Dataset
insurance = pd.read_csv('insurance.csv')
```

In [4]:
```
# Quick glance at the data
insurance
```

Out[4]:

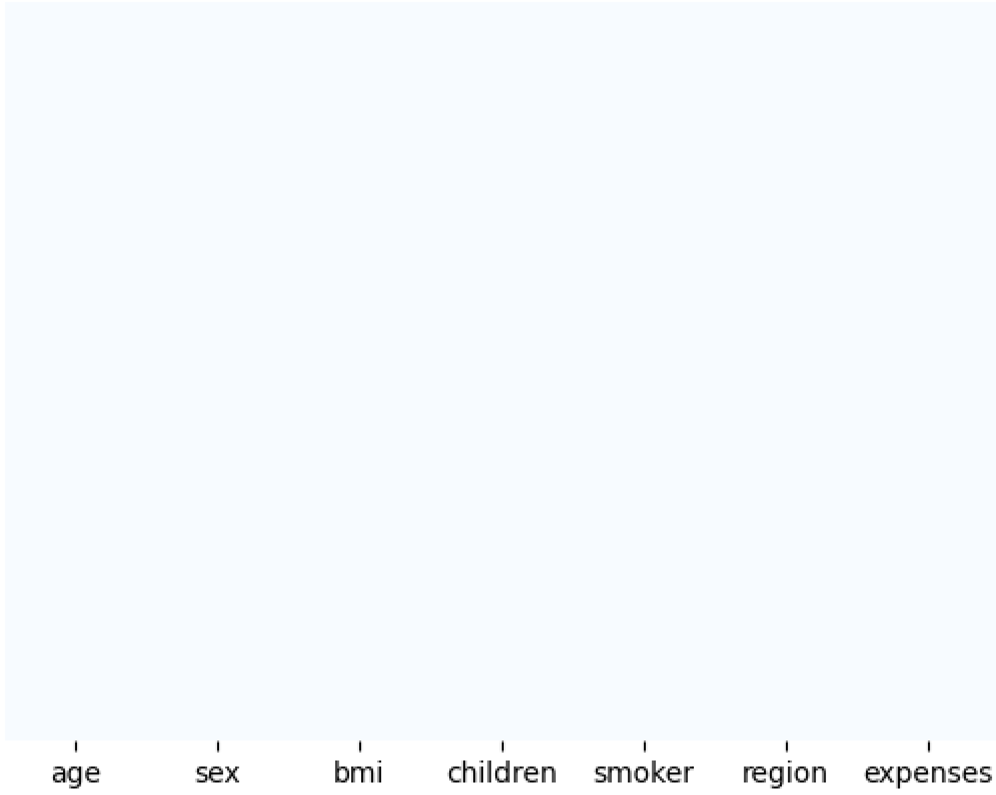
	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86
...	...	...	...	...	...	...	...
1333	50	male	31.0	3	no	northwest	10600.55
1334	18	female	31.9	0	no	northeast	2205.98
1335	18	female	36.9	0	no	southeast	1629.83
1336	21	female	25.8	0	no	southwest	2007.95
1337	61	female	29.1	0	yes	northwest	29141.36

1338 rows × 7 columns

Performing Exploratory Data Analysis

In [5]:
```
#Checking for null values in the Dataset
sns.heatmap(insurance.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[5]: `<AxesSubplot: >`



In [6]:
```
# Grouping by region to see any relationship between region and charges
region = insurance.groupby(by='region').mean()
region
# It reveals that the south east region has higher BMI values when compared to northeast
```

```
/tmp/ipykernel_105/2587700523.py:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is dep
recated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns
which should be valid for the function.
  region = insurance.groupby(by='region').mean()
```

Out[6]:

	age	bmi	children	expenses
region				
northeast	39.268519	29.176235	1.046296	13406.384691
northwest	39.196923	29.201846	1.147692	12417.575169
southeast	38.939560	33.359341	1.049451	14735.411538
southwest	39.455385	30.596615	1.141538	12346.937908

In [7]:
```
# Grouping by age
age = insurance.groupby(by='age').mean()
age
```

```
/tmp/ipykernel_105/2356038905.py:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is dep
recated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns
which should be valid for the function.
  age = insurance.groupby(by='age').mean()
```

Out[7]:

age	bmi	children	expenses
18	31.333333	0.449275	7086.217971
19	28.598529	0.426471	9747.909706
20	30.627586	0.862069	10159.697931
21	28.189286	0.785714	4730.464286
22	31.092857	0.714286	10012.932857
23	31.460714	1.000000	12419.820357
24	29.142857	0.464286	10648.015714
25	29.689286	1.285714	9838.365000
26	29.435714	1.071429	6133.825714
27	29.342857	0.964286	12184.701429
28	29.482143	1.285714	9069.187500
29	29.374074	1.259259	10430.159630
30	30.555556	1.555556	12719.111111
31	29.922222	1.407407	10196.980000
32	31.588462	1.269231	9220.299615
33	31.165385	1.538462	12351.532308
34	30.273077	1.153846	11613.528462
35	31.392000	1.680000	11307.183200
36	29.368000	1.240000	12204.477600
37	31.216000	1.520000	18019.911600
38	29.004000	1.480000	8102.732800
39	29.908000	2.200000	11778.243600
40	30.144444	1.592593	11772.251481
41	31.518519	1.407407	9653.745556
42	30.337037	1.000000	13061.038519
43	30.207407	1.629630	19267.279630
44	30.848148	1.222222	15859.397037
45	29.782759	1.482759	14830.199310
46	31.341379	1.620690	14342.591379
47	30.655172	1.379310	17653.999655
48	31.927586	1.310345	14632.500000
49	30.314286	1.500000	12696.006071
50	31.134483	1.310345	15663.003103
51	31.731034	1.103448	15682.255517
52	32.941379	1.482759	18256.270345
53	30.371429	1.250000	16020.930357
54	31.232143	1.428571	18758.546429
55	31.950000	0.961538	16164.545000
56	31.600000	0.769231	15025.517692
57	30.838462	0.615385	16447.186154
58	32.728000	0.240000	13878.928000
59	30.576000	1.200000	18895.869600
60	30.330435	0.347826	21979.419130
61	32.552174	0.739130	22024.457391
62	32.347826	0.565217	19163.856087
63	31.930435	0.565217	19884.999130
64	32.986364	0.772727	23275.530909

In [8]:

```
# Conversion of categorical value to numerical value to avoid training issues with the machine learning model
insurance['sex'] = insurance['sex'].apply(lambda x: 0 if x == 'female' else 1)
insurance['smoker'] = insurance['smoker'].apply(lambda x: 1 if x == 'yes' else 0)
```

In [9]:

```
# Check unique values in 'region' column
insurance['region'].unique()
```

Out[9]: array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)

In [10]:
```
region_dummies = pd.get_dummies(insurance['region'], drop_first = True)
```

In [11]:
```
insurance = pd.concat([insurance, region_dummies], axis = 1)
```

In [12]:
```
insurance.describe()
```

Out[12]:

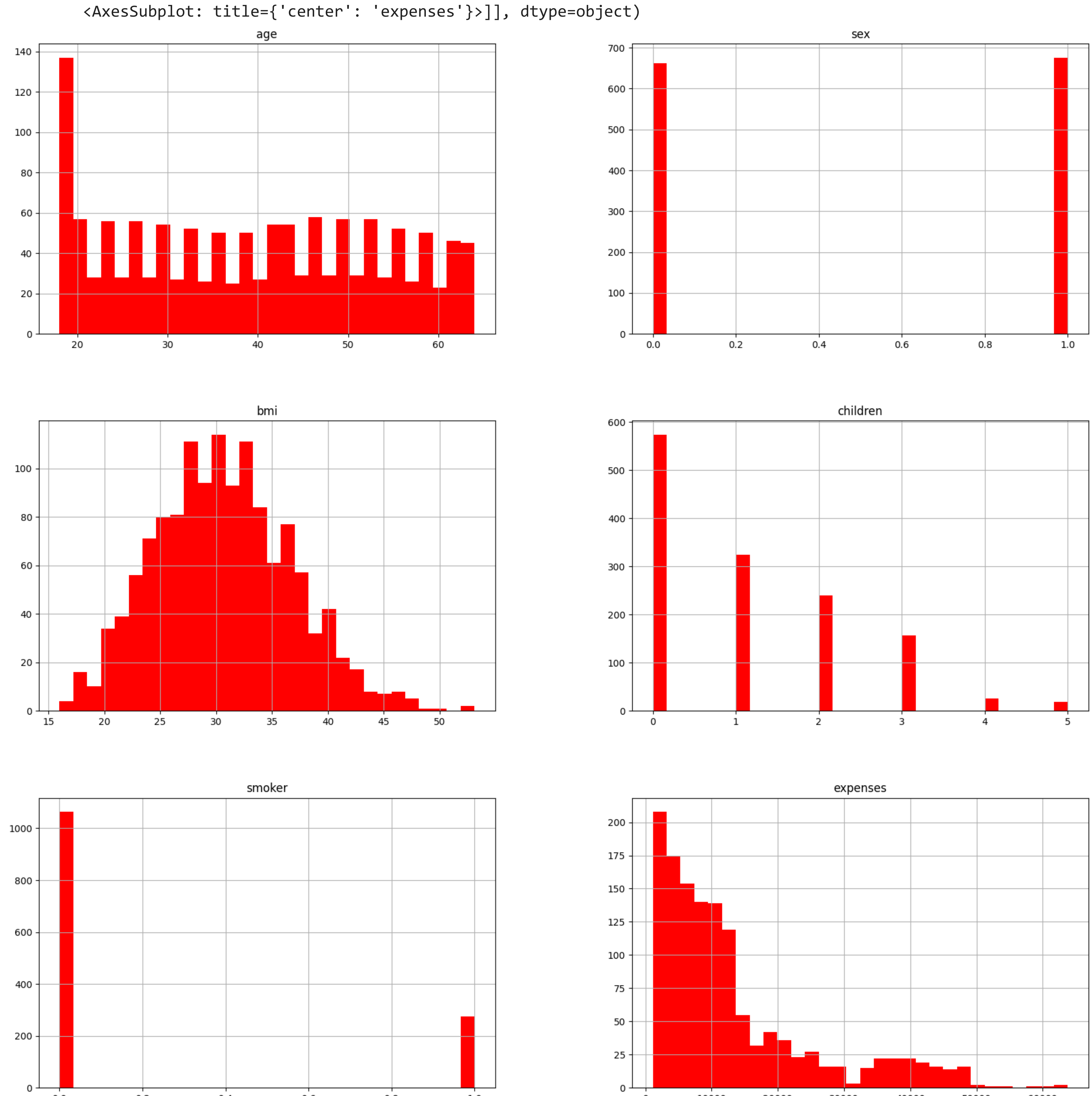
	age	sex	bmi	children	smoker	expenses	northwest	southeast	southwest
count	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	0.505232	30.665471	1.094918	0.204783	13270.422414	0.242900	0.272048	0.242900
std	14.049960	0.500160	6.098382	1.205493	0.403694	12110.011240	0.428995	0.445181	0.428995
min	18.000000	0.000000	16.000000	0.000000	0.000000	1121.870000	0.000000	0.000000	0.000000
25%	27.000000	0.000000	26.300000	0.000000	0.000000	4740.287500	0.000000	0.000000	0.000000
50%	39.000000	1.000000	30.400000	1.000000	0.000000	9382.030000	0.000000	0.000000	0.000000
75%	51.000000	1.000000	34.700000	2.000000	0.000000	16639.915000	0.000000	1.000000	0.000000
max	64.000000	1.000000	53.100000	5.000000	1.000000	63770.430000	1.000000	1.000000	1.000000

Plotting the Dataset graph using Seaborn

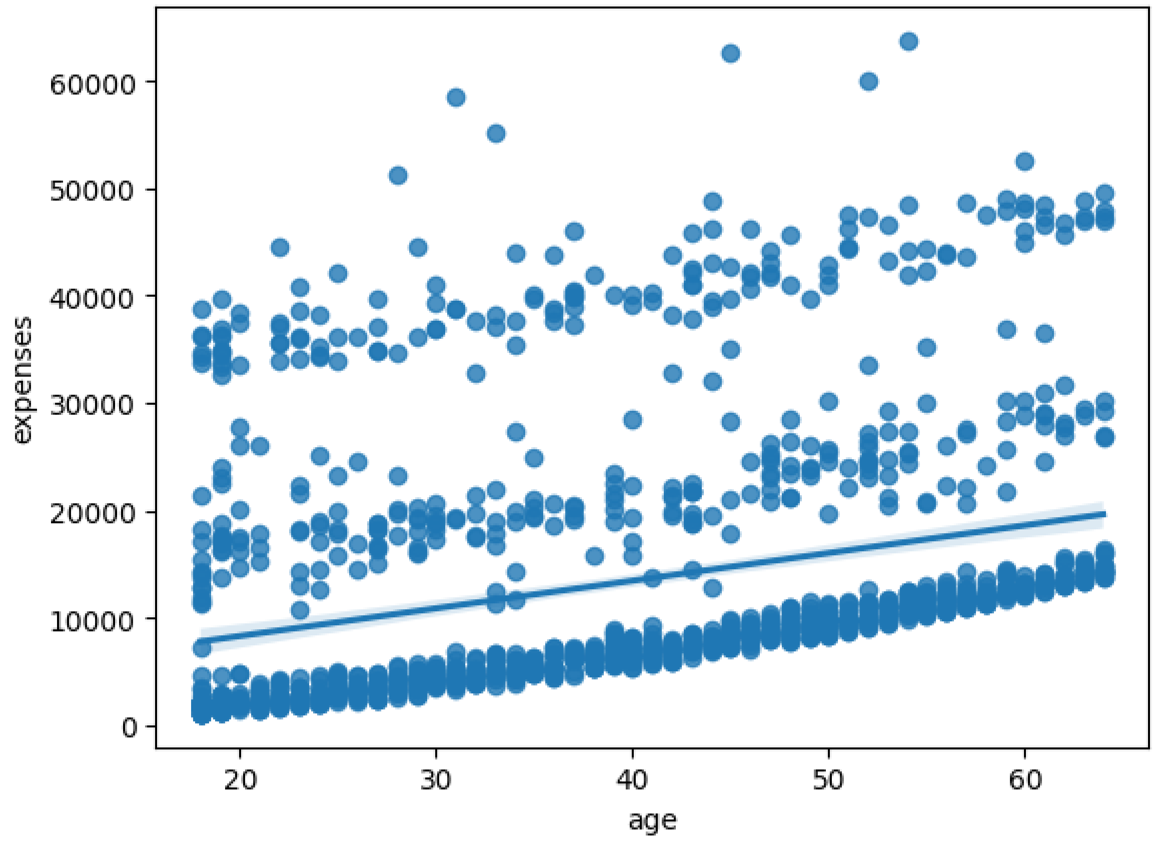
1. Graphs showing the overview of aspects like Age, Sex, BMI, Children, Smoker, Expenses

In [13]:
```
insurance[['age', 'sex', 'bmi', 'children', 'smoker', 'expenses']].hist(bins = 30, figsize = (20,20), color = 'r')
```

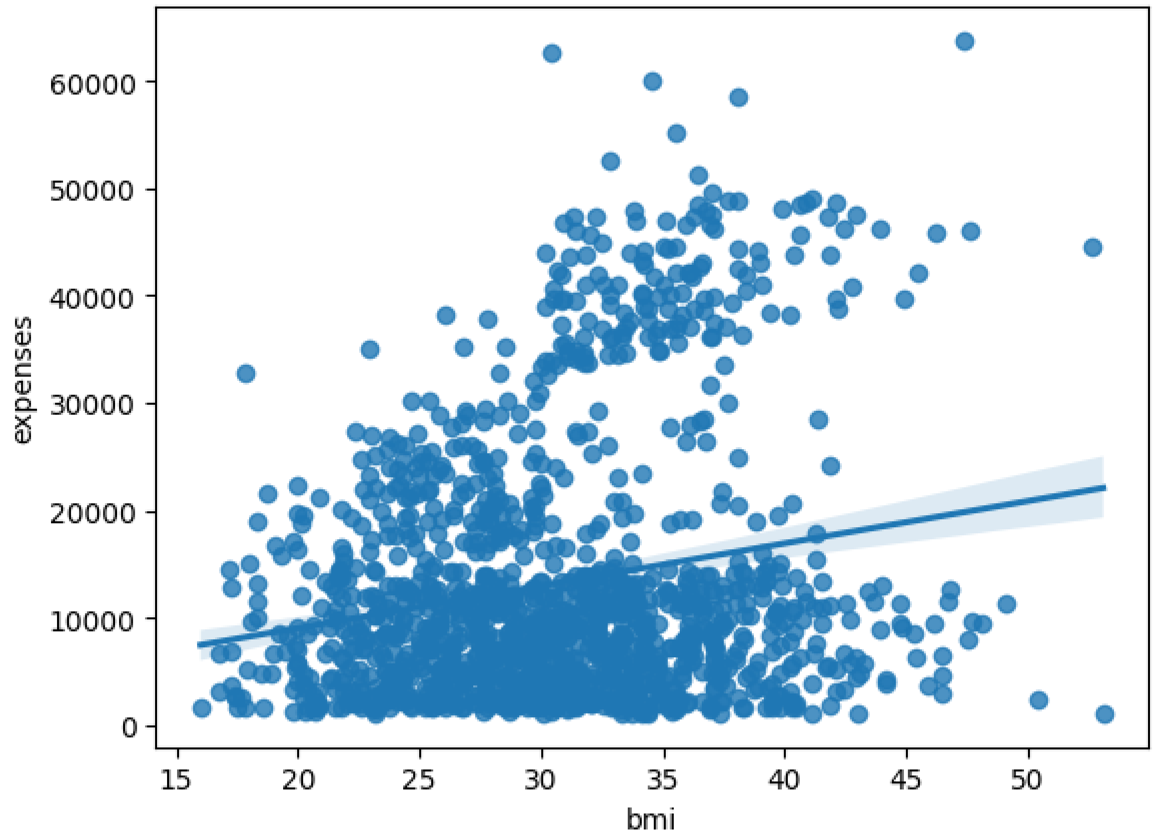
Out[13]:
```
array([[<AxesSubplot: title={'center': 'age'}>,
        <AxesSubplot: title={'center': 'sex'}>],
       [<AxesSubplot: title={'center': 'bmi'}>,
        <AxesSubplot: title={'center': 'children'}>],
       [<AxesSubplot: title={'center': 'smoker'}>,
        <AxesSubplot: title={'center': 'expenses'}>]], dtype=object)
```



In [14]:
```
# Plotting age and expenses
sns.regplot(x = 'age', y = 'expenses', data = insurance)
plt.show()
```



In [15]:
```
# Plotting BMI and expenses
sns.regplot(x = 'bmi', y = 'expenses', data = insurance)
plt.show()
```



In [16]:
```
# Finding the corealation among them
corelation = insurance.corr()
corelation
```

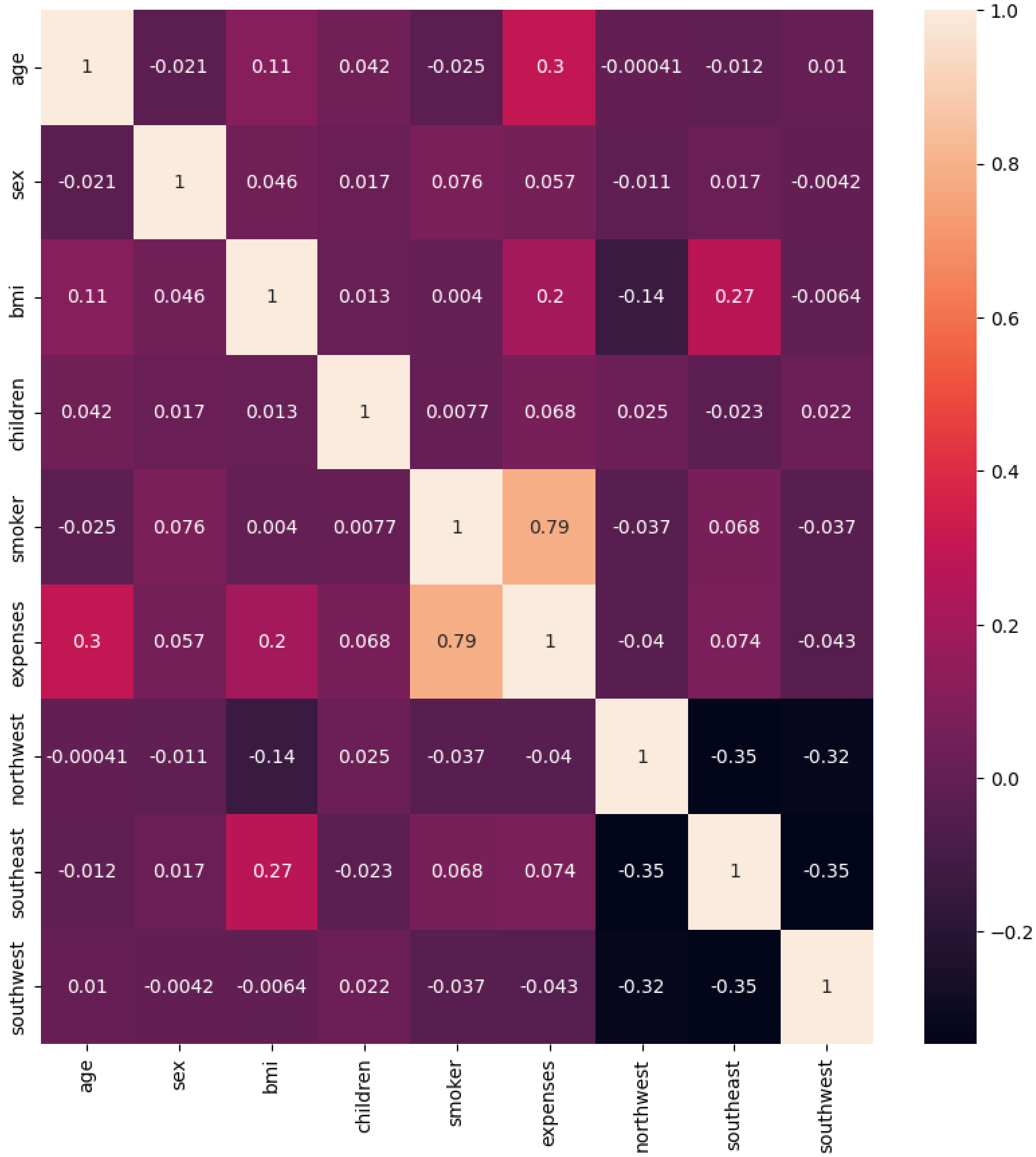
```
/tmp/ipykernel_105/2549767591.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to
silence this warning.
  corelation = insurance.corr()
```

Out[16]:

	age	sex	bmi	children	smoker	expenses	northwest	southeast	southwest
age	1.000000	-0.020856	0.109341	0.042469	-0.025019	0.299008	-0.000407	-0.011642	0.010016
sex	-0.020856	1.000000	0.046380	0.017163	0.076185	0.057292	-0.011156	0.017117	-0.004184
bmi	0.109341	0.046380	1.000000	0.012645	0.003968	0.198576	-0.135992	0.270144	-0.006398
children	0.042469	0.017163	0.012645	1.000000	0.007673	0.067998	0.024806	-0.023066	0.021914
smoker	-0.025019	0.076185	0.003968	0.007673	1.000000	0.787251	-0.036945	0.068498	-0.036945
expenses	0.299008	0.057292	0.198576	0.067998	0.787251	1.000000	-0.039905	0.073982	-0.043210
northwest	-0.000407	-0.011156	-0.135992	0.024806	-0.036945	-0.039905	1.000000	-0.346265	-0.320829
southeast	-0.011642	0.017117	0.270144	-0.023066	0.068498	0.073982	-0.346265	1.000000	-0.346265
southwest	0.010016	-0.004184	-0.006398	0.021914	-0.036945	-0.043210	-0.320829	-0.346265	1.000000

In [ ]:
```
plt.figure(figsize = (10,10))
sns.heatmap(corelation , annot = True)
```

Out[ ]: <AxesSubplot: >



In [ ]:
```
#Dropping Region coloumn
insurance.drop(['region'], axis = 1, inplace = True)
```

In [ ]:
```
X = insurance.drop(columns =['expenses'])
y = insurance['expenses']
```

In [ ]:
```
# Only take the numerical variables and scale them
X
```

Out[ ]:

	age	sex	bmi	children	smoker	northwest	southeast	southwest
0	19	0	27.9	0	1	0	0	1
1	18	1	33.8	1	0	0	1	0
2	28	1	33.0	3	0	0	1	0
3	33	1	22.7	0	0	1	0	0
4	32	1	28.9	0	0	1	0	0
...	...	...	...	...	...	...	...	...
1333	50	1	31.0	3	0	1	0	0
1334	18	0	31.9	0	0	0	0	0
1335	18	0	36.9	0	0	0	1	0
1336	21	0	25.8	0	0	0	0	1
1337	61	0	29.1	0	1	1	0	0

1338 rows × 8 columns

In [ ]:
```
y
```

```
Out[ ]: 0        16884.92
        1         1725.55
        2         4449.46
        3        21984.47
        4         3866.86
                    ...
        1333    10600.55
        1334     2205.98
        1335     1629.83
        1336     2007.95
        1337    29141.36
        Name: expenses, Length: 1338, dtype: float64
```

```
In [31]: print('Shape of X:')
         X.shape
```

```
Shape of X:
Out[31]: (1338, 8)
```

```
In [32]: print('Shape of Y:')
         y.shape
```

```
Shape of Y:
Out[32]: (1338,)
```

```
In [33]: # Converting into Numpy Array
         X = np.array(X).astype('float32')
         y = np.array(y).astype('float32')
```

```
In [34]: y = y.reshape(-1,1)
```

```
In [35]: y.shape
```

```
Out[35]: (1338, 1)
```

```
In [36]: X
```

```
Out[36]: array([[19. ,  0. , 27.9, ...,  0. ,  0. ,  1. ],
                [18. ,  1. , 33.8, ...,  0. ,  1. ,  0. ],
                [28. ,  1. , 33. , ...,  0. ,  1. ,  0. ],
                ...,
                [18. ,  0. , 36.9, ...,  0. ,  1. ,  0. ],
                [21. ,  0. , 25.8, ...,  0. ,  0. ,  1. ],
                [61. ,  0. , 29.1, ...,  1. ,  0. ,  0. ]], dtype=float32)
```

Model Building

```
In [37]: from sklearn.preprocessing import StandardScaler, MinMaxScaler

         scaler_x = StandardScaler()
         X = scaler_x.fit_transform(X)

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

```
In [38]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
In [39]: X_train.shape
```

```
Out[39]: (936, 8)
```

```
In [40]: X_test.shape
```

```
Out[40]: (402, 8)
```

```
In [41]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error, accuracy_score

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

```
Out[41]: ▾ LinearRegression
         LinearRegression()
```

```
In [42]: regresssion_model_sklearn_accuracy = regresssion_model_sklearn.score(X_test, y_test)
         print("regression model accuracy is " + str(regresssion_model_sklearn_accuracy*100))
```

```
regression model accuracy is 72.29441537204815
```

```
In [43]: y_predict = regresssion_model_sklearn.predict(X_test)
```

In [44]:
```
y_predict_orig = scaler_y.inverse_transform(y_predict)
y_test_orig = scaler_y.inverse_transform(y_test)
```

In [45]:
```
k = X_test.shape[1]
n = len(X_test)
n
```

Out[45]: 402

Calculating metrics for our regression model

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)),'.3f'))
MSE = mean_squared_error(y_test_orig, y_predict_orig)
```

In [47]:
```
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)
```

In [48]:
```
print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 6433.626
MSE = 41391544.0
MAE = 4358.8115
R2 = 0.7229441512449756
Adjusted R2 = 0.7173043375298606
```

Using Artificial Neural Network

In [106…
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
```

In [107…
```
ANN_model = keras.Sequential()
ANN_model.add(Dense(50, input_dim = 8))
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)                450

 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: 38,351
Trainable params: 38,351
Non-trainable params: 0
_________________________________________________________________
```

In [108…
```
ANN_model.compile(optimizer='Adam', loss='mean_squared_error')
```

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

```
Epoch 1/100
38/38 [==============================] - 3s 13ms/step - loss: 0.7304 - val_loss: 0.4560
Epoch 2/100
38/38 [==============================] - 0s 8ms/step - loss: 0.4466 - val_loss: 0.3254
Epoch 3/100
38/38 [==============================] - 0s 7ms/step - loss: 0.3499 - val_loss: 0.2671
Epoch 4/100
38/38 [==============================] - 0s 8ms/step - loss: 0.2580 - val_loss: 0.2449
Epoch 5/100
38/38 [==============================] - 0s 6ms/step - loss: 0.2391 - val_loss: 0.2801
Epoch 6/100
38/38 [==============================] - 0s 9ms/step - loss: 0.2409 - val_loss: 0.2369
Epoch 7/100
38/38 [==============================] - 0s 7ms/step - loss: 0.2400 - val_loss: 0.2505
Epoch 8/100
38/38 [==============================] - 0s 8ms/step - loss: 0.2139 - val_loss: 0.2317
Epoch 9/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1907 - val_loss: 0.2389
Epoch 10/100
38/38 [==============================] - 0s 7ms/step - loss: 0.2154 - val_loss: 0.2594
Epoch 11/100
38/38 [==============================] - 0s 5ms/step - loss: 0.2221 - val_loss: 0.2371
Epoch 12/100
38/38 [==============================] - 0s 4ms/step - loss: 0.2163 - val_loss: 0.2262
Epoch 13/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1900 - val_loss: 0.2315
Epoch 14/100
38/38 [==============================] - 0s 5ms/step - loss: 0.2024 - val_loss: 0.2383
Epoch 15/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1876 - val_loss: 0.2616
Epoch 16/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1945 - val_loss: 0.2599
Epoch 17/100
38/38 [==============================] - 0s 6ms/step - loss: 0.2015 - val_loss: 0.2280
Epoch 18/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1746 - val_loss: 0.2302
Epoch 19/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1910 - val_loss: 0.2660
Epoch 20/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1824 - val_loss: 0.2292
Epoch 21/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1899 - val_loss: 0.2697
Epoch 22/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1831 - val_loss: 0.2364
Epoch 23/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1776 - val_loss: 0.2256
Epoch 24/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1792 - val_loss: 0.2385
Epoch 25/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1743 - val_loss: 0.2226
Epoch 26/100
38/38 [==============================] - 0s 8ms/step - loss: 0.1773 - val_loss: 0.2464
Epoch 27/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1771 - val_loss: 0.2371
Epoch 28/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1713 - val_loss: 0.2277
Epoch 29/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1578 - val_loss: 0.2332
Epoch 30/100
38/38 [==============================] - 0s 11ms/step - loss: 0.1628 - val_loss: 0.2376
Epoch 31/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1727 - val_loss: 0.2319
Epoch 32/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1724 - val_loss: 0.2431
Epoch 33/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1824 - val_loss: 0.2573
Epoch 34/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1725 - val_loss: 0.2634
Epoch 35/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1680 - val_loss: 0.2571
Epoch 36/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1679 - val_loss: 0.2473
Epoch 37/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1589 - val_loss: 0.2641
Epoch 38/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1700 - val_loss: 0.2468
Epoch 39/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1626 - val_loss: 0.2688
Epoch 40/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1545 - val_loss: 0.2405
Epoch 41/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1672 - val_loss: 0.2483
Epoch 42/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1549 - val_loss: 0.2508
Epoch 43/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1476 - val_loss: 0.2671
Epoch 44/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1509 - val_loss: 0.2441
Epoch 45/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1449 - val_loss: 0.2639
```

```
Epoch 46/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1507 - val_loss: 0.2396
Epoch 47/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1469 - val_loss: 0.2594
Epoch 48/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1468 - val_loss: 0.2399
Epoch 49/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1468 - val_loss: 0.2372
Epoch 50/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1490 - val_loss: 0.2361
Epoch 51/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1471 - val_loss: 0.2685
Epoch 52/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1544 - val_loss: 0.2461
Epoch 53/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1378 - val_loss: 0.2461
Epoch 54/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1402 - val_loss: 0.2313
Epoch 55/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1384 - val_loss: 0.2526
Epoch 56/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1518 - val_loss: 0.2530
Epoch 57/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1467 - val_loss: 0.2474
Epoch 58/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1491 - val_loss: 0.2343
Epoch 59/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1411 - val_loss: 0.2373
Epoch 60/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1254 - val_loss: 0.2591
Epoch 61/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1502 - val_loss: 0.2416
Epoch 62/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1383 - val_loss: 0.2400
Epoch 63/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1353 - val_loss: 0.2368
Epoch 64/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1461 - val_loss: 0.2644
Epoch 65/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1375 - val_loss: 0.2285
Epoch 66/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1435 - val_loss: 0.2368
Epoch 67/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1362 - val_loss: 0.2404
Epoch 68/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1384 - val_loss: 0.2491
Epoch 69/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1392 - val_loss: 0.2535
Epoch 70/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1408 - val_loss: 0.2424
Epoch 71/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1351 - val_loss: 0.2411
Epoch 72/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1253 - val_loss: 0.2396
Epoch 73/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1242 - val_loss: 0.2428
Epoch 74/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1472 - val_loss: 0.2834
Epoch 75/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1459 - val_loss: 0.2404
Epoch 76/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1260 - val_loss: 0.2402
Epoch 77/100
38/38 [==============================] - 0s 8ms/step - loss: 0.1369 - val_loss: 0.2677
Epoch 78/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1337 - val_loss: 0.2439
Epoch 79/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1292 - val_loss: 0.2493
Epoch 80/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1306 - val_loss: 0.2390
Epoch 81/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1279 - val_loss: 0.2368
Epoch 82/100
38/38 [==============================] - 0s 8ms/step - loss: 0.1464 - val_loss: 0.2760
Epoch 83/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1301 - val_loss: 0.2448
Epoch 84/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1270 - val_loss: 0.2354
Epoch 85/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1227 - val_loss: 0.2530
Epoch 86/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1382 - val_loss: 0.2448
Epoch 87/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1313 - val_loss: 0.2354
Epoch 88/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1270 - val_loss: 0.2410
Epoch 89/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1268 - val_loss: 0.2453
Epoch 90/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1354 - val_loss: 0.2420
```

```
Epoch 91/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1393 - val_loss: 0.2456
Epoch 92/100
38/38 [==============================] - 0s 4ms/step - loss: 0.1268 - val_loss: 0.2362
Epoch 93/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1223 - val_loss: 0.2444
Epoch 94/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1316 - val_loss: 0.2479
Epoch 95/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1263 - val_loss: 0.2512
Epoch 96/100
38/38 [==============================] - 0s 7ms/step - loss: 0.1223 - val_loss: 0.2510
Epoch 97/100
38/38 [==============================] - 0s 8ms/step - loss: 0.1175 - val_loss: 0.2511
Epoch 98/100
38/38 [==============================] - 0s 6ms/step - loss: 0.1329 - val_loss: 0.2507
Epoch 99/100
38/38 [==============================] - 0s 5ms/step - loss: 0.1478 - val_loss: 0.2442
Epoch 100/100
38/38 [==============================] - 0s 8ms/step - loss: 0.1264 - val_loss: 0.2466
```

In [109…

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

```
13/13 [==============================] - 0s 2ms/step - loss: 0.1594
Accuracy : 0.840594083070755
```

In [110…

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

Out[110]:

```
<matplotlib.legend.Legend at 0x7f1f54220f10>
```

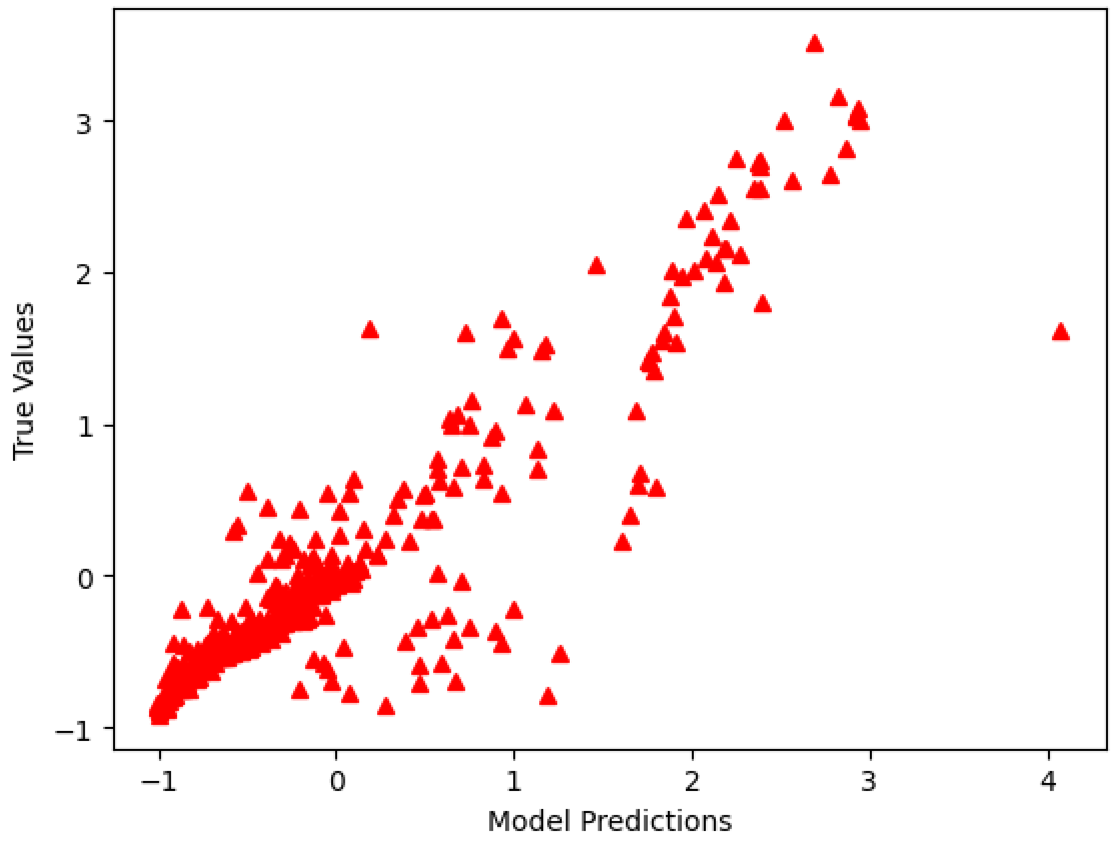
In [111…

```
y_predict = ANN_model.predict(X_test)
plt.plot(y_test, y_predict, "^", color = 'r')
plt.xlabel('Model Predictions')
plt.ylabel('True Values')
```

```
13/13 [==============================] - 0s 2ms/step
```

Out[111]:

```
Text(0, 0.5, 'True Values')
```



Deploying to Amazon Sagemaker

```
In [ ]: !pip install "sagemaker"
        !pip install boto3
```

```
In [ ]: import sagemaker
        import boto3

        sagemaker_session = sagemaker.Session()
        #Creating a sagemaker session and passing bucket and prefix values

        bucket = 'sagemaker-health1'
        prefix = 'linear_learner'

        role = sagemaker.get_execution_role()
        print(role)
```

```
In [ ]: import io
        import numpy as np
        import sagemaker.amazon.common as smac # sagemaker common libary

        buf = io.BytesIO()
        smac.write_numpy_to_dense_tensor(buf, X_train, y_train)
        buf.seek(0)
```

```
In [ ]: import os

        key = 'linear-train-data'
        boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).upload_fileobj(buf)
        s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)
        print('uploaded training data location: {}'.format(s3_train_data))
```

```
In [ ]: output_location = 's3://{}/{}/output'.format(bucket, prefix)
        print('Training artifacts will be uploaded to: {}'.format(output_location))
```

```
In [ ]: # This part of code will allow us to use training containers of sagemaker trained model.

        from sagemaker.amazon.amazon_estimator import image_uris
        container = image_uris.retrieve(region=boto3.Session().region_name, framework='linear-learner')
```

```
In [ ]: # Parameters are passed using Estimator function that is available to us.

        linear = sagemaker.estimator.Estimator(container,
                                               role,
                                               instance_count = 1,
                                               instance_type = 'ml.c4.xlarge',
                                               output_path = output_location,
                                               sagemaker_session = sagemaker_session)

        linear.set_hyperparameters(feature_dim = 8,
                                   predictor_type = 'regressor',
                                   mini_batch_size = 100,
                                   epochs = 100,
                                   num_models = 32,
                                   loss = 'absolute_loss')

        linear.fit({'train': s3_train_data})
```