- Aim of the problem is to find the health insurance cost incured by Individuals based on thier age, gender, BMI, number of children, smoking habit and geo-location.
- Features available are:
  - sex: insurance contractor gender, female, male
  - bmi: Body mass index (ideally 18.5 to 24.9)
  - children: Number of children covered by health insurance / Number of dependents
  - smoker: smoking habits
  - region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
  - charges: Individual medical costs billed by health insurance

TYPES OF AVAILABLE SAGEMAKER IMAGES

- Data Science [datascience-1.0]: Data Science is a Conda image with the most commonly used Python packages and libraries, such as NumPy and SciKit Learn.
- Base Python [python-3.6]
- MXNet (optimized for CPU) [mxnet-1.6-cpu-py36]
- MXNet (optimized for GPU) [mxnet-1.6-gpu-py36]
- PyTorch (optimized for CPU) [pytorch-1.4-cpu-py36]
- PyTorch (optimized for GPU) [pytorch-1.4-gpu-py36]
- TensorFlow (optimized for CPU) [tensorflow-1.15-cpu-py36]
- TensorFlow (optimized for GPU) [tensorflow-1.15-gpu-py36]
- TensorFlow 2 (optimized for CPU) [tensorflow-2.1-cpu-py36]
- TensorFlow 2 (optimized for GPU) [tensorflow-2.1-gpu-py36]

In [1]:

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

In [5]:

```
# read the csv file
insurance_df = pd.read_csv('insurance.csv')
```

In [ ]:

```

```

PERFORM EXPLORATORY DATA ANALYSIS:

In [6]:

```
# check if there are any Null values
sns.heatmap(insurance_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

Out[6]:

```
<AxesSubplot:>
```

In [7]:

```
# check if there are any Null values
insurance_df.isnull().sum()
```

Out[7]:

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [8]:

```
# Check the dataframe info

insurance_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [9]:

```
# Grouping by region to see any relationship between region and charges
# Seems like south east region has the highest charges and body mass index
df_region = insurance_df.groupby(by='region').mean()
df_region
```

Out[9]:

	age	bmi	children	charges
region				
northeast	39.268519	29.173503	1.046296	13406.384516
northwest	39.196923	29.199785	1.147692	12417.575374
southeast	38.939560	33.355989	1.049451	14735.411438
southwest	39.455385	30.596615	1.141538	12346.937377

MINI CHALLENGE

- **Group data by 'age' and examine the relationship between 'age' and 'charges'**

In [ ]:

```

```

In [10]:

```
# Check unique values in the 'sex' column
insurance_df['sex'].unique()
```

Out[10]:

```
array(['female', 'male'], dtype=object)
```

In [11]:

```
# convert categorical variable to numerical

insurance_df['sex'] = insurance_df['sex'].apply(lambda x: 0 if x == 'female' else 1)
```

In [12]:

```
insurance_df.head()
```

Out[12]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	yes	southwest	16884.92400
1	18	1	33.770	1	no	southeast	1725.55230
2	28	1	33.000	3	no	southeast	4449.46200
3	33	1	22.705	0	no	northwest	21984.47061
4	32	1	28.880	0	no	northwest	3866.85520

In [13]:

```
# Check the unique values in the 'smoker' column
insurance_df['smoker'].unique()
```

Out[13]:

```
array(['yes', 'no'], dtype=object)
```

In [14]:

```
# Convert categorical variable to numerical

insurance_df['smoker'] = insurance_df['smoker'].apply(lambda x: 0 if x == 'no' else 1)
```

In [15]:

```
insurance_df.head()
```

Out[15]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520

In [16]:

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

Out[16]:

```
array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

In [17]:

```
region_dummies = pd.get_dummies(insurance_df['region'], drop_first = True)
```

In [18]:

```
region_dummies
```

Out[18]:

	northwest	southeast	southwest
0	0	0	1
1	0	1	0
2	0	1	0
3	1	0	0
4	1	0	0
...	...	...	...
1333	1	0	0
1334	0	0	0
1335	0	1	0
1336	0	0	1
1337	1	0	0

1338 rows × 3 columns

In [19]:

```
insurance_df = pd.concat([insurance_df, region_dummies], axis = 1)
```

In [20]:

```
insurance_df.head()
```

Out[20]:

	age	sex	bmi	children	smoker	region	charges	northwest	southeast	southwest
0	19	0	27.900	0	1	southwest	16884.92400	0	0	1
1	18	1	33.770	1	0	southeast	1725.55230	0	1	0
2	28	1	33.000	3	0	southeast	4449.46200	0	1	0
3	33	1	22.705	0	0	northwest	21984.47061	1	0	0
4	32	1	28.880	0	0	northwest	3866.85520	1	0	0

In [21]:

```
# Let's drop the original 'region' column
insurance_df.drop(['region'], axis = 1, inplace = True)
```

In [22]:

```
insurance_df.head()
```

Out[22]:

	age	sex	bmi	children	smoker	charges	northwest	southeast	southwest
0	19	0	27.900	0	1	16884.92400	0	0	1
1	18	1	33.770	1	0	1725.55230	0	1	0
2	28	1	33.000	3	0	4449.46200	0	1	0
3	33	1	22.705	0	0	21984.47061	1	0	0

	age	sex	bmi	children	smoker	charges	northwest	southeast	southwest
3	33	1	22.705	0	0	21984.47061	1	0	0
4	32	1	28.880	0	0	3866.85520	1	0	0

MINI CHALLENGE

- **Calculate the mean and standard deviation of the age, charges and bmi**

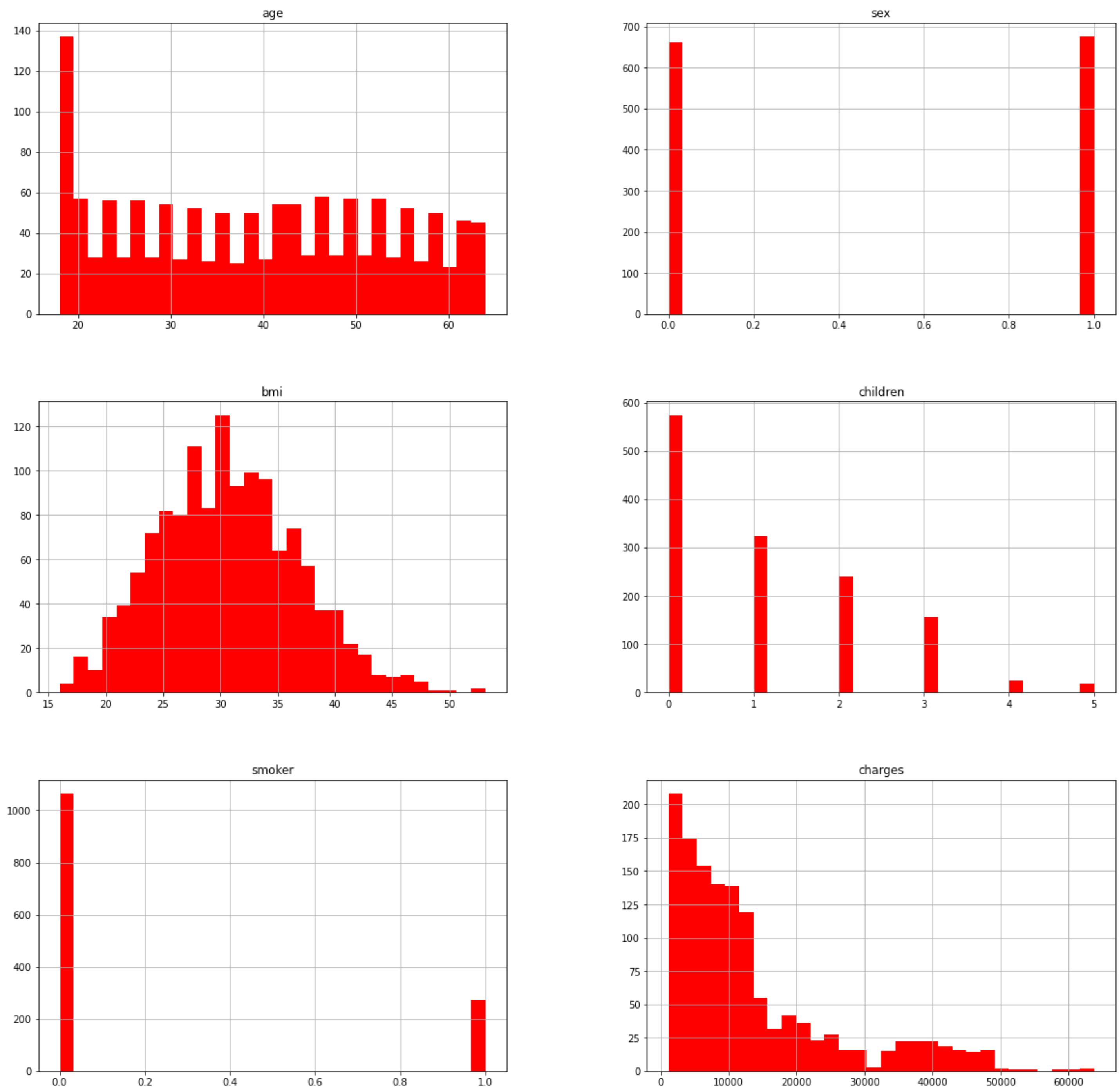
VISUALIZE DATASET

In [23]:

```
insurance_df[['age', 'sex', 'bmi', 'children', 'smoker', 'charges']].hist(bins = 30, fig
size = (20,20), color = 'r')
```

Out[23]:

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



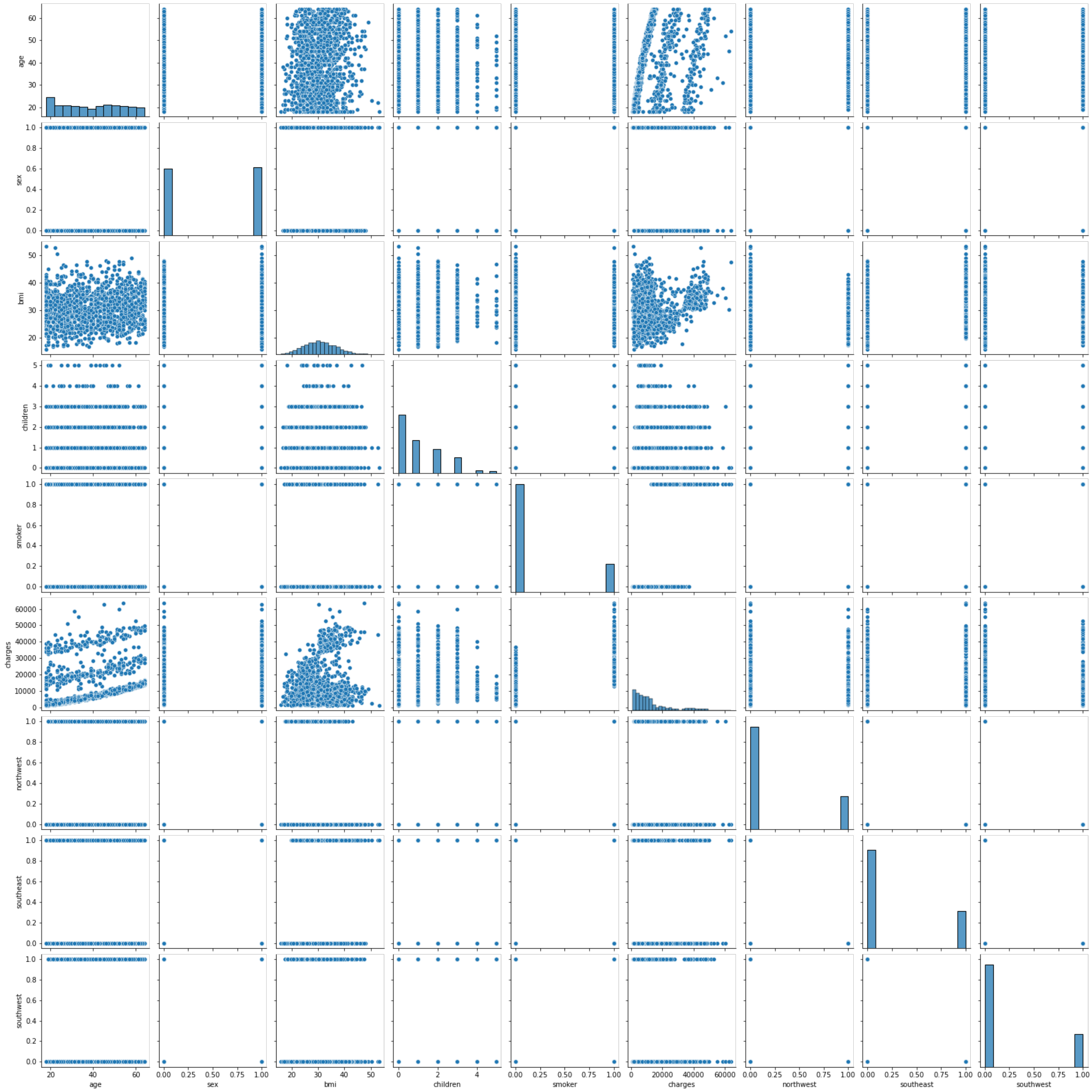
In [24]:

```
# plot pairplot

sns.pairplot(insurance_df)
```

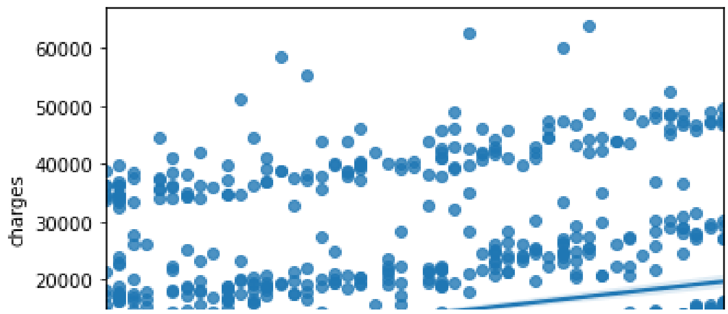
Out[24]:

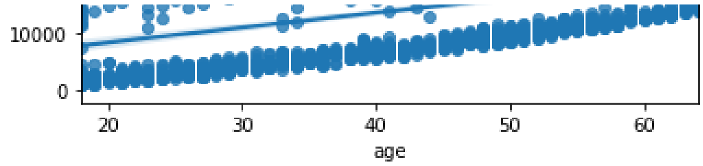
<seaborn.axisgrid.PairGrid at 0x7f3020c6a3c8>



In [25]:

```
sns.regplot(x = 'age', y = 'charges', data = insurance_df)
plt.show()
```





In [26]:

```
# smoker and age have positive correlations with charges
```

CREATE TRAINING AND TESTING DATASET

In [27]:

```
insurance_df.columns
```

Out[27]:

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'charges', 'northwest',
       'southeast', 'southwest'],
      dtype='object')
```

In [28]:

```
X = insurance_df.drop(columns =['charges'])
y = insurance_df['charges']
```

In [29]:

```
X
```

Out[29]:

	age	sex	bmi	children	smoker	northwest	southeast	southwest
0	19	0	27.900	0	1	0	0	1
1	18	1	33.770	1	0	0	1	0
2	28	1	33.000	3	0	0	1	0
3	33	1	22.705	0	0	1	0	0
4	32	1	28.880	0	0	1	0	0
...	...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	0	0
1334	18	0	31.920	0	0	0	0	0
1335	18	0	36.850	0	0	0	1	0
1336	21	0	25.800	0	0	0	0	1
1337	61	0	29.070	0	1	1	0	0

1338 rows × 8 columns

In [30]:

```
y
```

Out[30]:

```
0        16884.92400
1         1725.55230
2         4449.46200
3        21984.47061
4         3866.85520
             ...
1333     10600.54830
1334      2205.98080
1335      1629.83350
```

```
1336      2007.94500
1337     29141.36030
Name: charges, Length: 1338, dtype: float64
```

In [31]:

```
X.shape
```

Out[31]:

```
(1338, 8)
```

In [32]:

```
y.shape
```

Out[32]:

```
(1338,)
```

In [33]:

```
X = np.array(X).astype('float32')
y = np.array(y).astype('float32')
```

In [34]:

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

In [35]:

```
# Only take the numerical variables and scale them
X
```

Out[35]:

```
array([[19.  ,  0.  , 27.9 , ...,  0.  ,  0.  ,  1.  ],
       [18.  ,  1.  , 33.77, ...,  0.  ,  1.  ,  0.  ],
       [28.  ,  1.  , 33.  , ...,  0.  ,  1.  ,  0.  ],
       ...,
       [18.  ,  0.  , 36.85, ...,  0.  ,  1.  ,  0.  ],
       [21.  ,  0.  , 25.8 , ...,  0.  ,  0.  ,  1.  ],
       [61.  ,  0.  , 29.07, ...,  1.  ,  0.  ,  0.  ]], dtype=float32)
```

In [42]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4
2)
```

In [45]:

```
#scaling the data before feeding the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)

scaler_y = StandardScaler()
y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)
```

TRAIN AND TEST A LINEAR REGRESSION MODEL IN SK-LEARN (NOTE THAT SAGEMAKER BUILT-IN ALGORITHMS ARE NOT USED HERE)

In [46]:

In [46]:

```
# using linear regression model
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[46]:

```
LinearRegression()
```

In [47]:

```
regresssion_model_sklearn_accuracy = regresssion_model_sklearn.score(X_test, y_test)
regresssion_model_sklearn_accuracy
```

Out[47]:

```
0.7835929760314282
```

In [48]:

```
y_predict = regresssion_model_sklearn.predict(X_test)
```

In [49]:

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

In [50]:

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

Out[50]:

```
268
```

In [51]:

```
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 [52]:

```
print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r
2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-52-c68a241a5aba> in <module>
----> 1 print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =
', adj_r2)

NameError: name 'MAE' is not defined
```

TRAIN A LINEAR LEARNER MODEL USING SAGEMAKER

In [53]:

```
# Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python
# Boto3 allows Python developer to write software that makes use of services like Amazon
S3 and Amazon EC2

import sagemaker
```

```
import boto3
from sagemaker import Session

# Let's create a Sagemaker session
sagemaker_session = sagemaker.Session()
bucket = Session().default_bucket()
prefix = 'linear_learner' # prefix is the subfolder within the bucket.

# Let's get the execution role for the notebook instance.
# This is the IAM role that you created when you created your notebook instance. You pass
the role to the training job.
# Note that AWS Identity and Access Management (IAM) role that Amazon SageMaker can assum
e to perform tasks on your behalf (for example, reading training results, called model ar
tifacts, from the S3 bucket and writing training results to Amazon S3).
role = sagemaker.get_execution_role()
print(role)
```

arn:aws:iam::542063182511:role/service-role/AmazonSageMaker-ExecutionRole-20191104T033920

In [54]:

```
X_train.shape
```

Out[54]:

(1070, 8)

In [55]:

```
y_train.shape
```

Out[55]:

(1070, 1)

In [56]:

```
# y_train = y_train[:,0]
```

In [63]:

```
y_train.shape
```

Out[63]:

(1070, 1)

In [64]:

```
import io # The io module allows for dealing with various types of I/O (text I/O, binary
I/O and raw I/O).
import numpy as np
import sagemaker.amazon.common as smac # sagemaker common libary

# Code below converts the data in numpy array format to RecordIO format
# This is the format required by Sagemaker Linear Learner

buf = io.BytesIO() # create an in-memory byte array (buf is a buffer I will be writing to
)
smac.write_numpy_to_dense_tensor(buf, X_train, y_train.reshape(-1))
buf.seek(0)
# When you write to in-memory byte arrays, it increments 1 every time you write to it
# Let's reset that back to zero
```

Out[64]:

0

In [65]:

```
import os

# Code to upload RecordIO data to S3
```

```
# Key refers to the name of the file
key = 'linear-train-data'

# The following code uploads the data in record-io format to S3 bucket to be accessed lat
er for training
boto3.resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).upload_fi
leobj(buf)

# Let's print out the training data location in s3
s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)
print('uploaded training data location: {}'.format(s3_train_data))
```

```
uploaded training data location: s3://sagemaker-us-east-2-542063182511/linear_learner/tra
in/linear-train-data
```

In [66]:

```
# create an output placeholder in S3 bucket to store the linear learner output

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

```
Training artifacts will be uploaded to: s3://sagemaker-us-east-2-542063182511/linear_lear
ner/output
```

In [67]:

```
# This code is used to get the training container of sagemaker built-in algorithms
# all we have to do is to specify the name of the algorithm, that we want to use

# Let's obtain a reference to the linearLearner container image
# Note that all regression models are named estimators
# You don't have to specify (hardcode) the region, get_image_uri will get the current reg
ion name using boto3.Session


from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'linear-learner')
```

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
Defaulting to the only supported framework/algorithm version: 1. Ignoring framework/algor
ithm version: 1.
```

In [ ]:

```
# We have pass in the container, the type of instance that we would like to use for train
ing
# output path and sagemaker session into the Estimator.
# We can also specify how many instances we would like to use for training

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


# We can tune parameters like the number of features that we are passing in, type of pred
ictor like 'regressor' or 'classifier', mini batch size, epochs
# Train 32 different versions of the model and will get the best out of them (built-in pa
rameters optimization!)

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

```python
# Now we are ready to pass in the training data from S3 to train the linear learner model

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

# Let's see the progress using cloudwatch logs
```

DEPLOY AND TEST THE TRAINED LINEAR LEARNER MODEL

In [69]:

```python
# Deploying the model to perform inference

linear_regressor = linear.deploy(initial_instance_count = 1,
                                          instance_type = 'ml.m4.xlarge')
```

```
--------------!
```

In [70]:

```python
from sagemaker.predictor import csv_serializer, json_deserializer

# Content type overrides the data that will be passed to the deployed model, since the de
ployed model expects data in text/csv format.

# Serializer accepts a single argument, the input data, and returns a sequence of bytes i
n the specified content type

# Deserializer accepts two arguments, the result data and the response content type, and
return a sequence of bytes in the specified content type.

# Reference: https://sagemaker.readthedocs.io/en/stable/predictors.html

# linear_regressor.content_type = 'text/csv'
linear_regressor.serializer = csv_serializer
linear_regressor.deserializer = json_deserializer
```

In [71]:

```python
# making prediction on the test data

result = linear_regressor.predict(X_test)
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
The json_deserializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

In [ ]:

```python
result # results are in Json format
```

In [73]:

```python
# Since the result is in json format, we access the scores by iterating through the score
s in the predictions

predictions = np.array([r['score'] for r in result['predictions']])
```

In [ ]:

```python
predictions
```

In [75]:

```python
predictions.shape
```

Out[75]:

(268,)

In [76]:

```
y_predict_orig = scaler_y.inverse_transform(predictions)
y_test_orig = scaler_y.inverse_transform(y_test)
```

In [77]:

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

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

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r
2)
```

```
RMSE = 0.509
MSE = 0.25952588072932026
MAE = 0.2824190891977032
R2 = 0.7550276842210936
Adjusted R2 = 0.7474609717646022
```

In [78]:

```
# Delete the end-point

linear_regressor.delete_endpoint()
```

A MORE COMPLEX ARTIFICIAL NEURAL NETWORK-BASED REGRESSION MODEL

In [ ]:

```
!pip install tensorflow
```

In [ ]:

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

In [ ]:

```
# optimizer = Adam()
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()
```

In [ ]:

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

```
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)
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
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'])
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
plt.plot(y_test_orig, y_predict_orig, "^", color = 'r')
plt.xlabel('Model Predictions')
plt.ylabel('True Values')
```

In [ ]:

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

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

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

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r
2)
```