In [1]:

```
#QUESTION 1 : MEDICAL INSURANCE DATASET
#a. i.The Machine learning(ML) is SUPERVISED learning because the dataset has labelled data which enables ML algorithm to find the relationship between any two points(variables)
```

In [2]:

```
# ii. The Machine learning task best for this dataset is classification as it has labelled data. Logistic regression is not the best type of classification for this dataset as it is best for making a prediction about a categorica
```

In [3]:

```
#b. Data Exploration : When observing the dataset, it can be seen that the older the individual, the higher the medical cost. Also, the higher the bmi, the higher the medical cost of those people. However, exploratory data tools
```

In [4]:

```
# To start this assessment, data exploration comes first;using the relevant data exploration tools, some relevant libraries would be imported and the dataset uploaded
```

In [5]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from matplotlib import style
style.use('seaborn-whitegrid')
plt.rcParams['figure.figsize'] = (20,10)
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [6]:

```
dataset = pd.read_csv('insurance.csv')
dataset
```

Out[6]:

	age	sex	bmi	children	smoker	region	medicalCost
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

In [7]:

```
# Exploration of the dataset
dataset.isnull()
```

Out[7]:

	age	sex	bmi	children	smoker	region	medicalCost
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False

1338 rows × 7 columns

In [8]:

```
# No null values. The dataset has 7 Columns and 1338 rows
```

In [9]:

```
# Check the type of data whether integers or float
dataset.dtypes
```

Out[9]:

```
age              int64
sex             object
bmi            float64
children         int64
smoker          object
region          object
medicalCost    float64
dtype: object
```

In [10]:

```
# Only bmi and medical cost columns have float variables, the rest have objects
```

In [11]:

```
# first obtain the statistical values of all the variables
```

In [12]:

```
dataset.describe()
```

Out[12]:

	age	bmi	children	medicalCost
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

In [13]:

```
# mean value for age column is 39, bmi mean is 30.66, children column mean value is 1.09 and medical cost mean value is 13270
```

In [14]:

```
# check for null values
```

In [15]:

```
dataset.isnull()
```

Out[15]:

	age	sex	bmi	children	smoker	region	medicalCost
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False

1338 rows × 7 columns

In [16]:

```
# view the first 5 rows of the dataset
```

In [17]:

```
dataset.head()
```

Out[17]:

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

In [18]:

```
 # Logistic regression as a machine model will not be best for this dataset as it can work best only with categorical variables, as seen below:
x = dataset.drop(['sex','smoker','region','medicalCost','age','bmi'], axis = 1)
y = dataset['medicalCost']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(x_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_1664/3216416094.py in <module>
      5 from sklearn.linear_model import LogisticRegression
      6 lr_model = LogisticRegression()
----> 7 lr_model.fit(x_train, y_train)

~\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py in fit(self, X, y, sample_weight)
   1345                                    order="C",
   1346                                    accept_large_sparse=solver != 'liblinear')
-> 1347         check_classification_targets(y)
   1348         self.classes_ = np.unique(y)
   1349

~\anaconda3\lib\site-packages\sklearn\utils\multiclass.py in check_classification_targets(y)
    181     if y_type not in ['binary', 'multiclass', 'multiclass-multioutput',
    182                       'multilabel-indicator', 'multilabel-sequences']:
--> 183         raise ValueError("Unknown label type: %r" % y_type)
    184
    185

ValueError: Unknown label type: 'continuous'
```

In [ ]:

```
# As seen above, there is a ValueError: 'Unknown label type: 'continuous'' error because Logistic Regression as a machine model will not be able to predict a continuous variable only categorical variables.
```

In [19]:

```
# Clustering as a machine model will not be best for this dataset as it can work best only with unlabelled data;
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2, random_state = 42)
y_kmeans = kmeans.fit_predict(dataset)
kmeans.cluster_centers_
from sklearn.cluster import KMeans
clustersum = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state =0)
    kmeans.fit(dataset)
    clustersum.append(kmeans.inertia_)
plt.plot(range(1,11), clustersum)
plt.title('Medical Insurance dataset')
plt.xlabel('age')
plt.ylabel('medicalCost')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_1664/3075149227.py in <module>
      2 from sklearn.cluster import KMeans
      3 kmeans = KMeans(n_clusters = 2, random_state = 42)
----> 4 y_kmeans = kmeans.fit_predict(dataset)
      5 kmeans.cluster_centers_
      6 from sklearn.cluster import KMeans

~\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py in fit_predict(self, X, y, sample_weight)
   1075             Index of the cluster each sample belongs to.
   1076         """
-> 1077         return self.fit(X, sample_weight=sample_weight).labels_
   1078
   1079     def fit_transform(self, X, y=None, sample_weight=None):

~\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py in fit(self, X, y, sample_weight)
    977             Fitted estimator.
    978         """
--> 979         X = self._validate_data(X, accept_sparse='csr',
    980                                 dtype=[np.float64, np.float32],
    981                                 order='C', copy=self.copy_x,

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    419             out = X
    420         elif isinstance(y, str) and y == 'no_validation':
--> 421             X = check_array(X, **check_params)
    422             out = X
    423         else:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~\anaconda3\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order, like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

~\anaconda3\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order, like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104

ValueError: could not convert string to float: 'female'
```

In [ ]:

```
# there is a ValueError: could not convert string to float: 'female' as clustering as a machine model works best with unlabelled data.
```

In [ ]:

```
# Transform the categorical values to numerical values using feature engineering so the variables are the same type
```

In [20]:

```
dataset['sex'].replace({'male':1,'female':2}, inplace=True)
dataset['region'].replace({'northeast':1,'northwest':2,'southeast':3, 'southwest':4}, inplace=True)
dataset['smoker'].replace({'yes':1,'no':2}, inplace=True)
dataset
```

Out[20]:

	age	sex	bmi	children	smoker	region	medicalCost
0	19	2	27.900	0	1	4	16884.92400
1	18	1	33.770	1	2	3	1725.55230
2	28	1	33.000	3	2	3	4449.46200
3	33	1	22.705	0	2	2	21984.47061
4	32	1	28.880	0	2	2	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	2	2	10600.54830
1334	18	2	31.920	0	2	1	2205.98080
1335	18	2	36.850	0	2	3	1629.83350
1336	21	2	25.800	0	2	4	2007.94500
1337	61	2	29.070	0	1	2	29141.36030

1338 rows × 7 columns

In [ ]:

```
# Convert float values to integers using astype() function:
```

In [22]:

```
dataset1 = dataset.copy()   # Create a copy of DataFrame
dataset1 = dataset1.astype({'bmi': int, 'medicalCost': int})
dataset1
```

Out[22]:

	age	sex	bmi	children	smoker	region	medicalCost
0	19	2	27	0	1	4	16884
1	18	1	33	1	2	3	1725
2	28	1	33	3	2	3	4449
3	33	1	22	0	2	2	21984
4	32	1	28	0	2	2	3866
...	...	...	...	...	...	...	...
1333	50	1	30	3	2	2	10600
1334	18	2	31	0	2	1	2205
1335	18	2	36	0	2	3	1629
1336	21	2	25	0	2	4	2007
1337	61	2	29	0	1	2	29141

1338 rows × 7 columns

In [ ]:

```
# Visualize the dataset for further exploration using bar plot:
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'age', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(10,4))
ax = sns.barplot(x = 'sex', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'bmi', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'children', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(10,4))
ax = sns.barplot(x = 'smoker', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'region', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
# c. To determine the correlation(relationships) between each of the predictors(x variables) and medical cost(y variable):
```

In [ ]:

```
from plotnine.data import mpg
from plotnine import ggplot, aes, geom_point

ggplot(dataset1) + aes(x="age", y="medicalCost") + geom_point()
```

In [ ]:

```
# Medical cost of insurance of individuals increased as they got older with highest point(cost) at around age 54 and lowest point(cost) at around age 10
```

In [ ]:

```
ggplot(dataset1) + aes(x="bmi", y="medicalCost") + geom_point()
```

In [ ]:

```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'age', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

In [ ]:

```
# Medical cost of insurance of individuals with bmi(body mass index) between 30 and 50 had the highest medical cost which at its highest was above 60,000
```

In [ ]:

```
ggplot(dataset1) + aes(x="smoker", y="medicalCost") + geom_point()
```

In [ ]:

```
# Medical cost of insurance of individuals who were smokers represented as 1 was higher( as high as 60,000 plus) while the medical cost of non-smokers represented as 2 was low( as low as 38,000)
```

In [ ]:

```
ggplot(dataset1) + aes(x="sex", y="medicalCost") + geom_point()
```

In [ ]:

```
# Medical cost of insurance of males respresented as 1 was a bit lower than that of females which is represented as 2
```

In [ ]:

```
ggplot(dataset1) + aes(x="region", y="medicalCost") + geom_point()
```

In [ ]:

```
# Medical cost of individuals in region 3(southeast) was the highest, above 60,000 while those from region 4(southwest) was the lowest, around 51,000
```

In [ ]:

```
ggplot(dataset1) + aes(x="children", y="medicalCost") + geom_point()
```

In [ ]:

```
# Medical cost of individuals with no children(0) is the highest while those with 5 children have the lowest medical cost
```

In [ ]:

```
sns.pairplot(dataset1[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'medicalCost']])
```

In [ ]:

```
#d. The bivariate plot above shows the correlation between the variables. For example, smoker(1) have the higher medical cost of insurance compared to non-smoker(2). Region southeast(3) has people with the highest medical cost of
```

In [ ]:

```
# To determine the variables with measures of relationships
dataset1.corr()
```

In [ ]:

```
# # Another way ValueError: could not convert string to float: 'female'to show the correlation between the Predictors and 'medicalCost' is Pearson correlation, as seen below:
```

In [ ]:

```
dataset1.corr(method='pearson')
```

In [ ]:

```
# From the above, 'age', 'bmi' and 'children' have positive correlation with 'medicalCost'. While 'sex','smoker' and 'region' have negative correlation with 'medicalCost'
```

In [ ]:

```
# To show the correlation values more clearly;

# Correlation between the three predictors and medical cost:
dataset1['age'].corr(dataset['medicalCost'])
```

In [ ]:

```
dataset1['bmi'].corr(dataset['medicalCost'])
```

In [ ]:

```
dataset1['children'].corr(dataset['medicalCost'])
```

In [ ]:

```
# Build correlation matrix to show correlation between the variables
sns.heatmap(dataset1.corr())
plt.show()
```

In [ ]:

```
# From the heat map above, the predictors: 'age', 'bmi' have strong positive correlation (orange boxes) with 'medicalCost' and 'children' also has a positive correlation (pinkish-orange) with 'medicalCost, but not as strong as 'a
```

In [ ]:

```
# Scatterplots of the three predictors and 'medicalCost'
```

In [ ]:

```
%matplotlib notebook
```

In [ ]:

```
dataset1.plot(kind='scatter', x='age', y='medicalCost')
```

In [ ]:

```
dataset1.plot(kind='scatter', x='bmi', y='medicalCost')
```

In [ ]:

```
dataset1.plot(kind='scatter', x='children', y='medicalCost')
```

In [ ]:

```
# Building simple linear regression models using each of the three predictors above and 'medcialCost':
```

In [ ]:

```
# Linear regression model with predictor 'age' and 'medicalCost'
X_var = dataset1[['age']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [ ]:

```
# Using Statsmodels and scikit-learn and then showing the statistical summary:
import statsmodels.api as sm
lr_model = sm.OLS(y_var, X_var) # Ordinary Least Squares
lr = lr_model.fit()
print(lr.summary())
```

In [ ]:

```
# The R squared is 86.43 which shows that linear regression model is not doing so well as a machine learning model for these features
```

In [ ]:

```
# Linear regression model with predictor 'bmi' and 'medicalCost'
X_var = dataset1[['bmi']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [ ]:

```
# Using Statsmodels and scikit-learn and then showing the statistical summary:
import statsmodels.api as sm
lr_model = sm.OLS(y_var, X_var) # Ordinary Least Squares
lr = lr_model.fit()
print(lr.summary())
```

In [ ]:

```
# The R squared value is 90.51 which shows that this machine model, linear regression is not doing so well to predict medical cost
```

In [24]:

```
# Linear regression model with predictor 'children' and 'medicalCost'
X_var = dataset1[['children']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [25]:

```
# Using Statsmodels and scikit-learn and then showing the statistical summary:
import statsmodels.api as sm
lr_model = sm.OLS(y_var, X_var) # Ordinary Least Squares
lr = lr_model.fit()
print(lr.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:            medicalCost   R-squared (uncentered):                 148.126
Model:                            OLS   Adj. R-squared (uncentered):            148.236
Method:                 Least Squares   F-statistic:                             -1346.
Date:                Tue, 12 Jul 2022   Prob (F-statistic):                        1.00
Time:                        01:15:33   Log-Likelihood:                         -14784.
No. Observations:                1338   AIC:                                 2.957e+04
Df Residuals:                    1337   BIC:                                 2.958e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
children    5854.9955    255.710     22.897      0.000    5353.360    6356.631
==============================================================================
Omnibus:                      178.401   Durbin-Watson:                   1.583
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              262.297
Skew:                           0.955   Prob(JB):                     1.10e-57
Kurtosis:                       4.029   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [ ]:

```
# The R squared value is 148.12 which shows that this machine model, linear regression is not doing so well to predict medical costs. When R squared is closer to 1, the better the model is for predicting the target variable
```

In [26]:

```
# f. Building 2 multivariate regression models with
# i. The predictors: 'age', 'bmi' and 'children'
from sklearn.model_selection import train_test_split as holdout
from sklearn.linear_model import LinearRegression
from sklearn import metrics
x = dataset1.drop(['sex','smoker','region','medicalCost'], axis = 1)
y = dataset1['medicalCost']
x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2, random_state=0)
Lin_reg = LinearRegression()
Lin_reg.fit(x_train, y_train)
print(Lin_reg.intercept_)
print(Lin_reg.coef_)
print(Lin_reg.score(x_test, y_test))
```

```
-5027.40998731274
[220.70451175 293.78774538 664.26983106]
0.16201886771966845
```

In [ ]:

```
# ii. With all the predictors
from sklearn.model_selection import train_test_split as holdout
from sklearn.linear_model import LinearRegression
from sklearn import metrics
x = dataset1.drop(['medicalCost'], axis = 1)
y = dataset1['medicalCost']
x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2, random_state=0)
Lin_reg = LinearRegression()
Lin_reg.fit(x_train, y_train)
print(Lin_reg.intercept_)
print(Lin_reg.coef_)
print(Lin_reg.score(x_test, y_test))
```

In [ ]:

```
# iii. Evaluating the above statistical significance of the two results from the models above:
# The multivariate regression with the 3 predictors of 'age', 'bmi' and 'children' has a score of about 16% which is a low accuracy. This regression model is not the best using only the 3 predictors to predict medicalCost for the
```

In [ ]:

```
# Alternatively,using Random forest regressor with:
# i.all the predictors and medical cost and all
# ii.the three predictors and medical cost
```

In [27]:

```
from sklearn.ensemble import RandomForestRegressor as rfr
x = dataset1.drop(['sex','smoker','region','medicalCost'], axis=1)
y = dataset1.medicalCost
Rfr = rfr(n_estimators = 100, criterion = 'mse',
                              random_state = 1,
                              n_jobs = -1)
Rfr.fit(x_train,y_train)
x_train_pred = Rfr.predict(x_train)
x_test_pred = Rfr.predict(x_test)

print('MSE train data: %.3f, MSE test data: %.3f' %
      (metrics.mean_squared_error(x_train_pred, y_train),
       metrics.mean_squared_error(x_test_pred, y_test)))
print('R2 train data: %.3f, R2 test data: %.3f' %
      (metrics.r2_score(y_train,x_train_pred, y_train),
       metrics.r2_score(y_test,x_test_pred, y_test)))
```

```
MSE train data: 36522206.930, MSE test data: 158361506.629
R2 train data: 0.684, R2 test data: -0.315
```

```
C:\Users\toyin\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass sample_weight=621     40182
194      1137
240     38511
1168     4670
1192    13019
        ...
763      3070
835      7160
1216     5415
559      1646
684      4766
Name: medicalCost, Length: 1070, dtype: int32 as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
C:\Users\toyin\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass sample_weight=578      9724
610      8547
569     45702
1034    12950
198      9644
        ...
1084    15019
726      6664
1132    20709
725     40932
963      9500
Name: medicalCost, Length: 268, dtype: int32 as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

In [28]:

```
from sklearn.ensemble import RandomForestRegressor as rfr
x = dataset1.drop(['medicalCost'], axis=1)
y = dataset1.medicalCost
Rfr = rfr(n_estimators = 100, criterion = 'mse',
                              random_state = 1,
                              n_jobs = -1)
Rfr.fit(x_train,y_train)
x_train_pred = Rfr.predict(x_train)
x_test_pred = Rfr.predict(x_test)

print('MSE train data: %.3f, MSE test data: %.3f' %
      (metrics.mean_squared_error(x_train_pred, y_train),
       metrics.mean_squared_error(x_test_pred, y_test)))
print('R2 train data: %.3f, R2 test data: %.3f' %
      (metrics.r2_score(y_train,x_train_pred, y_train),
       metrics.r2_score(y_test,x_test_pred, y_test)))
```

```
MSE train data: 36522206.930, MSE test data: 158361506.629
R2 train data: 0.684, R2 test data: -0.315
```

```
C:\Users\toyin\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass sample_weight=621     40182
194      1137
240     38511
1168     4670
1192    13019
        ...
763      3070
835      7160
1216     5415
559      1646
684      4766
Name: medicalCost, Length: 1070, dtype: int32 as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
C:\Users\toyin\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass sample_weight=578      9724
610      8547
569     45702
1034    12950
198      9644
        ...
1084    15019
726      6664
1132    20709
725     40932
963      9500
Name: medicalCost, Length: 268, dtype: int32 as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

In [ ]: