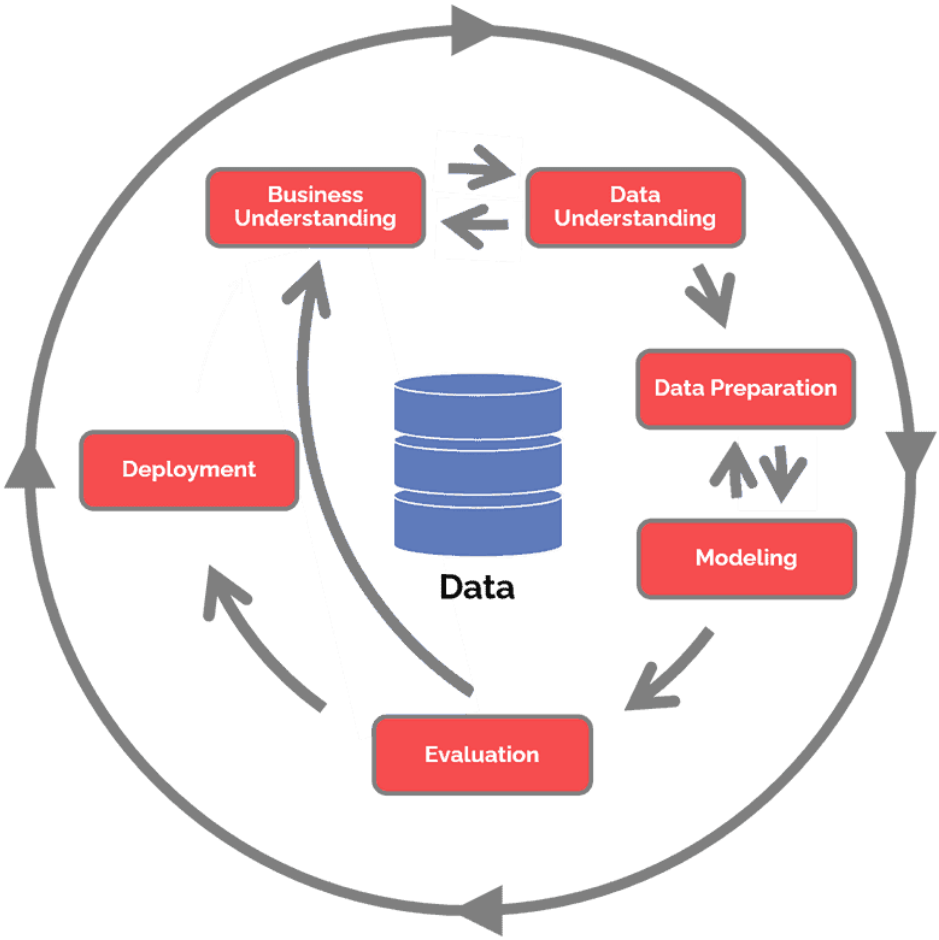
CT7205 MACHINE LEARNING AND OPTIMISATION

MACHINE LEARNING EXERCISES ASSESSMENT

BY

OLUWATOYIN ODENIYI(S4115181)

MACHINE LEARNING AND OPTIMISATION CYCLE



CRISP-DM (CROSS INDUSTRY STANDARD PROCESS FOR DATA MINING) is a process model (Methodology) as seen in the illustration above to guide Data Scientists in Data Science Projects for businesses or industries like the two for this assessment. The 6 Phases of CRISP-DM are:

1. Business understanding: what does the organization or business need? What are the data mining goals?

The goal of the medical insurance industry, for this case, is to be able to predict medical expenses of their payees based on the following parameters: age, sex, bmi, children, smoker, region and medical cost.

2. Data understanding: which data is available and is it clean (to verify data quality)?
   Data available for this assessment is a medical insurance dataset with 7 columns with the parameters as column names.

3. Data preparation ('data munging'): how to organize the data for modelling? how to format the data for modelling?
   Exploration of the dataset; cleaning the dataset and writing observations.

4. Data modelling: what modelling techniques should be applied? (After splitting the data to generate a test design, build and interpret the model)
   Data modelling using classification, clustering, regression analysis and simple linear regressions for each predictor

5. Evaluation: which model best meets the organisations or business' objective? (Summary of findings with possibility of further iterations)
   Using statistical performance measures to evaluate the significance of my results.

6. Build 2 multivariate regression models i. with 3 predictors above. ii. with all the predictors. iii. Evaluate and compare the two models.

7. Deployment: how do the relevant stakeholders access the results of the data mining process?
   Present a summary of the data mining process and results and recommendations and critical analysis of the models with conclusion.

QUESTION 1 : MEDICAL INSURANCE DATASET

A. i. The required Machine learning (ML) for forecasting or predicting medical cost for insurers with this specific dataset is SUPERVISED learning. This is because the dataset has labelled data which enables the required ML algorithm to find the relationship between any two points(variables), which for this assessment is certain predictors and medical cost.

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 often works well for making a prediction about a categorical variable which is not the type of variable needed to make a prediction in this Medical Insurance dataset analysis. So Linear regression would be better for prediction of the 'medicalCost' (dependent variable) as it is a model which can establish the relationship between a continuous dependent variable and independent variables. Clustering works best for unsupervised learning (dataset with unlabelled variables) therefore, it will not work well with this dataset which has labelled variables (supervised learning). Below, after the data has been uploaded, are the models to illustrate which ML is best and why (Linear regression model is at the tail end of this assessment).

B. Data Exploration: When observing the dataset, the older the individual, the higher the medical cost of insurance. Also, the higher the bmi, the higher the medical cost of insurance of those people. Individuals with children also had a higher medical cost; the more children they had, the higher the medical cost of insurance. However, exploratory data tools will be used to give a clear picture of the relationships between the variables and medical cost of insurance. 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 [6]:
```
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 [8]:
```
dataset = pd.read_csv('insurance.csv')
dataset
```

Out[8]:

	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

There are 1338 rows and 7 columns in this dataset. The columns are Age, Sex, Bmi, Children, Smoker, Region, Medical Cost.

```
In [9]: # Exploration of the dataset
        dataset.isnull()
```

Out[9]:

	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
```

There are no null values in the dataset, hence the 'False' response in the output using the python code above.

```
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
```

Bmi and Medical Cost have float variables and the others have objects and integers.

To further explore the dataset, the describe () function is used to show the statistical values of the variables as seen below. This includes mean, standard deviation, minimum and maximum. The mean value for 'age' is about 39, for 'bmi' it is 30.66. Average number of children is 1. It can be observed that as the age of the individual increased (the 75 percentile), the higher the medical cost at about 16,639 and the lower their ages, the lower their medical cost (the 25 percentile and lower medical cost of 4,740). Also, in the dataset's statistical exploration, the minimum age of people is 18 and their medical cost is about 1,121 while the maximum age is 64 and their medical cost is 63,770.

It can also be observed from the output below that the higher the body max index (bmi) of the individual, the higher their medical cost. People with bmi of 15.96, had medical cost

```
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 [10]:
```
# Logistic regression as a machine model will not be best for this dataset as it can work best only with categorical variables,
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_15768/3216416094.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'
```

```
dataset.head()
```

	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

```
    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, orde
r, 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)
```

```
    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 [1]:
```
# Transform the categorical values to numerical values using feature engineering so the variables are the same type
```

In [12]:
```
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[12]:

	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

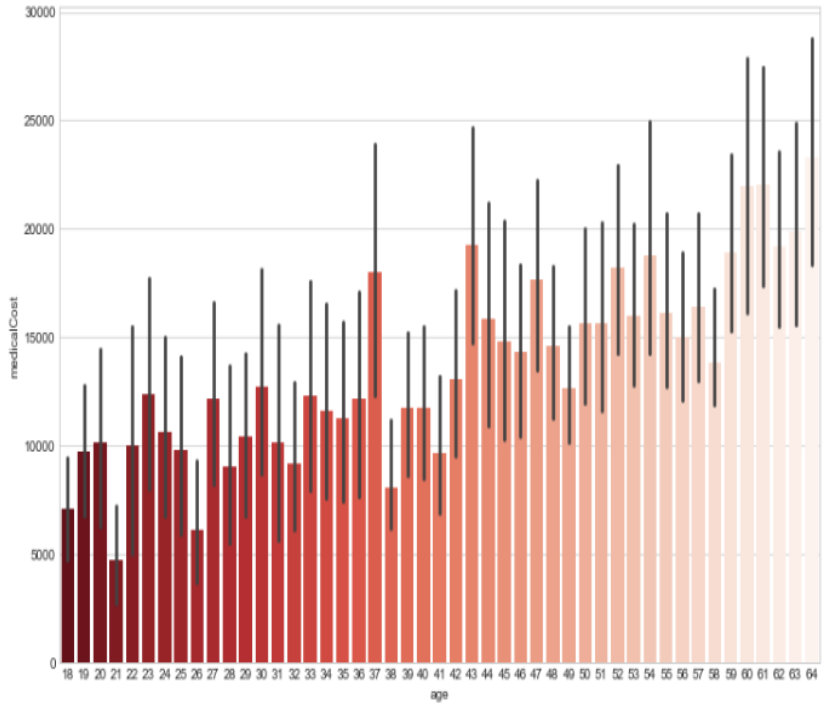
In [13]:
```
dataset1 = dataset.copy()   # Create a copy of DataFrame
dataset1 = dataset1.astype({'bmi': int, 'medicalCost': int})
dataset1
```

Out[13]:

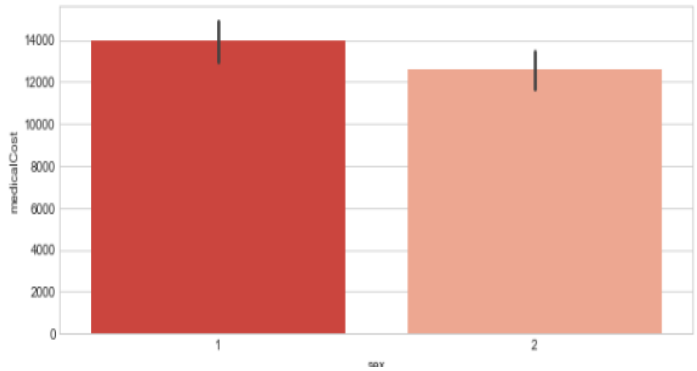
	age	sex	bmi	children	smoker	region	medicalCost
0	19	2	27	0	1	4	16884
1	18	1	33	1	2	3	1725
2	28	1	33	3	2	3	4449
3	33	1	22	0	2	2	21984
4	32	1	28	0	2	2	3866
...	...	...	...	...	...	...	...
1333	50	1	30	3	2	2	10600
1334	18	2	31	0	2	1	2205
1335	18	2	36	0	2	3	1629
1336	21	2	25	0	2	4	2007
1337	61	2	29	0	1	2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

1338 rows × 7 columns

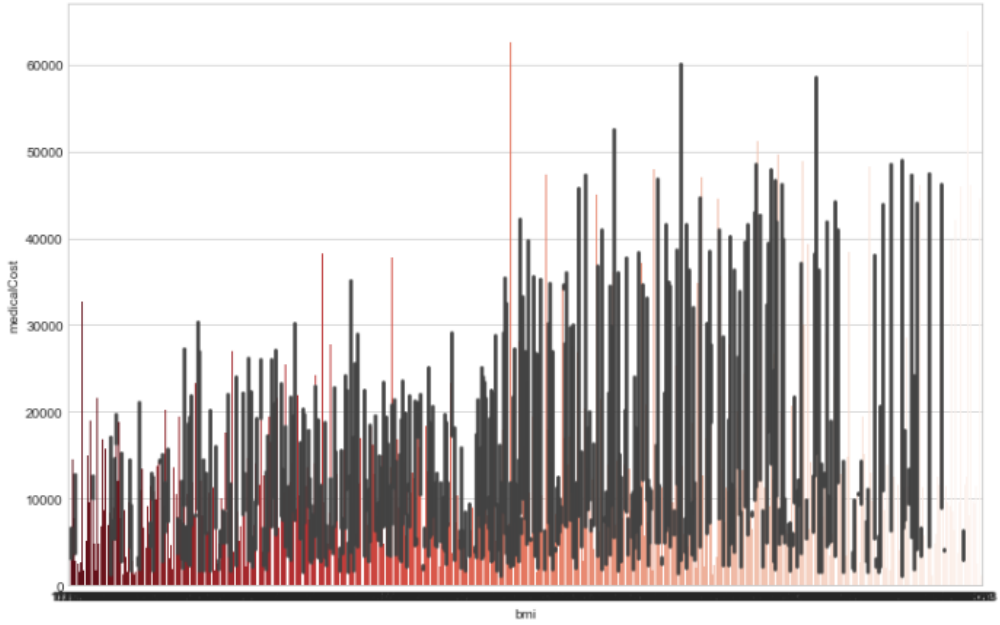
In [15]:
```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'age', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```



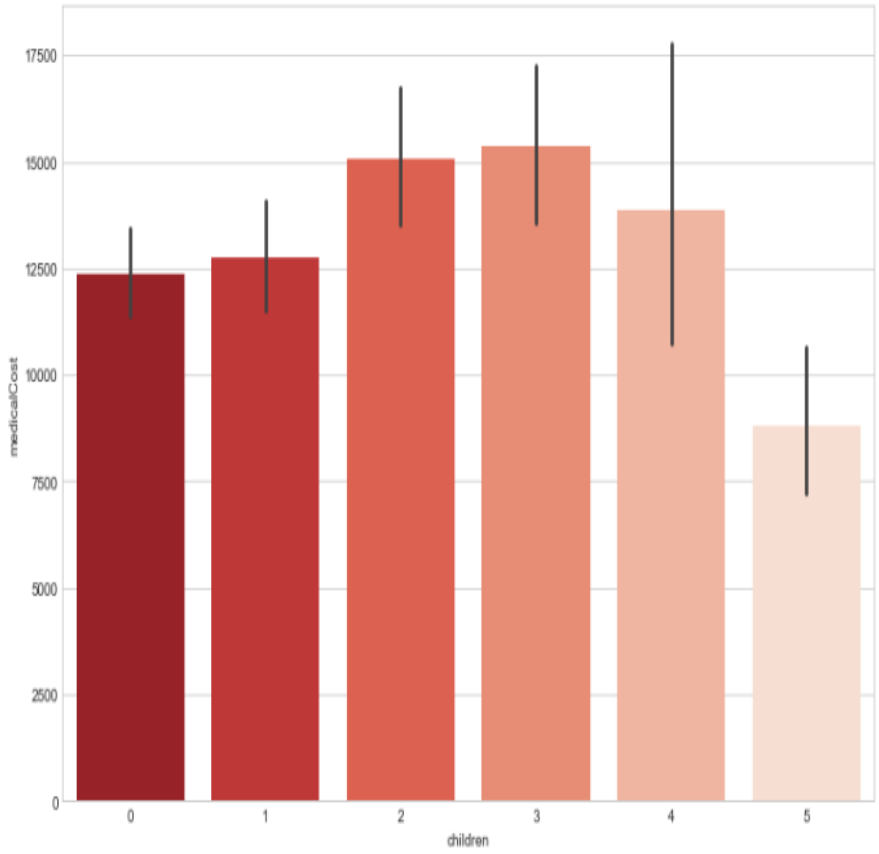
In [16]:
```
f, ax = plt.subplots(1,1, figsize=(10,4))
ax = sns.barplot(x = 'sex', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```



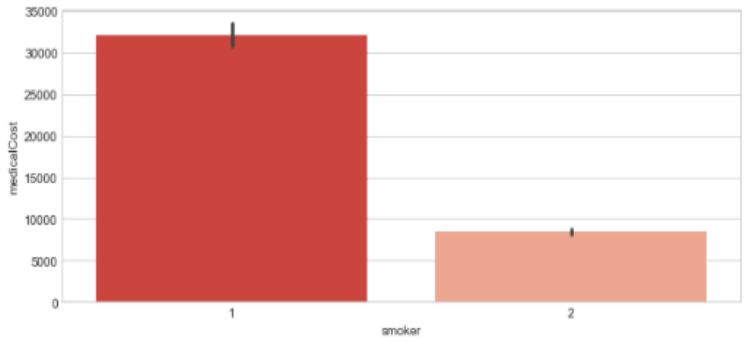
In [17]:
```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'bmi', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```



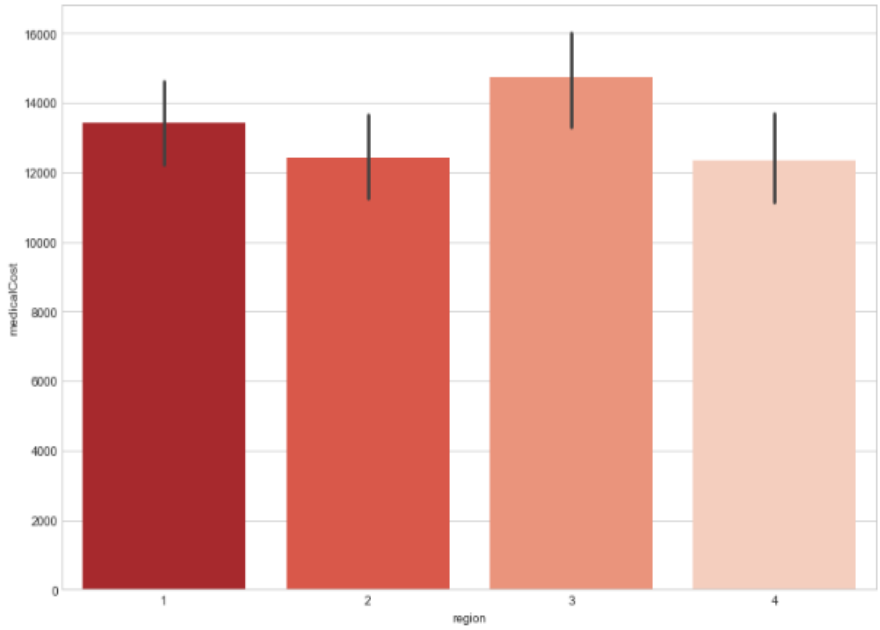
In [18]:
```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'children', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```



In [19]:
```
f, ax = plt.subplots(1,1, figsize=(10,4))
ax = sns.barplot(x = 'smoker', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```



In [20]:
```
f, ax = plt.subplots(1,1, figsize=(12,8))
ax = sns.barplot(x = 'region', y = 'medicalCost',
                 data=dataset, palette='Reds_r')
```

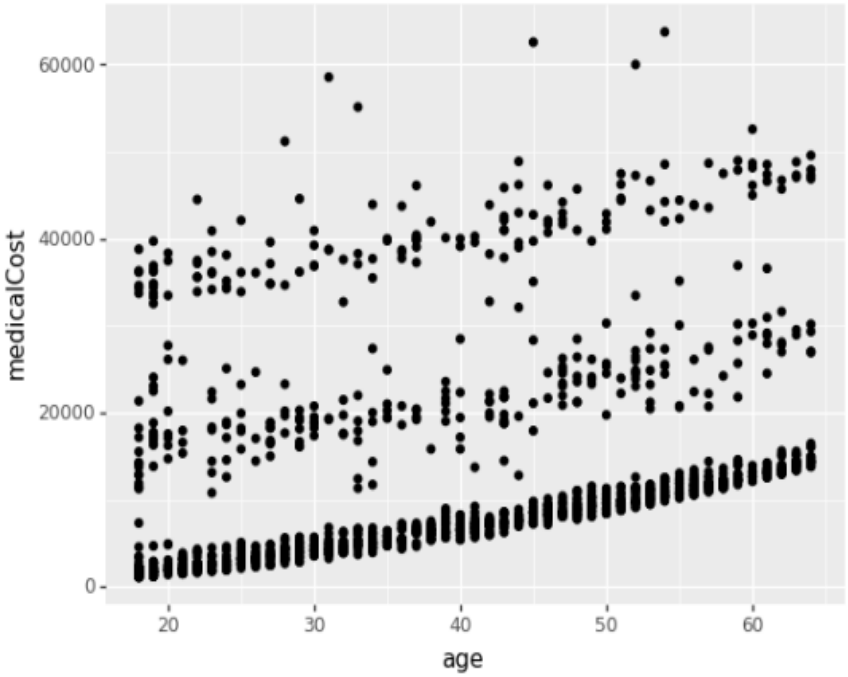


The above bar plots show the varying correlations between each predictor and medical cost. As the age of the insured people increased, the medical cost the insurers bore, increased. People aged 60 and above had medical insurance cost of between 20,000 and 25,000. But people between ages 18 and 21, cost the insurers less in medical cost as theirs was between 4000 and about 10,000. The difference between the medical costs of these age groups is largely significant. Body mass index (bmi) increased as age increased, therefore considering that the older people had higher medical costs, the same can be inferred for bmi. However, males had higher medical cost than females, but the difference in their costs is not that significant as it is about 1000. This might not be best to be considered as a predictor of medical cost. People with children numbering 5, had medical cost of about 8750 while those with 3 children had the highest medical cost at above 15,000 and interestingly, those with no children had medical cost close to 12,500. The significance of having children or not, in relation to medical cost of insurance, is significant. Individuals who smoked (represented by 1) cost insurers more as their medical cost is above 30,000 while those who did not smoke (represented by 2) had low medical cost of about 8000 which is a significant difference in costs. People from the southeast region (represented by 3) had the highest medical cost at nearly 15,000, however the people from northwest region (represented by 2), had the lowest medical cost

at nearly 13,000. The difference in medical costs based on regions is not as high as that of the difference in medical costs based on age.

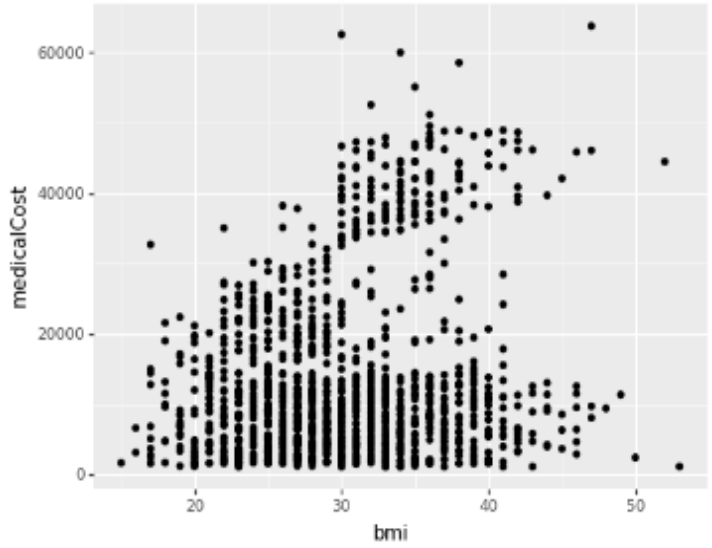
In [22]:
```
from plotnine.data import mpg
from plotnine import ggplot, aes, geom_point

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



Out[22]: <ggplot: (137401397655)>

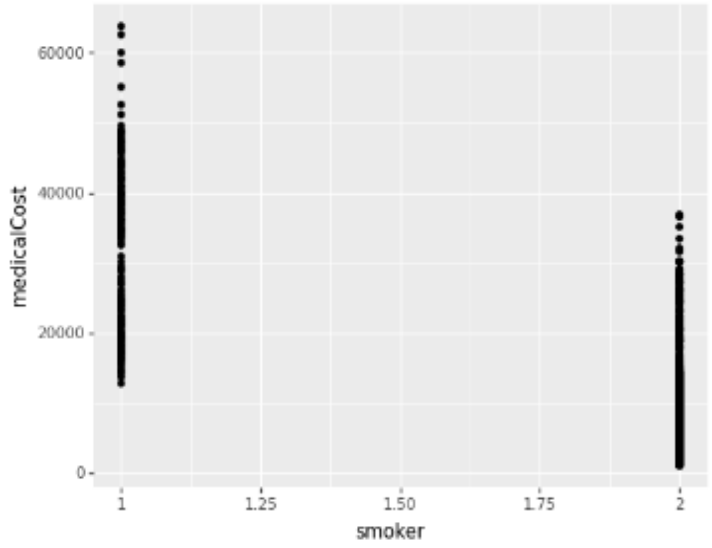
```
ggplot(dataset1) + aes(x="bmi", y="medicalCost") + geom_point()
```



Out[24]: <ggplot: (137401269824)>

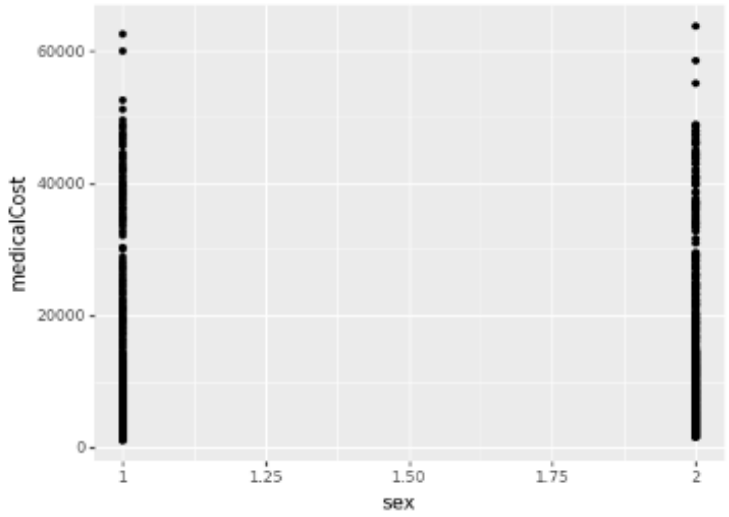
In [26]:
```
# Medical cost of insurance of individuals with bmi(body mass index) between 30 and 50 had the highest medical cost which at its
```

In [27]:
```
ggplot(dataset1) + aes(x="smoker", y="medicalCost") + geom_point()
```



Out[27]: <ggplot: (137403025282)>

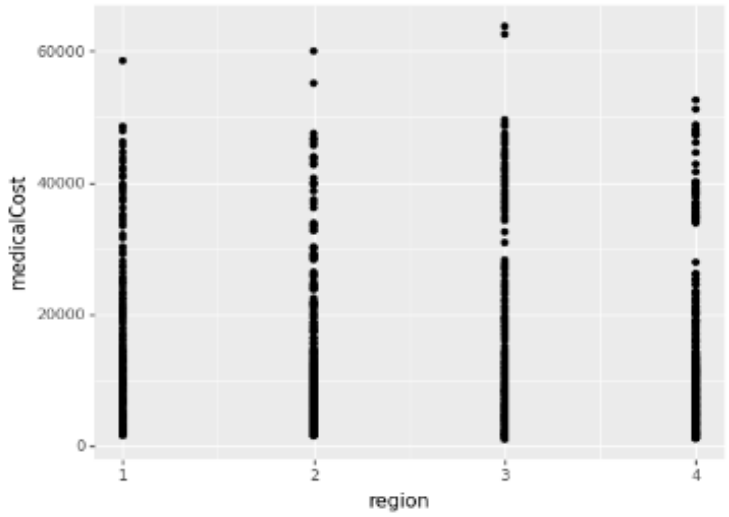
In [29]:
```
ggplot(dataset1) + aes(x="sex", y="medicalCost") + geom_point()
```



Out[29]: <ggplot: (137481656459)>

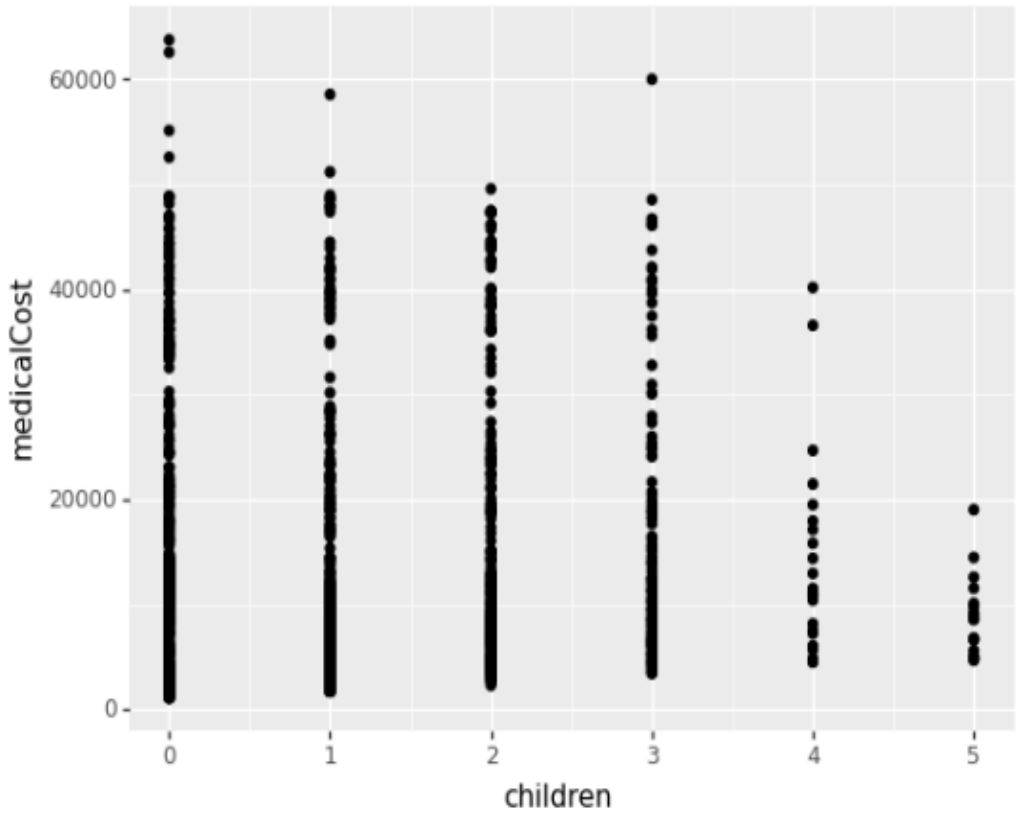
In [30]:
```
# Medical cost of insurance of males respresented as 1 was a bit lower than that of females which is represented as 2
```

In [31]:
```
ggplot(dataset1) + aes(x="region", y="medicalCost") + geom_point()
```



Out[31]: <ggplot: (137481955889)>

In [33]:
```
ggplot(dataset1) + aes(x="children", y="medicalCost") + geom_point()
```

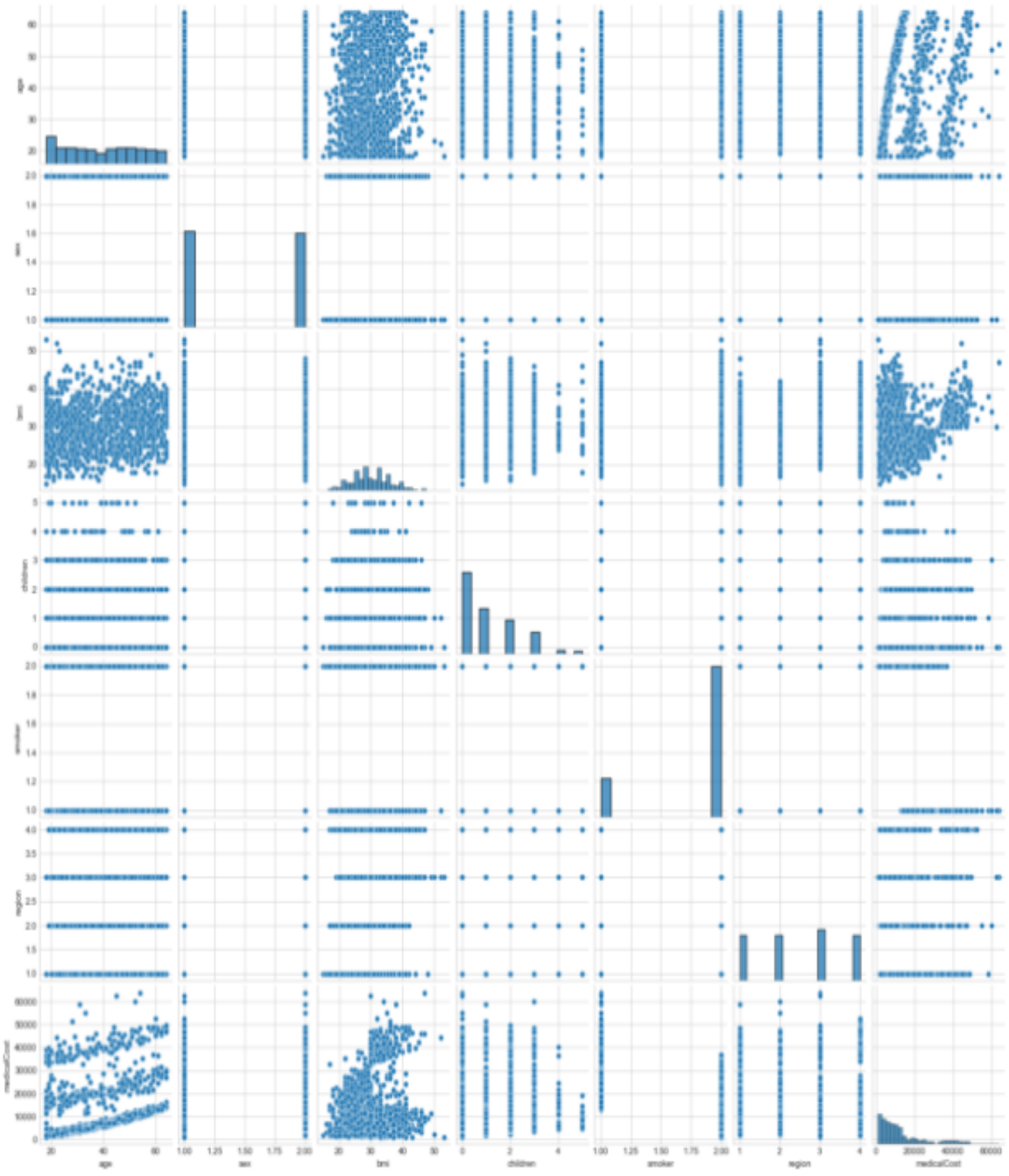


Out[33]:
```
<ggplot: (137403414089)>
```

In [34]:
```
# Medical cost of individuals with no children(0) is the highest while those with 5 children have the lowest medical cost
```

In [35]:
```
sns.pairplot(dataset1[['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'medicalCost']])
```

Out[35]: <seaborn.axisgrid.PairGrid at 0x1ffdb2aae80>



In [37]:
```
# To determine the variables with measures of relationships
dataset1.corr()
```

Out[37]:

	age	sex	bmi	children	smoker	region	medicalCost
age	1.000000	0.020856	0.108437	0.042469	0.025019	0.002127	0.299009
sex	0.020856	1.000000	-0.046503	-0.017163	0.076185	-0.004588	-0.057293
bmi	0.108437	-0.046503	1.000000	0.011097	-0.001669	0.159830	0.196188
children	0.042469	-0.017163	0.011097	1.000000	-0.007673	0.016569	0.067999
smoker	0.025019	0.076185	-0.001669	-0.007673	1.000000	0.002181	-0.787251
region	0.002127	-0.004588	0.159830	0.016569	0.002181	1.000000	-0.006209
medicalCost	0.299009	-0.057293	0.196188	0.067999	-0.787251	-0.006209	1.000000

In [38]:
```
# # Another way ValueError: could not convert string to float: 'female'to show the correlation between the Predictors and 'medic
```

In [39]:
```
dataset1.corr(method='pearson')
```

Out[39]:

	age	sex	bmi	children	smoker	region	medicalCost
age	1.000000	0.020856	0.108437	0.042469	0.025019	0.002127	0.299009
sex	0.020856	1.000000	-0.046503	-0.017163	0.076185	-0.004588	-0.057293
bmi	0.108437	-0.046503	1.000000	0.011097	-0.001669	0.159830	0.196188
children	0.042469	-0.017163	0.011097	1.000000	-0.007673	0.016569	0.067999
smoker	0.025019	0.076185	-0.001669	-0.007673	1.000000	0.002181	-0.787251
region	0.002127	-0.004588	0.159830	0.016569	0.002181	1.000000	-0.006209
medicalCost	0.299009	-0.057293	0.196188	0.067999	-0.787251	-0.006209	1.000000

In [40]:
```
# From the above, 'age', 'bmi' and 'children' have positive correlation with 'medicalCost'. While 'sex','smoker' and 'region' ha
```

In [41]:
```
# To show the correlation values more clearly;

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

Out[41]: 0.29900819333064765

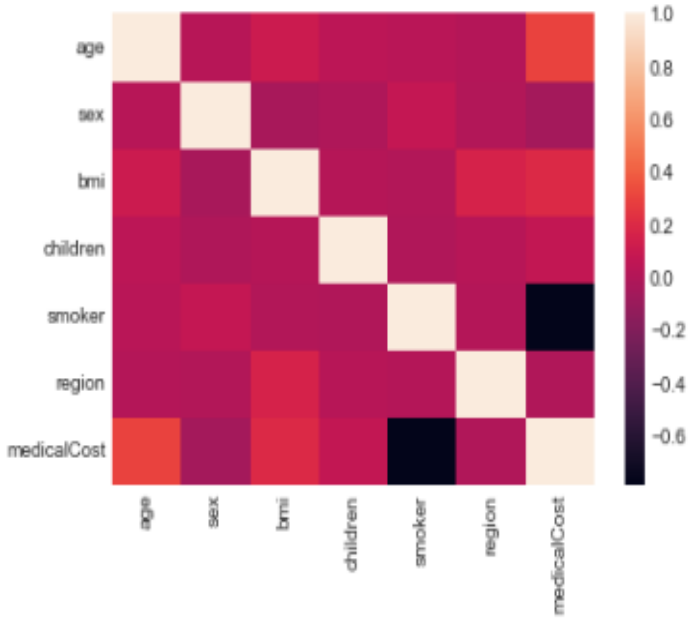
In [42]:
```
dataset1['bmi'].corr(dataset['medicalCost'])
```

Out[42]: 0.19618775134063157

In [43]:
```
dataset1['children'].corr(dataset['medicalCost'])
```

Out[43]: 0.06799822684790487

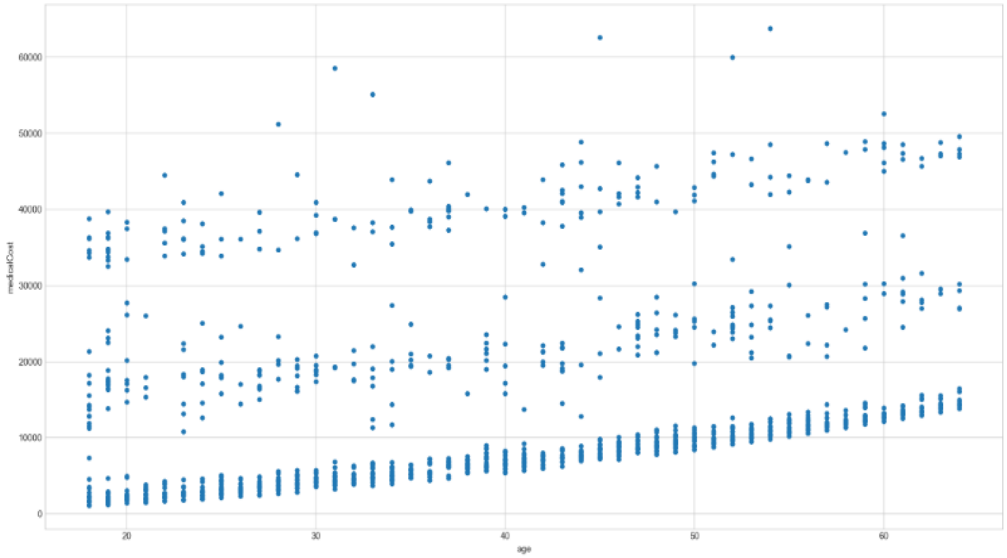
In [44]:
```
# Build correlation matrix to show correlation between the variables
sns.heatmap(dataset1.corr())
plt.show()
```



In [47]:
```
%matplotlib notebook
```

In [48]:
```
dataset1.plot(kind='scatter', x='age', y='medicalCost')
```

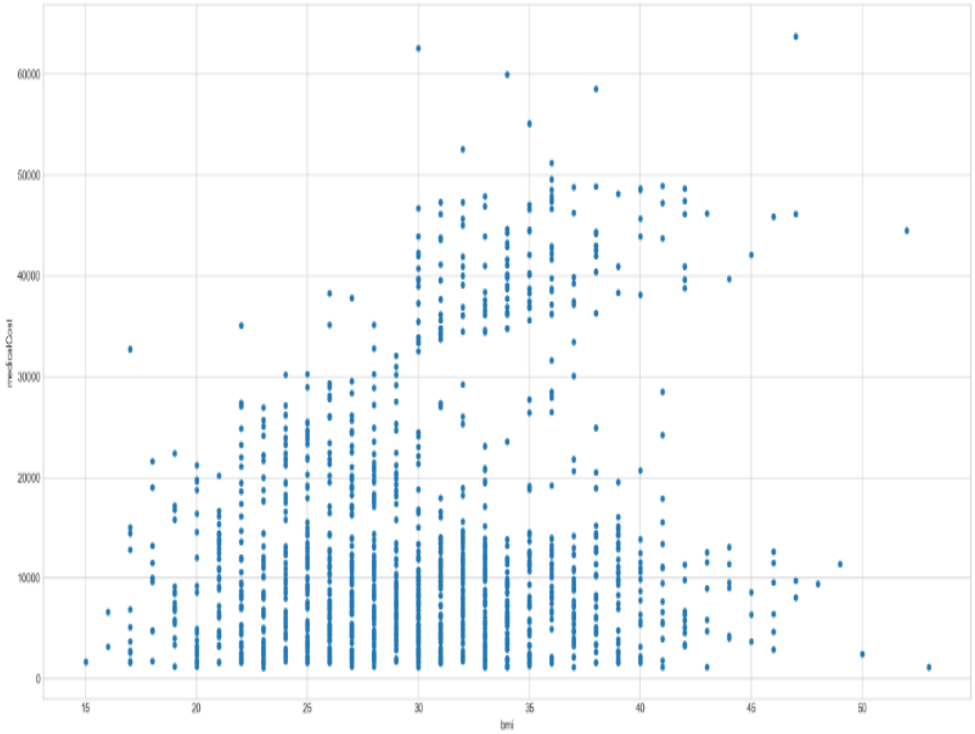
```
<IPython.core.display.Javascript object>
```



Out[48]:
```
<AxesSubplot:xlabel='age', ylabel='medicalCost'>
```

In [49]:
```
dataset1.plot(kind='scatter', x='bmi', y='medicalCost')
```

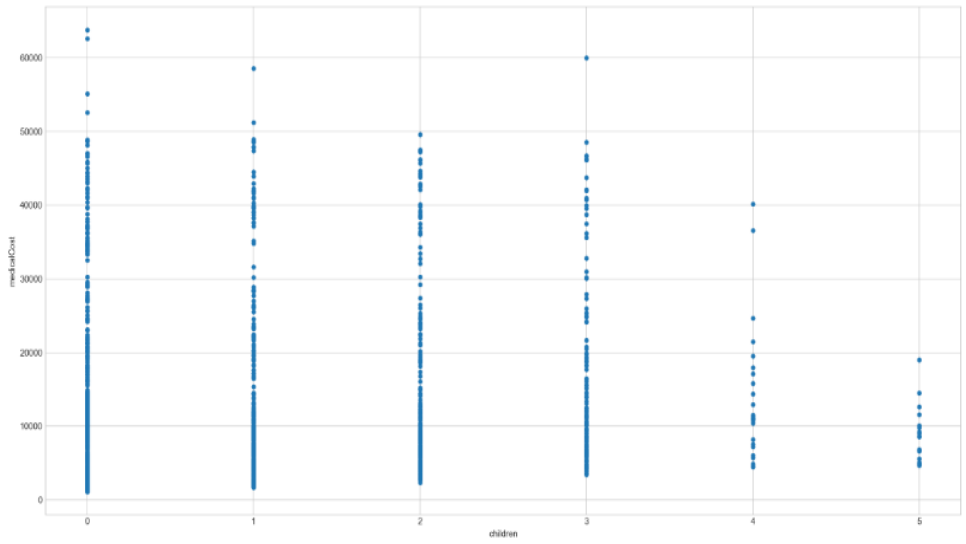
```
<IPython.core.display.Javascript object>
```



Out[49]:
```
<AxesSubplot:xlabel='bmi', ylabel='medicalCost'>
```

```
In [50]: dataset1.plot(kind='scatter', x='children', y='medicalCost')
```

```
<IPython.core.display.Javascript object>
```



To Predict the Medical costs values of potential clients with the represented features:

In [51]:
```
# Building simple linear regression models using each of the three predictors above and 'medcialCost':
```

In [52]:
```
# Linear regression model with predictor 'age' and 'medicalCost'
X_var = dataset1[['age']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [53]:
```
# 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):                  86.436
Model:                            OLS   Adj. R-squared (uncentered):             86.500
Method:                 Least Squares   F-statistic:                             -1353.
Date:                Tue, 12 Jul 2022   Prob (F-statistic):                       1.00
Time:                        17:48:47   Log-Likelihood:                        -14421.
No. Observations:                1338   AIC:                                  2.884e+04
Df Residuals:                    1337   BIC:                                  2.885e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
age          329.2762      7.618     43.223      0.000     314.331     344.221
==============================================================================
Omnibus:                      393.484   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              840.468
Skew:                           1.714   Prob(JB):                    3.12e-183
Kurtosis:                       4.823   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 [55]:
```
# Linear regression model with predictor 'bmi' and 'medicalCost'
X_var = dataset1[['bmi']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [56]:
```
# 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):                   90.510
Model:                            OLS   Adj. R-squared (uncentered):              90.577
Method:                 Least Squares   F-statistic:                             -1352.
Date:                Tue, 12 Jul 2022   Prob (F-statistic):                        1.00
Time:                        17:48:47   Log-Likelihood:                         -14452.
No. Observations:                1338   AIC:                                 2.891e+04
Df Residuals:                    1337   BIC:                                 2.891e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
bmi          437.8047     10.549     41.501      0.000     417.110     458.500
==============================================================================
Omnibus:                      248.823   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              401.174
Skew:                           1.259   Prob(JB):                     7.69e-88
Kurtosis:                       3.923   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 [58]:
```
# Linear regression model with predictor 'children' and 'medicalCost'
X_var = dataset1[['children']] # independent variable
y_var = dataset1['medicalCost'] # dependent variable
```

In [59]:
```
# 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:                        17:48:47   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 [76]:
```
# f. Building 2 multivariate regression models
# i. Using Multiple linear regression models with 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 [103]:
```
multi_lingreg = LinearRegression()
multi_lingreg.fit(x_train, y_train)
print(multi_lingreg.intercept_)
print(multi_lingreg.coef_)
print(multi_lingreg.score(x_test, y_test))
```

```
12537.726544968004
[  35.6019608    -4.71278257   82.98562337 -519.00431532 -223.14010113
 -780.18903   ]
-0.023445953779931417
```

In [105]:
```
# ii. Using Multiple linear regression 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))
```

```
35923.52928221541
[   254.02598346     24.98295267    327.61871331    448.04695482
 -23579.97255793   -292.66769323]
0.7994955354992438
```

In [106]:
```
multi_lingreg = LinearRegression()
multi_lingreg.fit(x_train, y_train)
print(multi_lingreg.intercept_)
print(multi_lingreg.coef_)
print(multi_lingreg.score(x_test, y_test))
```

```
35923.52928221541
[   254.02598346     24.98295267    327.61871331    448.04695482
 -23579.97255793   -292.66769323]
0.7994955354992438
```

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 insurance company. However, the second multivariate regression model with all the predictors had a score of about 79% which shows that the accuracy was good when the model had all the predictors included. Instead, other machine models for supervised learning should be built to ascertain if they would be better for predicting medical costs using the three predictors which are correlated with the dependent variable. It is important to note that Linear regression model works better with less independent variables (one independent variable) and one dependent variable. Multivariate regression models as the name indicates, works best with several independent variables and one or two dependent variables.

Predicting the Medical cost using all the predictors:

In [116]:
```
#To Predict the Medical costs; first defining the features:
#PREDICTORS
X = dataset[['age', 'sex', 'bmi', 'children','smoker','region']]

y = dataset['medicalCost']

# Then using the train_test_split method to split that dataset import the library:
from sklearn.model_selection import train_test_split
# splitting train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#Evaluation using Linear regression
# Import the relevant library:
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# Fit linear model on train set
model.fit(X_train,y_train)

predictions = model.predict(X_test)
predictions[0:3]
```

Out[116]:
```
array([12204.8232974 , 10415.41964421,  6975.10522622])
```

In [ ]:
```
# Above result are the prediction values for medical costs using all the predictors
```

In [117]:
```
# Visualisation of the above predictions and the test set:
plt.scatter(y_test, predictions)
```

Out[117]:
```
<matplotlib.collections.PathCollection at 0x216e3943340>
```

In [120]:
```
# Calculate the mean absolute error and mean squared error of the above results:
from sklearn import metrics
print(metrics.mean_absolute_error(y_test, predictions))
print(metrics.mean_squared_error(y_test, predictions))
```

```
4273.638823080483
38309551.88455172
```

Predicting medical costs using the three predictors:

In [121]:
```
#To Predict the Medical costs; first defining the features:
#PREDICTORS
X = dataset[['age','bmi','children']]

y = dataset['medicalCost']

# Then using the train_test_split method to split that dataset import the library:
from sklearn.model_selection import train_test_split
# splitting train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#Evaluation using Linear regression
# Import the relevant library:
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# Fit linear model on train set
model.fit(X_train,y_train)

predictions = model.predict(X_test)
predictions[0:3]
```

Out[121]:
```
array([ 5158.98397841, 20714.57696447, 14778.377244  ])
```

The above shows that the predicted medical costs will range from 6975 and 12204 using all the predictors while using the three predictors of 'age', 'bmi' and 'children', the medical cost prediction is between 5158 and 14,778. The three predictors gave a higher medical cost prediction than when all the predictors were factored into the calculation. It might be better for the insurance company to work with the higher predicted values as it could reduce their exposure risks. However, Linear regression model as a machine model is not the best forecast medical costs of clients for the insurance company as the error values are large. Therefore, other machine models which are multivariate regression models like Random Forest regressor or DecisionTreeRegressor can be considered as seen below:

RANDOM FOREST REGRESSOR:

In [66]:
```
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: 3890319.024, MSE test data: 19527912.910
R2 train data: 0.970, R2 test data: 0.882
```

In [67]:
```
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: 3890319.024, MSE test data: 19527912.910
R2 train data: 0.970, R2 test data: 0.882
```

From the above outputs, the Random Forest regressor produced high MSE values. The MSE represents the mean square error which is used to measure the accuracy of the model. It measures the distance between the predicted values and actual values. The MSE values is high for both train and test sets, however, it is important to bear in mind that the dataset is large, which could result in a higher MSE. This error value is the same measure when all the predictors were used and when only the three predictors of ‘age’, ‘bmi’ and ‘children’ were used for the model. However, considering that the R squared values (the degree of goodness of the machine model) for both test and train data 88% and 97% respectively, this model can be considered for predicting medical costs for the insurance company.

DECISION TREE REGRESSOR:

In [84]:
```
# DecisionTreeRegressor with only the three predictors:
x = dataset1.drop(['sex','smoker','region','medicalCost'], axis = 1)
y = dataset1['medicalCost']
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)
```

In [88]:
```
# Using the DecisionTreeRegressor of the tree library of Scikit-Learn to train the model and make predictions:
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

# To compare the predicted values of medical costs and its actual values:
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

Out[88]:

	Actual	Predicted
205	4337	4040.0
486	12475	11013.0
817	3597	37484.0
845	45008	12638.0
430	23082	36898.0
...	...	...
1314	18765	4931.0
631	1977	3392.0
716	9566	21098.0
402	14692	13393.0
1096	44641	10848.0

268 rows × 2 columns

In [ ]:
```
# The observations of the two columns varies as the train_test_split function splits the dataset randomly. However, some predicte
```

In [89]:
```
# Evaluating the model:
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 10932.186567164179
Mean Squared Error: 296118932.8955224
Root Mean Squared Error: 17208.106604025976
```

In [90]:
```
# DecisionTreeRegressor with all the predictors:
x = dataset1.drop(['medicalCost'], axis = 1)
y = dataset1['medicalCost']
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)
```

In [91]:
```
# Using the DecisionTreeRegressor of the tree library of Scikit-Learn to train the model and make predictions:
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

# To compare the predicted values of medical costs and its actual values:
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

Out[91]:

	Actual	Predicted
930	2927	3410.0
822	1621	1622.0
349	1635	16884.0
761	2416	27724.0
43	6313	6796.0
...	...	...
268	7441	8027.0
885	19719	3866.0
875	2690	18033.0
636	2709	1837.0
868	13129	25678.0

268 rows × 2 columns

In [ ]:
```
test_split function splits the dataset randomly. However, some of the predicted values are close in measure to the actual values.
```

In [92]:
```
# Evaluating the model:
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 10331.888059701492
Mean Squared Error: 270630188.8488806
Root Mean Squared Error: 16450.84158482114
```

The Decision Tree model for prediction of medical cost using all the predictors had lower error values (MAE, MSE, RMSE) than using the three predictors: 'age', 'bmi' and 'children'. This shows that for this model, the more the independent variables, the better the accuracy of prediction of the dependent variable value.

CONCLUSION

Medical insurance companies can save time and money when they have researchers (Data scientists) who can deploy machine learning models like linear regression (if the aim of the business question involves just one independent variable (like age) to predict a dependent variable (like medical cost). Also, random forest regressor or decision trees can be deployed as machine models to accurately forecast medical expenses of the insured or potential clients for the insurance company (if the dataset analysis requires using several independent variables like bmi and children with one more dependent variables). This would help the business to improve their revenue as they would have the required information of the cost of medical insurance of old and potential clients based on their age, bmi and if they have children (and how many they have). Also, people with higher bmi, tend to have more health issues as this factor seems to be strongly correlated with diseases according to the USA's Centers for Disease Control and Prevention (Prevention, n.d.). This means the higher the bmi of a person, the higher the medical cost for treating their diseases which will be higher than someone with lower bmi). People with children will have higher medical cost of insurance as each child could require medical treatment as some point of their lives, which is added cost insurance companies need to factor in to accurately predict their costs. This will not be the situation for people with no children and the medical cost will just be for them. Other factors can be considered to make this medical cost prediction; however, these three predictors are the best features to use for medical insurance companies to adequately save money and time in developing machine models best suited for them.

Bibliography

Prevention, C. f. D. C. a., n.d.
https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html. [Online].