In [2]:
```
#Installing the Libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
#Importing insurance.csv dataset
insurance_dataset=pd.read_csv("insurance.csv")
```

In [4]:
```
insurance_dataset
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges
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 [6]:
```
insurance_dataset.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 [7]:

```
insurance_dataset.describe()
```

Out[7]:

	age	bmi	children	charges
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 [8]:

```
insurance_dataset.isnull().sum()
```

Out[8]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [9]:
```
insurance_dataset['sex'].value_counts()
```

Out[9]:
```
male      676
female    662
Name: sex, dtype: int64
```

In [10]:
```
insurance_dataset['age'].value_counts()
```

Out[10]:

```
18    69
19    68
50    29
51    29
47    29
46    29
45    29
20    29
48    29
52    29
22    28
49    28
54    28
53    28
21    28
26    28
24    28
25    28
28    28
27    28
23    28
43    27
29    27
30    27
41    27
42    27
44    27
31    27
40    27
32    26
33    26
56    26
34    26
55    26
57    26
37    25
59    25
58    25
36    25
38    25
35    25
39    25
61    23
60    23
```

```
63    23
62    23
64    22
Name: age, dtype: int64
```

In [11]:

```
insurance_dataset['children'].value_counts()
```

Out[11]:

```
0    574
1    324
2    240
3    157
4     25
5     18
Name: children, dtype: int64
```

In [12]:

```
insurance_dataset['smoker'].value_counts()
```

Out[12]:

```
no     1064
yes     274
Name: smoker, dtype: int64
```

In [13]:

```
insurance_dataset['region'].value_counts()
```

Out[13]:

```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

In [14]:

```
insurance_dataset.columns
```

Out[14]:

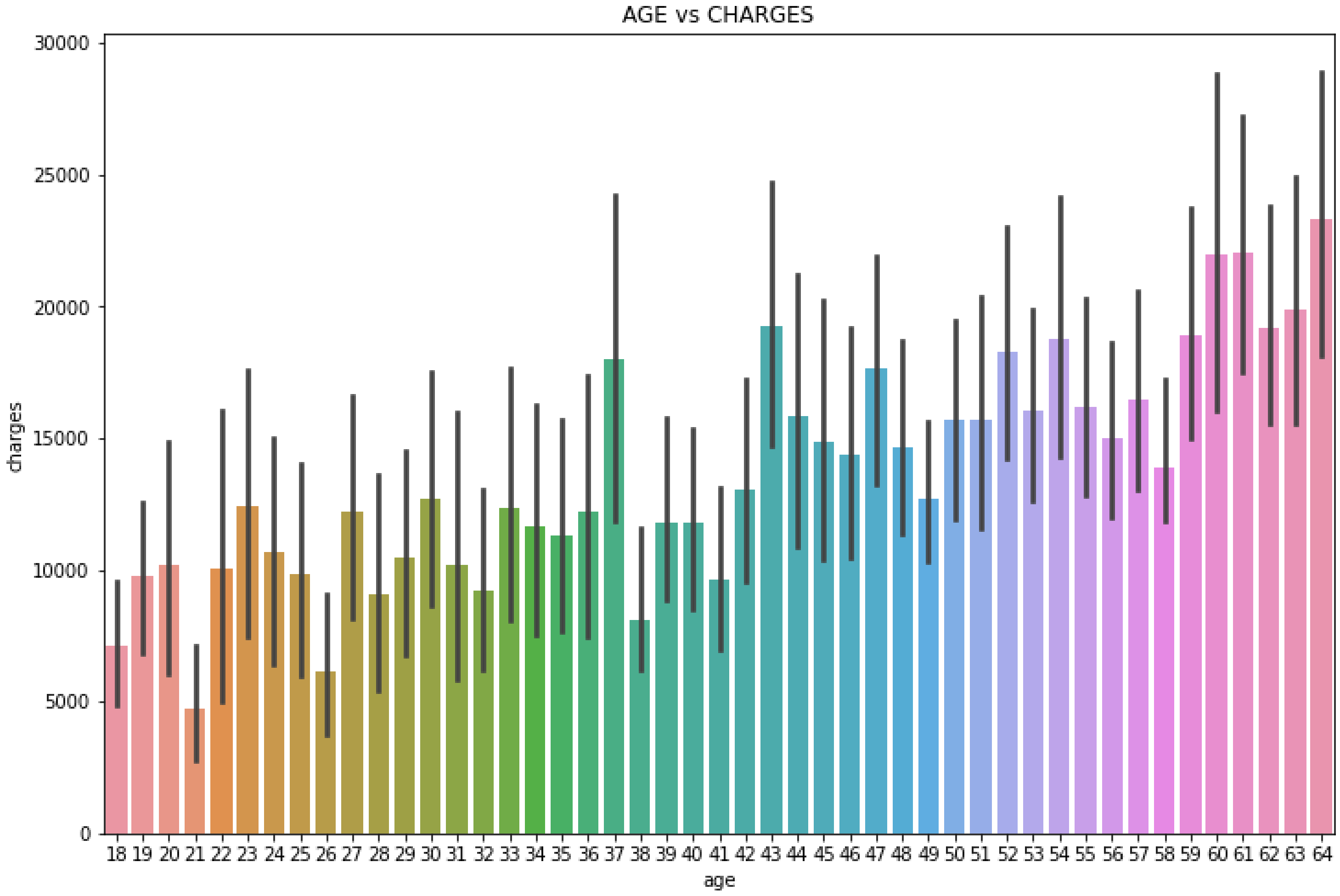
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

In [8]:

```
#Data Analysis

#Age vs Charges
plt.figure(figsize = (12, 8))
x=insurance_dataset['age']
y=insurance_dataset['charges']
sns.barplot(x,y)
```

```
plt.title('AGE vs CHARGES')
plt.show()
```

```
C:\Python\Python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

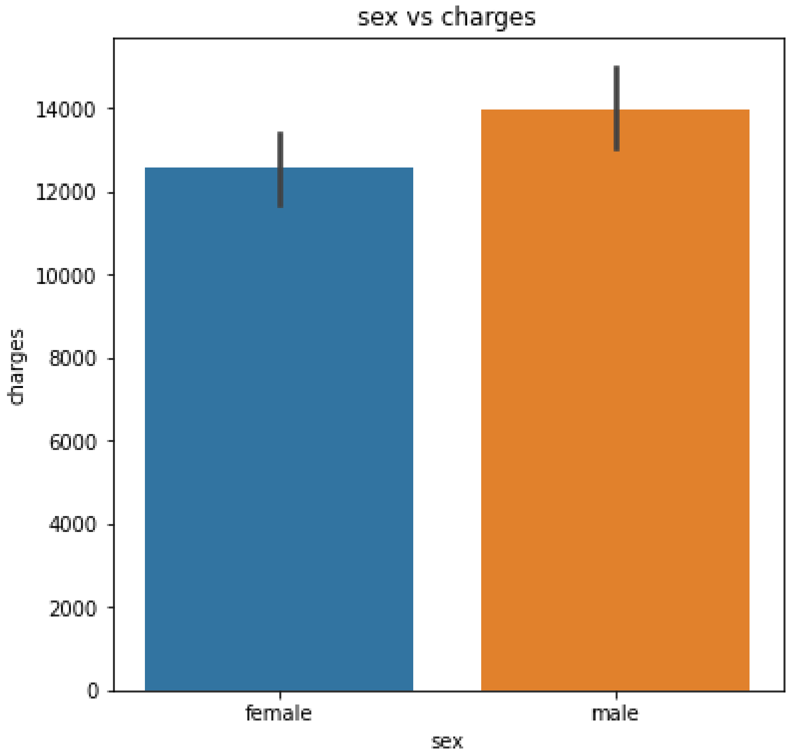


In [9]:

```
# sex vs charges
# males have slightly greater insurance charges than females in general

plt.figure(figsize = (6, 6))
```

```
x=insurance_dataset['sex']
y=insurance_dataset['charges']

sns.barplot(x,y)

plt.title('sex vs charges')
plt.show()
```

```
C:\Python\Python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

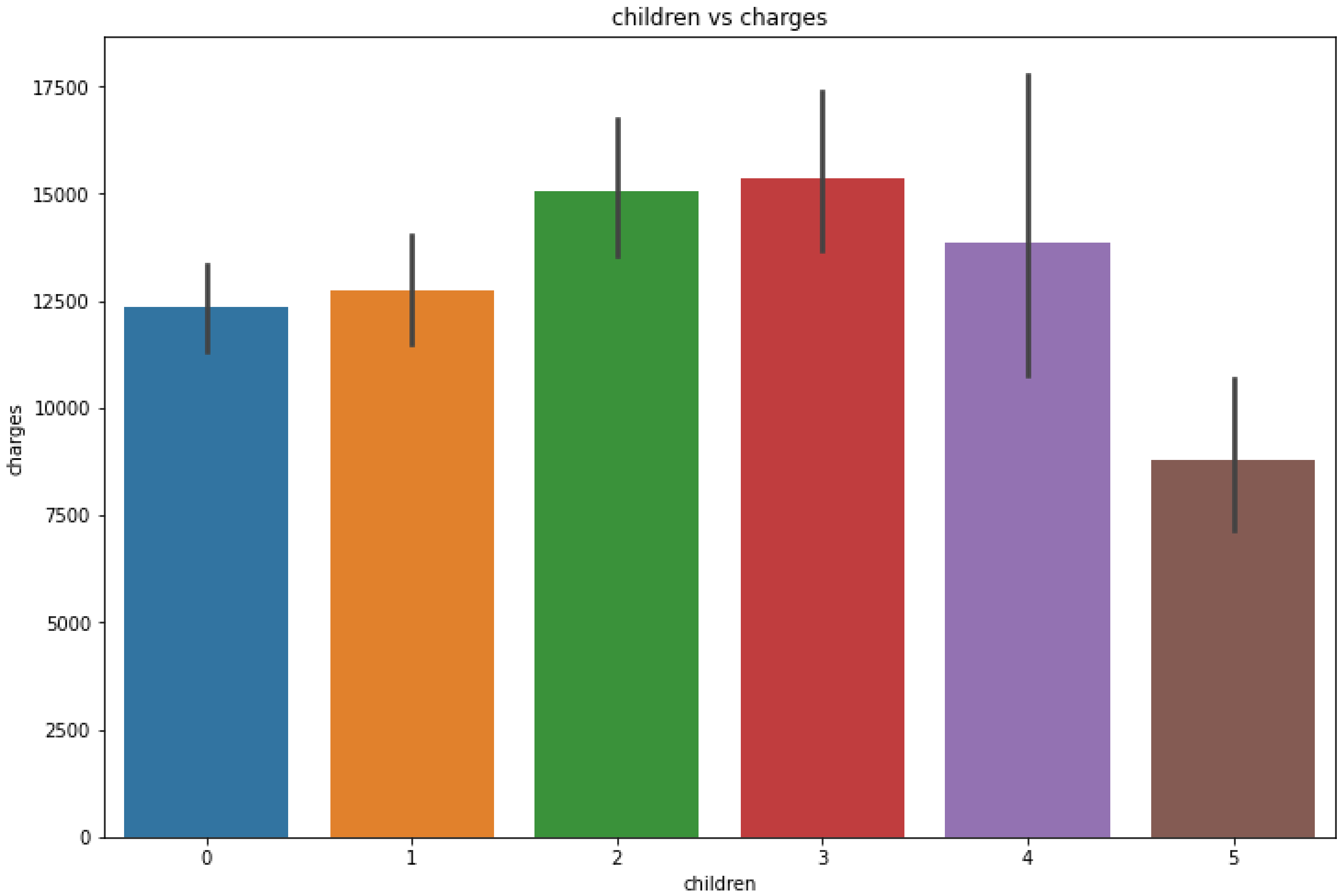


In [10]:

```
# children vs charges
# no. of childrens of a person has a very interesting dependency on insurance costs

plt.figure(figsize = (12, 8))
x=insurance_dataset['children']
y=insurance_dataset['charges']
```

```
sns.barplot(x,y)

plt.title('children vs charges')
plt.show()
```

```
C:\Python\Python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

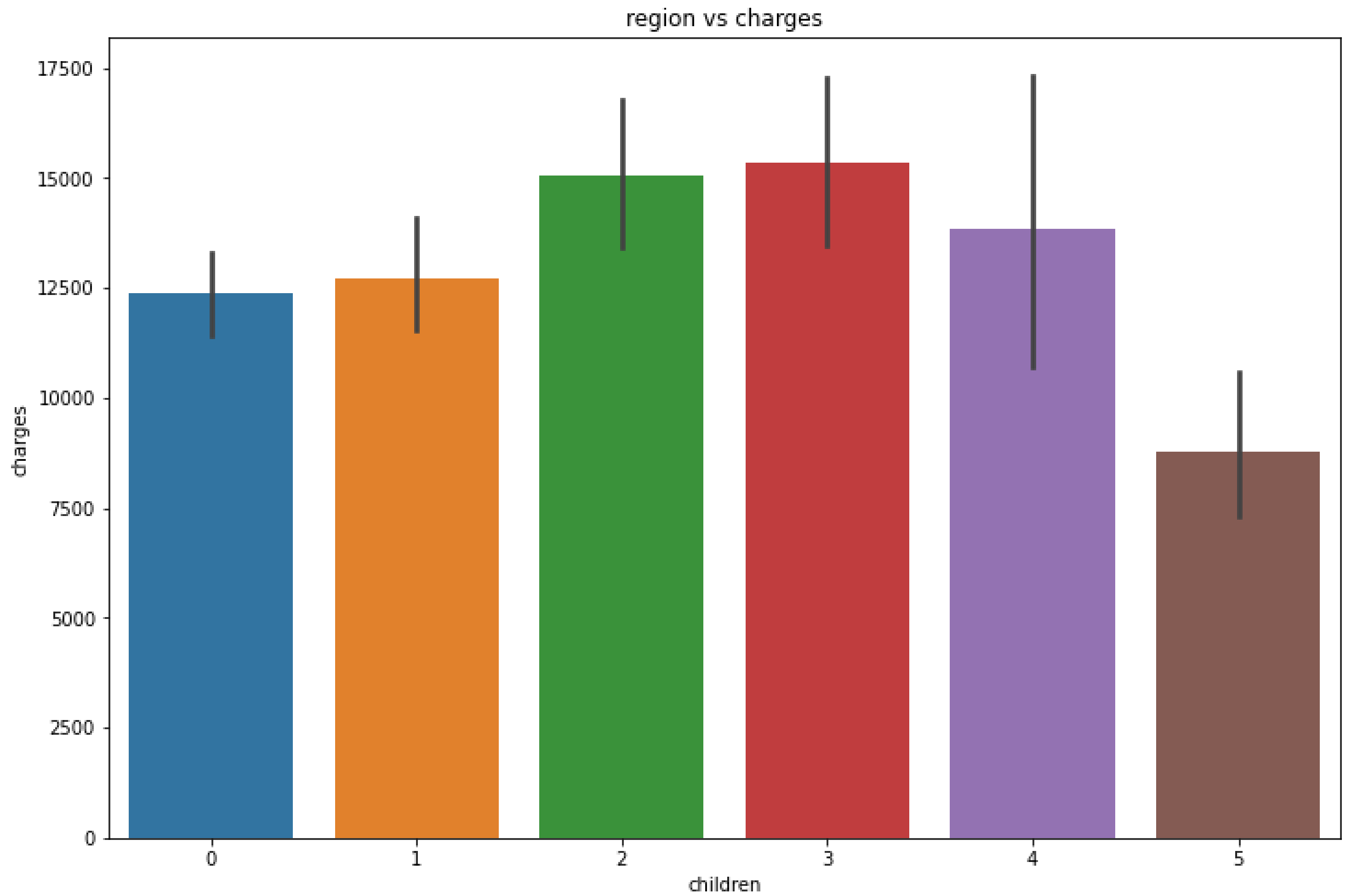


In [11]:

```
# region vs charges
```

```
# From the graph we can see that the region actually does not play any role in determining the insurance charges

plt.figure(figsize = (12, 8))
x=insurance_dataset['children']
y=insurance_dataset['charges']

sns.barplot(x,y)

plt.title('region vs charges')
plt.show()
```

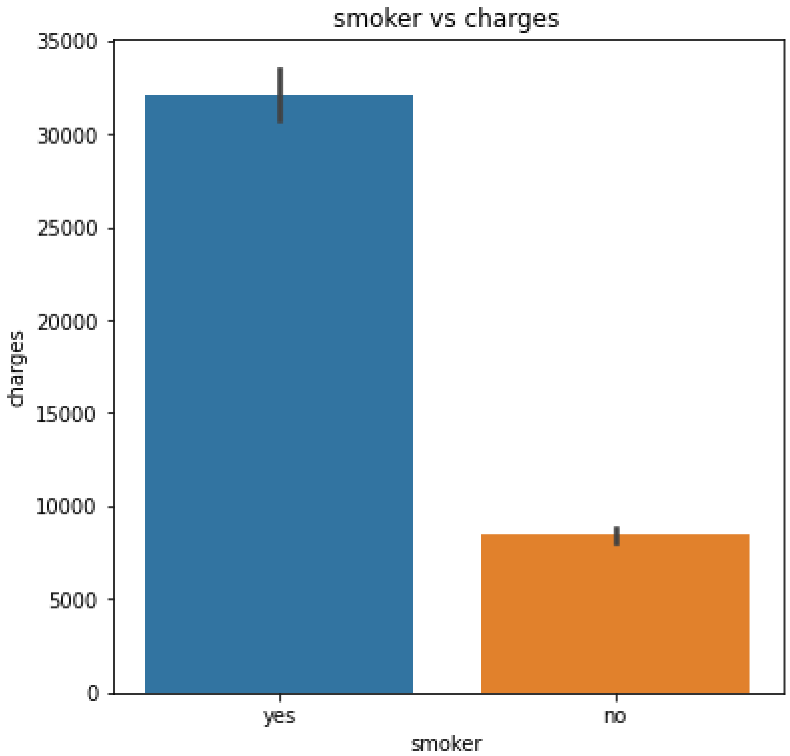
```
C:\Python\Python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [12]:

```
# smoker vs charges
# from the graph below, it is visible that smokers have more insurance charges than the non smokers

plt.figure(figsize = (6, 6))
x=insurance_dataset['smoker']
y=insurance_dataset['charges']
sns.barplot(x,y)

plt.title('smoker vs charges')
plt.show()
```

```
C:\Python\Python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

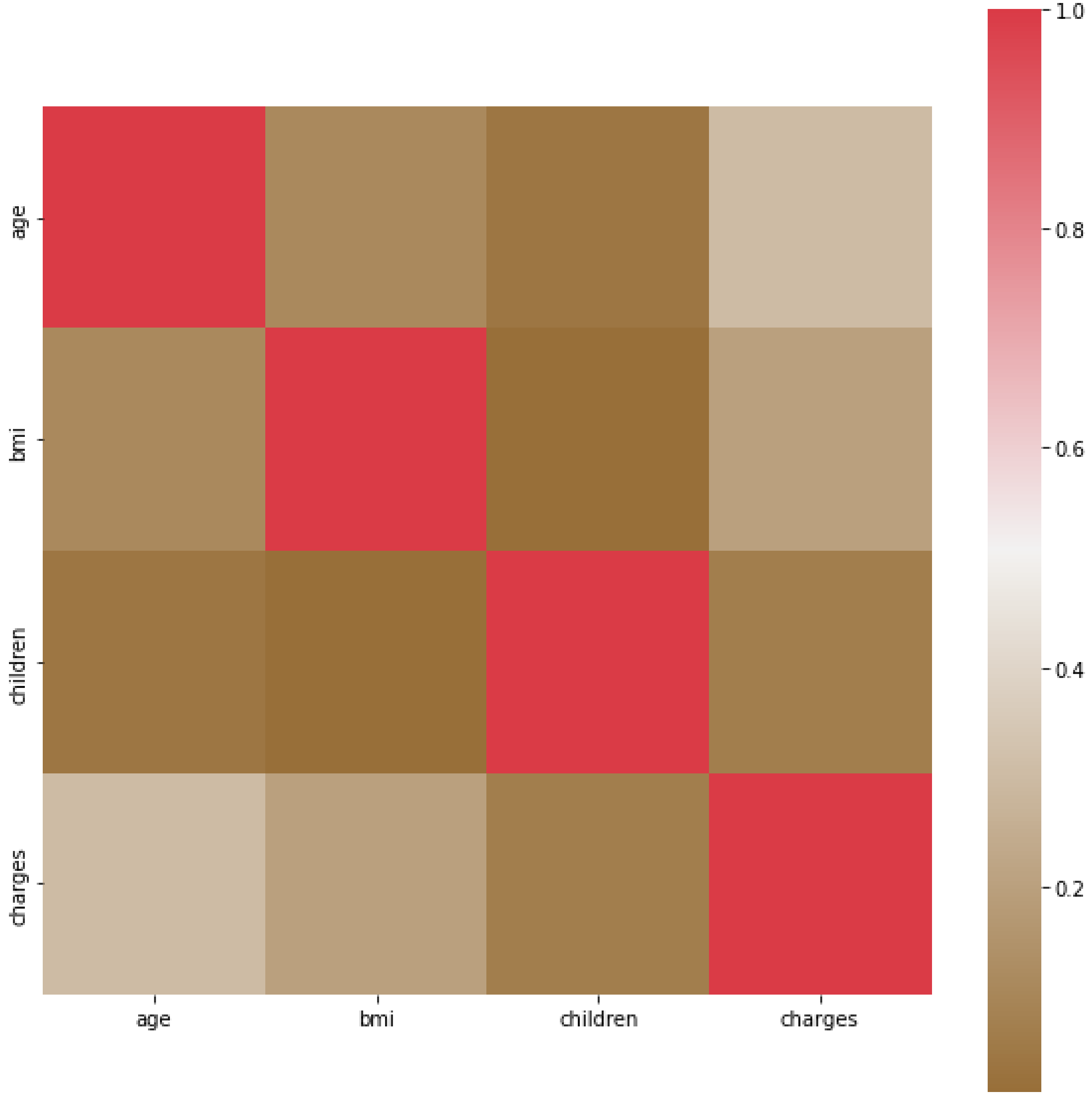


In [14]:

```python
# plotting the correlation plot for the dataset

f, ax = plt.subplots(figsize = (10, 10))

corr = insurance_dataset.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
            cmap = sns.diverging_palette(50, 10, as_cmap = True), square = True, ax = ax)
```

```
C:\Users\TOSHIBA\AppData\Local\Temp\ipykernel_9496\3944541768.py:6: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
```

Out[14]: <AxesSubplot:>



In [15]:

```
# removing unnecassary columns from the dataset

insurance_dataset = insurance_dataset.drop('region', axis = 1)

print(insurance_dataset.shape)
```

```
insurance_dataset.columns
```

Out[15]:
```
(1338, 6)
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'charges'], dtype='object')
```

In [17]:
```
# label encoding for sex and smoker

# importing label encoder
from sklearn.preprocessing import LabelEncoder

# creating a label encoder
le = LabelEncoder()


# label encoding for sex
# 0 for females and 1 for males
insurance_dataset['sex'] = le.fit_transform(insurance_dataset['sex'])

# label encoding for smoker
# 0 for smokers and 1 for non smokers
insurance_dataset['smoker'] = le.fit_transform(insurance_dataset['smoker'])
```

In [18]:
```
insurance_dataset['sex'].value_counts()
```

Out[18]:
```
1    676
0    662
Name: sex, dtype: int64
```

In [19]:
```
insurance_dataset['smoker'].value_counts()
```

Out[19]:
```
0    1064
1     274
Name: smoker, dtype: int64
```

In [22]:
```
# splitting the dependent and independent variable

x = insurance_dataset.iloc[:,:5]
y = insurance_dataset.iloc[:,5]
```

```
print(x.shape)
print(y.shape)
```

```
(1338, 5)
(1338,)
```

In [24]:

```
# splitting the dataset into training and testing sets

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 = 30)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1070, 5)
(268, 5)
(1070,)
(268,)
```

In [25]:

```
#Modelling

#Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


# creating the model
model = LinearRegression()

# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
```

```
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)

# Calculating the r2 score
r2 = r2_score(y_test, y_pred)
print("r2 score :", r2)
```

```
MSE : 37057975.783359274
RMSE : 6087.526244983202
r2 score : 0.759758528014896
```

In [26]:

```
y_test
```

Out[26]:

```
338    41919.09700
620     3659.34600
965     4746.34400
128    32734.18630
329     9144.56500
          ...
580    12913.99240
786    12741.16745
321    24671.66334
903     8125.78450
613     6753.03800
Name: charges, Length: 268, dtype: float64
```

In [27]:

```
y_pred
```

Out[27]:
```
array([35206.95988424,  5913.25371451,  5902.11903681, 26652.44561859,
       13037.68322943,  6976.63677636,  4714.42087021,  9784.76464047,
       33699.76519397, 11386.17430622,  8038.6451187 , 10230.0795605 ,
       35207.19003322, 11538.33660241,  1379.9011798 , 36009.29793685,
       11972.55413556,  9251.68230172, 29236.87908302, 11307.31833466,
        1438.78621227,  8722.85417   ,  8119.68124443, 11284.22664405,
       33324.77761469, 39447.6034078 , 14940.85243651,  3423.64409705,
       11041.14918239, 14793.12711084,  2187.2244934 , 31013.67789827,
        1465.9188615 , 15025.76344263, 12731.54461399,  8998.82427631,
        2405.26396426,  6992.56675683,   724.24627861,  8735.58652147,
        6939.69247432, 34200.98347795,  2536.40883799, 14066.35901945,
       10715.10391391, 30446.39967614,  5420.74682483,  3134.91154318,
        2864.74903307,  5583.45182778, 11557.18128072, 12987.4115041 ,
       12420.44483284,  1407.03382903, 11451.7642406 ,  7496.42488881,
       31522.42980901,  3573.16654523, 12963.45669772,  5015.13453229,
       15235.0742298 , 11213.15074468, 11460.11010792,  7949.0636769 ,
        -466.65361082,  8241.77639306, 10298.61718875,  1649.75256961,
        3006.35193724, 15048.85142823, 10915.66933214,  2398.74902057,
         294.41833049,  9764.87488799, 16276.17413659,  -668.29539324,
       12728.61382067,  9823.57715698, 31074.04522796,  4210.00456841,
        1090.07906619, 11224.20445106,  9559.40169037, 31013.16340893,
        1220.13786991, 16889.04045283,  9582.0497683 , 17464.36679108,
        9229.03051879, 11296.93510119, 14128.87209807, 13800.56919382,
       26876.29247035, 11030.42741716, 37112.86022059,  5720.08615622,
       10022.76185487,  1486.82090741,  7150.65202577, 35620.29227205,
       34904.1721693 ,  8618.65198729,  9969.88757236,  8334.46168583,
       33327.58703064, 26229.34215756,  7319.09663385, 36219.40755375,
       33958.86041332,  2137.49054761, 10466.7094654 ,  6852.63261838,
         842.56463401, 28219.28688021,  5983.40570178,  4206.47460942,
       12757.02910307, 10159.83689608, 10056.0169256 , 15214.6260922 ,
        4465.70189058, 12626.15166915, 36508.91544617, 14342.5798803 ,
       29835.75740984, 10946.16978243, 13980.73756495,  8818.76779201,
       31340.01781708, 12291.45890359, 11399.9694375 ,  5599.01628127,
       34078.41418993, 27318.72049284, 12377.22953576,  2996.6049593 ,
       40826.47213082, 25519.90286143, 11933.17025369,  6692.04326826,
       12707.36685336, 10447.66243841,  3634.87418919, 37619.58464454,
        8000.79440925, 10914.13964943,  6571.8635043 ,  3553.82508322,
        9849.83638049, 32982.15559928, 16134.58503209,  6630.67705599,
       11135.77707034,  6363.86174138,  9499.13244785,  8423.33578014,
        5115.63408719, 12824.4769563 , 17935.15610553,  5466.63615986,
       10799.19219598, 31715.39770325, 14406.3790818 , 10270.82457752,
       15338.84681479, 15542.22164896, 34742.01894654,  8224.02889979,
        7206.25393349,  2093.20237606,  8533.08450889, 11961.73449846,
        6079.68485076,  4885.9790351 , 33494.84028682,  7413.50756836,
```

```
        3428.18264445, 10443.50649208, 14845.70695833, 30251.01859315,
       10710.05746776, 14359.86397471,  1527.75878257,   581.96672718,
       39139.13606464, 28076.22867404, 27336.55309301,  7125.86478951,
       31653.19966523, 14799.8684985 ,  -317.13116264,  9585.74559021,
       14102.05448943, 12017.47178418,  1983.25398327, 12623.24438134,
       15827.42773357, 31508.63838275, 30456.37680305,  5752.4818162 ,
       10147.97426532,  3088.80934855,  7727.54533755,  7179.26675692,
       28057.49298264,  1339.18047336,  5099.13852687, 37067.88138681,
       35597.4849218 , 37339.03928978,  6680.41449149,  7540.69986481,
       39221.15007856,  6749.76704493,  5807.07884052, 38666.85806332,
        5892.52123654,  2964.42646805, 16984.90647405, 28616.60090621,
       10393.99281591, 38416.15549267, 12139.72620514,  1824.50875233,
        3135.80186945,  9975.92221665,  9433.91153017,  7503.13618036,
        7527.15178307,  3233.69886717,  9573.98845664, 11188.30953233,
       10215.98050366, 13241.2919176 ,  9970.93695567, 31930.81422667,
       30300.50527411, 11524.54888115,  7471.93208759, 28339.97432762,
       16622.83400972, 14994.93497148,  9195.53210353,  6020.93188011,
       16319.75185976, 11243.89475479,  1685.52901091,  4565.39252111,
        5548.69406955, 15496.09884877,  4821.55716243, 10765.44302618,
       13989.92964748, 37230.63028553, 25491.62934668,  4505.607283  ,
       11958.15693869, 11360.36158099, 11147.99432833, 27066.27189017,
        9734.1104876 , 15590.67935122,  5903.98415038, 11803.17203101,
       15255.93898488,  5938.44747358, 12282.81489623,  4325.27016334])
```

In [28]:

```python
#Support Vector Machine

from sklearn.svm import SVR

# creating the model
model = SVR()

# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
```

```
print("RMSE :", rmse)

# Calculating the r2 score
r2 = r2_score(y_test, y_pred)
print("r2 score :", r2)
```

```
MSE : 175634760.8944562
RMSE : 13252.726545675656
r2 score : -0.13861463280418374
```

In [29]:

```
#Random Forest

from sklearn.ensemble import RandomForestRegressor

# creating the model
model = RandomForestRegressor(n_estimators = 40, max_depth = 4, n_jobs = -1)

# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)

# Calculating the r2 score
r2 = r2_score(y_test, y_pred)
print("r2 score :", r2)
```

```
MSE : 23282926.865853906
RMSE : 4825.23852942566
r2 score : 0.8490601684486496
```

In [30]:

```
print(y_pred)
```

```
[45023.8249976   5639.67721012  6384.16753679 16871.65933407
 13028.5036422   5836.29716648  2551.33841792 10583.17884479
 45842.30371458 13884.64904764  7027.60150419 10529.80632979
 45369.70039144 13024.85448321  2551.33841792 24736.10876769
 10498.11468957 10294.73817716 17724.37010433 10573.49089461
  2505.86411698  6560.96787865 10406.79019751  9666.72650457
 25750.57493142 45592.57965972 13990.73313932  5781.70786169
  7261.46255252 13808.74403919  5847.33933626 38360.81721999
  4763.01071051 13567.3010879  14183.21630378  6560.96787865
  2742.57889882  7027.60150419  2505.86411698  6079.18113514
  6179.6715885  23924.40784553  4759.5571758  13211.36538649
  7197.20446162 39255.30714911  6179.6715885   2551.33841792
  5282.95898671  6375.53819724  8264.64012551 10445.91190019
 13925.09462838  2570.67545494  7334.51026652  9189.551811
 26503.63393383  5217.67915095 14183.21630378  6263.05825556
 13211.36538649  6718.31065152 10912.22503258  5557.47129699
  2505.86411698  5639.67721012 10692.81692405  5847.33933626
  6608.40410149 13211.36538649 14471.99020055  2677.76756087
  2961.8105536  10602.94195693 15616.21088009  3325.744939
 10157.47123256 10692.81692405 24315.78164489  4931.46061968
  2723.24186181 10293.81236002  6275.74363    18839.3202548
  2723.24186181 14678.67561072  6879.2414848  14678.67561072
 10445.0349841   7106.03531419 14394.10165927 13621.09215416
 36737.28421625  8220.36295948 26652.88211913  4915.68753843
 10615.9196883   6711.06516379  6722.84370049 45858.44006512
 45023.8249976  13200.46339318  7367.01372047  5803.72004579
 38523.85418439 17491.22707563  6793.30136319 45023.8249976
 25750.57493142  4716.64918416 13396.70903154  7282.80151318
  6236.86278904 18688.22316658  5039.25978079  6138.288779
 13211.36538649  6783.58085694  9936.14526458 10511.82855112
  2961.8105536   7766.90298964 25945.40353823 14048.38755868
 18839.3202548  14662.21926771 14584.33072643  9313.78622073
 25482.66705192 13401.59445365 13824.00443605  4915.68753843
 40623.579942   36464.19853444 13401.59445365  2570.67545494
 45692.20781011 18449.90486539 14295.74622458  6270.01579517
 13215.08355037 10660.07608054  5863.91377483 45592.57965972
  7005.91979204 13633.77536889  5675.10471918  6600.52810956
  9426.22786156 25133.70261607 13401.59445365 10330.16397071
  6372.29887344  9470.13454269  7100.64921818  6877.90195749
  2723.24186181 13401.59445365 13804.36742526  2723.24186181
 10610.81606687 37852.2824917  10384.66527525 10716.90613401
 10649.70140286 14768.02962255 41600.03644142  6722.84370049
  5717.43225794  4983.7937405   7213.26760507 13401.59445365
  7230.15635675  5639.67721012 38250.44883616 10256.49339057
```

```
  2505.86411698  5412.44651803 14584.33072643 24315.78164489
 10778.19367322 13401.59445365  2742.57889882  2742.57889882
 45592.57965972 39255.30714911 16871.65933407  6318.37609115
 38092.96630134 13066.21042425  2551.33841792  5558.76888791
 14584.33072643 13759.14813112  2723.24186181 13401.59445365
 15806.43994725 24181.54845896 37883.3934059   5813.91754985
  7164.90730909  2505.86411698  7100.64921818  6026.5359787
 38435.53379463  6481.81394692  4888.55262804 45592.57965972
 45616.41427775 42129.65362918  6270.01579517  6739.45614335
 45692.20781011 10330.16397071  5610.48131784 45692.20781011
  5039.25978079  2677.76756087 13621.09215416 18983.78150065
  6952.28919879 45592.57965972  9546.85719044  2570.67545494
  5263.49634549  8297.85279644  7027.60150419  4993.44258625
  7027.60150419  2723.24186181  7370.91591337 13462.58740549
 10716.90613401 13401.59445365 10635.41179149 36969.04285858
 39255.30714911 13401.59445365  6667.24362834 36837.26159522
 14488.44654356 15620.07423318 13166.47477502  2742.57889882
 15383.00789583 14662.21926771  2723.24186181  4636.01037091
  5039.25978079 13211.36538649  6306.67192811 10667.95019048
 10713.15578601 25945.40353823 20657.49595041  6059.96410782
 10634.73751955 13024.85448321 13812.75410249 21530.45804271
  7027.60150419 14584.33072643  5117.01482861 14490.51770446
 14277.56118807  6263.05825556 10307.7680395   6963.69270806]
```

In [31]:

```python
print(y_test)
```

```
338    41919.09700
620     3659.34600
965     4746.34400
128    32734.18630
329     9144.56500
          ...
580    12913.99240
786    12741.16745
321    24671.66334
903     8125.78450
613     6753.03800
Name: charges, Length: 268, dtype: float64
```

In [32]:

```python
#Decision Forest

from sklearn.tree import DecisionTreeRegressor
```

```
# creating the model
model = DecisionTreeRegressor()

# feeding the training data to the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# calculating the mean squared error
mse = np.mean((y_test - y_pred)**2, axis = None)
print("MSE :", mse)

# Calculating the root mean squared error
rmse = np.sqrt(mse)
print("RMSE :", rmse)

# Calculating the r2 score
r2 = r2_score(y_test, y_pred)
print("r2 score :", r2)
```

```
MSE : 44043931.24033023
RMSE : 6636.560196391669
r2 score : 0.7144695939399076
```

In [ ]: