Medical Costs Analysis Using Linear Regression

Linear Regration:

linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables

importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Reading the data

In [2]:
```
d=pd.read_csv(r'C:\Users\Anand Mishra\Downloads\insurance.csv')
d
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400

	age	sex	bmi	children	smoker	region	charges
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

describe()

This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

In [3]:
```
d.describe()
```

Out[3]:

	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

	age	bmi	children	charges
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

print all statistical information

In [4]:
```
d.describe(include='all')
```

Out[4]:

	age	sex	bmi	children	smoker	region	charges
count	1338.000000	1338	1338.000000	1338.000000	1338	1338	1338.000000
unique	NaN	2	NaN	NaN	2	4	NaN
top	NaN	male	NaN	NaN	no	southeast	NaN
freq	NaN	676	NaN	NaN	1064	364	NaN
mean	39.207025	NaN	30.663397	1.094918	NaN	NaN	13270.422265
std	14.049960	NaN	6.098187	1.205493	NaN	NaN	12110.011237
min	18.000000	NaN	15.960000	0.000000	NaN	NaN	1121.873900
25%	27.000000	NaN	26.296250	0.000000	NaN	NaN	4740.287150
50%	39.000000	NaN	30.400000	1.000000	NaN	NaN	9382.033000
75%	51.000000	NaN	34.693750	2.000000	NaN	NaN	16639.912515
max	64.000000	NaN	53.130000	5.000000	NaN	NaN	63770.428010

info()

print the information of the dataset

In [5]:
```
d.info()
```

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

To check size of the data set

In [6]:
```
d.size
```

Out[6]: 9366

To check shape of data set

In [7]:
```
d.shape
```

Out[7]: (1338, 7)

To check dimention of data set

In [8]:
```
d.ndim
```

Out[8]: 2

head()

To print top 5 rows

In [9]:
```
d.head()
```

Out[9]:

	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

tail()

To print bottom 5 rows

In [10]:
```
d.tail()
```

Out[10]:

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

To check is ther any missing value

In [11]:
```
d.isna()
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
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

index

To print the index

In [12]:
```
d.index
```

Out[12]:
```
RangeIndex(start=0, stop=1338, step=1)
```

Sum of null value

In [13]:
```
d.isnull().sum()
```

Out[13]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

To check the duplicated value

In [14]:
```
d.duplicated()
```

Out[14]:
```
0       False
1       False
2       False
3       False
4       False
        ...
1333    False
1334    False
1335    False
1336    False
1337    False
Length: 1338, dtype: bool
```

Count the region

In [15]:
```
d['region'].value_counts()
```

Out[15]:
```
southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

To view the record of male

In [16]:
```
dd=d[d['sex']=='male']
dd
```

Out[16]:

	age	sex	bmi	children	smoker	region	charges
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
8	37	male	29.830	2	no	northeast	6406.41070
...	...	...	...	...	...	...	...
1324	31	male	25.935	1	no	northwest	4239.89265
1325	61	male	33.535	0	no	northeast	13143.33665
1327	51	male	30.030	1	no	southeast	9377.90470
1329	52	male	38.600	2	no	southwest	10325.20600
1333	50	male	30.970	3	no	northwest	10600.54830

676 rows × 7 columns

To view the record of female

In [17]:
```
dd=d[d['sex']=='female']
dd
```

Out[17]:

	age	sex	bmi	children	smoker	region	charges

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.90	0	yes	southwest	16884.92400
5	31	female	25.74	0	no	southeast	3756.62160
6	46	female	33.44	1	no	southeast	8240.58960
7	37	female	27.74	3	no	northwest	7281.50560
9	60	female	25.84	0	no	northwest	28923.13692
...	...	...	...	...	...	...	...
1332	52	female	44.70	3	no	southwest	11411.68500
1334	18	female	31.92	0	no	northeast	2205.98080
1335	18	female	36.85	0	no	southeast	1629.83350
1336	21	female	25.80	0	no	southwest	2007.94500
1337	61	female	29.07	0	yes	northwest	29141.36030

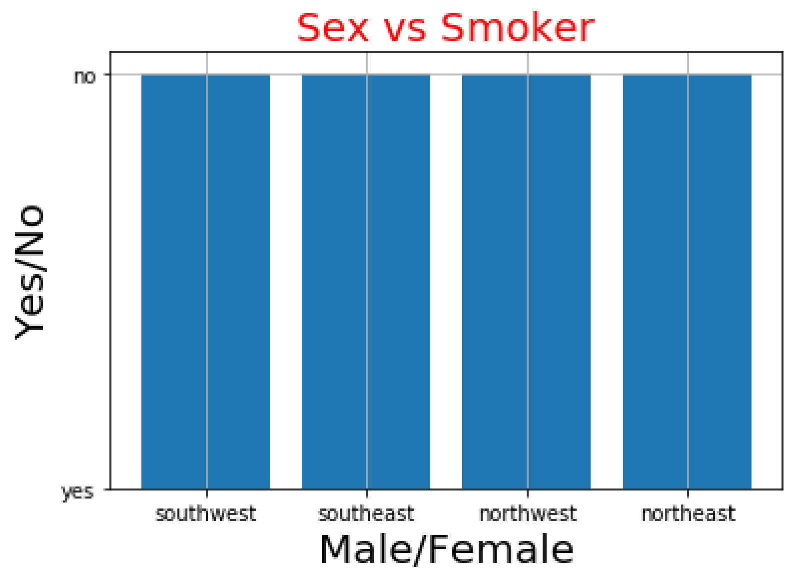
662 rows × 7 columns

Visualization of data set

Plot a graph of insurance dataset

In [18]:

```
plt.bar(dd['region'],dd['smoker'])
plt.title('Sex vs Smoker',fontsize=20,color='red')
plt.xlabel('Male/Female',fontsize=20)
plt.ylabel('Yes/No',fontsize=20)
plt.grid(True)
plt.show()
```



Scatter plot:

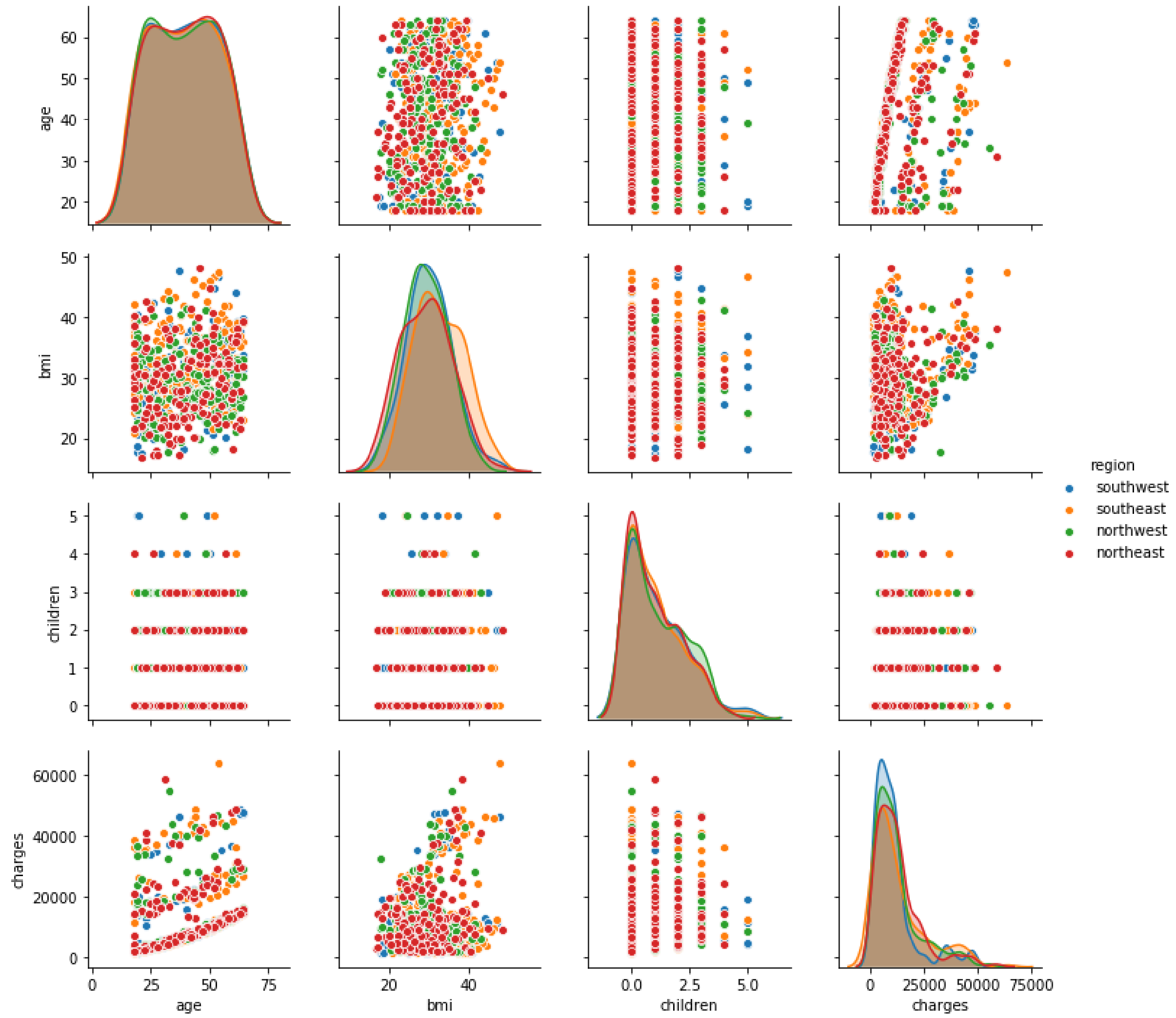
Plot a graph. which shows the relation between age and children

In [19]:
```
plt.scatter(x='age',y='children',label='Age of children',color='r',s=50
0,alpha=0.4,data=dd)
plt.xlabel('Age of children')
plt.ylabel('children')
plt.title('scatter plot')
plt.legend()
plt.show()
```

pairpolt()

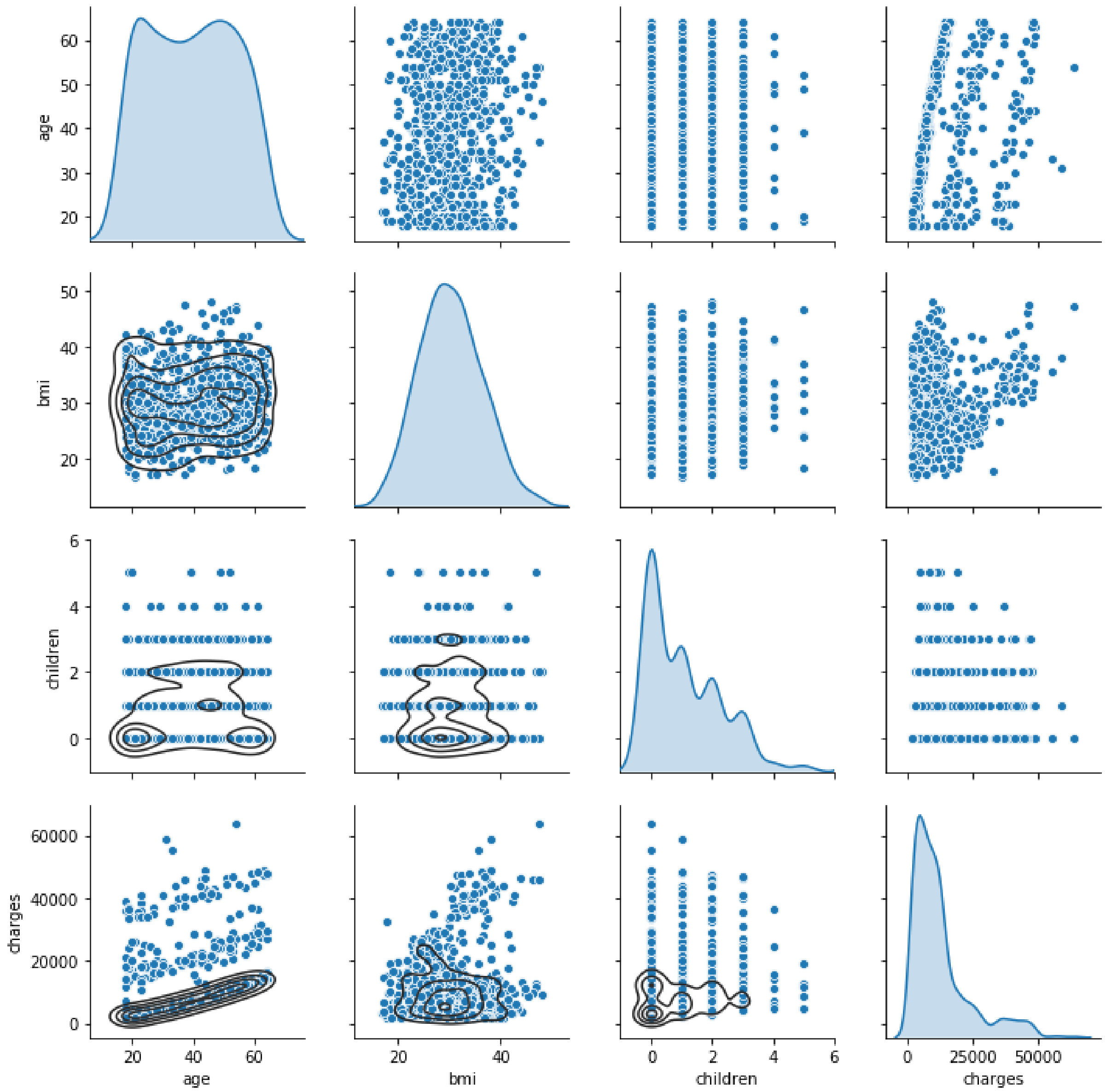
A pairplot plot a pairwise relationships in a dataset. The pairplot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column.

In [20]:
```
sns.pairplot(dd,diag_kind='kde',hue='region')
plt.show()
```



In [21]:
```
g = sns.pairplot(dd, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")
```

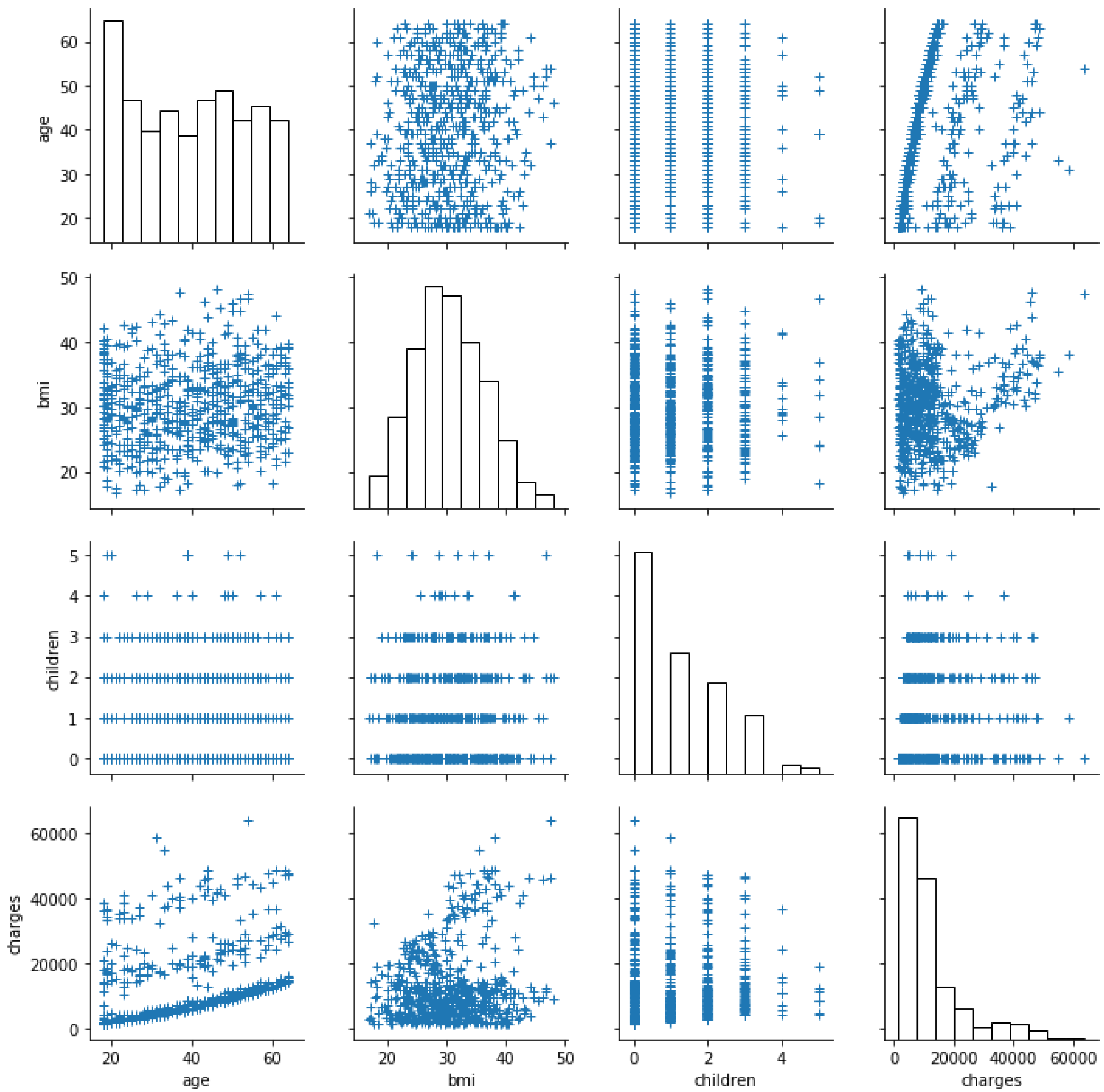
Out[21]: <seaborn.axisgrid.PairGrid at 0x1cfc223e788>



In [22]:
```
sns.pairplot(
    dd,
    plot_kws=dict(marker="+", linewidth=1),
```

```
    diag_kws=dict(fill=False),
)
```

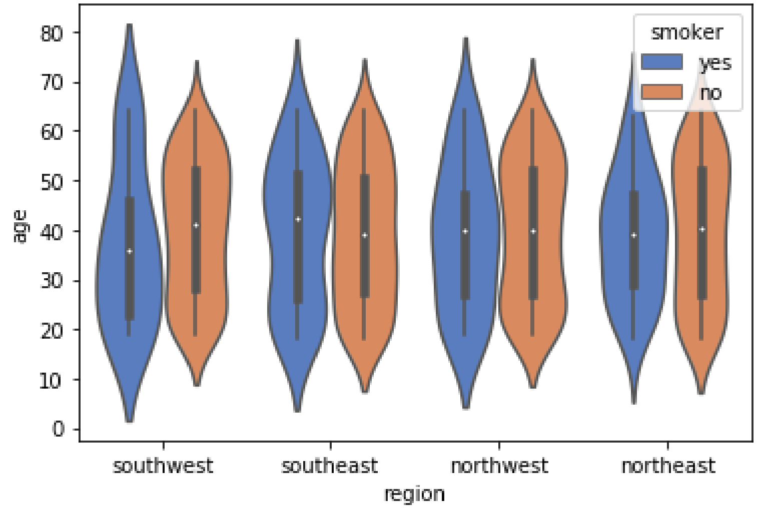
Out[22]: <seaborn.axisgrid.PairGrid at 0x1cfc3772bc8>



violinplot()

It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.

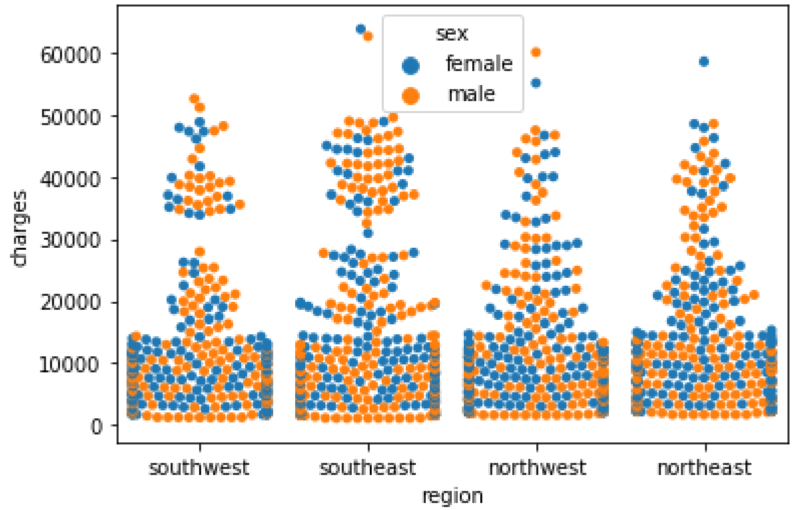
In [23]:
```
ax = sns.violinplot(x="region", y="age", hue="smoker",
                    data=dd, palette="muted")
```



swarmplot()

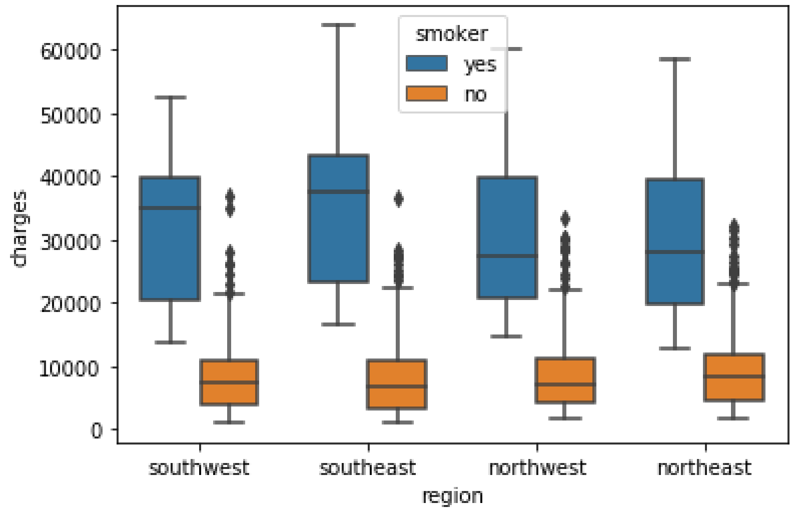
Swarm Plots, also called beeswarm plots, are similar to strip plots in that they plot all of the data points. Unlike strip plots, swarm plots attempt to avoid obscuring points by calculating non-overlapping positions instead of adding random jitter.2

In [24]:
```
sns.swarmplot(data=d,x='region',y='charges',hue='sex')
plt.show()
```



To plot box plot

In [25]:
```
sns.boxplot(x='region',y='charges',hue='smoker',width=0.7,data=d)
plt.show()
```



corr()

Is used to find the pairwise correlation of all columns in the dataframe. Any na values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

In [26]:
```
d.corr()
```

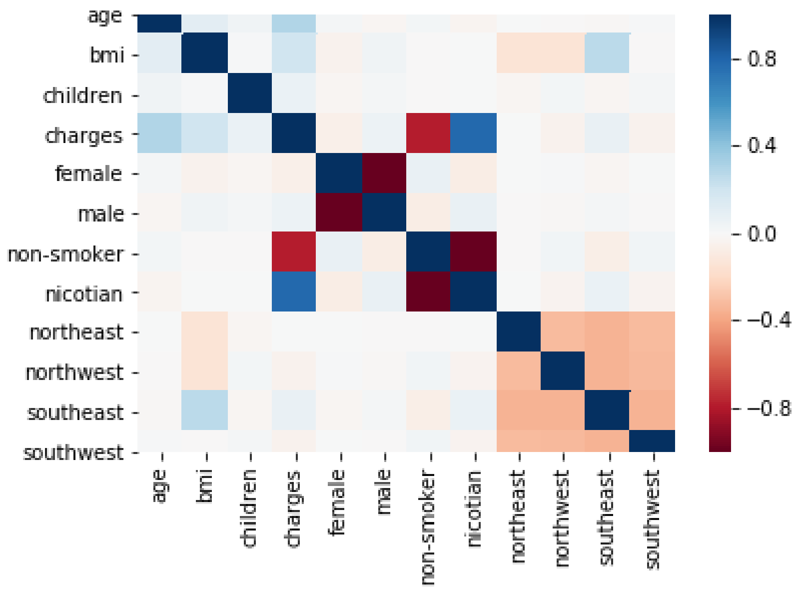
Out[26]:

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

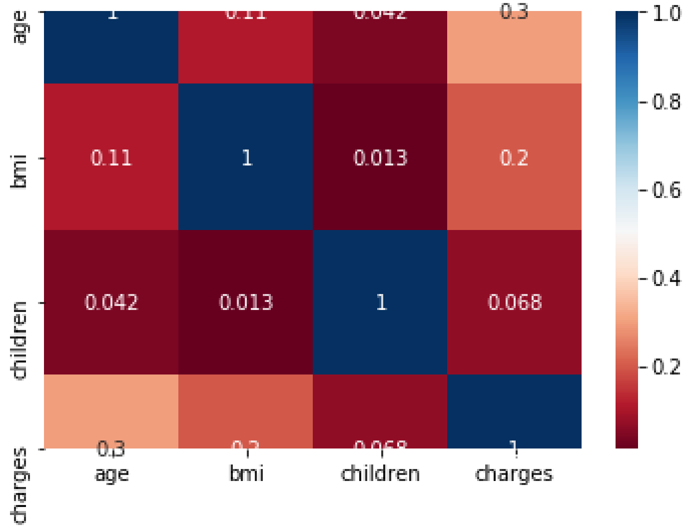
heatmap()

A heatmap is a graphical representation of data that uses a system of color-coding to represent different values.

In [49]:
```
sns.heatmap(d.corr(),cmap='RdBu',vmin=-1,vmax=1)
plt.show()
```

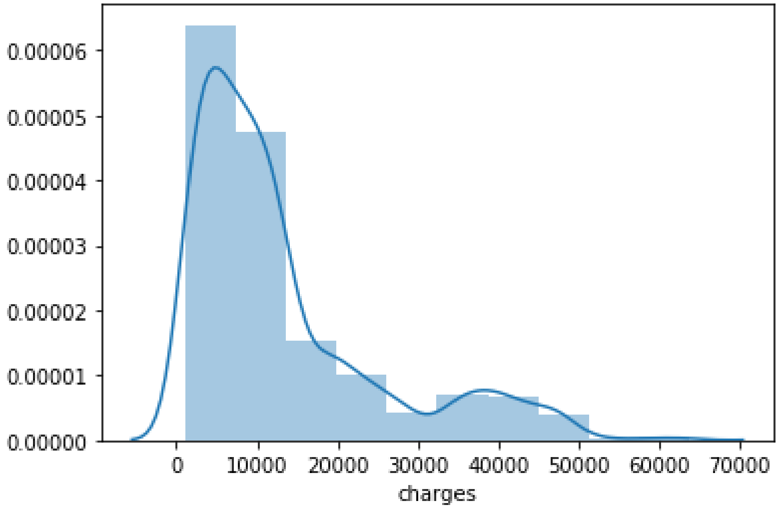


In [28]:
```
sns.heatmap(d.corr(),cmap='RdBu',annot=True)
plt.show()
```

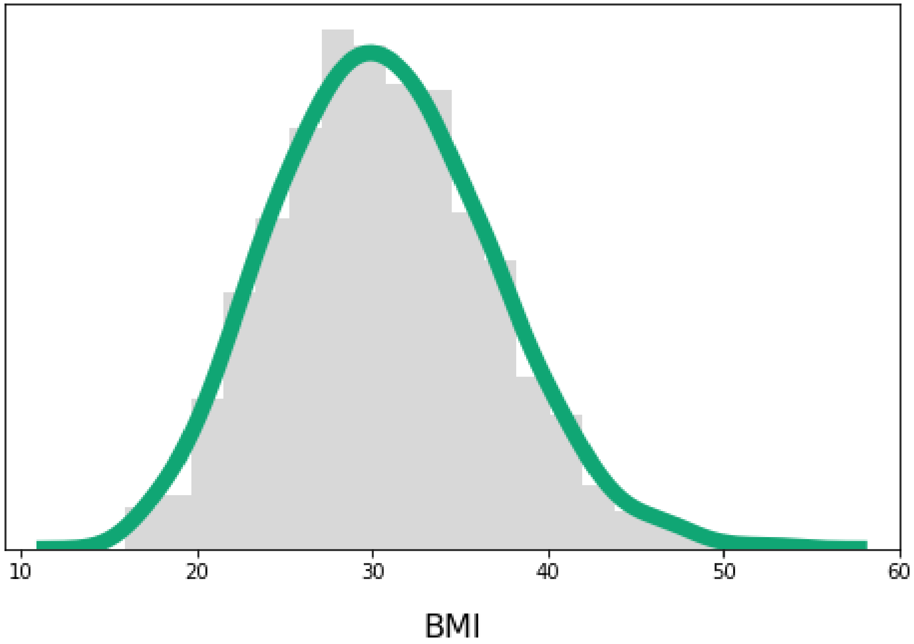


bins = 10 plots in equal distribution

In [30]:
```
sns.distplot(d.charges,bins=10)
plt.show()
```



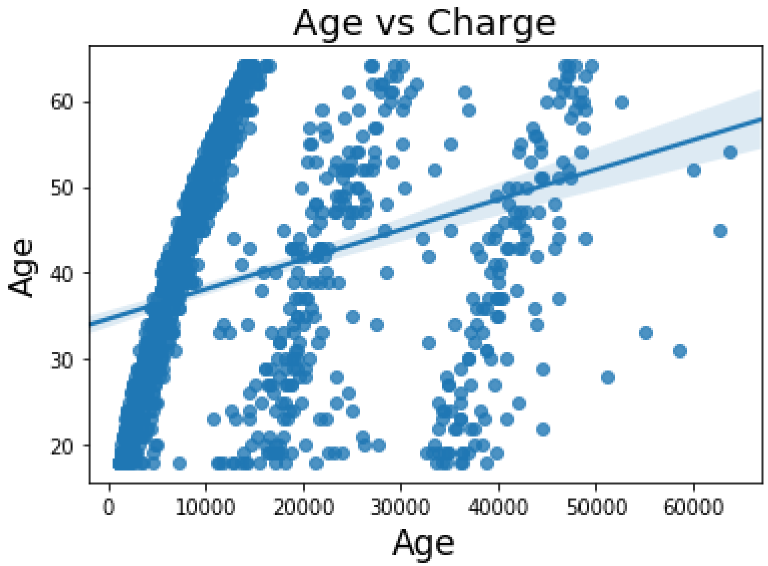
In [31]:
```
plt.figure(figsize=(8,5))
sns.distplot(d.bmi,
             bins=20,
             kde_kws={"lw":8,'color':'xkcd:bluish green'},
             hist_kws={'alpha':0.3,'color':'gray'}
)
plt.xlabel('BMI',fontsize=16,labelpad=15)
plt.yticks([])
plt.show()
```



regplot():

This method is used to plot data and a linear regression model fit. ... If strings

In [32]:
```
sns.regplot(y=d['age'],x=d['charges'])
plt.title('Age vs Charge',size=18)
plt.ylabel('Age',size=16)
plt.xlabel('Age',size=17)
plt.show()
```



import norm from scipy.stats

import StandardScaler from sklearn.preprocessing

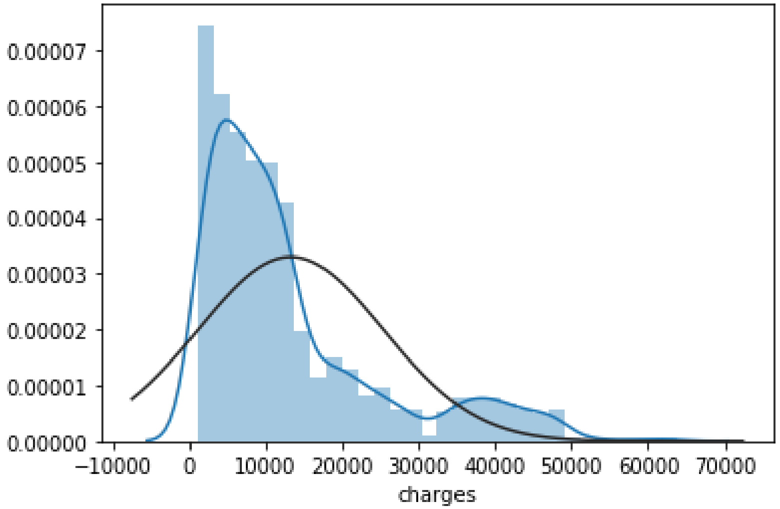
In [34]:
```
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
```

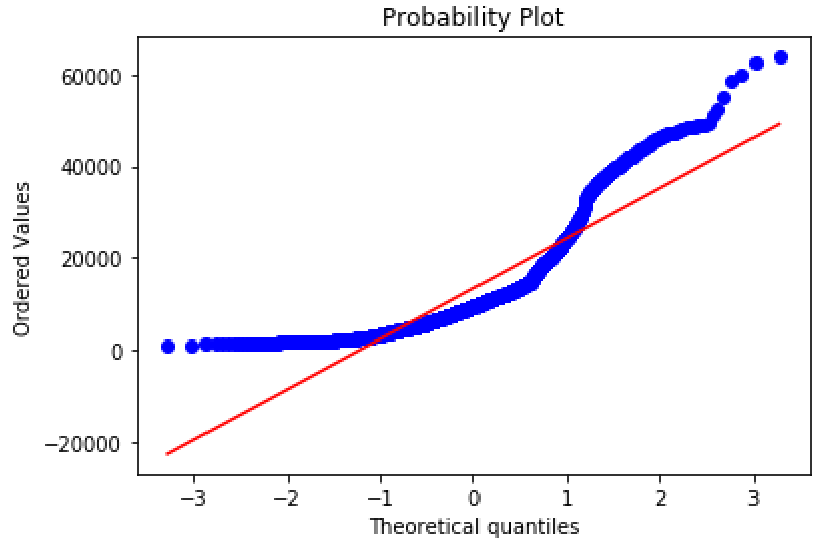
distplot()

The variable of interest is "charges". We want to predict what would be medical costs for specific individual, based on other given information. Let's see distribution of data for "charges".

From the figure bellow we can conclude that the variable "charges" do not possess normal distribution of data, but it has mixture distribution. That could be a problem for further assumptions.

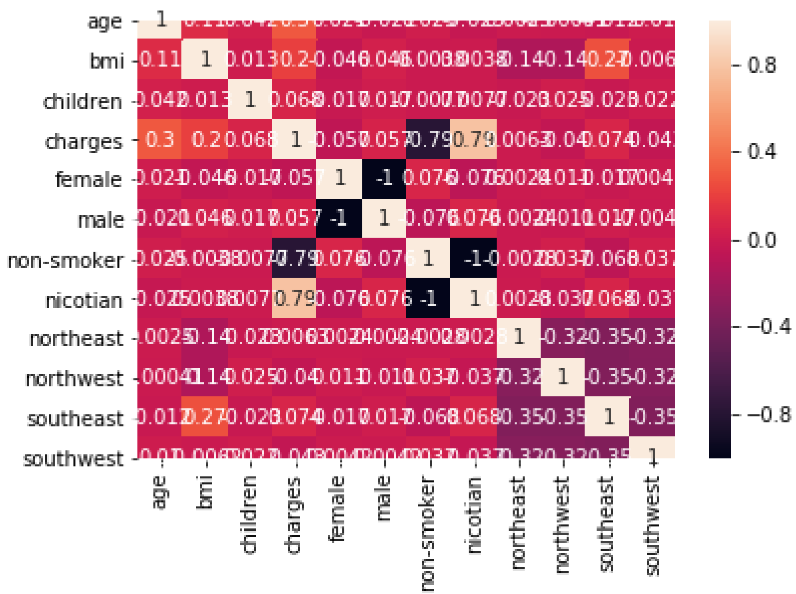
In [35]:
```
sns.distplot(d["charges"],fit=norm)
fig = plt.figure()
res = stats.probplot(d["charges"], plot=plt)
```





In [54]:
```
sns.heatmap(d.corr(),annot=True)
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x1cfc5ca4ec8>



To check column name

In [36]:
```
d.columns
```
Out[36]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'],
dtype='object')
```

rename()

We have three non numerical categories: sex, smoker and region. We want to use them too. So the next step is to convert these variables to binary values by using "One hot encoding" method.

In [37]:
```
sex_dummy = pd.get_dummies(d['sex'])
smoker_dummy = pd.get_dummies(d['smoker'])
region_dummy = pd.get_dummies(d['region'])

d = pd.concat([d,sex_dummy,smoker_dummy,region_dummy], axis=1)

d.rename(columns={'no': 'non-smoker', 'yes': 'nicotian'}, inplace=True)
```

We can see last 8 columns at dataframe below (which represent converted categories)

In [62]:
```
d.head(10)
```
Out[62]:

	age	sex	bmi	children	smoker	region	charges	female	male	non-smoker	...	sou
0	19	female	27.900	0	yes	southwest	16884.92400	1	0	0	...	
1	18	male	33.770	1	no	southeast	1725.55230	0	1	1	...	
2	28	male	33.000	3	no	southeast	4449.46200	0	1	1	...	

	age	sex	bmi	children	smoker	region	charges	female	male	non-smoker	...	sou
3	33	male	22.705	0	no	northwest	21984.47061	0	1	1	...	
4	32	male	28.880	0	no	northwest	3866.85520	0	1	1	...	
5	31	female	25.740	0	no	southeast	3756.62160	1	0	1	...	
6	46	female	33.440	1	no	southeast	8240.58960	1	0	1	...	
7	37	female	27.740	3	no	northwest	7281.50560	1	0	1	...	
8	37	male	29.830	2	no	northeast	6406.41070	0	1	1	...	
9	60	female	25.840	0	no	northwest	28923.13692	1	0	1	...	

10 rows × 23 columns

drop()

To delete selected rows

In [38]:
```
d = d.drop(['sex','smoker','region'], axis=1)
```

We have prepared our data for further processing. Finally, we can import, initialize and use the Linear Regression model.

In [39]:
```
d.head(10)
```

Out[39]:

	age	bmi	children	charges	female	male	non-smoker	nicotian	northeast	northwest	sou
0	19	27.900	0	16884.92400	1	0	0	1	0	0	

	age	bmi	children	charges	female	male	non-smoker	nicotian	northeast	northwest	sou
1	18	33.770	1	1725.55230	0	1	1	0	0	0	
2	28	33.000	3	4449.46200	0	1	1	0	0	0	
3	33	22.705	0	21984.47061	0	1	1	0	0	1	
4	32	28.880	0	3866.85520	0	1	1	0	0	1	
5	31	25.740	0	3756.62160	1	0	1	0	0	0	
6	46	33.440	1	8240.58960	1	0	1	0	0	0	
7	37	27.740	3	7281.50560	1	0	1	0	0	1	
8	37	29.830	2	6406.41070	0	1	1	0	1	0	
9	60	25.840	0	28923.13692	1	0	1	0	0	1	

In [40]:
```
from sklearn.model_selection import train_test_split
```

Eleven dataframe categories will be used as inputs X for the model. And we want to fit our model according to "charges" category - output Y of the model

In [41]:
```
X = d[['age', 'female','male','non-smoker','nicotian','northeast','nort
hwest','southeast','southwest','bmi', 'children']]
y = d['charges']
```

We will use train_test_split function to divide our data to training and testing data.

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4
```

```
)
```

In [60]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(802, 11)
(536, 11)
(802,)
(536,)
```

In [63]:
```
print(np.mean(X_train))
print(np.mean(X_test))
print(np.mean(y_train))
print(np.mean(y_test))
```

```
age             39.430175
female           0.478803
male             0.521197
non-smoker       0.801746
nicotian         0.198254
northeast        0.243142
northwest        0.253117
southeast        0.265586
southwest        0.238155
bmi             30.572151
children         1.129676
dtype: float64
age             38.873134
female           0.518657
male             0.481343
non-smoker       0.785448
nicotian         0.214552
northeast        0.240672
northwest        0.227612
southeast        0.281716
southwest        0.250000
bmi             30.799925
```

```
children       1.042910
dtype: float64
13126.857807518705
13485.233263300363
```

Procedure for importing and fitting the model.

In [43]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Out[43]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normaliz
e=False)
```

We will create a new dataframe to present estimated coeffcieints of our model. First one is the intercept, and other coefficients are in correlation with specific categories.

In [44]:
```
print(lm.intercept_)
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
print(coeff_df)
```

```
-293.7460154216242
             Coefficient
age           243.533353
female        -87.429238
male           87.429238
non-smoker -11999.973124
nicotian    11999.973124
northeast     445.648198
northwest     350.264132
southeast    -521.551674
```

```
southwest    -274.360656
bmi           336.968721
children      673.820362
```

Final part is to use fitted model for predicting new values (based on prepared X_test array)

In [45]:
```
predictions = lm.predict(X_test)
print("Predicted medical costs values:", predictions)
```

```
Predicted medical costs values: [ 5513.00189848 27670.14341082 38885.07
19728  30468.90964232
  2166.47224579  2728.95854267  1943.04528313  2044.86969931
  8637.65226843 11410.57627765 11888.27978298 13696.57646576
 11288.38866417  9654.29004717  7266.64849971 12083.99723272
  5848.68948665 13018.864278    9961.03718989 24394.56778051
 11581.08045672  6716.97873298 10247.37797573  6081.96707085
  9580.07862572  5668.69817738  5353.51101931 16502.90816923
  8007.14402134  9633.06195022 12341.51464651 25106.09244355
 11158.87119012 16247.21849742 12614.07690612  6002.94910148
  6573.84710034   183.10020814 11242.28595197  8981.27743195
 12491.05349926 12905.15113901 34637.74323436 13395.05956506
   637.50286583 33437.67194049 14804.19941229  7893.4675565
  9113.95922086  7426.70864218  3642.62579018  9791.12407567
 28631.04461934 14625.24207227  8768.53645814 16696.59986447
  6311.96424229  1945.32953919   821.77268874  8000.41867329
 15378.34419046 11702.54341083  9458.5461902   8339.05345364
 11183.45750113  6908.72009742   389.97071226 11178.53869914
 35560.29838351 39576.34881769 15404.79998259  6193.15957287
  7557.49198954  4940.02396665 12349.08129699 15020.58110363
 11109.23745189 11486.54798743  3257.59183561  4719.95600458
  6809.82648111 15221.04747229 28359.23981619 10930.95351157
 28193.0095626  29223.73326318  4782.03677787 11198.76101776
 12857.28199908  2772.16526059 27314.92240969 12217.46933217
  2341.07912542 15063.17716026  3903.3276467   7372.27043825
 10784.46785087  1109.73404173  3790.13201059  6825.67598929
 10733.04646844 11065.88790343   145.41158222  4634.09860368
```

```
30173.48692001 11429.61021998 29091.98870627  5773.35251571
  207.40284397  2634.81150781 -1513.51999442 10179.55918967
10404.00998267  6995.78463193 39495.38926522  2380.88226588
 9369.06249142 14490.44741433 10523.73006101 35484.06235869
14857.5267938   6326.36329709 29939.24668374 12027.50691149
 6835.27351851  4712.34055589  9041.24696262  8158.11192633
35973.52414435 10705.84353472  5781.94795821  8659.3850537
33943.88147603  1970.73658069  6988.00902292  2521.56172698
31139.01474999 38681.01038519  8761.60258827   763.1444263
 9570.59880661 36352.87957147 34259.73301988  7142.4611686
 3642.97702947 30477.39917969 12891.95338968 33518.94964827
15392.06292697  4898.99476616 10909.25498242 30377.86737399
11786.46061269 33758.85930605   272.2971471   3123.08789357
 1907.00633063 39668.35869022  7150.91521032  4494.95429881
12727.05657069  8055.94634246 33877.74438108  5613.12816211
 4200.51335926  5180.35901913 14380.86295705   488.0742083
14851.0582467  10346.8249434   7681.88912542  1204.33346314
31970.85937481  4190.87811301 13327.06673386  5858.21676257
12895.38970356  5635.72451915  1875.64455966 11305.12452071
12234.03811648 35886.26481713 29936.27775147 15970.61454101
 8185.87437774 11383.94691085 11091.66567316  4739.49775396
39280.61513698  5030.33750186 11860.87010176  4036.82537449
 8904.33241627  5795.72051397  6852.1174543   3237.33343145
35809.46120223  5520.53321621 10643.2033385  14340.82700484
34879.90954589 11742.04080421  8055.34364234  4699.81468019
12110.55984405  2390.33226129 13554.92433132 37183.87089861
10673.26817294  2886.92624251  9399.54355267 12176.44560062
 6697.42829735 13572.28594607 16308.28376026  3087.93479426
 9384.15626427 27426.93503656 26037.76070064 25480.46296281
10395.73133386  3134.44806769  -878.26433239 14522.02577317
12611.39910512  8607.17937914  3747.36588061 16197.81859327
10198.48768749 29980.42207664 10328.74397609 27096.01934054
26363.3095323  13784.09423307 29012.02625863 32111.76429983
12278.47194495  7534.93713237 33426.46995272  3320.40875898
 5391.47499973 11622.59007262 11132.14341955 12623.09008577
 9105.56227969  7237.72261206  6779.69182707  6858.3400369
15724.52444373  7204.60785113  5819.33097223 25589.20882706
12839.11234595 13324.59561253   899.27569121 31190.48176618
12549.83584107 10454.86624285 35149.47271771 28640.23944021
 8759.67674105 31856.75813209  6773.89280976 10782.89858306
```

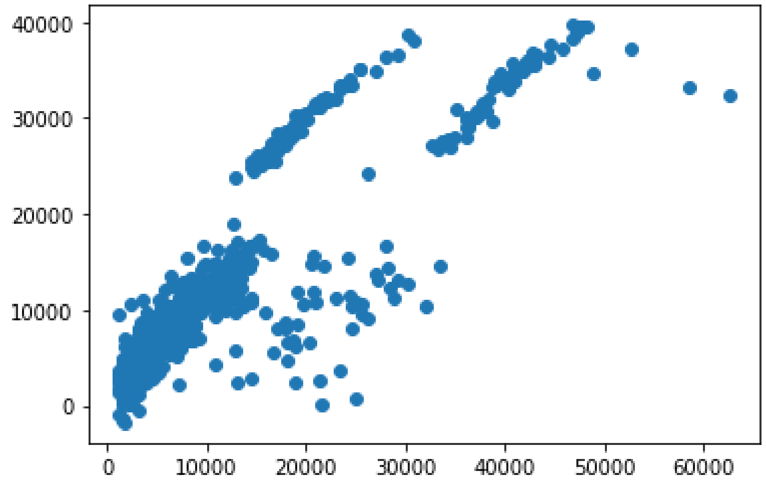
9265.72369823	9033.53206727	12407.39310472	9829.54793633
3163.01197734	18883.42380397	24196.68941631	5811.91438406
8807.86247086	14975.0205067	16903.6141874	35475.36268428
4452.61085413	6901.06484946	1959.76978896	10344.00630413
7246.09969873	10335.99981688	4803.65619838	34617.53032241
10322.23145648	804.78452514	11920.59742337	8450.8455033
5565.08514677	7374.14781271	9558.90691826	1233.16853192
4096.01162372	8565.61025103	2489.62813013	10521.64939996
27320.46162339	8467.59551502	4804.49199175	8988.8329078
2595.91320144	10785.18251367	4432.48233077	8781.78104745
7705.64669015	3359.72329217	10869.57160375	14261.10454486
9286.97314201	6994.38313804	26516.18946116	5539.45769061
6187.34768032	35165.48555702	-481.35036765	35643.77693382
2603.45149843	26881.41424913	1202.18228082	7084.28611202
6242.09846906	11241.51832968	27527.09575296	33448.18921316
16560.91586903	11318.36206929	33303.26981233	36333.75241613
5288.91071975	13426.08889672	8167.38806688	11241.95206079
11249.49035779	31641.13843155	10117.03874887	35140.00172303
12582.45845665	33044.20944373	10853.80891794	6571.75137642
11016.37135322	9672.91058289	35239.47894809	29714.40531919
13485.68153239	3845.1270906	7266.81746871	7157.9968708
31955.16097182	11635.9061143	8772.48793106	8125.40670431
16264.78893474	11391.48520785	10250.23119657	9460.63165994
10746.78939715	15344.59349911	7128.87936636	3752.333352
8022.22036409	31358.96029855	34864.61605275	36567.93430595
6202.44953678	4638.87233162	15062.65502194	8348.82320977
10989.04859625	29425.188028	5050.1382198	2390.21518152
10767.24865284	6979.20793726	26711.99670744	15343.24910577
29267.51876414	8853.57160402	38056.3459272	3436.21642921
35028.49223784	8477.79048494	32987.64184199	34913.07025967
26489.67984977	5270.0249882	625.96576791	30646.19928384
5522.11665494	5822.41402843	2167.56484648	8568.25658919
13776.63138079	1249.21797756	4027.84336631	27906.63775399
10867.77685629	4183.80465077	11186.02396214	9818.49304229
10772.63131453	1152.48437438	4100.48417218	10177.61766755
7079.99235941	5058.42821296	4019.10340464	3709.01284198
30669.49000609	11096.42535417	13400.70470602	5133.48972114
38274.73950924	7945.89740865	7842.07097673	12511.85923637
36743.18101411	7834.3113276	8078.06603238	11766.20220216
2013.29255307	3197.78577524	51.61095458	14628.75135822

```
 25215.23339079 13203.49792408 26012.6225628  35549.82406259
 11912.99497873 33793.43524872 33495.26190951  9386.90923136
  3722.19869797  1811.99602253 25538.13112003  5001.09639903
 12874.96368167  8498.55594115 14153.51897423 10968.96485404
 27109.91360514 11008.13431955 10823.38659001 31160.83079261
  3637.11652898 10226.21280747 34027.11902271 27678.51482554
 10290.27131722  6021.20025038 12778.02423868 11224.18070411
  6998.77879928 39368.77227218  5621.59388    30913.20428126
 14737.05664727  8000.08356314  8562.32647212 12496.61435832
 14501.17854257  6626.94019351  2124.85450974 31921.15327651
 25372.99823138 17348.03183612 15693.85947941  1115.59454223
  6916.12181935 15209.9126269  17026.6492026  10071.12128752
  2275.79291649  7816.34567848  4718.20688451  7761.2691769
 11043.35169533  1446.8198425   4659.9060683  11244.17184375
 14759.89996487  5119.83887421 36638.89046486 27898.14391301
  9052.30635055 32029.15744754 37137.13527063  2872.1106017
  4201.15377004  3708.78422559  8955.84207071  3994.09936286
 39495.90451263 -1760.75712211  7882.73642826  3416.31508611
 28866.67617728  6015.20236616 16727.80847013 13646.08261314
 14445.68580781 -1338.67882646  2573.02290867 28135.8819557
 32461.47591095 16723.14319417  4196.91534066 30272.29601736
  5443.58820053  9083.29794815  8045.93229505 10775.00383546
 37612.71426484  6229.8224214  14534.33062857 12227.43866625
  7964.32078445 11837.50507955 23796.73480294  9980.06293398
 24823.84468759 15652.73269446  7240.91811896  7513.64322807
 36620.27723385  7759.34332968  5106.61230362 12322.31754276
 14112.70720786 31458.12493244  9457.1565013   6037.0224075
  8111.6982962   9205.44576237 33124.62378991  4009.32589036]
```

Graphical comparison of expected values (y_test) and predicted values (predictions)

In [56]:
```
plt.scatter(y_test, predictions)
```

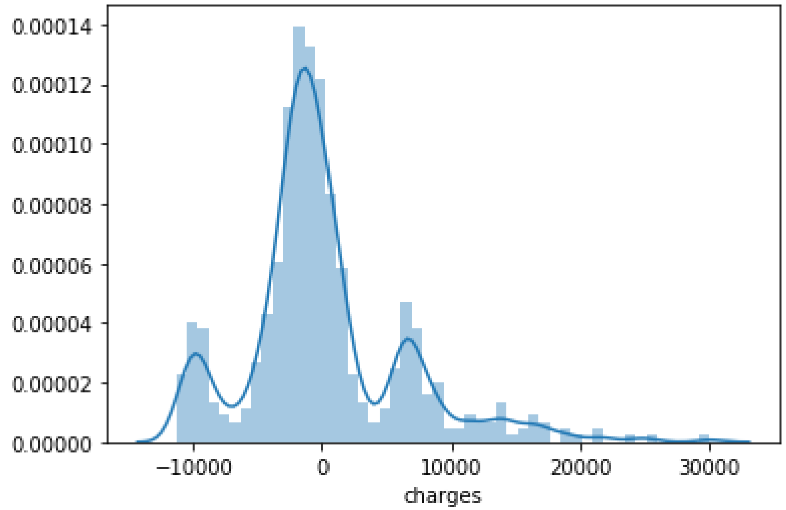
Out[56]:
```
<matplotlib.collections.PathCollection at 0x1cfc4b0c9c8>
```



Also, let's see error distribution graph of our predictions. Very close to normally distributed data.

In [57]:
```
sns.distplot((y_test-predictions), bins=50)
```

Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cfc3837748>
```



Finally, let's print MAE and MSE erorrs for entire test data.

In [58]:
```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test, predictions))
print(metrics.mean_squared_error(y_test, predictions))
```

```
4344.908424140114
38753945.045067705
```

In [ ]:

CONCLUSION

DATA SET: insurance

Medical Costs analysis using Simple Linear Regression. It shows the cleare picture of whole data set of insurance so that anyone can understand the output. with the help of sex row we can see the gender(male/female) who smoke age define the catogary of age people who smoke.

Graph represent data set in nice look which is understandable for anyone. with several graph we can plot the rows and colunms. with the help of linear regression we can predict the relation between two variable in dataset and it is allows you to estimate how a dependent variable changes as the independent variable(s) change. Simple linear regression is used to estimate the relationship between two quantitative variables.

In [ ]:

```

```