In [64]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/medical-cost-data/Medical cost data.csv
```

In [65]:

```
# Import library
#Data manipulation
import pandas  as pd

#Data manipulation
import numpy as np

# Visualization
import matplotlib.pyplot as plt

# Visualization
import seaborn as sns
plt.rcParams['figure.figsize'] = [8,5]
plt.rcParams['font.size'] =14
plt.rcParams['font.weight']= 'bold'
plt.style.use('seaborn-whitegrid')
```

In [63]:

```
# Import dataset
#path for the dataset
path = '../input/'
df = pd.read_csv("/kaggle/input/medical-cost-data/Medical cost data.csv")
print('\nNumber of rows and columns in the data set: ',df.shape)
print('')

#Lets look into top few rows and columns in the dataset
df.head()
```

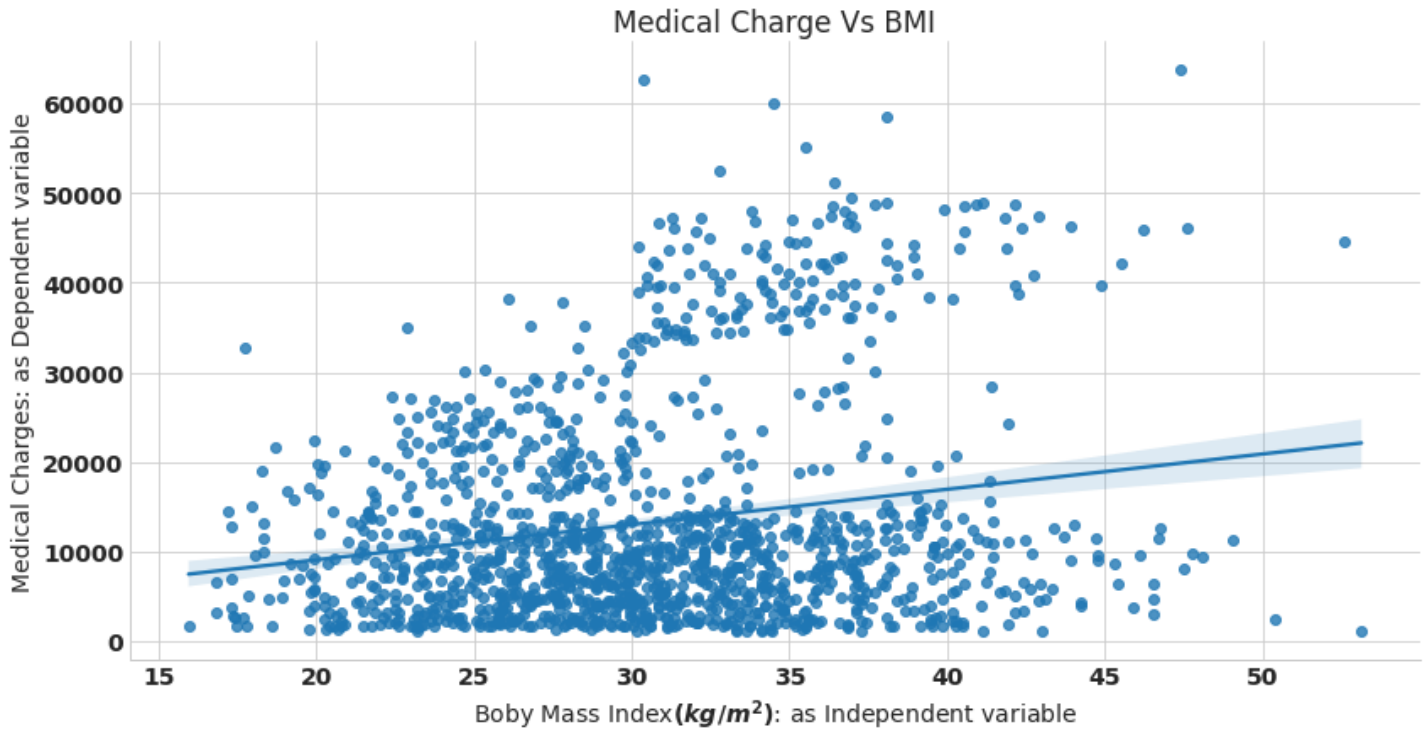
```
Number of rows and columns in the data set:  (1338, 7)
```

Out[63]:

	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

In [39]:

```
""" for our visualization purpose will fit line using seaborn library only for bmi as ind
ependent variable
and charges as dependent variable"""

#fitting the labels to plot the graph
#visualising the dataset on a graph
sns.lmplot(x='bmi',y='charges',data=df,aspect=2,height=6)
plt.xlabel('Boby Mass Index$(kg/m^2)$: as Independent variable')
plt.ylabel('Medical Charges: as Dependent variable')
plt.title('Medical Charge Vs BMI');
```



In [40]:

```
#Summary of the dataset
# Summary statistics
df.describe()
```

Out[40]:

	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 [41]:

```
#Cleaning the dataset
#Checking columns which have missing values
df.isnull().sum()[df.isnull().sum()>0]
```

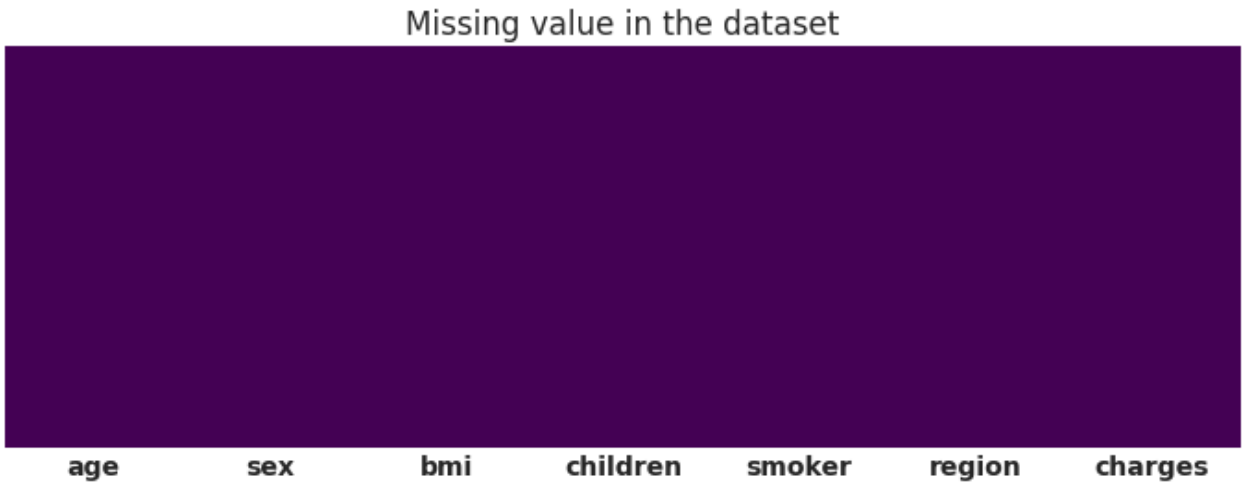
Out[41]:

```
Series([], dtype: int64)
```

In [42]:

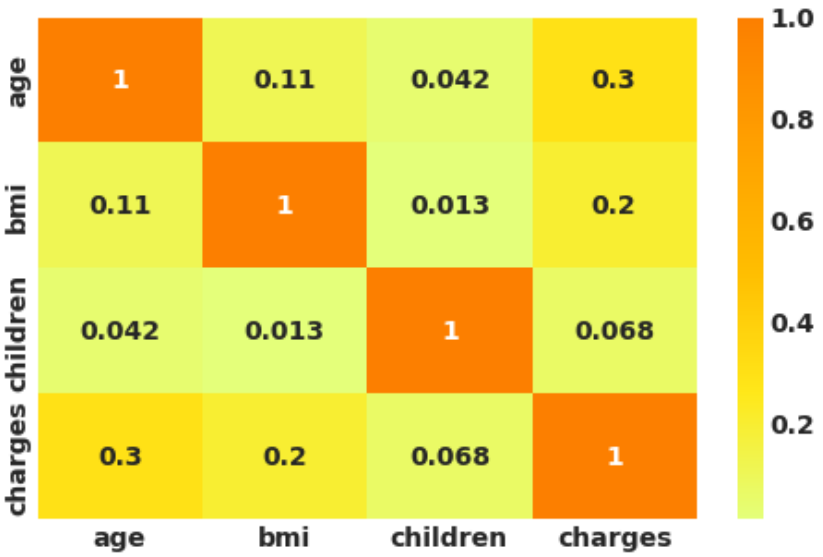
```
#Cleaning the dataset by checking missing values
```

```
plt.figure(figsize=(12,4))
#plotting the values for the missing values
sns.heatmap(df.isnull(),cbar=False,cmap='viridis',yticklabels=False)
plt.title('Missing value in the dataset');
```



In [43]:

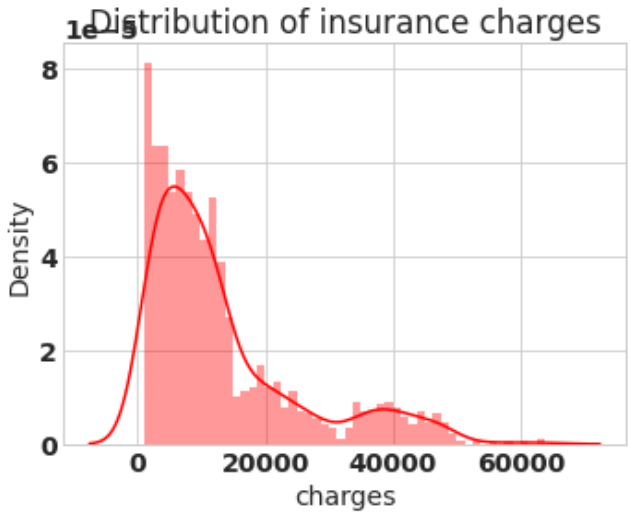
```
# Checking correlation for the dataset
corr = df.corr()
#correlation plot
sns.heatmap(corr, cmap = 'Wistia', annot= True);
```

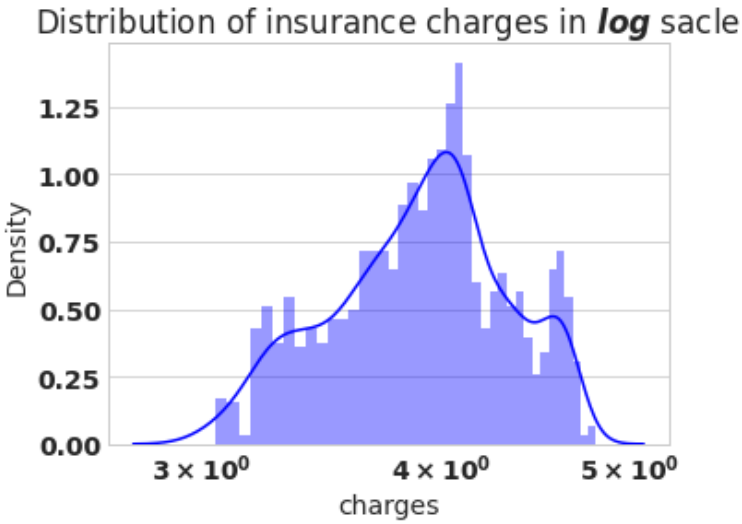


In [44]:

```
f= plt.figure(figsize=(12,4))

ax=f.add_subplot(121)
sns.distplot(df['charges'],bins=50,color='r',ax=ax)
ax.set_title('Distribution of insurance charges')

ax=f.add_subplot(122)
sns.distplot(np.log10(df['charges']),bins=40,color='b',ax=ax)
ax.set_title('Distribution of insurance charges in $log$ sacle')
ax.set_xscale('log');
```

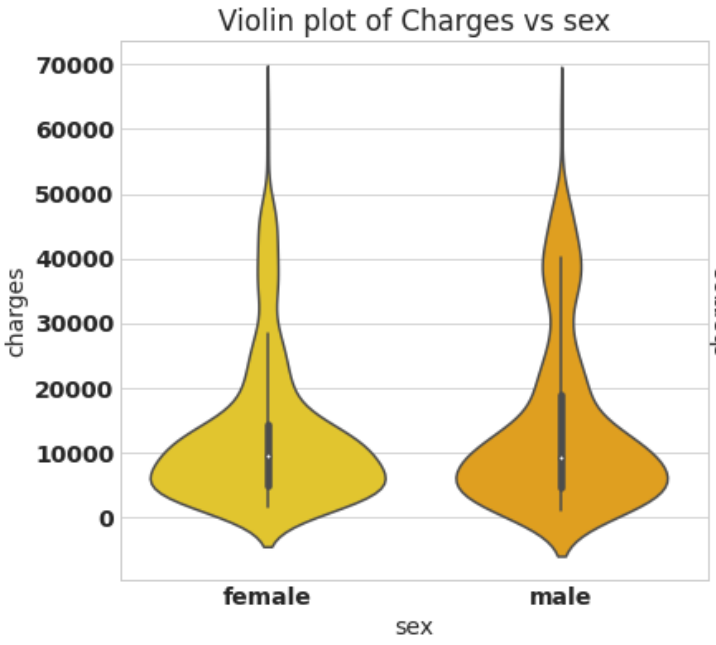
```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning: `dis
tplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning: `dis
tplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





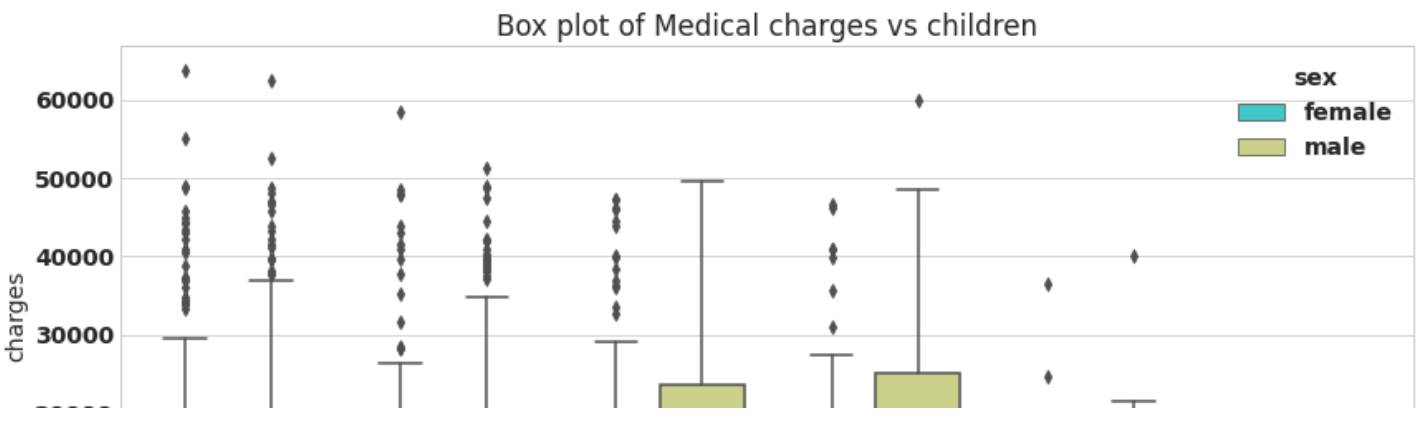
In [45]:

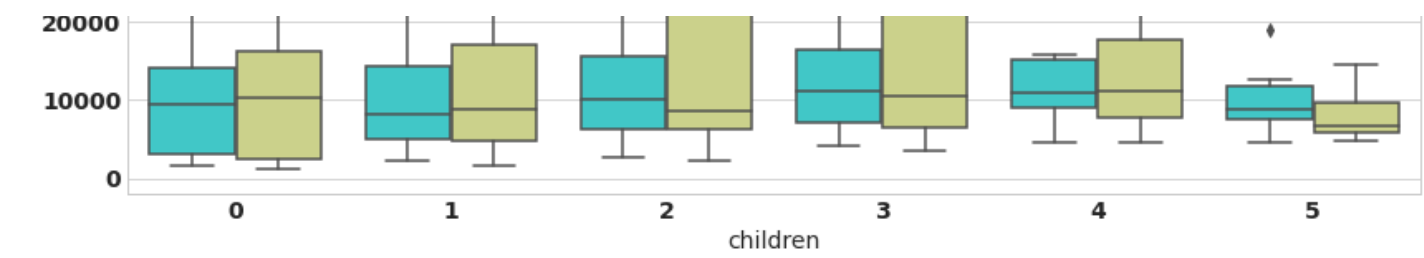
```
f = plt.figure(figsize=(14,6))
ax = f.add_subplot(121)
sns.violinplot(x='sex', y='charges',data=df,palette='Wistia',ax=ax)
ax.set_title('Violin plot of Charges vs sex')

ax = f.add_subplot(122)
sns.violinplot(x='smoker', y='charges',data=df,palette='magma',ax=ax)
ax.set_title('Violin plot of Charges vs smoker');
```



In [46]:

```
#visualization
plt.figure(figsize=(14,6))
sns.boxplot(x='children', y='charges',hue='sex',data=df,palette='rainbow')
plt.title('Box plot of Medical charges vs children');
```





In [47]:

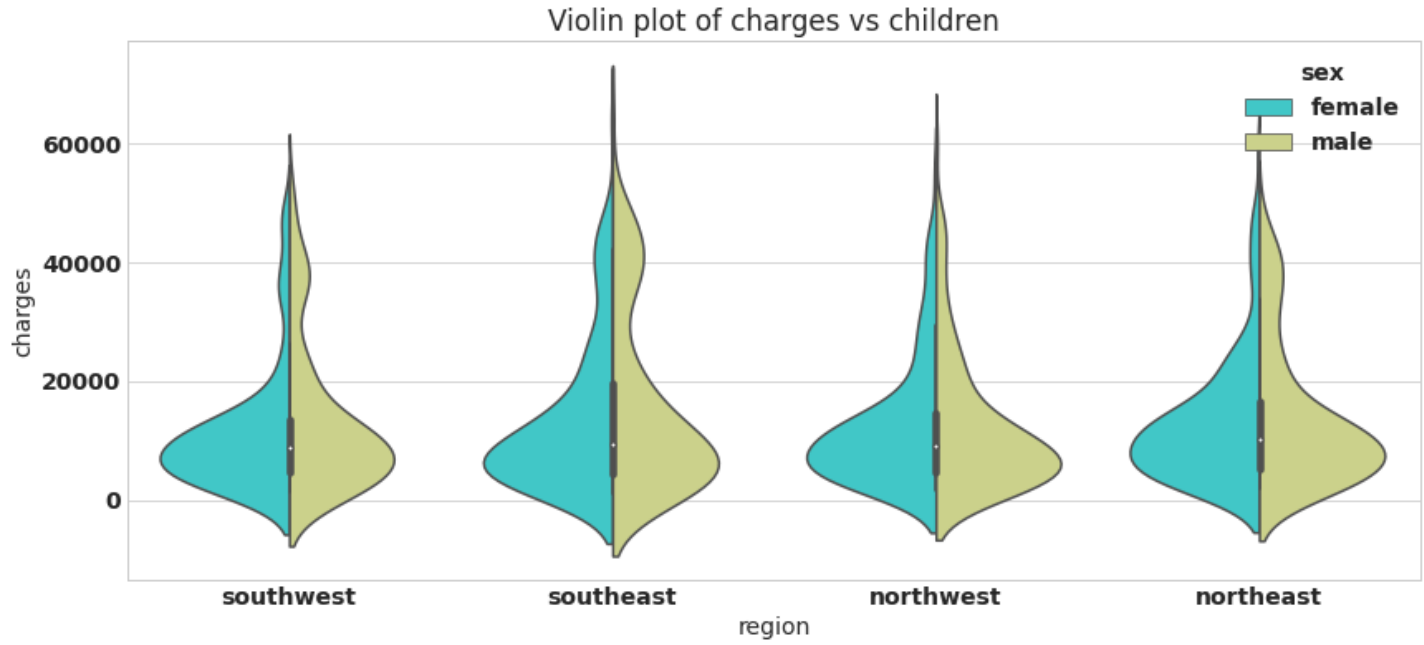
```
df.groupby('children').agg(['mean','min','max'])['charges']
```

Out[47]:

children	mean	min	max
0	12365.975602	1121.8739	63770.42801
1	12731.171832	1711.0268	58571.07448
2	15073.563734	2304.0022	49577.66240
3	15355.318367	3443.0640	60021.39897
4	13850.656311	4504.6624	40182.24600
5	8786.035247	4687.7970	19023.26000

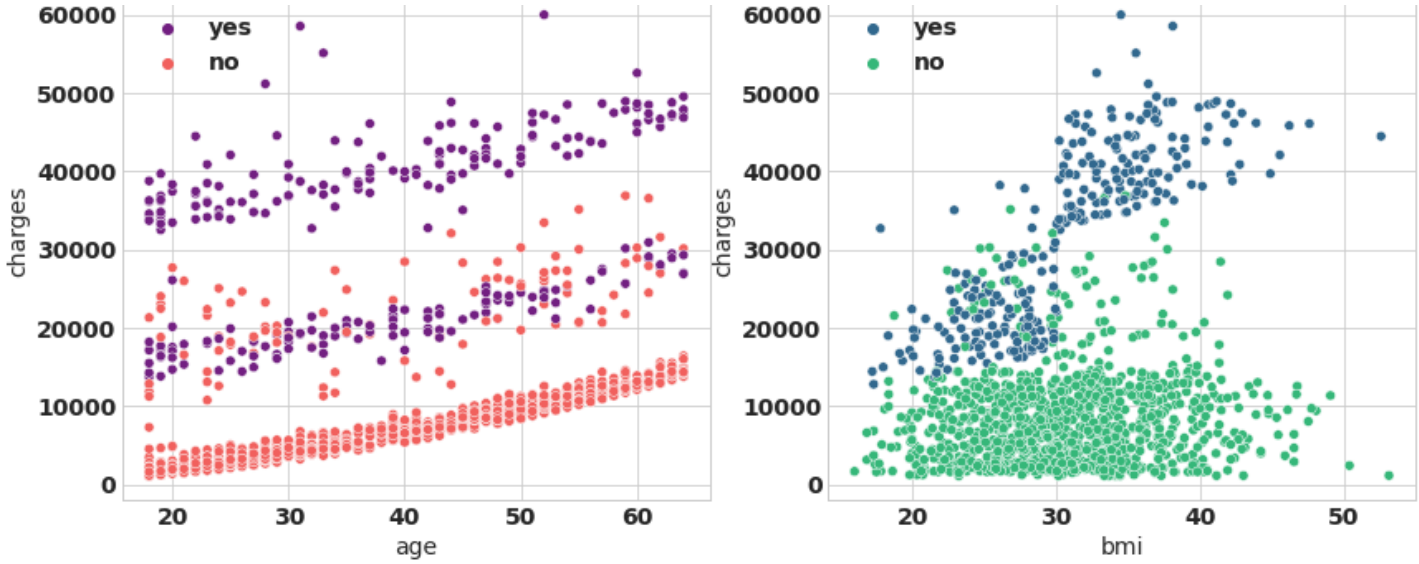
In [48]:

```
plt.figure(figsize=(14,6))
sns.violinplot(x='region', y='charges',hue='sex',data=df,palette='rainbow',split=True)
plt.title('Violin plot of charges vs children');
```



In [49]:

```
f = plt.figure(figsize=(14,6))
ax = f.add_subplot(121)
sns.scatterplot(x='age',y='charges',data=df,palette='magma',hue='smoker',ax=ax)
ax.set_title('Scatter plot of Charges vs age')

ax = f.add_subplot(122)
sns.scatterplot(x='bmi',y='charges',data=df,palette='viridis',hue='smoker')
ax.set_title('Scatter plot of Charges vs bmi')
plt.savefig('sc.png');
```



In [50]:

```
#Data Preprocessing techniques
#Categorical Features
# Dummy variable
categorical_columns = ['sex','children', 'smoker', 'region']
df_encode = pd.get_dummies(data = df, prefix = 'OHE', prefix_sep='_',
               columns = categorical_columns,
               drop_first =True,
              dtype='int8')
```

In [51]:

```
#One hot encoding
# Lets verify the dummay variable process
print('Columns in original data frame:\n',df.columns.values)
print('\nNumber of rows and columns in the dataset:',df.shape)
print('\nColumns in data frame after encoding dummy variable:\n',df_encode.columns.value
s)
print('\nNumber of rows and columns in the dataset:',df_encode.shape)
```

```
Columns in original data frame:
 ['age' 'sex' 'bmi' 'children' 'smoker' 'region' 'charges']

Number of rows and columns in the dataset: (1338, 7)

Columns in data frame after encoding dummy variable:
 ['age' 'bmi' 'charges' 'OHE_male' 'OHE_1' 'OHE_2' 'OHE_3' 'OHE_4' 'OHE_5'
 'OHE_yes' 'OHE_northwest' 'OHE_southeast' 'OHE_southwest']

Number of rows and columns in the dataset: (1338, 13)
```

In [52]:

```
#Normalization
#Box -Cox transformation
#A Box Cox transformation is a way to transform
     #non-normal dependent variables into a normal shape.
#Normality is an important assumption for
     #many statistical techniques
from scipy.stats import boxcox
y_bc,lam, ci= boxcox(df_encode['charges'],alpha=0.05)

#df['charges'] = y_bc
# it did not perform better for this model, so log transform is used
ci,lam
```

Out[52]:

```
((-0.01140290617294196, 0.0988096859767545), 0.043649053770664956)
```

In [53]:

```
## Log transform
df_encode['charges'] = np.log(df_encode['charges'])
```

In [54]:

```
#Model building
#Train Test split
from sklearn.model_selection import train_test_split
X = df_encode.drop('charges',axis=1) # Independet variable
y = df_encode['charges'] # dependent variable
# linear regression equation
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=23)
```

In [55]:

```
# Step 1: add x0 =1 to dataset
X_train_0 = np.c_[np.ones((X_train.shape[0],1)),X_train]
X_test_0 = np.c_[np.ones((X_test.shape[0],1)),X_test]

# Step2: build model
theta = np.matmul(np.linalg.inv( np.matmul(X_train_0.T,X_train_0) ), np.matmul(X_train_0
.T,y_train))
```

In [56]:

```
# The parameters for linear regression model
parameter = ['theta_'+str(i) for i in range(X_train_0.shape[1])]
columns = ['intersect:x_0=1'] + list(X.columns.values)
parameter_df = pd.DataFrame({'Parameter':parameter,'Columns':columns,'theta':theta})
```

In [57]:

```
# Scikit Learn module
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train,y_train) # Note: x_0 =1 is no need to add, sklearn will take care of
it.

#Parameter
sk_theta = [lin_reg.intercept_]+list(lin_reg.coef_)
parameter_df = parameter_df.join(pd.Series(sk_theta, name='Sklearn_theta'))
parameter_df
```

Out[57]:

	Parameter	Columns	theta	Sklearn_theta
0	theta_0	intersect:x_0=1	7.059171	7.059171
1	theta_1	age	0.033134	0.033134
2	theta_2	bmi	0.013517	0.013517
3	theta_3	OHE_male	-0.067767	-0.067767
4	theta_4	OHE_1	0.149457	0.149457
5	theta_5	OHE_2	0.272919	0.272919
6	theta_6	OHE_3	0.244095	0.244095
7	theta_7	OHE_4	0.523339	0.523339
8	theta_8	OHE_5	0.466030	0.466030
9	theta_9	OHE_yes	1.550481	1.550481
10	theta_10	OHE_northwest	-0.055845	-0.055845
11	theta_11	OHE_southeast	-0.146578	-0.146578
12	theta_12	OHE_southwest	-0.133508	-0.133508

In [58]:

```
# Normal equation
y_pred_norm =  np.matmul(X_test_0,theta)

#Evaluvation: MSE
J_mse = np.sum((y_pred_norm - y_test)**2)/ X_test_0.shape[0]

# R_square
sse = np.sum((y_pred_norm - y_test)**2)
sst = np.sum((y_test - y_test.mean())**2)
R_square = 1 - (sse/sst)
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse)
print('R square obtain for normal equation method is :',R_square)
```

```
The Mean Square Error(MSE) or J(theta) is:  0.18729622322982067
R square obtain for normal equation method is : 0.7795687545055298
```

In [59]:

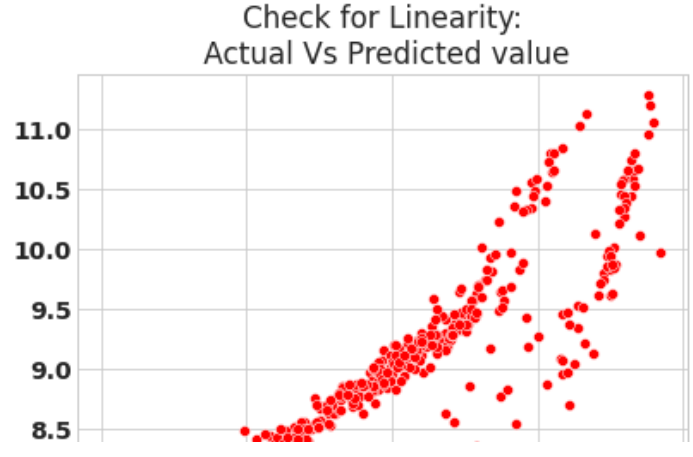
```
# sklearn regression module
y_pred_sk = lin_reg.predict(X_test)

#Evaluvation: MSE
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(y_pred_sk, y_test)

# R_square
R_square_sk = lin_reg.score(X_test,y_test)
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse_sk)
print('R square obtain for scikit learn library is :',R_square_sk)
```

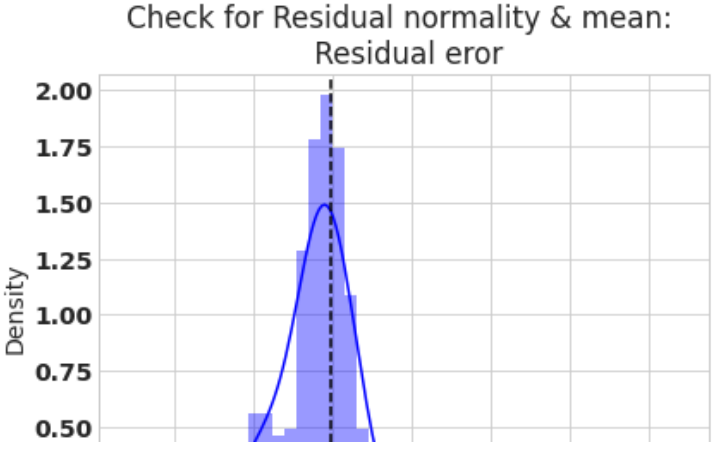
```
The Mean Square Error(MSE) or J(theta) is:  0.1872962232298189
R square obtain for scikit learn library is : 0.7795687545055319
```

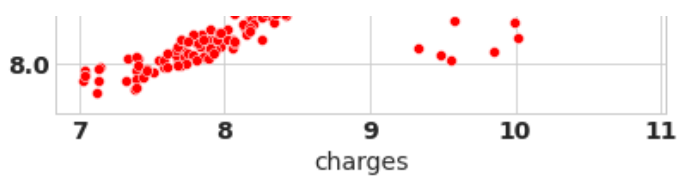
In [60]:

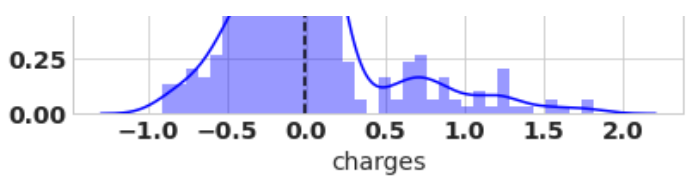
```
# Check for Linearity
f = plt.figure(figsize=(14,5))
ax = f.add_subplot(121)
sns.scatterplot(y_test,y_pred_sk,ax=ax,color='r')
ax.set_title('Check for Linearity:\n Actual Vs Predicted value')

# Check for Residual normality & mean
ax = f.add_subplot(122)
sns.distplot((y_test - y_pred_sk),ax=ax,color='b')
ax.axvline((y_test - y_pred_sk).mean(),color='k',linestyle='--')
ax.set_title('Check for Residual normality & mean: \n Residual eror');
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: 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.
  FutureWarning
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



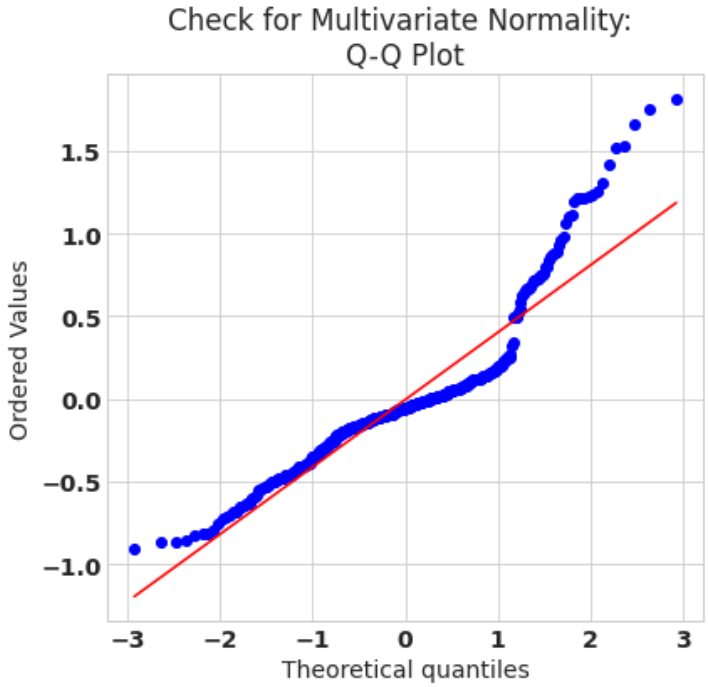


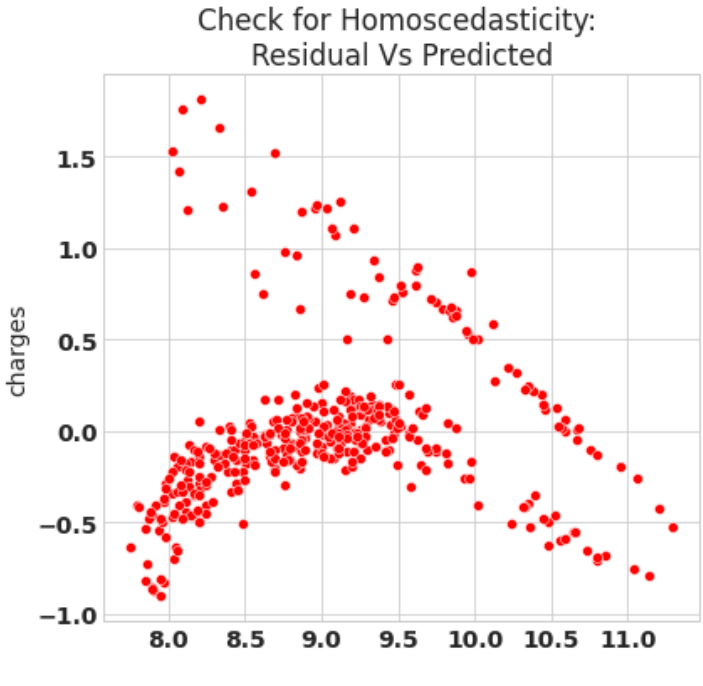




In [61]:

```
# Check for Multivariate Normality
# Quantile-Quantile plot
f,ax = plt.subplots(1,2,figsize=(14,6))
import scipy as sp
_,(_,_,r)= sp.stats.probplot((y_test - y_pred_sk),fit=True,plot=ax[0])
ax[0].set_title('Check for Multivariate Normality: \nQ-Q Plot')

#Check for Homoscedasticity
sns.scatterplot(y = (y_test - y_pred_sk), x= y_pred_sk, ax = ax[1],color='r')
ax[1].set_title('Check for Homoscedasticity: \nResidual Vs Predicted');
```





In [62]:

```
# Check for Multicollinearity
#Variance Inflation Factor
VIF = 1/(1- R_square_sk)
VIF
```

Out[62]:

```
4.536561945911138
```

In [ ]:

```
```