Import Libraries

In [1]:
```python
# Following libraries should be imported to run the experiment......
import pandas  as pd #Data manipulation
import numpy as np #Data manipulation
import matplotlib.pyplot as plt # Visualization
import seaborn as sns #Visualization
plt.rcParams['figure.figsize'] = [8,5]
plt.rcParams['font.size'] =14
plt.rcParams['font.weight']= 'bold'
plt.style.use('seaborn-whitegrid')
```

Load Dataset

In [2]:
```python
# Download and save csv file ("insurance.csv") in your folder.


df = pd.read_csv('....................../insurance.csv')
df
```

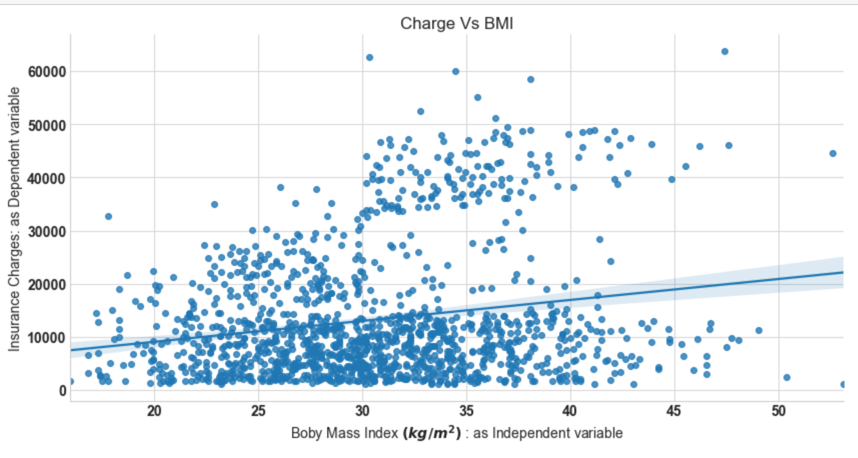
Out[2]:

	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

Visualization

In [3]:
```python
# Fit the line using seaborn library only for bmi as independent variable and charges as dependent variable.


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('Insurance Charges: as Dependent variable')
plt.title('Charge Vs BMI');
```



Data Analysis

In [4]:
```python
df.describe()
```

Out[4]:

	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

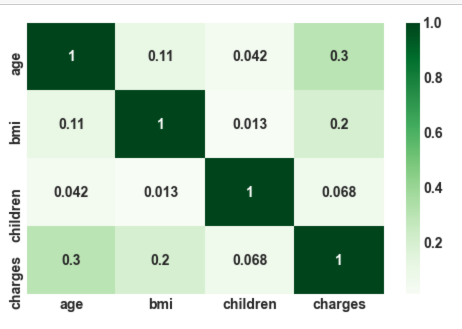
Remove Missing Values

In [5]:
```python
df.isnull().sum()
```

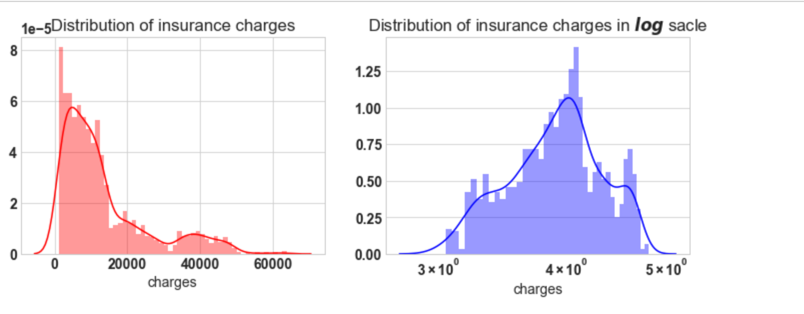
Out[5]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

Plots

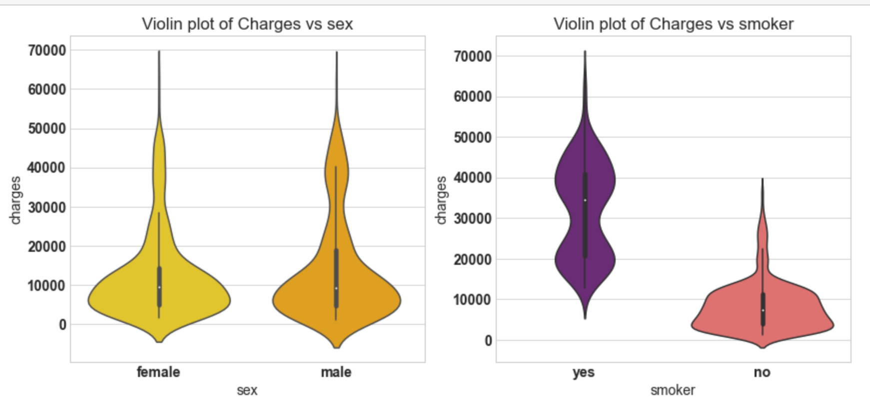
In [6]:
```python
# Correlation plot
corr = df.corr()
sns.heatmap(corr, cmap = 'Greens', annot= True);
```



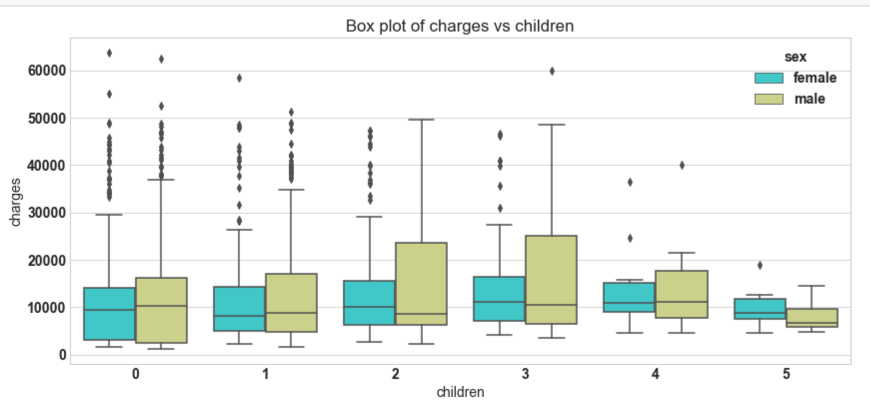
In [7]:
```python
# No Correlation found among variables in above plot.


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');
```



In [8]:
```python
# Observe at the left plot the charges varies from 1120 to 63500, the plot is right skewed.
# In right plot we will apply natural log, then plot approximately tends to normal.
# For further analysis we will apply log on target variable charges.


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 [9]:
```python
# Left plot indicates the insurance charge for male and female is approximatley in same range i.e., average around 5
000 bucks.
# Following facts are observed in right plot i.e.,
#   1.  Insurance charge for smokers is much wide range compare to non smokers.
#   2.  Average charges for non smoker is approximately 5000 bucks.
#   3.  For smoker the minimum insurance charge is itself 5000 bucks.


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



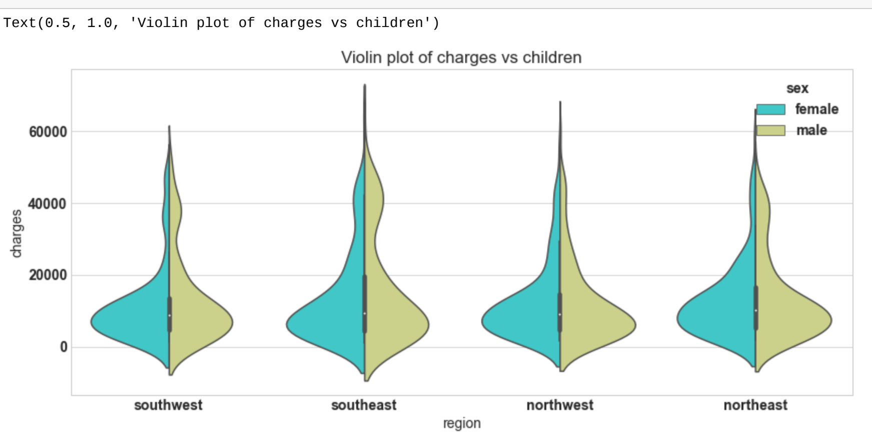
In [10]:
```python
df.groupby('children').agg(['mean','min','max'])['charges']
```

Out[10]:

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 [11]:
```python
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')
```

Out[11]:
```
Text(0.5, 1.0, 'Violin plot of charges vs children')
```



Data Preprocessing

In [12]:
```python
# Convert categorical data into numbers
# Dummy variable is a scenario in which the independent variable are multicollinear.
# A scenario in which two or more variables are highly correlated in simple term one variable can be predicted from
 the others.


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 [13]:
```python
# 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.values)
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)
```

Box - Cox transformation

In [14]:
```python
#  Box Cox transformation is a way to transform non-normal dependent variables into a normal shape.
#  All that we need to perform this transformation is to find lambda value.

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[14]:
```
((-0.01140290617294196, 0.0988096859767545), 0.043649053770664956)
```

In [15]:
```python
# Log transform
df_encode['charges'] = np.log(df_encode['charges'])
```

Train Test split

In [16]:
```python
from sklearn.model_selection import train_test_split
X = df_encode.drop('charges',axis=1) # Independent variable
y = df_encode['charges'] # dependent variable

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=23)
```

Model building

In [17]:
```python
# We build model using our linear regression equation  θ=(X^TX)^(−1)X^Ty .
# Need to add a feature  x0=1  to our original data set.

# 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 [18]:
```python
# 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 [19]:
```python
# 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[19]:

	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

Model evaluation

In [20]:
```python
#  Predict value for target variable by using our model parameter for test data set.
#  And, compare the predicted value with actual value in test set.
#  Compute Mean Square Error (MSE)
# 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:  0.18729622322981912
R square obtain for normal equation method is : 0.7795687545055316
```

In [21]:
```python
# 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:  0.18729622322981887
R square obtain for scikit learn library is : 0.7795687545055319
```

References

[1] https://www.kaggle.com/sudhirnl7/linear-regression-tutorial/notebook