We will build a Linear regression model for Medical cost dataset. The dataset consists of age, sex, BMI(body mass index), children, smoker and region feature, which are independent and charge as a dependent feature. We will predict individual medical costs billed by health insurance.

Definition & Working principle

Let's build model using **Linear regression**.

Linear regression is a **supervised learining** algorithm used when target / dependent variable **continues** real number. It establishes relationship between dependent variable y and one or more independent variable x using best fit line. It work on the principle of ordinary least square (OLS) / Mean square errror (MSE). In statistics ols is method to estimated unkown parameter of linear regression function, it's goal is to minimize sum of square difference between observed dependent variable in the given data set and those predicted by linear regression fuction.

Hypothesis representation

We will use x_i to denote the independent variable and y_i to denote dependent variable. A pair of (x_i, y_i) is called training example. The subscripe i in the notation is simply index into the training set. We have m training example then $i=1,2,3,\dots m$.

The goal of supervised learning is to learn a *hypothesis function* h, for a given training set that can used to estimate y based on x. So hypothesis fuction represented as

$$h_\theta(x_i)=\theta_0+\theta_1 x_i$$

θ_0, θ_1 are parameter of hypothesis.This is equation for **Simple / Univariate Linear regression**.

For **Multiple Linear regression** more than one independent variable exit then we will use x_{ij} to denote indepedent variable and y_i to denote dependent variable. We have n independent variable then $j=1,2,3\dots..n$. The hypothesis function represented as

$$h_\theta(x_i)=\theta_0+\theta_1 x_{i1}+\theta_2 x_{i2}+\dots..\theta_j x_{ij}\dots...\theta_n x_{mn}$$

$\theta_0, \theta_1, \dots.\theta_j \dots.\theta_n$ are parameter of hypothesis, m Number of training exaples, n Number of independent variable, x_{ij} is i^{th} training exaple of j^{th} feature.

Import Library and Dataset

Now we will import couple of python library required for our analysis and import dataset

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

```python
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')

# Import dataset
#path ='dataset/'
path = '../input/'
df = pd.read_csv(path+'insurance.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)


   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
```

Now we have import dataset. When we look at the shape of dataset it has return as (1338,7).So there are $m=1338$ training exaple and $n=7$ independent variable. The target variable here is charges and remaining six variables such as age, sex, bmi, children, smoker, region are independent variable. There are multiple independent variable, so we need to fit Multiple linear regression. Then the hypothesis function looks like

$$h_\theta(x_i)=\theta_0+\theta_1 age+\theta_2 sex+\theta_3 bmi+\theta_4 children+\theta_5 smoker+\theta_6 region$$

This multiple linear regression equation for given dataset.
If $i=1$ then

$$h_\theta(x_1)=\theta_0+\theta_1 19+\theta_2 female+\theta_3 27.900+\theta_4 1+\theta_5 yes+\theta_6 southwest$$

$$y_1=16884.92400$$

If $i=3$ then

$$h_\theta(x_3)=\theta_0+\theta_1 28+\theta_2 male+\theta_3 33.000+\theta_4 3+\theta_5 no+\theta_6 northwest$$

$$y_3=4449.46200$$

Note: In python index starts from 0.

$$x_1 = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \end{pmatrix} = \begin{pmatrix} 19 & female & 27.900 & 1 & no & northwest \end{pmatrix}$$

Matrix Formulation

In general we can write above vector as

$$x_{ij} = \begin{pmatrix} x_{i1} & x_{i2} & . & . & . & x_{in} \end{pmatrix}$$

Now we combine all aviable individual vector into single input matrix of size (m,n) and denoted it by X input matrix, which consist of all training exaples,

$$X = \begin{pmatrix} x_{11} & x_{12} & . & . & . & . & x_{1n} \\ x_{21} & x_{22} & . & . & . & . & x_{2n}\text{¿.¿.¿.¿.¿.¿.¿.¿¿}x_{m1}\text{¿}x_{m2}\text{¿.¿.¿.¿.¿.¿.}x_{mn}\text{¿} \\ x_{31} & x_{32} & . & . & . & . & x_{3n} \\ . & . & . & . & . & . & \text{¿} \end{pmatrix}_{(m,n)}$$

We represent parameter of function and dependent variable in vactor form as

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ \theta_j \\ . \\ . \\ \theta_n \end{pmatrix}_{(n+1,1)} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ y_i \\ . \\ . \\ y_m \end{pmatrix}_{(m,1)}$$

So we represent hypothesis function in vectorize form

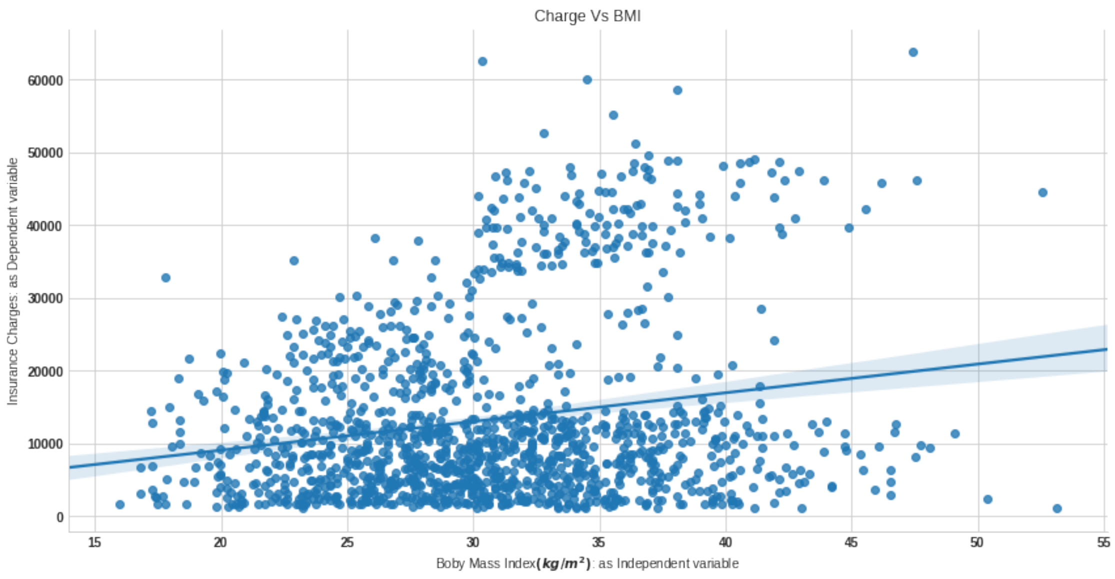
$$h_\theta(x) = X\theta$$

.

```
""" for our visualization purpose will fit 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');
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`.
In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a
```

```
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In above plot we fit regression line into the variables.

Cost function

A cost function measures how much error in the model is in terms of ability to estimate the relationship between x and y. We can measure the accuracy of our hypothesis function by using a cost function. This takes an average difference of observed dependent variable in the given the dataset and those predicted by the hypothesis function.

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i-y_i)^2$$

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)^2$$

To implement the linear regression, take training example add an extra column that is x_0 feature, where $x_0=1$. $x_o=(x_{i0} \quad x_{i1} \quad x_{i2} \quad . \quad . \quad . \quad x_{mi})$,where $x_{i0}=0$ and input matrix will become as

$$X=\begin{pmatrix} x_{10} & x_{11} & x_{12} & . & . & . & . & x_{1n} \\ x_{20} & x_{21} & x_{22} & . & . & . & . & x_{2n}¿.¿.¿.¿.¿.¿.¿.¿.¿.¿¿x_{m0}¿x_{m1}¿x_{m2}¿.¿.¿.¿.¿.¿.x_{mn}¿ \\ x_{30} & x_{31} & x_{32} & . & . & . & . & x_{3n} \\ . & . & . & . & . & . & . & ¿ \end{pmatrix}_{(m,n+1)}$$

Each of the m input samples is similarly a column vector with n+1 rows x_0 being 1 for our convenience, that is $x_{10}, x_{20}, x_{30} \ldots x_{m0} = 1$. Now we rewrite the ordinary least square cost function in matrix form as

$$J(\theta) = \frac{1}{m}(X\theta - y)^T (X\theta - y)$$

Let's look at the matrix multiplication concept,the multiplication of two matrix happens only if number of column of firt matrix is equal to number of row of second matrix. Here input matrix X of size $(m, n+1)$, parameter of function is of size $(n+1, 1)$ and dependent variable vector of size $(m, 1)$. The product of matrix $X_{(m,n+1)}\theta_{(n+1,1)}$ will return a vector of size $(m, 1)$, then product of $(X\theta - y)_{(1,m¿¿T)(X\theta-y)_{(m,1)}¿}$ will return size of unit vector.

Normal Equation

The normal equation is an analytical solution to the linear regression problem with a ordinary least square cost function. To minimize our cost function, take partial derivative of $J(\theta)$ with respect to θ and equate to 0. The derivative of function is nothing but if a small change in input what would be the change in output of function.

$$min_{\theta_0, \theta_1 .. \theta_n} J(\theta_0, \theta_1 .. \theta_n)$$

$$\frac{\partial J(\theta_j)}{\partial \theta_j} = 0$$

where $j = 0, 1, 2, \ldots . n$

Now we will apply partial derivative of our cost function,

$$\frac{\partial J(\theta_j)}{\partial \theta_j} = \frac{\partial}{\partial \theta}\frac{1}{m}(X\theta - y)^T (X\theta - y)$$

I will throw $\frac{1}{m}$ part away since we are going to compare a derivative to 0. And solve $J(\theta)$,

$$J(\theta) = (X\theta - y)^T (X\theta - y)$$

$$¿(X\theta)^T - y^T ¿(X\theta - y)$$

$$¿(\theta^T X^T - y^T)(X\theta - y)$$

$$¿\theta^T X^T X\theta - y^T X\theta - \theta^T X^T y + y^T y$$

$$¿\theta^T X^T X\theta - 2\theta^T X^T y + y^T y$$

Here $y^T_{(1,m)} X_{(m,n+1)} \theta_{(n+1,1)} = \theta^T_{(1,n+1)} X^T_{(n+1,m)} y_{(m,1)}$ because unit vector.

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\theta^T X^T X \theta - 2\theta^T X^T y + y^T y\right)$$

$$¿ X^T X \frac{\partial \theta^T \theta}{\partial \theta} - 2 X^T y \frac{\partial \theta^T}{\partial \theta} + \frac{\partial y^T y}{\partial \theta}$$

Partial derivative $\frac{\partial x^2}{\partial x} = 2x, \frac{\partial k x^2}{\partial x} = kx, \frac{\partial Constact}{\partial x} = 0$

$$\frac{\partial J(\theta)}{\partial \theta} = X^T X 2\theta - 2 X^T y + 0$$

$$0 = 2 X^T X \theta - 2 X^T y$$

$$X^T X \theta = X^T$$

$$\theta = \left(X^T X\right)^{-1} X^T y$$

this the normal equation for linear regression

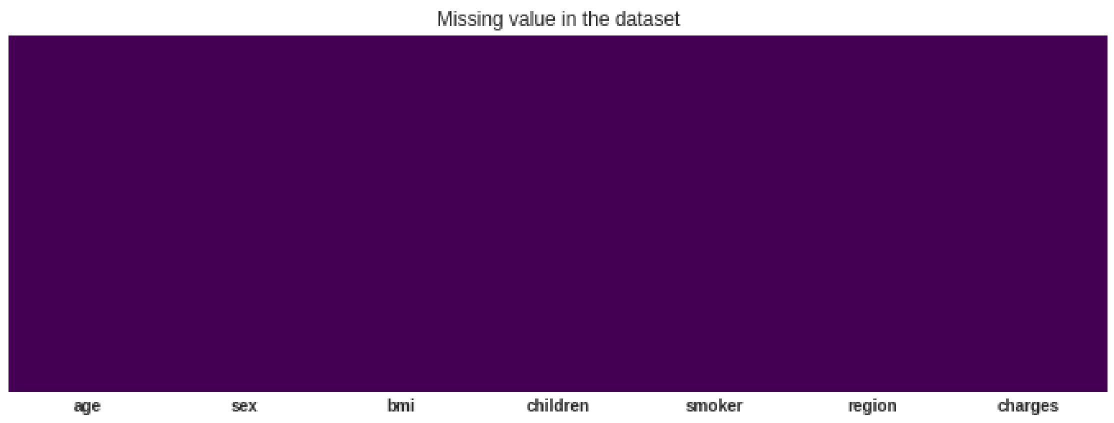
Exploratory data analysis

```
df.describe()
```

```
                 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
```

Check for missing value

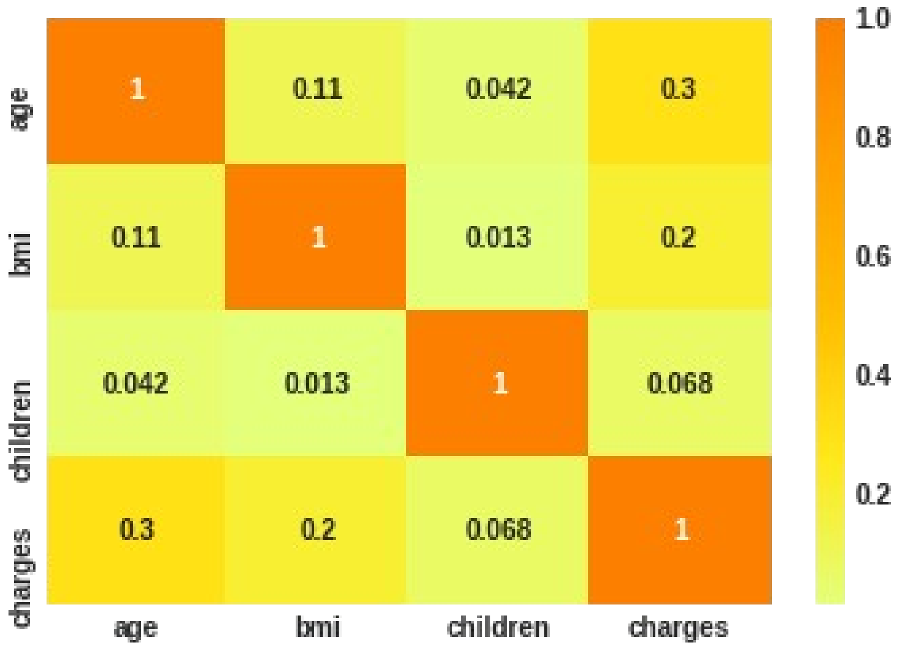
```
plt.figure(figsize=(12,4))
sns.heatmap(df.isnull(),cbar=False,cmap='viridis',yticklabels=False)
plt.title('Missing value in the dataset');
```



There is no missing value in the data sex

Plots

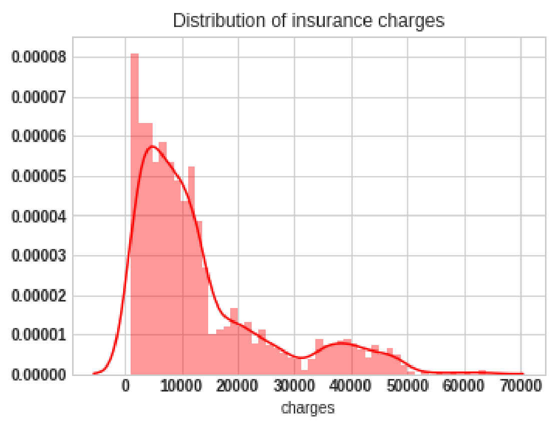
```
# correlation plot
corr = df.corr()
sns.heatmap(corr, cmap = 'Wistia', annot= True);
```

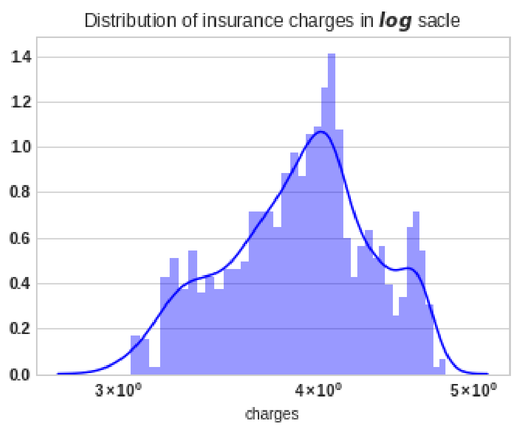


Thier no correlation among valiables.

```
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/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`.
In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

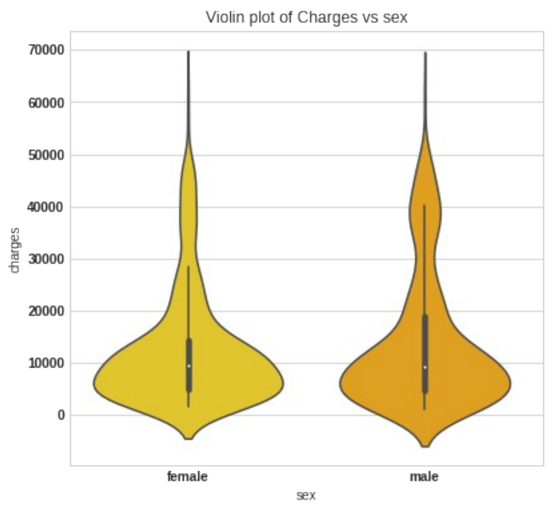


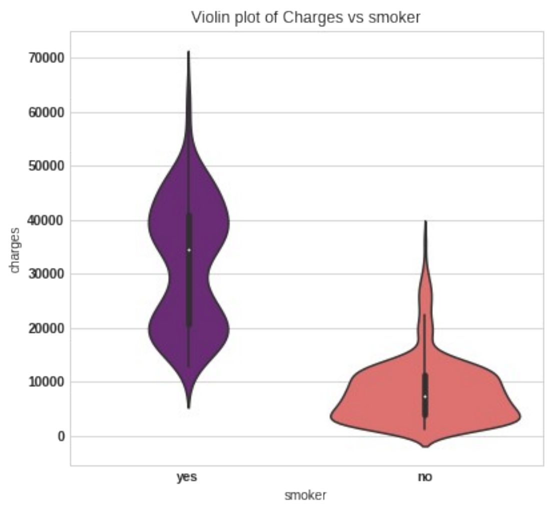


If we look at the left plot the charges varies from 1120 to 63500, the plot is right skewed. In right plot we will apply natural log, then plot approximately tends to normal. for further analysis we will apply log on target variable charges.

```
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');
```

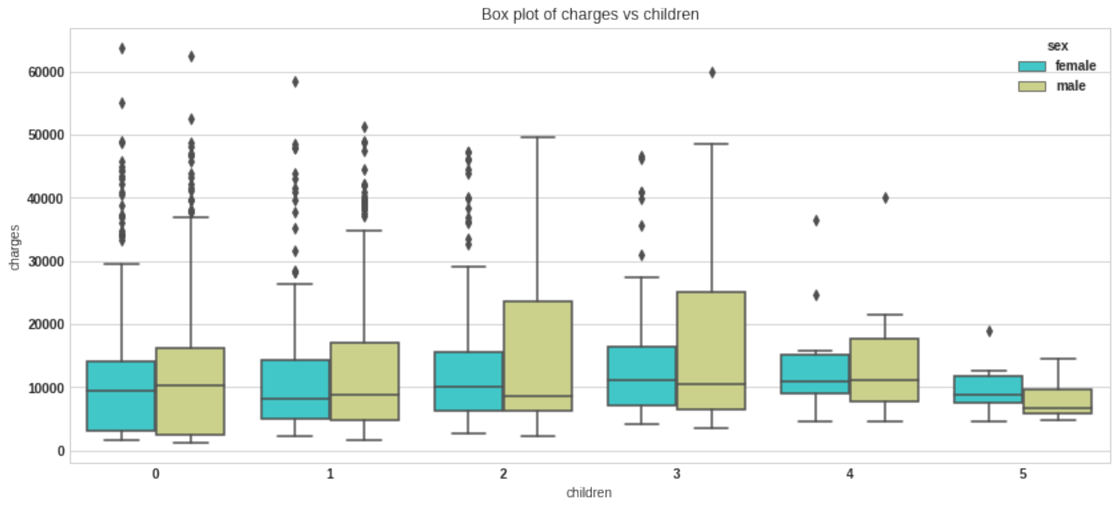
```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`.
In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





From left plot the insurance charge for male and female is approximatley in same range,it is average around 5000 bucks. In right plot the insurance charge for smokers is much wide range compare to non smokers, the average charges for non smoker is approximately 5000 bucks. 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');
```

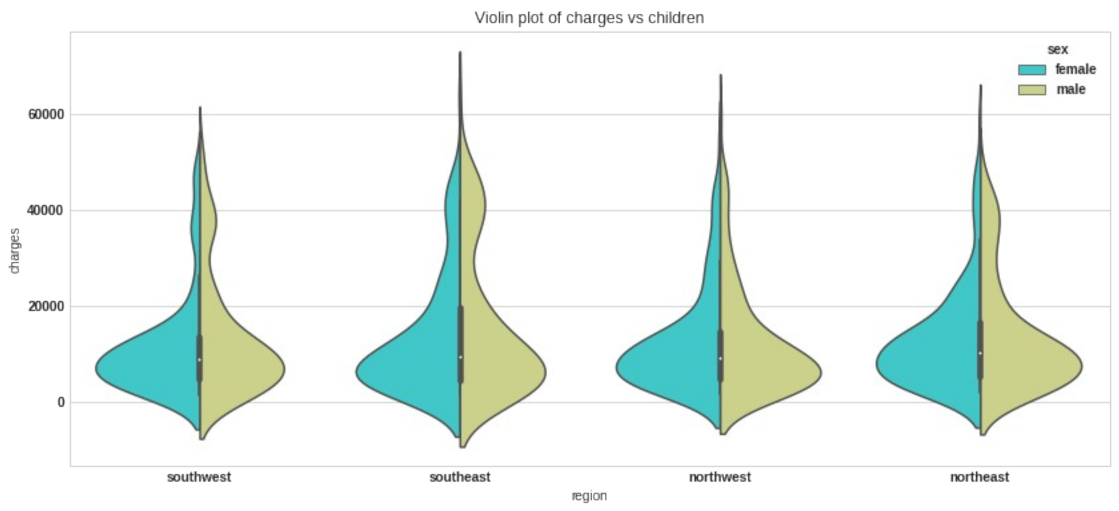


```
df.groupby('children').agg(['mean','min','max'])['charges']
```

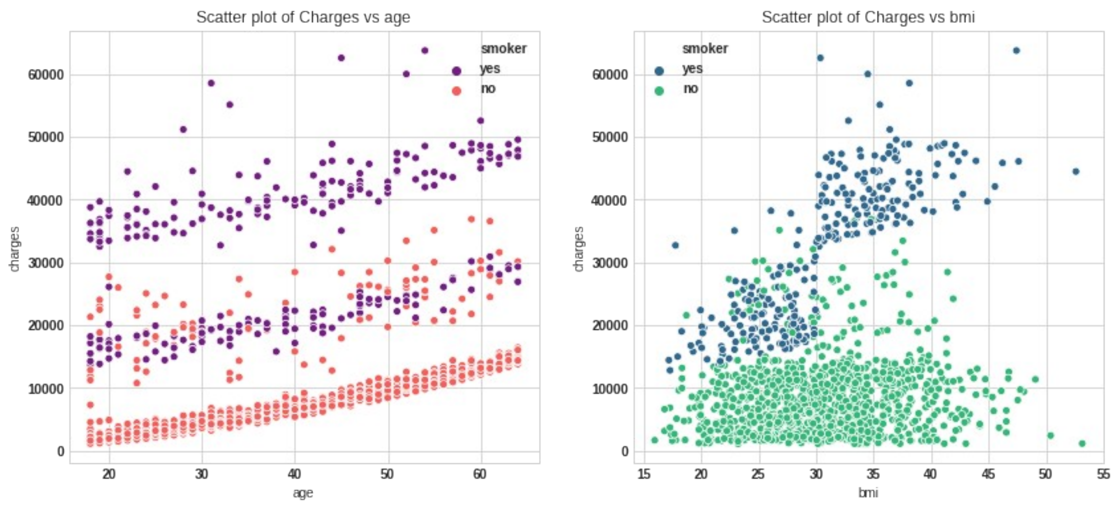
```
                  mean        min          max
children
0         12365.975602  1121.8739  63770.42801
1         12731.171832  1711.0268  58571.07448
2         15073.563734  2304.0022  49577.66240
3         15355.318367  3443.0640  60021.39897
4         13850.656311  4504.6624  40182.24600
5    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

```
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');
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`.
In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



```
f = plt.figure(figsize=(14,6))
ax = f.add_subplot(121)
sns.scatterplot(x='age',y='charges',data=df,palette='magma',hue='smoke
r',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='smo
ker')
ax.set_title('Scatter plot of Charges vs bmi')
plt.savefig('sc.png');
```



From left plot the minimum age person is insured is 18 year. There is slabs in policy most of non smoker take 1^{st} and 2^{nd} slab, for smoker policy start at 2^{nd} and 3^{rd} slab.

Body mass index (BMI) is a measure of body fat based on height and weight that applies to adult men and women. The minimum bmi is $16 kg/m^2$ and maximum upto $54 kg/m^2$

Data Preprocessing

Encoding

Machine learning algorithms cannot work with categorical data directly, categorical data must be converted to number.

1. Label Encoding
2. One hot encoding
3. Dummy variable trap

Label encoding refers to transforming the word labels into numerical form so that the algorithms can understand how to operate on them.

A **One hot encoding** is a representation of categorical variable as binary vectors.It allows the representation of categorical data to be more expresive. This first requires that the categorical values be mapped to integer values, that is label encoding. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

The **Dummy variable trap** 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.

By using *pandas get_dummies* function we can do all above three step in line of code. We will this fuction to get dummy variable for sex, children,smoker,region features. By setting *drop_first =True* function will remove dummy variable trap by droping one variable and original variable.The pandas makes our life easy.

```
# 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')

# 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

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; if your data isn't normal, applying a Box-Cox means that you are able to run a broader number of tests. All that we need to perform this transformation is to find lambda value and apply the rule shown below to your variable.

$$\left\{ \begin{matrix} \frac{y^{\lambda}-1}{\lambda}, & y_i \neg = 0 \\ log\left(y_i\right) & \lambda = 0 \end{matrix} \right)$$

The trick of Box-Cox transformation is to find lambda value, however in practice this is quite affordable. The following function returns the transformed variable, lambda value,confidence interval

```
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
```

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

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

The original categorical variable are remove and also one of the one hot encode varible column for perticular categorical variable is droped from the column. So we completed all three encoding step by using get dummies function.

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

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

Model building

In this step build model using our linear regression equation $\theta=\left(X^T X\right)^{-1} X^T y$. In first step we need to add a feature $x_0=1$ 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))

# 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})

# 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
```

```
   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
```

The parameter obtained from both the model are same.So we succefull build our model using normal equation and verified using sklearn linear regression module. Let's move ahead, next step is prediction and model evaluvation.

Model evaluation

We will predict value for target variable by using our model parameter for test data set. Then compare the predicted value with actual valu in test set. We compute **Mean Square Error** using formula

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

R^2 is statistical measure of how close data are to the fitted regression line. R^2 is always between 0 to 100%. 0% indicated that model explains none of the variability of the response data around it's mean. 100% indicated that model explains all the variablity of the response data around the mean.

$$R^2=1-\frac{SSE}{SST}$$

SSE = Sum of Square Error
SST = Sum of Square Total

$$SSE=\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

$$SST=\sum_{i=1}^{m}(y_i - \acute{y}_i)^2$$

Here $\hat{y}$ is predicted value and $\acute{y}$ is mean value of y.

```
# 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 : 0.7795687545055299
```

```
# 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 : 0.7795687545055318
```

The model returns R^2 value of 77.95%, so it fit our data test very well, but still we can imporve the the performance of by diffirent technique. Please make a note that we have transformer out variable by applying natural log. When we put model into production antilog is applied to the equation.

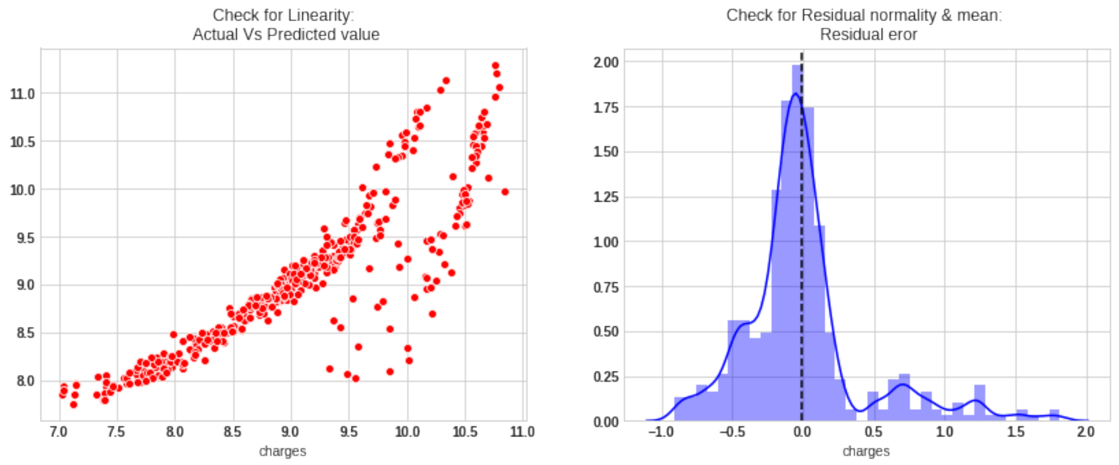
Model Validation

In order to validated model we need to check few assumption of linear regression model. The common assumption for *Linear Regression* model are following

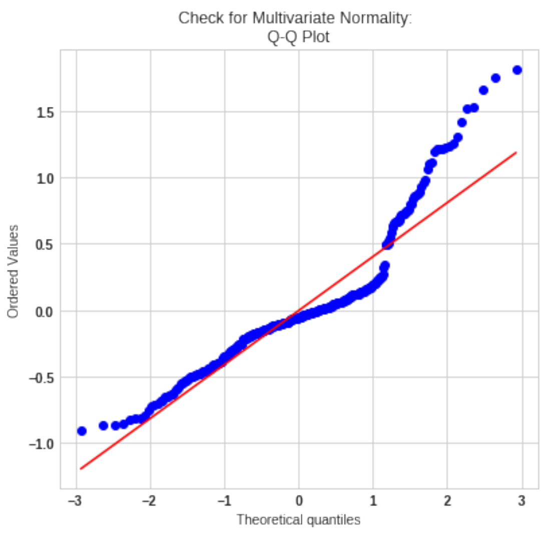
1. Linear Relationship: In linear regression the relationship between the dependent and independent variable to be *linear*. This can be checked by scatter ploting Actual value Vs Predicted value
2. The residual error plot should be *normally* distributed.
3. The *mean* of *residual error* should be 0 or close to 0 as much as possible
4. The linear regression require all variables to be multivariate normal. This assumption can best checked with Q-Q plot.
5. Linear regession assumes that there is little or no *Multicollinearity in the data. Multicollinearity occurs when the independent variables are too highly correlated with each other. The variance inflation factor* VIF* identifies correlation between independent variables and strength of that correlation. $VIF=\frac{1}{1-R^2}$, If VIF >1 & VIF <5 moderate correlation, VIF < 5 critical level of multicollinearity.
6. Homoscedasticity: The data are homoscedastic meaning the residuals are equal across the regression line. We can look at residual Vs fitted value scatter plot. If heteroscedastic plot would exhibit a funnel shape pattern.

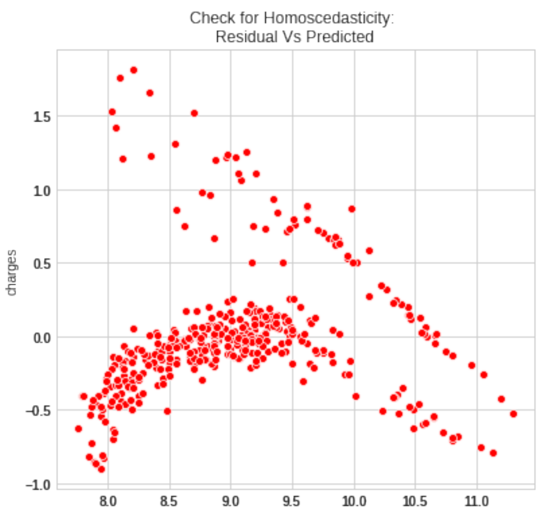
```
# Check for Linearity
f = plt.figure(figsize=(14,5))
ax = f.add_subplot(121)
sns.scatterplot(y_test,y_pred_sk,ax=ax,color='r')
ax.set_title('Check for Linearity:\n Actual Vs Predicted value')

# 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/python3.6/site-packages/scipy/stats/stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`.
In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



```python
# 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');
```





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

```
4.536561945911135
```

The model assumption linear regression as follows

1. In our model the actual vs predicted plot is curve so linear assumption fails
2. The residual mean is zero and residual error plot right skewed
3. Q-Q plot shows as value log value greater than 1.5 trends to increase
4. The plot is exhibit heteroscedastic, error will insease after certian point.
5. Variance inflation factor value is less than 5, so no multicollearity.

1) What do you mean by Linear Regression ?

-> Linear regression is a statistical technique used to analyze the relationship between a dependent variable (often called the "outcome" or "response" variable) and one or more independent variables (often called the "predictors" or "explanatory" variables).

The main idea of linear regression is to find the line that best fits the data points, based on a mathematical formula that defines the relationship between the variables. This line is often called the "regression line" or "line of best fit."

The basic form of a linear regression model can be written as:

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_r x_r + \varepsilon$

where y is the dependent variable, x_1, x_2, ..., x_r are the independent variables, β_0, β_1, β_2, ..., β_r are the regression coefficients, and ε is the error term (representing the difference between the predicted and actual values of y).

The goal of linear regression is to estimate the values of the regression coefficients that minimize the sum of the squared errors (the difference between the predicted and actual values of y). Once these coefficients have been estimated, they can be used to predict the value of y for new values of $x_1, x_2, \ldots, x_r$.

2) Why is linear regression called as supervised learning?

-> Linear regression is called supervised learning because it involves using a set of input-output pairs (also known as training data) to learn a mapping function that predicts the output variable (also known as the dependent variable) based on the input variables (also known as the independent variables).

In other words, in supervised learning, we have a set of labeled data where we know the input values and their corresponding output values, and we use this data to train a model that can make predictions on new, unseen data.

In the case of linear regression, we use a set of input-output pairs to estimate the coefficients of the regression equation that can predict the output variable based on the input variables. The input variables are used to predict the output variable, which is also known as the dependent variable.

Therefore, linear regression is a supervised learning technique because it learns from labeled data, where the output variable is known and used to train the model.

3) If the input data has some feature which is nominal can that column be used for Solving regression, why?

-> If the input data has a feature that is nominal (i.e., categorical data where there is no inherent order or hierarchy among the categories), it can still be used for solving regression, but it requires some pre-processing.

One common approach is to use a technique called "one-hot encoding," which involves creating a binary column for each category in the nominal feature. Each binary column represents whether or not the observation belongs to a particular category.

For example, let's say we have a nominal feature called "color" with three categories: "red", "green", and "blue." In this case, we would create three binary columns called "color_red", "color_green", and "color_blue." Each binary column would have a value of 1 if the observation belongs to that category, and 0 otherwise.

Once the nominal feature has been one-hot encoded, it can be used as input to a regression model. The regression model will treat each binary column as an independent variable and estimate the corresponding regression coefficient.

It's important to note that one-hot encoding can increase the number of independent variables in the model, which can lead to overfitting if the number of observations is small relative to the number of independent variables. Therefore, it's important to carefully consider the trade-off between the number of independent variables and the number of observations when using nominal features in regression

4) What is role of sklearn package in tutorial?

-> The scikit-learn (sklearn) package is a popular Python library for machine learning, which provides a range of tools for data analysis, modeling, and evaluation. In the context of a tutorial on machine learning or data analysis, sklearn can be used to implement and apply the various techniques covered in the tutorial.

The sklearn package provides a unified interface for a wide range of machine learning algorithms, including regression, classification, clustering, and dimensionality reduction, among others. It also includes a variety of pre-processing and feature extraction methods, as well as tools for model selection and evaluation.

In a tutorial, the sklearn package can be used to demonstrate how to implement various machine learning algorithms using Python, and how to apply them to real-world data. This can include tasks such as data cleaning and pre-processing, feature selection and extraction, model training and testing, and hyperparameter tuning.

Additionally, the sklearn package provides a range of visualization tools that can be used to explore and visualize data, as well as the results of machine learning models. This can be particularly useful in a tutorial, as it allows the user to gain a better understanding of the underlying patterns in the data and the performance of the models.

Overall, the sklearn package is a valuable resource for anyone learning about machine learning or data analysis in Python, and is often used in tutorials to illustrate the practical application of these techniques.

5) What is violin plot? Explain what information is displayed in any one of the plots shown in notebook.

-> A violin plot is a type of data visualization that shows the distribution of numerical data across different categories. It combines the features of a box plot and a kernel density plot to provide a richer visualization of the data.

In a violin plot, each category is represented by a "violin" shape, which shows the density of the data at different values. The width of the violin at a given point represents the density of data at that point. The height of the violin is typically standardized to show the same area for each category.

In a notebook, a violin plot might be used to visualize the distribution of a numerical variable across different categories. For example, suppose we have a dataset that includes information about the height of plants in different gardens. We might be interested in comparing the distribution of heights across different gardens.

A violin plot for this dataset would show a separate "violin" shape for each garden, with the width of the shape representing the density of heights at different values. We might also include a box plot inside each violin shape to show the quartiles of the data distribution, and any outliers.

For example, let's say we have a dataset with four gardens, and we want to compare the distribution of plant heights across the gardens using a violin plot. The plot might look like this:

Violin plot example

In this example, the x-axis shows the garden names, and the y-axis shows the height values. Each garden is represented by a violin shape, with the width of the shape representing the density of height values at different values. The box plot inside each shape shows the quartiles of the data distribution, and the whiskers indicate the range of the data (excluding outliers).

From this plot, we can see that Garden 1 has the widest distribution of plant heights, with a significant number of outliers. Garden 2 has a fairly symmetric distribution with few outliers. Garden 3 has a bimodal distribution, with a cluster of shorter plants and a cluster of taller plants. Garden 4 has the narrowest distribution, with all plants being relatively similar in height.