What is K-NN Regression?

K-NN stands for K-Nearest Neighbors. It is an algorithm used for the prediction of a continuous variable. A non-parametric and a prediction problem; it does not care about the relationship between the predictor x the response variable y. It takes k nearest neighbors whose distances form that point are minimum and computes the average of those values.

KNN Algorithm

The following are the steps for K-NN Regression:

1. Find the k nearest neighbors based on distances for x.
2. Average the output of the K-Nearest Neighbors of x.

In [1]:
```
# Import the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [2]:
```
df=pd.read_csv('Advertising.csv')
df.head(5)
```

Out[2]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

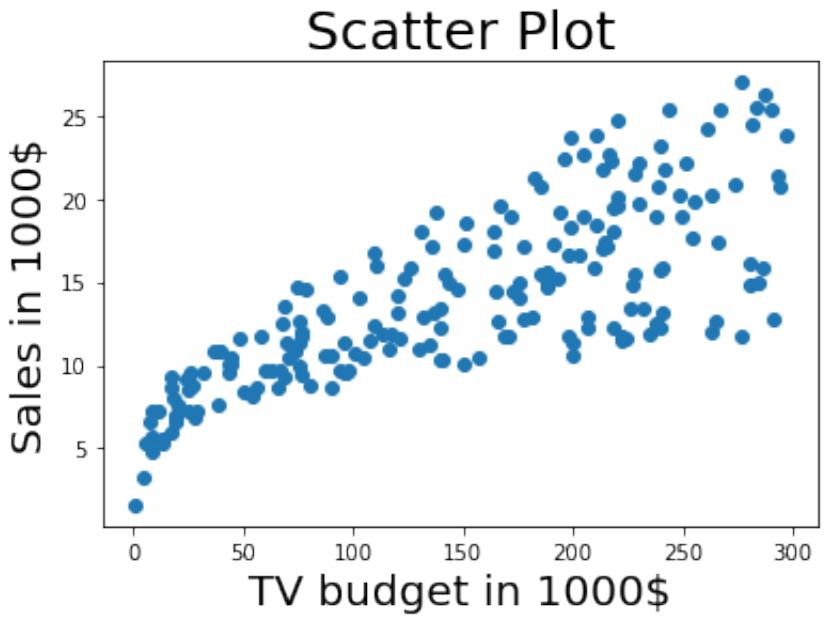
We Will be regressing Sales on TV here.

Now, select variable x(predictor) and y(response) and plot a scatter plot

In [3]:
```
# Select variables and draw a scatter Plot
x=df[['TV']].values # predictor
y=df['Sales'].values #response or output variable

plt.scatter(x,y)
plt.xlabel('TV budget in 1000$' ,fontsize=20)
plt.ylabel('Sales in 1000$',fontsize=20)
plt.title('Scatter Plot',fontsize=25)
```

Out[3]:
```
Text(0.5, 1.0, 'Scatter Plot')
```



In [ ]:
```
```

We split our data set now, let's do it then. Take 70% of your data set as train set.

In [4]:
```
#Split Data set
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7)
```

We need to get the optimal value of k for our model, how ?

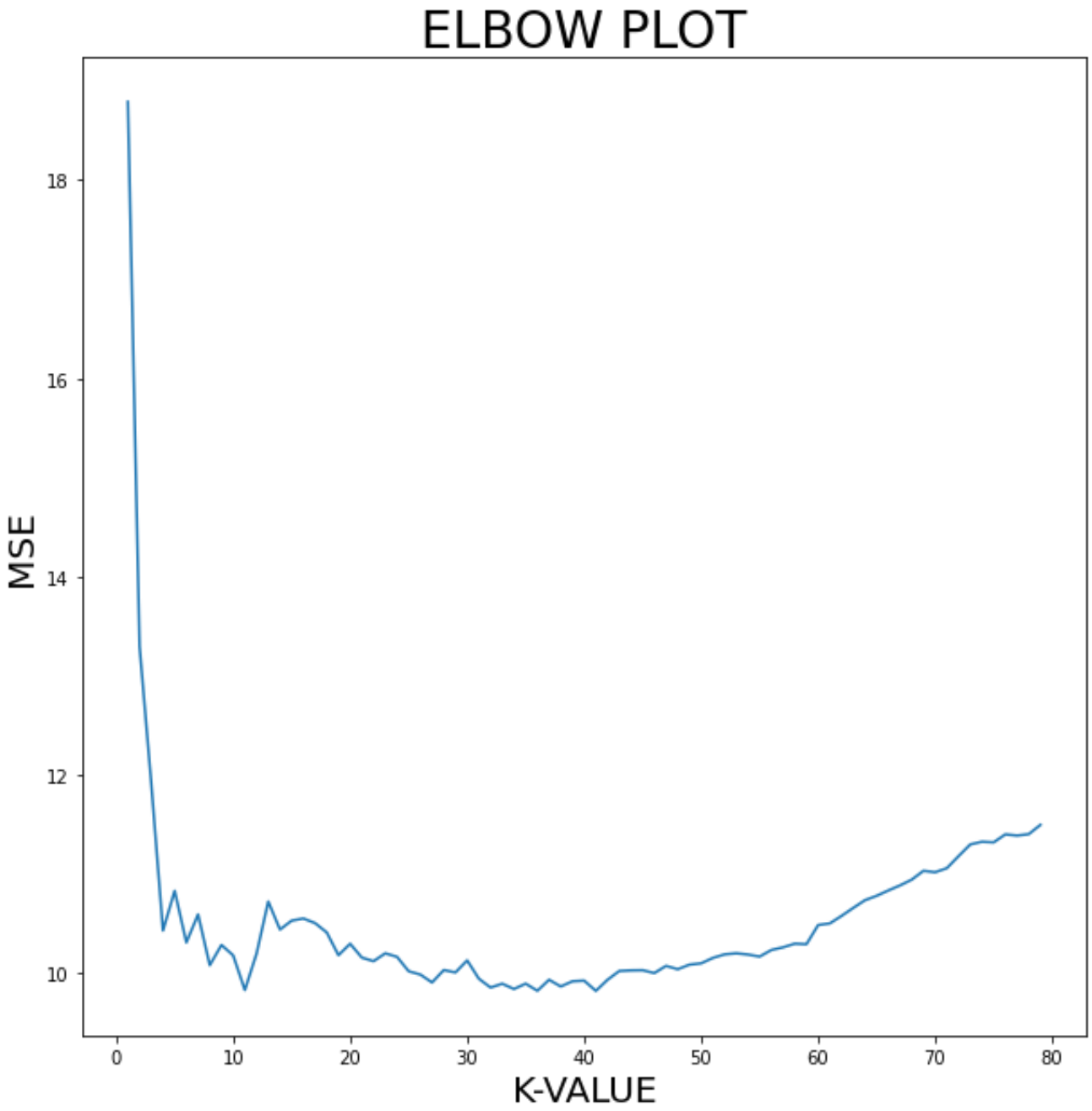
We will plot a graph between mse and k values and we will choose the one with the lowest mse. This graph is known as elbow plot and you will know why.

We create an elbow plot by developing a model taking k =[1,n] while recording MSE for each model, you can choose n of your choice and we will take upto 80. You can experiment with this number, generally depends on you number of observations.

In [5]:

```
fig,ax=plt.subplots(figsize=(10,10))
k_list=np.arange(1,80,1)
knn_dict={} # To store k and mse pairs
for i in k_list:
#Knn Model Creation
    knn=KNeighborsRegressor(n_neighbors=int(i))
    model_knn=knn.fit(x_train,y_train)
    y_knn_pred=model_knn.predict(x_test)
#Storing MSE
    mse=mean_squared_error(y_test,y_knn_pred)
    knn_dict[i]=mse
#Plotting the results
ax.plot(knn_dict.keys(),knn_dict.values())
ax.set_xlabel('K-VALUE', fontsize=20)
ax.set_ylabel('MSE' ,fontsize=20)
ax.set_title('ELBOW PLOT' ,fontsize=28)
```

Out[5]: Text(0.5, 1.0, 'ELBOW PLOT')



In [6]:
```
mean_squared_error(y_test,y_knn_pred)
```

Out[6]: 11.487973134647229

It clearly shows our MSE is minimum when k=47. We build our model similar to the way we did above and we just substitute the value of n_neighbors as 9 while doing that. Let's see the results.

The mean Square error turns out to be 12.06

In [7]:
```
###### K-Nearest Neighbour(KNN) Regression in Python #######
from sklearn.neighbors import KNeighborsRegressor
RegModel = KNeighborsRegressor(n_neighbors=47)

#Printing all the parameters of KNN
print(RegModel)
```

```
KNeighborsRegressor(n_neighbors=47)
```

In [8]:
```
#Creating the model on Training Data
KNN=RegModel.fit(x_train,y_train)
prediction=KNN.predict(x_test)


#Measuring Goodness of fit in Training data
from sklearn import metrics
print('R2 Value:',metrics.r2_score(y_train, KNN.predict(x_train)))
```

```
R2 Value: 0.5918909982277922
```

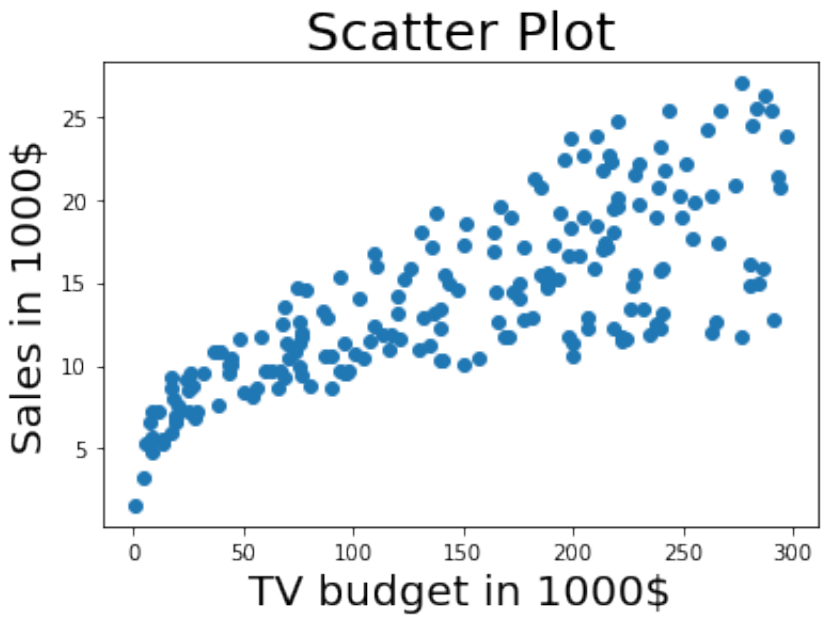
In [9]:
```

#Measuring accuracy on Testing Data
print('Accuracy',100- (np.mean(np.abs((y_test - prediction) / y_test)) * 100))
```

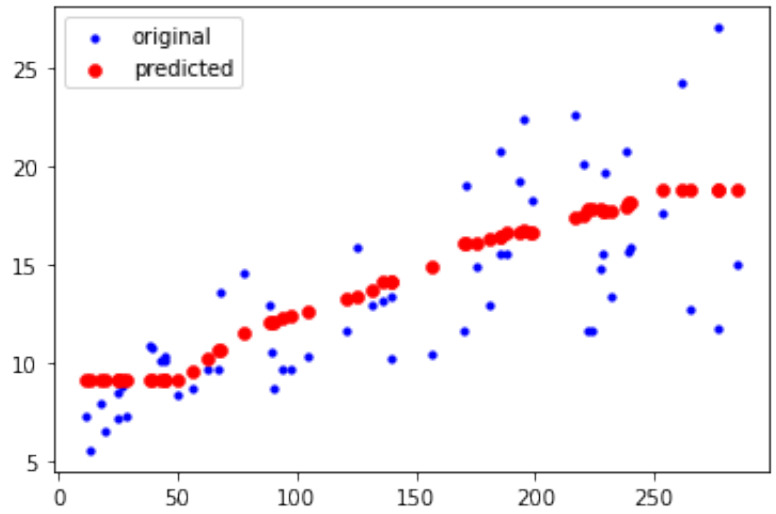
```
Accuracy 79.52138527277296
```

In [10]:
```
plt.scatter(x,y)
plt.xlabel('TV budget in 1000$' ,fontsize=20)
plt.ylabel('Sales in 1000$',fontsize=20)
plt.title('Scatter Plot',fontsize=25)
```

Out[10]: Text(0.5, 1.0, 'Scatter Plot')



In [11]:
```
plt.scatter(x_test, y_test, s=10, color="blue", label="original")
plt.scatter(x_test, prediction, lw=0.5, color="red", label="predicted")
plt.legend()
plt.show()
```



In [ ]: