In [2]:

```
# Author: GreHiDeL
#ML Reference: https://www.askpython.com/python/examples/python-predict-function
```

In [3]:

```
import pandas as pd
import os
import pandas as pd
from binance import Client
from sklearn.tree import DecisionTreeRegressor
from datetime import date,timedelta
```

In [4]:

```
# Collect data from Binance websockets unlike the CSV used in Part 1

link="https://www.cryptodatadownload.com/cdd/Binance_BTCUSDT_d.csv"
btc = pd.read_csv(link, skiprows=1, usecols=['date', 'close','open'])
```

In [5]:

```
today=date.today()
today_date=today.strftime("%Y-%m-%d")
#diff_date=timedelta(date=10)#Timedelta for difference between two dates
btc=btc.loc[:today_date]
btc.tail()
print(len(btc))
```

```
203
```

In [6]:

```
rows_count=len(btc.index)
#Conventional 70-30 rule for training and validation data set"""
train_rows=int(0.7*rows_count)
test_rows=int(0.3*rows_count)
data= btc.head(train_rows)
data_test=btc.tail(test_rows) # Validate the data model with 30% of the data

data

#print(data_test)
```

Out[6]:

	date	open	close
0	2022-05-02 00:00:00	38468.35	38698.97
1	2022-05-01 00:00:00	37630.80	38468.35
2	2022-04-30 00:00:00	38596.11	37630.80
3	2022-04-29 00:00:00	39742.06	38596.11
4	2022-04-28 00:00:00	39235.72	39742.07
...	...	...	...
137	2021-12-16 00:00:00	48864.98	47632.38
138	2021-12-15 00:00:00	48336.95	48864.98
139	2021-12-14 00:00:00	46702.76	48343.28

	date	open	close
140	2021-12-13 00:00:00	50053.90	46702.75
141	2021-12-12 00:00:00	49389.99	50053.90

142 rows × 3 columns

In [7]:

```
btc_open_data=data.open.values.reshape(-1,1)
btc_close_data=data.close.values.reshape(-1,1)
btc_close_test_data=data_test.close.values.reshape(-1,1)
print(btc_close_data)
```

```
[[38698.97]
 [38468.35]
 [37630.8 ]
 [38596.11]
 [39742.07]
 [39235.72]
 [38112.65]
 [40426.08]
 [39450.13]
 [39441.6 ]
 [39709.18]
 [40480.01]
 [41358.19]
 [41493.18]
 [40801.13]
 [39678.12]
 [40378.71]
 [40551.9 ]
 [39942.38]
 [41147.79]
 [40074.94]
 [39530.45]
 [42158.85]
 [42753.97]
 [42252.01]
 [43444.19]
 [43170.47]
 [45497.55]
 [46580.51]
 [46407.35]
 [45811.  ]
 [46283.49]
 [45510.34]
 [47067.99]
 [47434.8 ]
 [47122.21]
 [46827.76]
 [44511.27]
 [44313.16]
 [43991.46]
 [42882.76]
 [42364.13]
 [41002.25]
 [41262.11]
 [42201.13]
```

```
 [41757.51]
 [40917.9 ]
 [41114.  ]
 [39280.33]
 [39671.37]
 [37777.34]
 [38807.36]
 [38729.57]
 [39422.  ]
 [41941.71]
 [38730.63]
 [37988.  ]
 [38420.81]
 [39397.96]
 [39148.66]
 [42454.  ]
 [43892.98]
 [44421.2 ]
 [43160.  ]
 [37699.07]
 [39116.72]
 [39219.17]
 [38327.21]
 [37250.01]
 [38230.33]
 [37008.16]
 [38386.89]
 [40079.17]
 [39974.44]
 [40515.7 ]
 [43873.56]
 [44544.86]
 [42535.94]
 [42053.66]
 [42217.87]
 [42373.73]
 [43495.44]
 [44372.72]
 [44042.99]
 [43839.99]
 [42380.87]
 [41382.59]
 [41574.25]
 [37311.61]
 [36896.36]
 [38694.59]
 [38466.9 ]
 [37881.76]
 [38166.84]
 [37716.56]
 [37160.1 ]
 [36809.34]
 [36958.32]
 [36660.35]
 [36244.55]
 [35071.42]
 [36445.31]
 [40680.91]
 [41660.01]
 [42352.12]
```

```
 [42201.62]
 [43071.66]
 [43084.29]
 [43059.96]
 [42560.11]
 [43902.66]
 [42729.29]
 [41822.49]
 [41864.62]
 [41679.74]
 [41566.48]
 [43082.31]
 [43451.13]
 [45832.01]
 [46446.1 ]
 [47286.18]
 [47722.65]
 [46216.93]
 [47120.87]
 [46464.66]
 [47543.74]
 [50701.44]
 [50775.49]
 [50399.66]
 [50820.  ]
 [50838.81]
 [48588.16]
 [48889.88]
 [46914.16]
 [46681.23]
 [46834.48]
 [46131.2 ]
 [47632.38]
 [48864.98]
 [48343.28]
 [46702.75]
 [50053.9 ]]
```

In [8]:

```
#Decision Tree Regressor
data_model=DecisionTreeRegressor(max_depth=100).fit(btc_open_data,btc_close_data)
DTR_data_predict=data_model.predict(btc_close_test_data)
print(DTR_data_predict)
```

```
[47434.8  46464.66 50775.49 47543.74 50775.49 50053.9  50053.9  50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.   50820.   50820.   50820.   50820.
 50820.   50820.   50820.   50820.  ]
```

Technique 2 K-Nearest Neighbour Regression

In [9]:

```
#KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline
```

```python
from sklearn.preprocessing import StandardScaler # Scaling for the training data
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
from matplotlib import pyplot as plt
import numpy as np
```

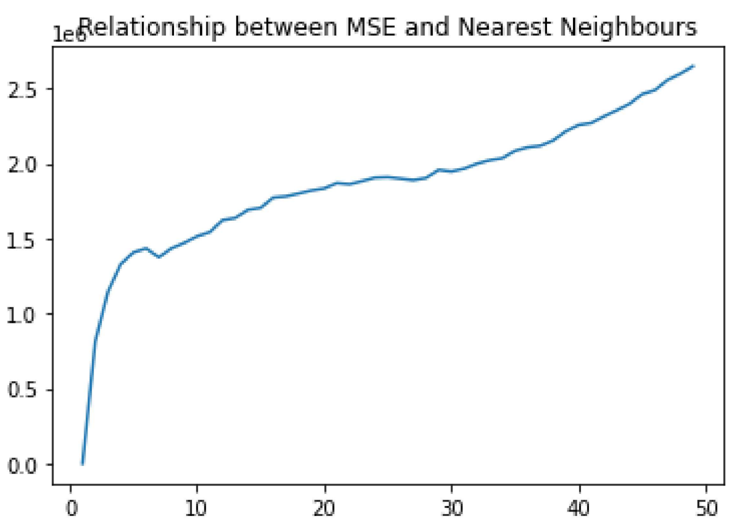
X= Training Data, Y= Target values

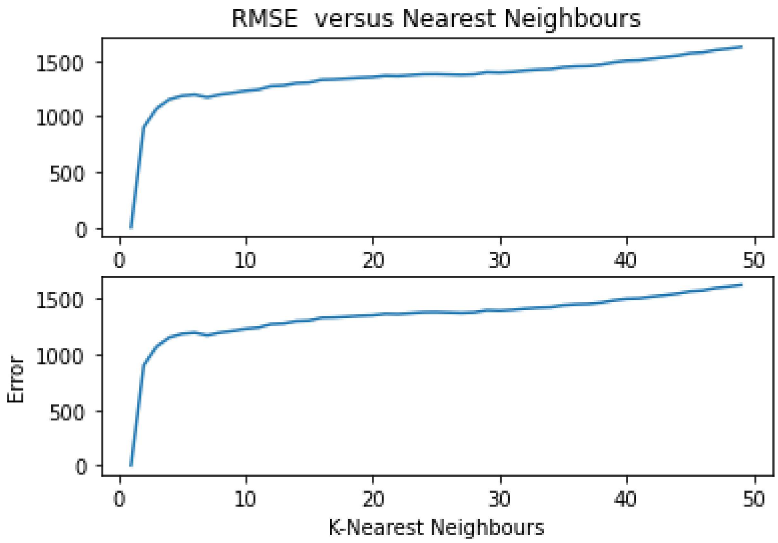
In [10]:

```python
# Choosing the K-Nearest Neighbour based on Loop
mse_error=[]
rmse_error=[]
x_range=range(1,50)# K-Nearest neighbour (range of 1-50)
for k in x_range:
    KNN_model = KNeighborsRegressor(n_neighbors=k).fit(btc_open_data,btc_close_data)
    closeData_predict=KNN_model.predict(btc_open_data)
    mse_error.append(mean_squared_error(btc_close_data,closeData_predict))
    rmse_error.append(sqrt(mean_squared_error(btc_close_data,closeData_predict)))
plt.title("Relationship between MSE and Nearest Neighbours")
plt.plot(x_range,mse_error, label='MSE')
figure, axis =plt.subplots(2)  # Minimum is two but we need one.
axis[0].plot(x_range,rmse_error)
axis[0].set_title("RMSE  versus Nearest Neighbours")
axis[1].plot(x_range,rmse_error)
axis[0].set_title("RMSE  versus Nearest Neighbours") # The other is redundant
plt.xlabel("K-Nearest Neighbours")
plt.ylabel("Error")
```

Out[10]:

```
Text(0, 0.5, 'Error')
```





The graph above helps in determining the nearest neighbours below.

So by observing the appropriate K-Nearest Neighbour the target price predictions can proceed normally

In [11]:

```
KNN_model = KNeighborsRegressor(n_neighbors=4).fit(btc_open_data,btc_close_data) # Fit(
closeData_KNN_predict = KNN_model.predict(btc_open_data)  # Y_predict=Predict (X)
print(closeData_KNN_predict)
```

```
[[38097.43   ]
 [39866.64   ]
 [37980.18   ]
 [39529.7925]
 [39842.6875]
 [38274.53   ]
 [39368.5975]
 [39256.6475]
 [39256.6475]
 [39529.7925]
 [39543.745  ]
 [41305.33   ]
 [41700.3475]
 [40515.0325]
 [39529.7925]
 [39368.5975]
 [39543.745  ]
 [40041.4375]
 [41056.665  ]
 [40041.4375]
 [39670.18   ]
 [41299.585  ]
 [43278.2325]
 [42105.465  ]
 [42788.06   ]
 [43125.8825]
 [44828.11   ]
```

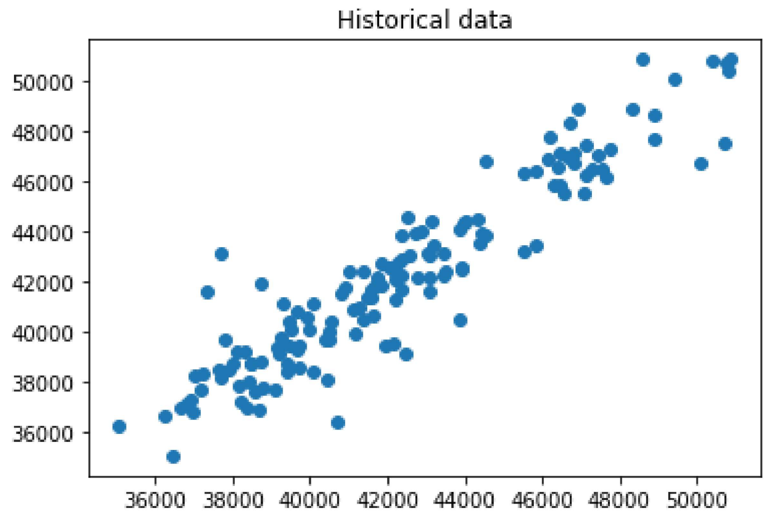
```
[46968.965 ]
[46336.0975]
[44828.11  ]
[46737.16  ]
[44828.11  ]
[47012.9875]
[46527.4875]
[46402.0425]
[47759.15  ]
[44522.435 ]
[44681.8625]
[43424.9975]
[43284.315 ]
[42650.1575]
[41245.48  ]
[40951.3775]
[41299.585 ]
[41868.9875]
[41633.18  ]
[41056.665 ]
[39598.4   ]
[39529.7925]
[39866.2775]
[38855.6925]
[38855.6925]
[39256.6475]
[41627.43  ]
[38855.6925]
[38578.7525]
[38097.43  ]
[39256.6475]
[38988.955 ]
[42437.845 ]
[42393.2   ]
[44681.8625]
[43125.8825]
[39866.64  ]
[38988.955 ]
[39842.6875]
[37866.335 ]
[38962.0875]
[38396.665 ]
[37516.96  ]
[38227.48  ]
[40041.4375]
[40041.4375]
[39543.745 ]
[42393.2   ]
[44522.435 ]
[42648.3675]
[40687.625 ]
[42105.465 ]
[42650.1575]
[42788.06  ]
[44681.8625]
[43424.9975]
[42393.2   ]
[42650.1575]
[41305.33  ]
[41275.3575]
```

```
 [38962.0875]
 [37377.845 ]
 [38819.0575]
 [38097.43  ]
 [38758.935 ]
 [38396.665 ]
 [39866.64  ]
 [38962.0875]
 [37059.8425]
 [37377.845 ]
 [36625.3625]
 [36462.5475]
 [36233.66  ]
 [36462.5475]
 [39572.91  ]
 [41401.965 ]
 [42650.1575]
 [41299.585 ]
 [42481.0125]
 [42481.0125]
 [42481.0125]
 [42664.035 ]
 [42393.2   ]
 [43278.2325]
 [41543.7275]
 [41543.7275]
 [41532.3125]
 [41275.3575]
 [42481.0125]
 [42788.06  ]
 [45744.1125]
 [46257.735 ]
 [46791.455 ]
 [46737.5075]
 [46737.16  ]
 [46402.0425]
 [46257.735 ]
 [46737.5075]
 [49866.21  ]
 [49866.21  ]
 [48930.855 ]
 [49866.21  ]
 [49866.21  ]
 [48981.0825]
 [49278.3125]
 [47050.915 ]
 [46969.3   ]
 [47759.15  ]
 [46737.16  ]
 [46737.5075]
 [49278.3125]
 [48981.0825]
 [46969.3   ]
 [48768.97  ]
 [48244.2975]]
```

In [12]:

```
plt.title("Historical data")
plt.scatter(btc_open_data,btc_close_data, label='Historical data')
```

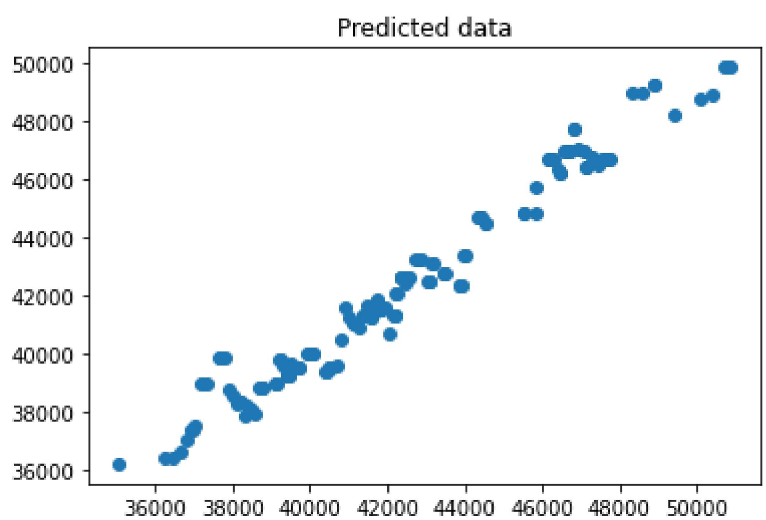
Out[12]: <matplotlib.collections.PathCollection at 0x1fd4460c310>



In [13]:

```
plt.title("Predicted data")
plt.scatter(btc_open_data,closeData_KNN_predict, label='Historical data')
```

Out[13]: <matplotlib.collections.PathCollection at 0x1fd3f9a3640>



In [14]:

```
MSE=mean_squared_error(btc_close_data,closeData_KNN_predict)
print("Mean Squared Error:" +str(MSE))
RMSE=sqrt(mean_squared_error(btc_close_data,closeData_KNN_predict))
print("RMSE:" +str(RMSE))
R2_Score=r2_score(btc_close_data,closeData_KNN_predict)
print("R2_Score:" +str(R2_Score))
```

```
Mean Squared Error:1328546.6784948057
RMSE:1152.625992460176
R2_Score:0.9025684746148769
```

In [15]:

```
# Target Price
Target_Price_data=btc_close_data[0]
Target_Price_predicted=closeData_KNN_predict[0]
print("The original Target_Price was :"+str(Target_Price_data))
print("The predicted Target_Price :"+str(Target_Price_predicted))
```

```
The original Target_Price was :[38698.97]
The predicted Target_Price :[38097.43]
```

In [23]:

```
import statistics as st
import numpy as np
def confidence_interval(array,confidence): #99% Confidence interval
    count=len(array)
    dataArray=np.array(array) #Convert your data into an array
    sum=np.sum(dataArray)
    average=sum/count
    stdev=np.std(dataArray)
    z_value=confidence     #Confidence value in Z_distribution e.g 4.417 for 99.999% Co
    sqrt_n=np.sqrt(count)
    variation=z_value*(stdev/sqrt_n)
    lower_CI=average-variation
    upper_CI=average+variation
    return lower_CI,upper_CI

print(confidence_interval(btc_close_data,4.417))
```

Out[23]:

```
(40887.498208823046, 43624.98404469808)
```

In [ ]:

In [ ]: