```
!pip·install·yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: yfinance in /usr/local/lib/python3.8/dist-packages (0.2.11)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.8/dist-packages (from yfinance) (2.3.4)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-packages (from yfinance) (4.9.2)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.8/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.8/dist-packages (from yfinance) (2.28.2)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: cryptography>=3.3.2 in /usr/local/lib/python3.8/dist-packages (from yfinance) (39.0.1)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.8/dist-packages (from yfinance) (4.11.2)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.8/dist-packages (from yfinance) (2022.7.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.8/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.8/dist-packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.8/dist-packages (from html5lib>=1.1->yfinance) (1.15.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (1.24.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2.1.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2022.12.7)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from cffi>=1.12->cryptography>=3.3.2->yfinance) (2.2
```

```
import seaborn as sns
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
```

```
# Get the data
btc= yf.Ticker("BTC-USD")
df = btc.history(period="max")
df.head(7)
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2014-09-17 00:00:00+00:00	465.864014	468.174011	452.421997	457.334015	21056800	0.0	0.0
2014-09-18 00:00:00+00:00	456.859985	456.859985	413.104004	424.440002	34483200	0.0	0.0
2014-09-19 00:00:00+00:00	424.102997	427.834991	384.532013	394.795990	37919700	0.0	0.0
2014-09-20 00:00:00+00:00	394.673004	423.295990	389.882996	408.903992	36863600	0.0	0.0
2014-09-21 00:00:00+00:00	408.084991	412.425995	393.181000	398.821014	26580100	0.0	0.0
2014-09-22 00:00:00+00:00	399.100006	406.915985	397.130005	402.152008	24127600	0.0	0.0
2014-09-23 00:00:00+00:00	402.092010	441.557007	396.196991	435.790985	45099500	0.0	0.0

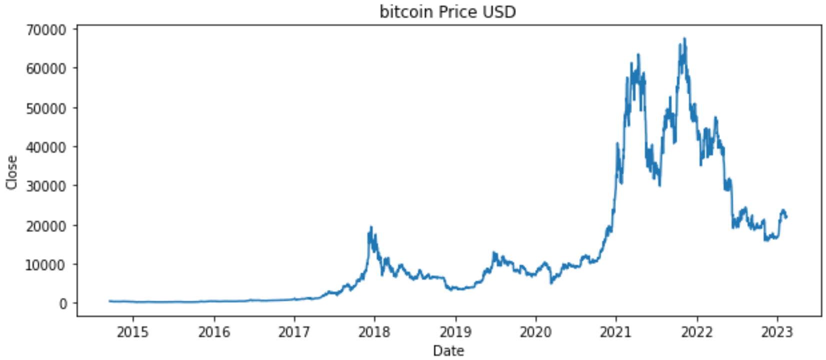
```
df.tail()
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2023-02-11 00:00:00+00:00	21651.841797	21891.410156	21618.449219	21870.875000	16356226232	0.0	0.0
2023-02-12 00:00:00+00:00	21870.902344	22060.994141	21682.828125	21788.203125	17821046406	0.0	0.0
2023-02-13 00:00:00+00:00	21787.000000	21898.414062	21460.087891	21808.101562	23918742607	0.0	0.0
2023-02-14 00:00:00+00:00	21801.822266	22293.140625	21632.394531	22220.804688	26792596581	0.0	0.0
2023-02-15 00:00:00+00:00	22206.128906	22206.128906	22086.865234	22133.824219	27030398976	0.0	0.0

```
df.shape
```

```
(3074, 7)
```

```
df.dropna()
plt.figure(figsize=(10, 4))
plt.title("bitcoin Price USD")
plt.xlabel("Date")
plt.ylabel("Close")
plt.plot(df["Close"])
plt.show()
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3074 entries, 2014-09-17 00:00:00+00:00 to 2023-02-15 00:00:00+00:00
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          3074 non-null   float64
 1   High          3074 non-null   float64
 2   Low           3074 non-null   float64
 3   Close         3074 non-null   float64
 4   Volume        3074 non-null   int64  
 5   Dividends     3074 non-null   float64
 6   Stock Splits  3074 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 256.7 KB
```

```
df.isna()
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2014-09-17 00:00:00+00:00	False	False	False	False	False	False	False
2014-09-18 00:00:00+00:00	False	False	False	False	False	False	False
2014-09-19 00:00:00+00:00	False	False	False	False	False	False	False
2014-09-20 00:00:00+00:00	False	False	False	False	False	False	False
2014-09-21 00:00:00+00:00	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
2023-02-11 00:00:00+00:00	False	False	False	False	False	False	False
2023-02-12 00:00:00+00:00	False	False	False	False	False	False	False
2023-02-13 00:00:00+00:00	False	False	False	False	False	False	False
2023-02-14 00:00:00+00:00	False	False	False	False	False	False	False
2023-02-15 00:00:00+00:00	False	False	False	False	False	False	False

3074 rows × 7 columns

```
#to get the count of null values in each column
df.isna().sum()
```

```
Open            0
High            0
Low             0
Close           0
```

```
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

```
#to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values
df.describe()
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
count	3074.000000	3074.000000	3074.000000	3074.000000	3.074000e+03	3074.0	3074.0
mean	12978.747174	13303.421306	12617.983413	12984.267020	1.644408e+10	0.0	0.0
std	16101.305842	16511.990523	15627.118783	16097.065496	1.986080e+10	0.0	0.0
min	176.897003	211.731003	171.509995	178.102997	5.914570e+06	0.0	0.0
25%	701.501984	707.074997	684.512238	701.905762	1.072450e+08	0.0	0.0
50%	7241.401611	7388.366455	7081.487305	7247.584473	7.920519e+09	0.0	0.0
75%	17331.447266	17830.788086	16920.371582	17402.766602	2.785963e+10	0.0	0.0
max	67549.734375	68789.625000	66382.062500	67566.828125	3.509679e+11	0.0	0.0

```
#Splitting the dataset
X = df.drop( ['Close'], axis = 1)
Y = df['Close']
```

```
#Splitting the data as the trainning & testing as 70-30
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=2)
```

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 923 entries, 2017-04-03 00:00:00+00:00 to 2015-08-27 00:00:00+00:00
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Open          923 non-null    float64
 1   High          923 non-null    float64
 2   Low           923 non-null    float64
 3   Volume        923 non-null    int64  
 4   Dividends     923 non-null    float64
 5   Stock Splits  923 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 50.5 KB
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingRegressor
```

```
knr = KNeighborsRegressor(n_neighbors=2)
knr.fit(X_train, y_train)
y_pred_knr = knr.predict(X_test)
mse_knr=metrics.mean_squared_error(y_test,y_pred_knr)
print(mse_knr)
r2knr = metrics.r2_score(y_test, y_pred_knr)
print(r2knr)
```

```
154395041.4525137
0.42415532609805706
```

```
rf = RandomForestRegressor()
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
mse_rf=metrics.mean_squared_error(y_test,y_pred_rf)
print(mse_rf)
r2rf = metrics.r2_score(y_test, y_pred_rf)
print(r2rf)
```

```
178727.44799469842
0.9993334031453368
```

```
dt = tree.DecisionTreeRegressor()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
mse_dt=metrics.mean_squared_error(y_test,y_pred_dt)
print(mse_dt)
r2dt = metrics.r2_score(y_test, y_pred_dt)
print(r2dt)
```

```
328218.0040770396
0.9987758506507178
```

```
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
Y_pred_gbr = gbr.predict(X_test)
mse_gbr=metrics.mean_squared_error(y_test,Y_pred_gbr)
print(mse_gbr)
r2gbr = metrics.r2_score(y_test, Y_pred_gbr)
print(r2gbr)
```

```
216451.98614230033
0.9991927025492339
```

```
print("*"*10, "R2 score", "*"*10)

print("-"*30)
print("K nearest neighbors: ", r2knr)
print("K nearest neighbors: ",mse_knr)
print("-"*30)


print("-"*30)
print("random forest: ", r2rf)
print("random forest: ",mse_rf)
print("-"*30)


print("-"*30)
print("decision tree: ", r2dt)
print("decision tree: ",mse_dt)
print("-"*30)


print("-"*30)
print("gradient boosting: ", r2gbr)
print("gradient boosting: ",mse_gbr)
print("-"*30)
```

```
********** R2 score **********
------------------------------
K nearest neighbors:  0.42415532609805706
K nearest neighbors:  154395041.4525137
------------------------------
------------------------------
random forest:  0.9993334031453368
random forest:  178727.44799469842
------------------------------
------------------------------
decision tree:  0.9987758506507178
decision tree:  328218.0040770396
------------------------------
------------------------------
gradient boosting:  0.9991927025492339
gradient boosting:  216451.98614230033
------------------------------
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split


df['Price'] = df['Close']

# Add the year, month, and day columns to the dataframe
df['Year'] = df.index.year
df['Month'] = df.index.month
df['Day'] = df.index.day
# Split the data into a training set and a testing set
```

```
x_train, x_test, y_train,y_test = train_test_split(df.drop('Price', axis=1), df['Price'], test_size=0.2)

# Train a linear regression model on the training data
model = RandomForestRegressor()
model.fit(x_train[['Year', 'Month', 'Day']], y_train)
# Evaluate the model on the testing data
print(f"Accuracy: {model.score(x_test[['Year', 'Month', 'Day']], y_test)}")

# Use the trained model to make predictions about future prices
future_prices = model.predict(np.array([[2023, 2, 15]]))
print(f"Predicted price: {future_prices[0]}")
```

```
Accuracy: 0.9981630412918572
Predicted price: 22020.4910546875
```

```
str = ['Siddhant Bhalke']
print("The str  ")
```