```
# DOGE COIN PREDICTION
```

```
import numpy as np
import pandas as pd
```

```
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed current browser session. Please rerun this cell to enable.

```
Saving DOGE-INR.csv to DOGE-INR.csv
{'DOGE-INR.csv': b'Date,Open,High,Low,Close,Adj Close,Volume\n2020-11-15,0.206694,0.2
```

```
df = pd.read_csv('DOGE-INR.csv')
df = df.set_index(pd.DatetimeIndex(df['Date'].values))
df
```

	Date	Open	High	Low	Close	Adj Close	
2020-11-15	2020-11-15	0.206694	0.209708	0.204163	0.206873	0.206873	1873
2020-11-16	2020-11-16	0.206873	0.223988	0.205262	0.214756	0.214756	3874
2020-11-17	2020-11-17	0.216400	0.221485	0.214734	0.218885	0.218885	3476
2020-11-18	2020-11-18	0.218868	0.223266	0.212416	0.215966	0.215966	3671
2020-11-19	2020-11-19	0.215972	0.219532	0.212982	0.215717	0.215717	2988
...	...	...	...	...	...	...	
2021-05-11	2021-05-11	33.110958	40.095493	32.710129	36.337738	36.337738	1068856
2021-05-12	2021-05-12	36.228504	38.192429	28.395218	28.395218	28.395218	635236
2021-05-13	2021-05-13	28.896252	38.151993	26.317286	36.015686	36.015686	1370723
2021-05-14	2021-05-14	35.773029	43.211235	34.124741	41.007305	41.007305	1499743
2021-05-15	2021-05-15	40.769993	41.144543	37.930008	37.930008	37.930008	1147925

182 rows × 7 columns

```
df = df[['Close']]
df
```

	Close
2020-11-15	0.206873
2020-11-16	0.214756
2020-11-17	0.218885
2020-11-18	0.215966
2020-11-19	0.215717
...	...
2021-05-11	36.337738

```
prediction_days = 1
df['Prediction'] = df[['Close']].shift(-prediction_days)
df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
```

	Close	Prediction
2020-11-15	0.206873	0.214756
2020-11-16	0.214756	0.218885
2020-11-17	0.218885	0.215966
2020-11-18	0.215966	0.215717
2020-11-19	0.215717	0.221327
...	...	...
2021-05-11	36.337738	28.395218
2021-05-12	28.395218	36.015686
2021-05-13	36.015686	41.007305
2021-05-14	41.007305	37.930008
2021-05-15	37.930008	NaN

182 rows × 2 columns

```
X = np.array(df.drop(['Prediction'], 1))
X = X[:len(df) -prediction_days -1]
print(X)
```

```
[[ 0.206873]
 [ 0.214756]
 [ 0.218885]
 [ 0.215966]
 [ 0.215717]
 [ 0.221327]
 [ 0.260058]
 [ 0.246872]
 [ 0.270841]
 [ 0.310908]
 [ 0.274996]
 [ 0.239946]
 [ 0.242157]
 [ 0.251833]
 [ 0.254961]
 [ 0.262768]
 [ 0.245667]
 [ 0.248565]
 [ 0.253435]
 [ 0.243396]
 [ 0.250684]
 [ 0.250074]
 [ 0.246698]
 [ 0.2344  ]
 [ 0.233124]
 [ 0.229096]
 [ 0.227025]
 [ 0.233241]
 [ 0.240089]
 [ 0.237955]
 [ 0.235707]
 [ 0.251459]
 [ 0.274252]
 [ 0.283992]
 [ 0.289037]
 [ 0.340359]
 [ 0.353963]
 [ 0.334126]
 [ 0.278204]
 [ 0.336195]
 [ 0.337048]
 [ 0.33055 ]
 [ 0.335281]
 [ 0.337702]
 [ 0.329478]
 [ 0.339271]
 [ 0.342073]
 [ 0.415527]
 [ 0.775838]
 [ 0.714098]
 [ 0.713881]
 [ 0.725858]
 [ 0.765446]
 [ 0.715538]
 [ 0.722521]
 [ 0.748031]
 [ 0.723264]
 [ 0.649504]
 [ 0.589769]
```

```
y = np.array(df['Prediction'])
y = y[:-prediction_days - 1]
print(y)
```

```
[ 0.214756  0.218885  0.215966  0.215717  0.221327  0.260058  0.246872
  0.270841  0.310908  0.274996  0.239946  0.242157  0.251833  0.254961
  0.262768  0.245667  0.248565  0.253435  0.243396  0.250684  0.250074
```

```
   0.246698  0.2344    0.233124  0.229096  0.227025  0.233241  0.240089
   0.237955  0.235707  0.251459  0.274252  0.283992  0.289037  0.340359
   0.353963  0.334126  0.278204  0.336195  0.337048  0.33055   0.335281
   0.337702  0.329478  0.339271  0.342073  0.415527  0.775838  0.714098
   0.713881  0.725858  0.765446  0.715538  0.722521  0.748031  0.723264
   0.649504  0.589769  0.630459  0.686812  0.685452  0.677528  0.663181
   0.671228  0.665918  0.660563  0.595775  0.62221   0.626271  0.637041
   0.611707  0.601925  0.546444  0.915767  3.43842   2.054217  2.711994
   2.550038  2.298398  2.704501  3.889393  3.416299  4.192222  5.734924
   5.748727  5.104978  5.307402  5.066114  5.086341  4.823641  4.539874
   4.110701  3.903513  3.594242  4.323034  4.000434  3.946707  4.062029
   3.892825  3.430348  4.094377  3.669811  3.715136  3.685921  3.541098
   3.709454  3.683102  3.693224  3.660217  3.630395  3.731076  3.814438
   4.542781  4.220882  4.071832  4.061831  4.023272  4.538134  4.259066
   4.139745  4.249959  4.18389   4.17002   4.225426  4.271229  4.144476
   3.978162  3.888802  3.755918  3.735831  3.908651  3.939478  3.890313
   3.931832  3.958744  3.927914  4.542806  4.231146  4.094653  4.212069
   4.374503  4.736458  4.391816  4.581807  4.609608  4.771112  5.578495
   5.306897  7.025336  9.117004 13.615935 27.280514 21.186237 23.895748
  30.499327 24.106075 23.151365 19.596518 18.605471 20.230602 18.800257
  20.250553 20.29851  24.053448 22.614897 25.011761 29.118603 27.863157
  32.614586 39.93919  48.532417 42.756138 50.187649 46.648445 41.780777
  33.072441 36.337738 28.395218 36.015686 41.007305]
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)
```

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators =2, random_state = 587)
forest.fit(x_train, y_train)
print(forest.score(x_test, y_test))
```

```
0.8032434714110614
```

```
prediction = forest.predict(x_test)
print(prediction)
print()
print(y_test)
```

```
[ 3.669811   0.6120675 50.187649   3.710876   0.312616   4.434337
  2.6272695  0.2344     0.342073   3.731076   0.342073   3.6006575
  3.76686    3.946707   5.104978   0.26851    5.578495  22.097129
  3.76686    5.578495   3.669811   0.7198695 23.813329   0.260058
 24.106075   7.110991   2.6272695  0.274252   3.715136  24.106075
  0.244894   0.719401   0.289037   3.735831  22.097129   4.259066 ]

[ 4.212069  0.630459 33.072441  3.814438  0.283992  4.771112  2.298398
  0.248565  0.353963  4.323034  0.415527  3.693224  3.931832  4.538134
  5.748727  0.243396  4.539874 18.800257  3.755918  4.391816  3.903513
  0.722521 19.596518  0.215966 36.337738 13.615935  3.889393  0.254961
  3.630395 39.93919   0.242157  0.714098  0.340359  3.908651 24.053448
  4.110701]
```

```
temp_df = df[:prediction_days]
x_val = temp_df.tail(1)['Close'][0]
print(x_val)
```

```
0.206873
```

```
prediction = forest.predict([[x_val]])
print('the price of Dogecoin in', prediction_days, 'day(s) is predicted to be', prediction
print('the actual price was', temp_df.tail(1)['Prediction'][0])
```

```
the price of Dogecoin in 1 day(s) is predicted to be [0.2168205]
the actual price was 0.214756
```

```
from sklearn.svm import SVR
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.00001)
svr_rbf.fit(x_train, y_train)
```

```
SVR(C=1000.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=1e-05,
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
svr_rbf_confidence = svr_rbf.score(x_test, y_test)
print("svr_rbf accuracy:", svr_rbf_confidence)
```

```
svr_rbf accuracy: 0.9509470567271271
```