

ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ ΠΡΟΟΔΟΥ

ΠΡΑΚΤΙΚΗΣ ΑΣΚΗΣΗΣ

Όνομα ασκούμενου: ΚΑΛΥΒΑΣ ΝΙΚΟΛΑΟΣ

Αρ. Μητρώου: 3130273

Φορέας Υλοποίησης Π.Α.: ΚΜ QUBE ASSET MANAGEMENT

Επιβλέπων Καθηγητής: ΑΝΤΩΝΗΣ ΔΗΜΑΚΗΣ

Εργασιακός Επιβλέπωντας: ΓΡΑΦΕΙΟ ΠΡΑΚΤΙΚΗΣ ΑΣΚΗΣΗΣ Ο.Π.Α

Ακαδημαϊκό Εξάμηνο: Χειμώνας 2020 - Ακ. Έτος Εξάμηνο 2020 - 2021

Αθήνα, 19/3/2021

Η εταιρεία:

Η KM Cube είναι μια εταιρεία διαχείρισης περιουσιακών στοιχείων που προσφέρει εξελιγμένες επενδυτικές υπηρεσίες σε ιδιώτες επενδυτές και ιδρύματα. Η ομάδα μας στελεχώνεται από έμπειρους επαγγελματίες με πολυετή σταδιοδρομία σε ιδρύματα μεγάλου κύρους και ισχυρές αναλυτικές δεξιότητες. Η εταιρεία μας είναι το αποτέλεσμα μιας ομαδικής προσπάθειας ανθρώπων με ταλέντο και μεγάλη

εμπειρία στην διαχείριση περιουσίας. Δραστηριοποιούμαστε στην Ελλάδα

και στην Ευρωπαϊκή Ένωση στον τομέα της συμβουλευτικής και εν λευκώ

διαχείρισης χαρτοφυλακίου. Το πλεονέκτημα μας είναι η χρήση αλγοριθμικών μοντέλων με αντιστάθμιση κινδύνου, έτσι ώστε να μεγιστοποιήσουμε τις αποδόσεις και την ασφάλεια για τους πελάτες μας.

Ο Ρόλος μου:

Βοηθός έρευνας και ανάπτυξης λογισμικού για πρόβλεψη χρηματοοικονομικών μεταβλητών με την χρήση μηχανικής εκμάθησης (Machine Learning) σε κρυπτονομίσματα. Αυτή την θέση εργασίας έχω στο περιβάλλον της πρακτικής μου στην KM QUBE ASSET MANAGEMENT , μαζί με τον Κ. Κώστα Μεταξά. Η πρακτική ξεκίνησε στις 16/11/2020 και πλέον ένα μήνα αργότερα

μπορώ να αναφέρω με σιγουριά, πως με τον Κ. Κώστα έχουμε πάρα πολύ καλή επικοινωνία, ο Κ. Κώστας είναι απίστευτα μεταδοτικός, έχει πολύ υψηλές γνώσεις προγραμματισμού, γνωρίζει ακριβώς ποιες είναι οι ανάγκες της αγοράς , χρησιμοποιεί τις πιο πρόσφατες μορφές τεχνολογίας και εργαλείων της εποχής μας, είναι καινοτόμος και τον περιβάλλει πάντα ένα κλίμα εφευρετικότητας και εμπιστοσύνης.

Λόγο πανδημίας του covid-19 η πρακτική πραγματοποιείται με την μορφή τηλεργασίας, επικοινωνούμε με τον Κ. Κώστα και συζητάμε για το project, πολλές φορές γίνεται share screen και μου εξηγεί την λειτουργία διαφόρων εργαλείων, άλλες φορές παράγει κάποιο κώδικα και τον επεξηγεί, άλλες στιγμές θα λύσει τις απορίες μου και πάντα στο τέλος της τηλεργασίας καταλήγουμε στο τι να μελετήσω ή στο τι να προγραμματίσω , μέχρι την επόμενη τηλεργασία.

Η ομάδα μας έχει στόχο να απορροφήσει όσο τον δυνατό περισσότερη γνώση γίνεται στα κρυπτονομίσματα και τις συμπεριφορές τους και να παρέχει στον χρήστη ένα web application το οποίο με την χρήση φίλτρων και μηχανικής εκμάθησης να προτείνει σε ποιο ή ποια κρυπτονομίσματα να επενδύσει την συγκεκριμένη χρονική στιγμή.

Μέρος Πρώτο

Χρήση εργαλείων στο project:

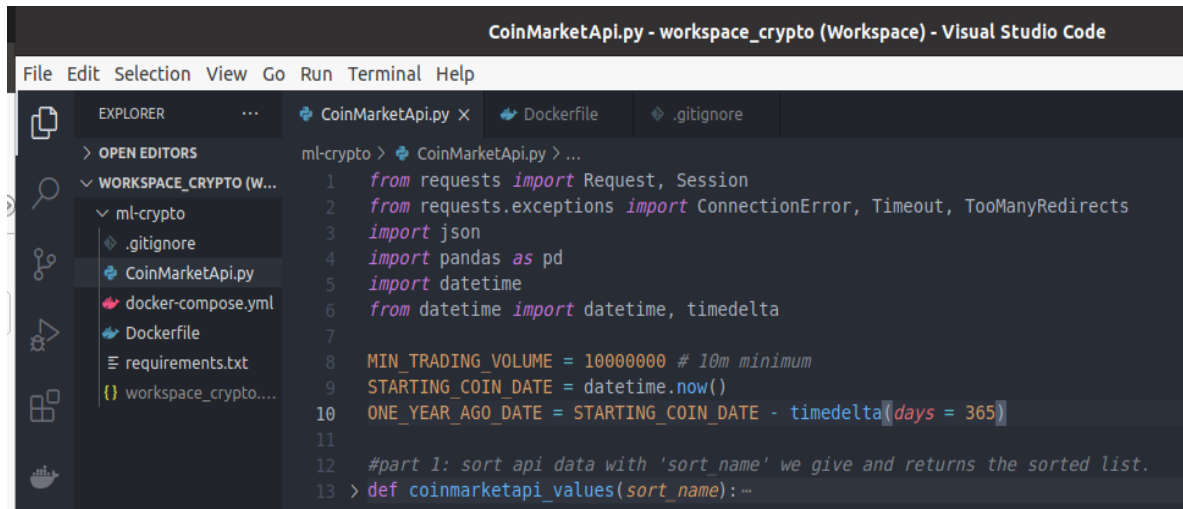
- 1. Λειτουργικό ubuntu**
- 2. Visual studio code editor**
- 3. Χρήση Git**
- 4. Χρήση Docker**
- 5. Programme language Python**

Εργαλεία και βιβλιοθήκες προγραμματισμού:

- 1. Χρήση Coin Market Api (with requests library)**
- 2. Χρήση pandas software library for Python**
- 3. Χρήση json file,datetime**

Screenshots of the above services:

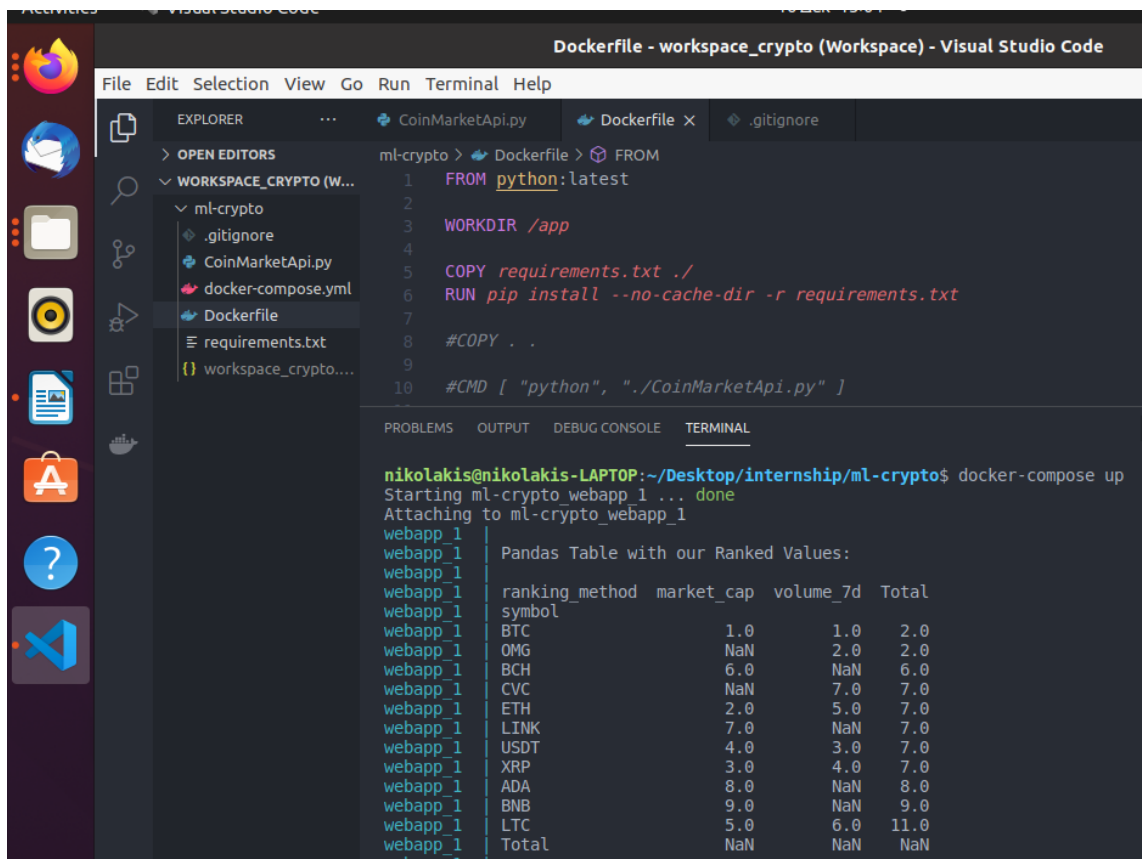
Visual studio code editor



The screenshot shows the Visual Studio Code editor with the file 'CoinMarketApi.py' open. The Explorer sidebar on the left shows the project structure with files like '.gitignore', 'CoinMarketApi.py', 'docker-compose.yml', 'Dockerfile', 'requirements.txt', and 'workspace_crypto...'. The main editor area displays the Python code for the CoinMarketApi.py file, which includes imports for requests, json, pandas, and datetime, and a function definition for 'coinmarketapi_values'.

```
1 from requests import Request, Session
2 from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
3 import json
4 import pandas as pd
5 import datetime
6 from datetime import datetime, timedelta
7
8 MIN_TRADING_VOLUME = 10000000 # 10m minimum
9 STARTING_COIN_DATE = datetime.now()
10 ONE_YEAR_AGO_DATE = STARTING_COIN_DATE - timedelta(days = 365)
11
12 #part 1: sort api data with 'sort_name' we give and returns the sorted list.
13 > def coinmarketapi_values(sort_name):...
```

Docker:

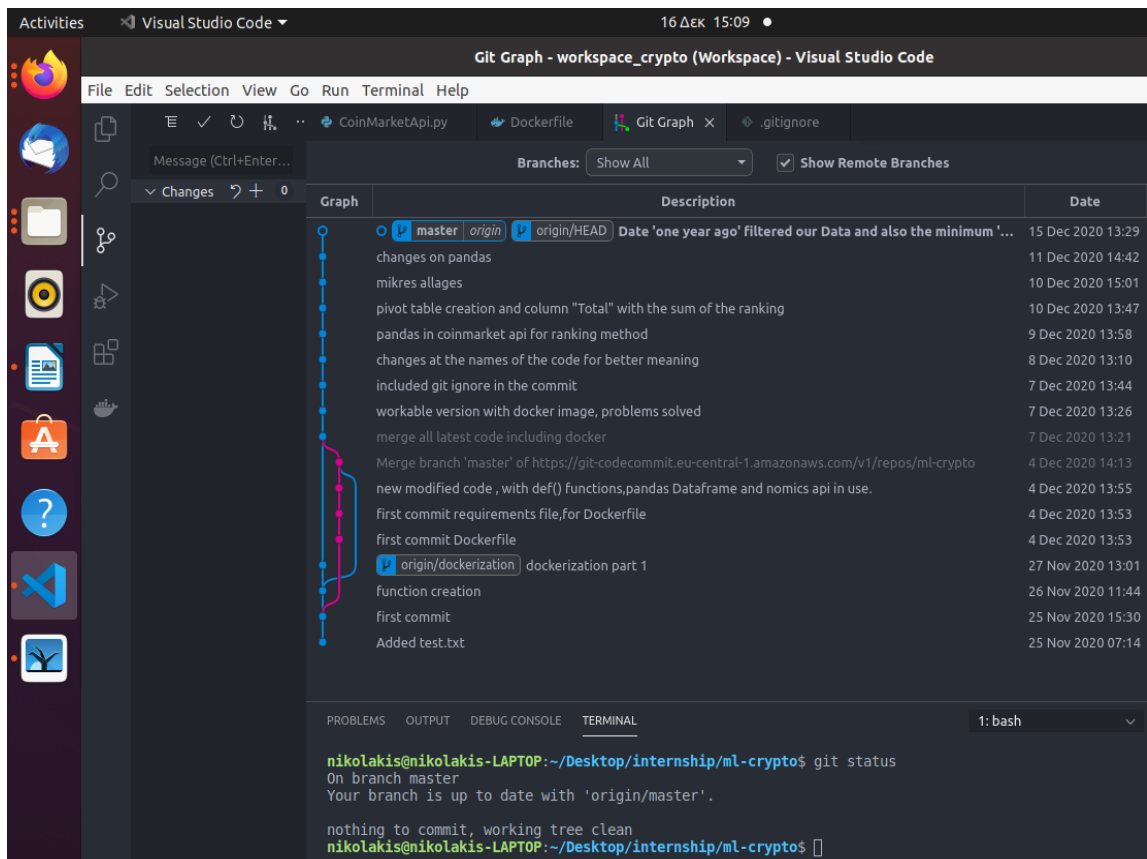


The screenshot shows the Visual Studio Code editor with the 'Dockerfile' file open. The Explorer sidebar on the left shows the project structure. The main editor area displays the Dockerfile content, which includes instructions for building a Docker image from a Python base image, setting the working directory, copying requirements, and running the application. Below the editor, the terminal shows the output of the 'docker-compose up' command, indicating that the application is running successfully and displaying a table of ranked values.

```
1 FROM python:latest
2
3 WORKDIR /app
4
5 COPY requirements.txt ./
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 #COPY . .
9
10 #CMD [ "python", "./CoinMarketApi.py" ]
```

nikolakis@nikolakis-LAPTOP:~/Desktop/internship/ml-crypto\$ docker-compose up
Starting ml-crypto_webapp_1 ... done
Attaching to ml-crypto_webapp_1
webapp_1 | Pandas Table with our Ranked Values:
webapp_1 | ranking_method market_cap volume_7d Total
webapp_1 | symbol
webapp_1 | BTC 1.0 1.0 2.0
webapp_1 | OMG NaN 2.0 2.0
webapp_1 | BCH 6.0 NaN 6.0
webapp_1 | CVC NaN 7.0 7.0
webapp_1 | ETH 2.0 5.0 7.0
webapp_1 | LINK 7.0 NaN 7.0
webapp_1 | USDT 4.0 3.0 7.0
webapp_1 | XRP 3.0 4.0 7.0
webapp_1 | ADA 8.0 NaN 8.0
webapp_1 | BNB 9.0 NaN 9.0
webapp_1 | LTC 5.0 6.0 11.0
webapp_1 | Total NaN NaN NaN

Presentation of Git tool, with Git Graph in Vs code:



The screenshot shows the Visual Studio Code interface with the Git Graph extension active. The top bar indicates the workspace is 'workspace_crypto (Workspace)'. The left sidebar shows the Explorer, Search, and Source Control views. The main editor area displays the Git Graph interface, which includes a commit history graph on the left and a list of commits in the center. The commits are listed in descending order of date, starting from 15 Dec 2020 13:29. The bottom panel shows the Terminal view with the output of a 'git status' command.

Graph	Description	Date
o master origin origin/HEAD	Date 'one year ago' filtered our Data and also the minimum '...	15 Dec 2020 13:29
	changes on pandas	11 Dec 2020 14:42
	mikres allages	10 Dec 2020 15:01
	pivot table creation and column "Total" with the sum of the ranking	10 Dec 2020 13:47
	pandas in coinmarket api for ranking method	9 Dec 2020 13:58
	changes at the names of the code for better meaning	8 Dec 2020 13:10
	included git ignore in the commit	7 Dec 2020 13:44
	workable version with docker image, problems solved	7 Dec 2020 13:26
	merge all latest code including docker	7 Dec 2020 13:21
	Merge branch 'master' of https://git-codecommit.eu-central-1.amazonaws.com/v1/repos/ml-crypto	4 Dec 2020 14:13
	new modified code , with def() functions,pandas Dataframe and nomics api in use.	4 Dec 2020 13:55
	first commit requirements file,for Dockerfile	4 Dec 2020 13:53
	first commit Dockerfile	4 Dec 2020 13:53
o origin/dockerization	dockerization part 1	27 Nov 2020 13:01
	function creation	26 Nov 2020 11:44
	first commit	25 Nov 2020 15:30
	Added test.txt	25 Nov 2020 07:14

```
nikolakis@nikolakis-LAPTOP:~/Desktop/internship/ml-crypto$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
nikolakis@nikolakis-LAPTOP:~/Desktop/internship/ml-crypto$
```

Presentation of Python code with Api ,json file and Pandas Library tools for processing our data:

```
CoinMarketApi.py • Dockerfile Git Graph .gitignore
ml-crypto > CoinMarketApi.py > ...

12 | #part 1: sort api data with 'sort_name' we give and returns a pandas table with all data in it..
13 | def coinmarketapi_values(sort_name):
14 |     url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
15 |     parameters = {
16 |         'start': '1',
17 |         'limit': '10',
18 |         'convert': 'USD',
19 |         'sort': sort_name
20 |     }
21 |     headers = {
22 |         'Accepts': 'application/json',
23 |         'X-CMC_PRO_API_KEY': '5044fdd3-f5fc-4a53-bdaa-d13c1ce5d4a0',
24 |     }
25 |
26 |     session = Session()
27 |     session.headers.update(headers)
28 |
29 |     try:
30 |         response = session.get(url, params=parameters)
31 |         data = json.loads(response.text)
32 |         tbl = pd.json_normalize(data["data"])
33 |
34 |         # now filter tbl to exclude coins with one year old in the Market.
35 |         tbl["date_added"] = pd.to_datetime(tbl["date_added"]).apply(lambda x: x.replace(tzinfo=None))
36 |         tbl = tbl[ tbl["date_added"] < ONE_YEAR_AGO_DATE ]
37 |
38 |         # now filter tbl to exclude coins with small volume
39 |         tbl = tbl[ tbl["quote.USD.volume_24h"] > MIN_TRADING_VOLUME ]
40 |         tbl["rank"] = range(1, tbl.shape[0]+1, 1) #creates column rank
41 |         tbl["ranking_method"] = sort_name #creates column ranking method
42 |     except (ConnectionError, Timeout, TooManyRedirects) as e:
43 |         print(e)
44 |     return(tbl)
45 |
46 | market_cap = coinmarketapi_values('market_cap')
47 | percent_change_7d = coinmarketapi_values('percent_change_7d')
48 | volume_7d = coinmarketapi_values('volume_7d')
49 |
50 | rnk1 = market_cap[["symbol","name","rank", "ranking_method"] ]
51 | rnk2 = percent_change_7d[["symbol","name","rank", "ranking_method"] ]
52 | rnk3 = volume_7d[["symbol","name","rank", "ranking_method"] ]
53 |
54 | tbl_rnk = rnk1.append(rnk2).append(rnk3)
55 | table_pivot = tbl_rnk.pivot_table(values='rank', index='symbol', columns='ranking_method')
56 | table_pivot = table_pivot.reset_index()
57 | table_pivot = table_pivot.rename(columns={'symbol': 'symbol', 'ranking_method': 'ranking_method'})
```

Το project μας, χωρίζεται σε δύο μέρη.

Το πρώτο μέρος αποτελεί την αξιολόγηση των κρυπτονομισμάτων με βάση πέντε βασικά κριτήρια:

1. Ο χρόνος από την ημέρα εκκίνησης του κρυπτονομίσματος, να μην είναι μικρότερος του ενός έτους.
2. Το μεγαλύτερο Market Cap ($\text{Market Cap} = \text{Τιμή} \times \text{αριθμός νομισμάτων που υπάρχουν}$)
3. Το μεγαλύτερο Ποσοστό αλλαγής της τιμής, τις τελευταίες 7 ημέρες.
4. Την μεγαλύτερη ένταση της τιμής τις τελευταίες 24 ώρες. (Volume 24 hours)
5. Επίσης το Volume 24 hours > 10 εκατομμύρια.

Τα 5 προηγούμενα κριτήρια λοιπόν, τα υλοποιούμε με την χρήση της ιστοσελίδας CoinMarketCap και το API του.

Όσον αφορά την ιστοσελίδα, μας παρέχει διάφορα δεδομένα σχετικά με πολλά αναφερόμενα νομίσματα, όπως η τιμή τους, η διαθέσιμη προσφορά, ο όγκος συναλλαγών τις τελευταίες 24 ώρες ή η κεφαλαιοποίηση της αγοράς.

Όσον αφορά το API, είναι μια σουίτα υψηλής απόδοσης RESTful JSON endpoints που έχουν σχεδιαστεί ειδικά για να ανταποκρίνονται στις κρίσιμες αποστολές των προγραμματιστικών εφαρμογών, των επιστημονικών δεδομένων και των επιχειρησιακών πλατφορμών.

Αυτή η αναφορά API περιλαμβάνει όλες τις τεχνικές τεκμηρίωσης που χρειάζονται οι προγραμματιστές για να ενσωματώσουν εφαρμογές και πλατφόρμες τρίτων, [CoinMarketCap API FAQ](#).

Παρακάτω γίνεται αναφορά του κώδικα σε python:

Κλήση του API από CoinMarketCap με ταξινομημένα τα δεδομένα, με βάση το μεγαλύτερο Market Cap και έπειτα φιλτράρουμε το $\text{Volume_24h} > 10.000.000$ και την ημερομηνία δημιουργίας του κρυπτονομίσματος, έτσι ώστε να είναι ένα χρόνο μεγαλύτερη.


```

MIN_TRADING_VOLUME = 10000000 # 10m minimum
STARTING_COIN_DATE = datetime.now()
ONE_YEAR_AGO_DATE = STARTING_COIN_DATE - timedelta(days = 365)
#part 1: sort api data with 'sort_name' we give and returns the
sorted table.
def coinmarketapi_values(sort_name):
    url =
'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
    parameters = {
        'start':'1',
        'limit':'100',
        'convert':'USD',
        'sort': sort_name
    }
    headers = {
        'Accepts': 'application/json',
        'X-CMC_PRO_API_KEY': '5044fdd3-f5fc-4a53-bdaa-d13c1ce5d4a0',
    }

    session = Session()
    session.headers.update(headers)

    try:
        response = session.get(url, params=parameters)
        data = json.loads(response.text)
        tbl = pd.json_normalize(data["data"])

        # now filter tbl to exclude coins with one year old in the
Market.
        tbl["date_added"] =
pd.to_datetime(tbl["date_added"]).apply(lambda x:
x.replace(tzinfo=None))
        tbl = tbl[ tbl["date_added"] < ONE_YEAR_AGO_DATE ]
        # now filter tbl to exclude coins with small volume
        tbl = tbl[ tbl["quote.USD.volume_24h"] > MIN_TRADING_VOLUME ]

    except (ConnectionError, Timeout, TooManyRedirects) as e:
        print(e)
    return(tbl)

```

Χρησιμοποιούμε Pandas DataFrame για την δημιουργία του 'table_rank', με τελευταία στήλη του table, τη στήλη 'Total', όπου και το αποτέλεσμα των αξιολογήσεων μας, με το πρώτο ως καλύτερο και ούτω καθεξής.

Η διαδικασία αξιολόγησης που ακολουθούμε είναι απλή.

Βαθμολογούμε με ένα αριθμό το κάθε κρυπτονόμισμα με βάση το 'Market Cap', 'Volume 24h' και 'percent change 7d', έπειτα προσθέτουμε τις μονάδες που συγκέντρωσε σε κάθε στήλη το νόμισμα και τις διαιρούμε με τον αριθμό 3. Δημιουργούμε την στήλη 'Total' με τα αποτελέσματα και την ταξινομούμε από το μικρότερο έως το μεγαλύτερο.

Τα 50 πρώτα μικρότερα που εμφανίζονται είναι και αυτά που μας ενδιαφέρουν.

```
def rank(tbl, rnkColumn):
    ntbl = tbl.sort_values(rnkColumn, ascending=False)
    ntbl["rank"] = range(1, tbl.shape[0]+1, 1) #creates column rank
    ntbl["ranking_method"] = rnkColumn #creates column ranking
    method
    return(ntbl)

def rank_coins():
    coinTble = coinmarketapi_values('market_cap')
    rnk1 = rank(coinTble, "quote.USD.market_cap")
    rnk2 = rank(coinTble, "quote.USD.percent_change_7d")
    rnk3 = rank(coinTble, "quote.USD.volume_24h")

    tbl_rnk = rnk1.append(rnk2).append(rnk3)
    table_pivot = tbl_rnk.pivot_table(values='rank', index='symbol',
    columns='ranking_method')
    table_pivot["Total"] = table_pivot.loc['Total',:] =
table_pivot.sum(axis=1)
    table_pivot["Total"] = table_pivot["Total"] / 3
    sort_table = table_pivot.sort_values('Total')
    # index_table = sort_table.index
    return(sort_table)

total_tbl = rank_coins()
with pd.option_context('display.max_rows', None,
'display.max_columns', None): # more options can be specified also
    print(total_tbl)
```

Εμφάνιση αποτελεσμάτων:

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL		
		Market_Cap	Percent_change_7d	Volume_24h	Total
name					
Binance	Coin	3.0	12.0	7.0	7.333333
Cardano		5.0	21.0	4.0	10.000000
Tether		4.0	27.0	1.0	10.666667
Bitcoin		1.0	35.0	2.0	12.666667
Dogecoin		12.0	23.0	10.0	15.000000
USD	Coin	11.0	29.0	12.0	17.333333
Huobi	Token	20.0	13.0	19.0	17.333333
Crypto.com	Coin	15.0	8.0	34.0	19.000000
Ethereum		2.0	52.0	3.0	19.000000
Fantom		36.0	1.0	23.0	20.000000
XRP		6.0	50.0	5.0	20.333333
NEM		14.0	14.0	37.0	21.666667
Binance	USD	31.0	30.0	9.0	23.333333
Chainlink		8.0	55.0	13.0	25.333333
Litecoin		7.0	64.0	6.0	25.666667
TRON		21.0	43.0	14.0	26.000000
Neo		24.0	36.0	20.0	26.666667
OKB		50.0	6.0	28.0	28.000000
Ravencoin		41.0	3.0	42.0	28.666667
Bitcoin	Cash	9.0	70.0	8.0	29.000000
EOS		17.0	60.0	11.0	29.333333
Stellar		10.0	61.0	17.0	29.333333
Dash		32.0	44.0	16.0	30.666667
Dai		30.0	28.0	36.0	31.333333
Polygon		53.0	9.0	35.0	32.333333
Wrapped	Bitcoin	13.0	34.0	52.0	33.000000
FTX	Token	25.0	15.0	60.0	33.333333
BitTorrent		42.0	18.0	40.0	33.333333
IOST		60.0	20.0	21.0	33.666667
Cosmos		16.0	62.0	24.0	34.000000

Μέρος Δεύτερο

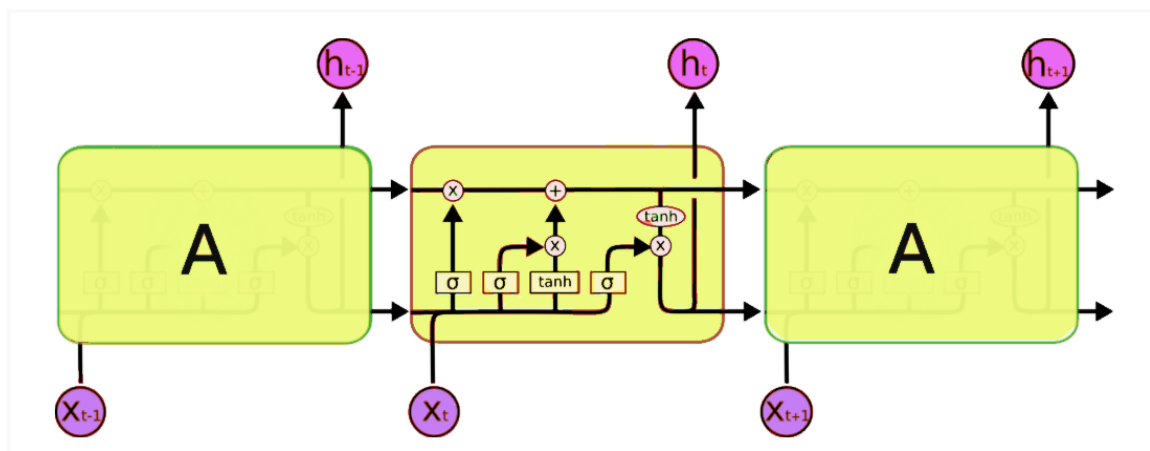
Αξιολογήσαμε λοιπόν τα κρυπτονομίσματα μας και τώρα θέλουμε να αναλύσουμε το κάθε ένα ξεχωριστά, με τεχνική προγραμματισμού την μηχανική εκμάθηση, για την πρόβλεψη της ανόδου ή τις καθόδου της τιμής του κρυπτονομίσματος .

Με αυτό τον τρόπο, θα μπορέσουμε να εξασφαλίσουμε και να ενισχύσουμε με παραπάνω ασφάλεια την απόφαση μας, για το πότε να αγοράσουμε ή να πουλήσουμε κάποιο κρυπτονόμισμα που ήδη διαθέτουμε.

Κάναμε λοιπόν έρευνα αναζήτησης για την εύρεση των πιο αποδοτικών μοντέλων μηχανικής εκμάθησης για την πρόβλεψη χρηματιστηριακών τιμών(μετοχές), που να συνδυάζει την γλώσσα προγραμματισμού python, τη βιβλιοθήκη tensorflow της google και να μπορεί να υλοποιηθεί στο χρονικό διάστημα της πρακτικής άσκησης των 4 μηνών.

Αποφασίσαμε το μοντέλο που θα ακολουθήσουμε για να υλοποιήσουμε είναι αυτό του Long Short Term Memory που σημαίνει 'μακροχρόνια βραχυπρόθεσμη μνήμη' ή κοινός LSTM.

Το LSTM είναι ένας τύπος επαναλαμβανόμενου νευρωνικού δικτύου, αλλά αντί να τροφοδοτούν απλώς το αποτέλεσμα στο επόμενο τμήμα του δικτύου, ένα LSTM κάνει μια σειρά μαθηματικών λειτουργιών, ώστε να μπορεί να έχει καλύτερη μνήμη.



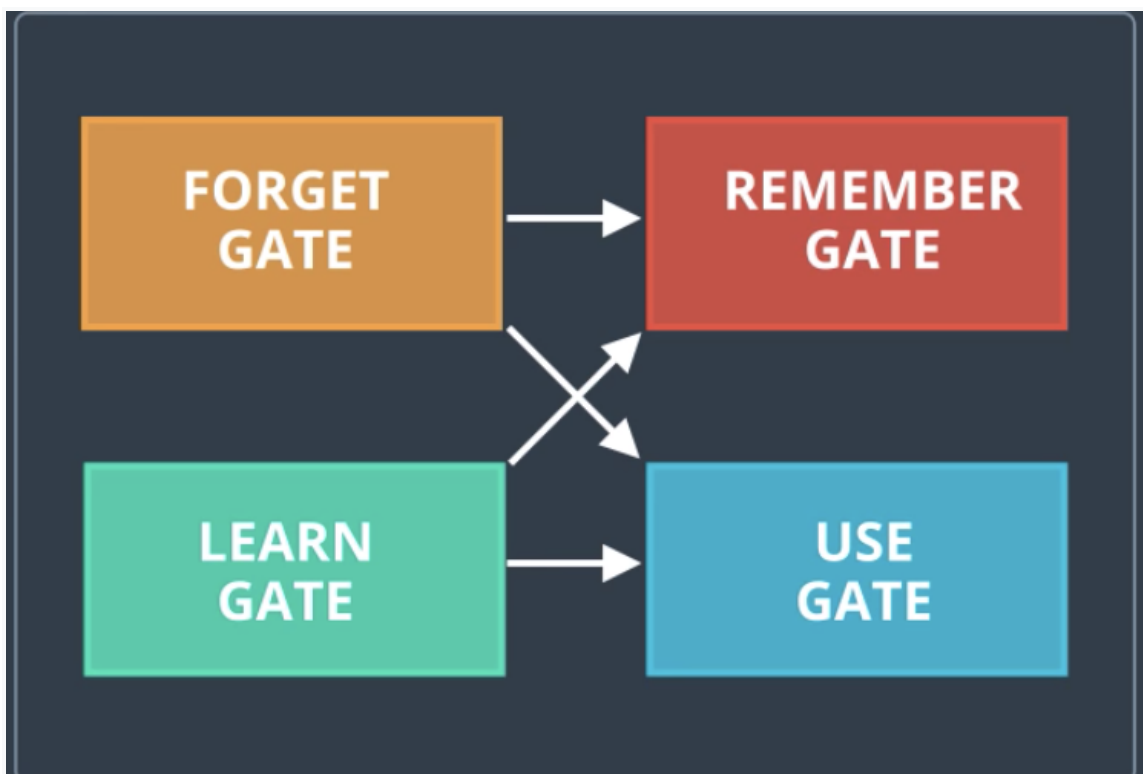
Στο παραπάνω σχήμα βλέπουμε ένα νευρωνικό δίκτυο και στο μεσαίο επίπεδο(Layer) την εφαρμογή του LSTM αλγορίθμου.

Ένα LSTM έχει τέσσερις «πύλες»:

1. Την Ξεχάστε (Forget Gate).
2. Την Θυμηθείτε (Remember Gate).
3. Την Μαθαίνεται (Learn Gate).
4. Την Χρησιμοποιήστε (use or output Gate).

Έχει επίσης τρεις εισόδους:

1. Τη Μακροπρόθεσμη μνήμη.
2. Τη Βραχυπρόθεσμη μνήμη.
3. Τα στοιχεία που ταΐζουμε κάθε φορά το μοντέλο μας, τα οποία ονομάζουμε «Ε» και θα πρέπει να συμψηφιστούν με τις πύλες για να γνωρίζουμε, τι θα αποθηκεύσουμε και τι θα ξεχάσουμε σε κάθε μνήμη ξεχωριστά, είτε είναι Μακροπρόθεσμη ή είτε είναι Βραχυπρόθεσμη μνήμη.



Βήμα 1: Όταν οι 3 εισόδοι εισέλθουν στο LSTM πηγαίνουν είτε στην πύλη 'Ξεχάστε' είτε στην πύλη "Μαθαίνεται".

Οι μακροπρόθεσμες πληροφορίες πηγαίνουν στην πύλη "Ξεχάστε", όπου, μερικά από αυτά θα ξεχαστούν (τα άσχετα μέρη).

Οι βραχυπρόθεσμες πληροφορίες και το «E» μπαίνουν στην πύλη “Μαθαίνεται”. Αυτή η πύλη αποφασίζει ποιες πληροφορίες θα μάθει το μοντέλο.

Βήμα 2: Οι πληροφορίες που περνούν την πύλη “Ξεχάστε” (δεν ξεχνάμε, οι ξεχασμένες πληροφορίες παραμένουν στην πύλη) και οι πληροφορίες που περνούν την πύλη “Μαθαίνεται” θα μεταβούν στην πύλη μνήμης (η οποία αποτελεί τη νέα μακροπρόθεσμη μνήμη) και η πύλη “Χρησιμοποιήστε” (η οποία ενημερώνει τη βραχυπρόθεσμη μνήμη + είναι το αποτέλεσμα του δικτύου).

Learn Gate

Η πύλη “Μαθαίνεται”, συνδυάζει STM(Βραχυπρόθεσμη μνήμη) + “E” (είσοδος) και επιλέγει να αγνοήσει τις περιττές πληροφορίες.

Αυτή η πύλη συνδυάζει την υπάρχουσα βραχυπρόθεσμη μνήμη (STM) και κάποια είσοδο “E”, πολλαπλασιάζεται με μια μήτρα (W) και προσθέτει b. Στη συνέχεια, τα συμπιέζει όλα σε μια λειτουργία tanh.

$$N_t = \tanh(W_n[STM_{t-1}, E_t] + b_n)$$

Αυτός ο συνδυασμός μας δίνει «N».

Στη συνέχεια, αγνοεί μέρος της βραχυπρόθεσμης μνήμης, πολλαπλασιάζοντας το συνδυασμένο αποτέλεσμα με έναν «παράγοντα αγνόησης».

Ο παράγοντας αγνόησης (I) υπολογίζεται συνδυάζοντας STM και E, με ένα νέο σύνολο W (βάρη) και b (μεροληψίες)

$$i_t = \sigma(W_i[STM_{t-1}, E_t] + b_i)$$

Μόλις έχουμε N και I, πολλαπλασιάζουμε μαζί, και αυτό είναι το αποτέλεσμα της πύλης μάθησης.

Forget Gate

“Ξεχάστε” είναι η πύλη που χρησιμοποιείται για να απορριφθούν όλες τις περιττές μακροπρόθεσμες πληροφορίες.

Κάτι σαν όταν μελετάς για μια μεγάλη εξέταση και την επόμενη μέρα ξεχνάς τα πάντα. Αυτή είναι η δύναμη της πύλης “Ξεχάστε”.

Βασικά, η μακροπρόθεσμη μνήμη (LTM) πολλαπλασιάζεται με έναν παράγοντα ξεχασμού (f). Αυτός ο παράγοντας θα κάνει μερικές από τις μακροπρόθεσμες πληροφορίες να «ξεχαστούν»

Ο παράγοντας ξεχάσεως είναι ο εξής:

$$f_t = \sigma(W_f[STM_{t-1}, E_t] + b_f)$$

Υπολογίζεται λαμβάνοντας τη βραχυπρόθεσμη μνήμη και την είσοδο (E), πολλαπλασιάζοντας τα με κάποια βάρη και μεροληψίες και συμπιέζοντας τα σε μια σιγμοειδή συνάρτηση.

Αυτή η συνάρτηση (f) πολλαπλασιάζεται με LTM - και boom, έχουμε μείνει με το LTM που χρειαζόμαστε.

Remember Gate

Η πύλη “Θυμηθείτε” παίρνει τις πληροφορίες από την πύλη “Ξεχάστε” και τις προσθέτει στις πληροφορίες από την πύλη “Μαθαίνεται”, για να υπολογίσει τη νέα μακροπρόθεσμη μνήμη.

Remember gate = Learn gate output + Forget gate output

Use Gate

Η πύλη “Χρησιμοποιήστε” παίρνει το LTM από την πύλη “Ξεχάστε” και το STM + E από την πύλη “Μαθαίνεται” και τα χρησιμοποιεί για να βρει μια νέα βραχυπρόθεσμη μνήμη ή έξοδο (το ίδιο πράγμα)

Για παράδειγμα, αν προσπαθούσαμε να ταξινομήσουμε εικόνες, η έξοδος θα ήταν η ταξινόμηση δικτύου.

Παίρνει την έξοδο της πύλης “Ξεχάστε” και το βάζει σε μια λειτουργία ενεργοποίησης tanh, όπως έτσι:

$$U_t = \tanh(W_u LTM_{t-1} f_t + b_u)$$

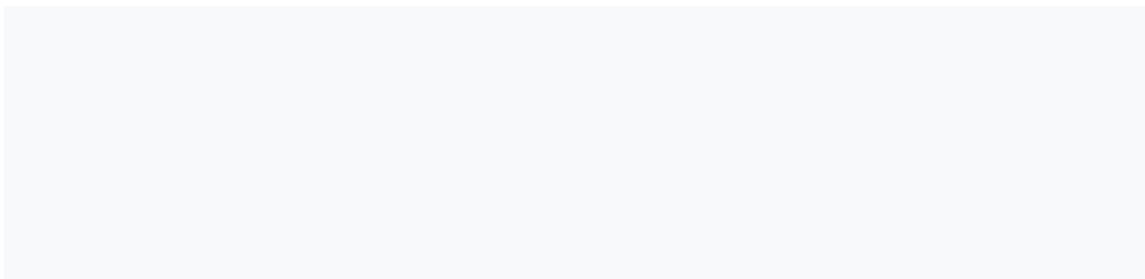
Παίρνει την έξοδο της πύλης “Μαθαίνεται” και εφαρμόζει μια λειτουργία σιγμοειδούς συνάρτησης, οπότε η εξίσωση μοιάζει με αυτήν:

$$V_t = \sigma(W_v [STM_{t-1}, E_t] + b_v)$$

Στη συνέχεια, η πύλη πολλαπλασιάζει το $V \times U$, για να αποκτήσει τη νέα βραχυπρόθεσμη μνήμη.

Ανάλυση συναισθημάτων

Η ανάλυση συναισθημάτων μπορεί να χρησιμοποιηθεί για μηχανές για την κατανόηση των ανθρώπινων συναισθημάτων, κάτι που είναι πάρα πολύ ενδιαφέρον. Τα πάντα κρύβουν ένα ίχνος συναισθήματος στις ανθρώπινες επιλογές, έτσι λοιπόν κάποια συναισθήματα κρύβει και η τιμή των κρυπτονομισμάτων. Ζούμε στην εποχή της πληροφορίας και χάρης αυτή έχουμε καταφέρει να προχωρήσουμε τεχνολογικά μπροστά και να βοηθήσουμε την ανθρωπότητα με υπηρεσίες και τεχνικές διευκόλυνσης για τον πλανήτη μας. Τα δεδομένα είναι ο επόμενος χρυσός της εποχής μας, αυτό τον χρυσό λοιπόν τον εκμεταλλευόμαστε και τον δίνουμε στο μοντέλο μας για φαγητό και μέσω της βραχύχρονης και μακρόχρονης μνήμης προσπαθούμε να αναλύσουμε και να κατανοήσουμε τα συναισθήματα που παρουσιάζουν οι τιμές. Όλοι μας μπορούμε να προβλέψουμε την επόμενη σκέψη μας στο μέλλον όταν κατανοούμε ότι κάτι που μόλις είδαμε ή ακούσαμε ή αισθανθήκαμε μας ξύπνησε μια ανάμνηση μας, τότε λοιπόν θυμόμαστε και περιγράφουμε αυτό το συναίσθημα ανασύροντας κομμάτια της μνήμης μας, συνδέοντας με αυτό τον τρόπο το παζλ. Όσον αφορά το Μοντέλο μας, επικεντρώνεται καθαρά στο κομμάτι του οφθαλμού. Δηλαδή το βάζουμε να παρατηρήσει τις τιμές του νομίσματος και να κρατήσει τις πιο σημαντικές συμπεριφορές από αυτές για να μπορεί να τις θυμηθεί και να κάνει την σωστή πρόβλεψη με αυτά που γνωρίζει και θυμάται.



Το RNN μας χρησιμοποιεί τέσσερα επίπεδα LSTM με 50 νευρώνες το κάθε ένα και “dropout = 0.2”, δηλ απο τα δεδομένα που κρατάνε οι πύλες μνήμης το 20% το πετάμε για να μην γεμίζουμε την μνήμη του μοντέλου με πάρα πολλά δεδομένα και το μπερδέψουμε ως προς την πρόβλεψη.

Έπειτα στο τέλος το μοντέλο μας έχει ένα “επίπεδο εξαγωγής” όπου είναι και το ζητούμενο αποτέλεσμα(Dense Layer).

```
# Initialising the RNN(recurrent neural network )
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape
= (x_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))
```

Δημιουργήσαμε το μοντέλο μας λοιπόν και καλούμαστε να το κάνουμε Compile, έτσι ώστε να προπονήσουμε τα δεδομένα μας και πάνω στην εκμάθηση αυτών να κάνουμε την πρόβλεψη.

Επιλέξαμε τον “ADAM OPTIMIZER” για το μοντέλο μας, το όνομα του οποίου δεν είναι τυχαίο. Το όνομα Adam προέρχεται από την προσαρμοστική εκτίμηση στιγμής (adaptive moment estimation).

Ακριβώς δηλαδή αυτό που χρειάζεται το μοντέλο μας, αφού δέχεται συνέχεια διαφορετικές τιμές μεταξύ τους και θα πρέπει να καταλάβει τις κοινές συσχετίσεις των τιμών αυτών, έτσι ώστε με βάση αυτές να κάνει την πρόβλεψη που πιστεύει.

```
# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
# Fitting the RNN to the Training set
regressor.fit(x_train, y_train, epochs = 200, batch_size = 512)
```

Όπως και ο άνθρωπος έχει μάθει σε κάθε περιβάλλον που βρίσκεται να προσαρμόζεται σε αυτό, είτε για να επιβιώσει, είτε για να περάσει όμορφα το χρόνο του, έτσι λοιπόν και με την Τεχνητή Νοημοσύνη (το λέει και η λέξη) προσπαθούμε αλγοριθμικά να περάσουμε αυτή την δύναμη της προσαρμογής από τον άνθρωπο στον υπολογιστή.

Ο Adam είναι ένας αλγόριθμος βελτιστοποίησης που μπορεί να χρησιμοποιηθεί αντί της διαδικασίας του κλασικού Stochastic Gradient Descent για την ενημέρωση επαναληπτικών βαρών δικτύου με βάση τα δεδομένα εκπαίδευσης.

Στην συνέχεια έρχεται η προπόνηση του μοντέλου μας όπου χρησιμοποιούμε διάφορους συνδυασμούς των Batch size και των Epoch για την εύρεση καλύτερης ακρίβειας του μοντέλου.

Όπως στην αρχή έτσι και στο τέλος κάναμε προετοιμασία των δεδομένων για να μπορέσουμε να τα δώσουμε στο μοντέλο στην μορφή με την οποία δέχεται τα δεδομένα. Επιλέγουμε το 80% των δεδομένων για εκπαίδευση και το υπόλοιπο 20% για τεστάρισμα.

```
Feed the Model data  
(6902, 60, 1)
```

Το μοντέλο δέχεται τιμές σε δυαδική μορφή μέσα σε ένα 3x3 numpy πίνακα και αυτό παρουσιάζεται με την χρήση του timestep. Το LSTM χρησιμοποιεί το timestep για να δημιουργήσει καλύτερη μνήμη στο μοντέλο. Για κάθε τιμή που θέλουμε να προβλέψουμε δίνουμε στο μοντέλο μας τον αριθμό (number of timesteps) των προηγούμενων τιμών που είχε το κρυπτονόμισμα και το κρατάμε στη μνήμη(numpy array) για την πρόβλεψη της κάθε τιμής.

Με την βοήθεια του MiniMaxScaler αναπαριστούμε τις τιμές μας στο διάστημα από 0 έως 1, διότι σε αυτή την μορφή διαβάζει το μοντέλο μας τα δεδομένα. Στο μοντέλο μας επιλέξαμε τα 60 timesteps και δημιουργήσαμε τον numpy array όπου και δώσαμε στο μοντέλο για training. Δηλαδή σε κάθε τιμή, αναλογούν άλλες 60.

```
#Convert the dataframe to a numpy array  
dataset = data.values  
#Get the number of rows to train the model(we use 80%)  
training_data_len = len(dataset) * .8  
training_data_len = int(training_data_len)  
#Scale the data  
scaler = MinMaxScaler(feature_range=(0,1))  
scaled_data = scaler.fit_transform(dataset)  
  
#Create the training data set  
#Create the scaled training data set  
train_data = scaled_data[0:training_data_len , :]  
#Split the data x_train and y_train data sets  
x_train = []  
y_train = []  
  
for i in range (60, len(train_data)):  
    x_train.append(train_data[i-60:i, 0])  
    y_train.append(train_data[i, 0])
```

```

#Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

#Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0],
x_train.shape[1],1))

```

Αφού τελειώσει το training λοιπόν, ακολουθούμε την ίδια διαδικασία των δεδομένων που μόλις αναφέραμε για την δημιουργία του 20% των δεδομένων σε numpy array, έτσι ώστε να είναι σε θέση το μοντέλο μας να κάνει την πρόβλεψη μέσω της μεθόδου predict().

```

#Create the testing Data set
#Create a new array containing scaled values from the remaining 0.2
data which left to test the model
test_data = scaled_data[training_data_len - 60: , :]
#Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

#Convert the data to a numpy array
x_test = np.array(x_test)

#Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

predictions = regressor.predict(x_test)
predictions = scaler.inverse_transform(predictions)

```

Η πηγή των δεδομένων που χρησιμοποιήθηκε είναι το API της Binance. Το Binance API είναι μια μέθοδος που μας επιτρέπει να συνδεθούμε με τους διακομιστές Binance μέσω Python ή αρκετών άλλων γλωσσών προγραμματισμού. Με αυτό, μπορούμε να αυτοματοποιήσουμε τις συναλλαγές μας. Πιο συγκεκριμένα, το Binance διαθέτει ένα RESTful API που χρησιμοποιεί αιτήματα HTTP για αποστολή και λήψη δεδομένων.

Πιο συγκεκριμένα χρησιμοποιούμε την βιβλιοθήκη της binance και με την βοήθεια των παρακάτω τριών μεθόδων:

1. `date_to_milliseconds`,
2. `interval_to_milliseconds`
3. `get_historical_klines`

```
def date_to_milliseconds(date_str):
    """Convert UTC date to milliseconds
    If using offset strings add "UTC" to date string e.g. "now UTC",
    "11 hours ago UTC"
    See dateparse docs for formats
    http://dateparser.readthedocs.io/en/latest/
    :param date_str: date in readable format, i.e. "January 01,
    2018", "11 hours ago UTC", "now UTC"
    :type date_str: str
    """
    # get epoch value in UTC
    epoch = datetime.datetime.fromtimestamp(0).replace(tzinfo=pytz.utc)
    # parse our date string
    d = dateparser.parse(date_str)
    # if the date is not timezone aware apply UTC timezone
    if d.tzinfo is None or d.tzinfo.utcoffset(d) is None:
        d = d.replace(tzinfo=pytz.utc)

    # return the difference in time
    return int((d - epoch).total_seconds() * 1000.0)

def interval_to_milliseconds(interval):
    """Convert a Binance interval string to milliseconds
    :param interval: Binance interval string 1m, 3m, 5m, 15m, 30m,
    1h, 2h, 4h, 6h, 8h, 12h, 1d, 3d, 1w
    :type interval: str
    :return:
```

```

        None if unit not one of m, h, d or w
        None if string not in correct format
        int value of interval in milliseconds
    """
    ms = None
    seconds_per_unit = {
        "m": 60,
        "h": 60 * 60,
        "d": 24 * 60 * 60,
        "w": 7 * 24 * 60 * 60
    }

    unit = interval[-1]
    if unit in seconds_per_unit:
        try:
            ms = int(interval[:-1]) * seconds_per_unit[unit] * 1000
        except ValueError:
            pass
    return ms

def get_historical_klines(symbol, interval, start_str,
end_str=None):
    """Get Historical Klines from Binance

    See dateparse docs for valid start and end string formats
http://dateparser.readthedocs.io/en/latest/

    If using offset strings for dates add "UTC" to date string e.g.
    "now UTC", "11 hours ago UTC"

    :param symbol: Name of symbol pair e.g BNBBTC
    :type symbol: str
    :param interval: Biannce Kline interval
    :type interval: str
    :param start_str: Start date string in UTC format
    :type start_str: str
    :param end_str: optional - end date string in UTC format
    :type end_str: str
    :return: list of OHLCV values
    """

    # create the Binance client, no need for api key
    client = Client("", "")

    # init our list
    output_data = []

    # setup the max limit
    limit = 1000

```

```

# convert interval to useful value in seconds
timeframe = interval_to_milliseconds(interval)

# convert our date strings to milliseconds
start_ts = date_to_milliseconds(start_str)

# if an end time was passed convert it
end_ts = None
if end_str:
    end_ts = date_to_milliseconds(end_str)

idx = 0
# it can be difficult to know when a symbol was listed on
Binance so allow start time to be before list date
symbol_existed = False
while True:
    # fetch the klines from start_ts up to max 500 entries or
the end_ts if set
    temp_data = client.get_klines(
        symbol=symbol,
        interval=interval,
        limit=limit,
        startTime=start_ts,
        endTime=end_ts
    )

    # handle the case where our start date is before the symbol
pair listed on Binance
    if not symbol_existed and len(temp_data):
        symbol_existed = True

    if symbol_existed:
        # append this loops data to our output data
        output_data += temp_data

        # update our start timestamp using the last value in the
array and add the interval timeframe
        start_ts = temp_data[len(temp_data) - 1][0] + timeframe
    else:
        # it wasn't listed yet, increment our start date
        start_ts += timeframe

    idx += 1

```



```

        # check if we received less than the required limit and exit
the loop
        if len(temp_data) < limit:
            # exit the while loop
            break

        # sleep after every 3rd call to be kind to the API
        if idx % 3 == 0:
            time.sleep(1)

    return output_data

```

Λαμβάνουμε την Ανοιχτή,Υψηλή,Χαμηλή και Κλειστή τιμή οποιουδήποτε κρυπτονομίσματος θέλουμε, για συγκεκριμενη ώρα ή λεπτό, μήνα ή μέρα ,χρόνο ή χρόνια από την ημερομηνία εκκίνησης της Binance το καλοκαίρι του 2017 έως σήμερα.

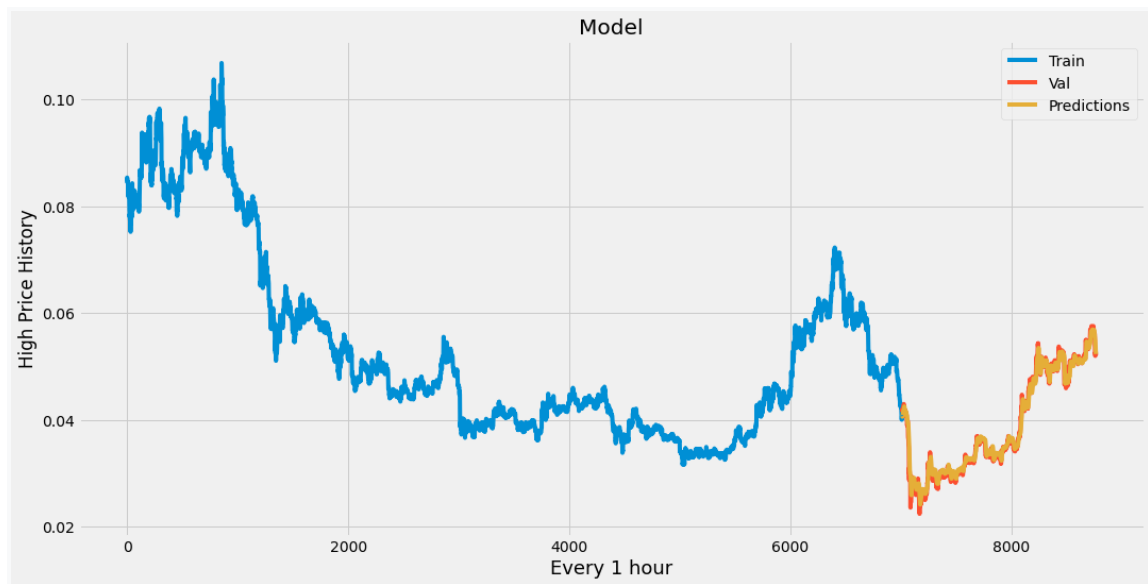
Επιλέξαμε την Binance συγκεκριμένα, διότι από τον Ιανουάριο του 2018, η Binance ήταν το μεγαλύτερο χρηματιστήριο κρυπτονομισμάτων στον κόσμο όσον αφορά τον όγκο συναλλαγών.

```

klines = get_historical_klines("BTCUSDT",
Client.KLINE_INTERVAL_1HOUR, "22 Aug, 2018", "22 Aug, 2019")
df = pd.DataFrame(klines)

```

Τέλος, παρουσιάζεται Διάγραμμα τιμών, όπου η μπλέ γραμμή αναπαριστά το 80% των δεδομένων εκμάθησης, η κόκκινη γραμμή τις πραγματικές τιμές ως το υπόλοιπο 20% των δεδομένων και η κίτρινη γραμμή τις προβλέψεις μας με βάση το 20% των δεδομένων για τρέστ.



Μπορούμε να διακρίνουμε πώς το μοντέλο μας ακολούθησε με αρκετά καλή επιτυχία την πορεία της τιμής.

