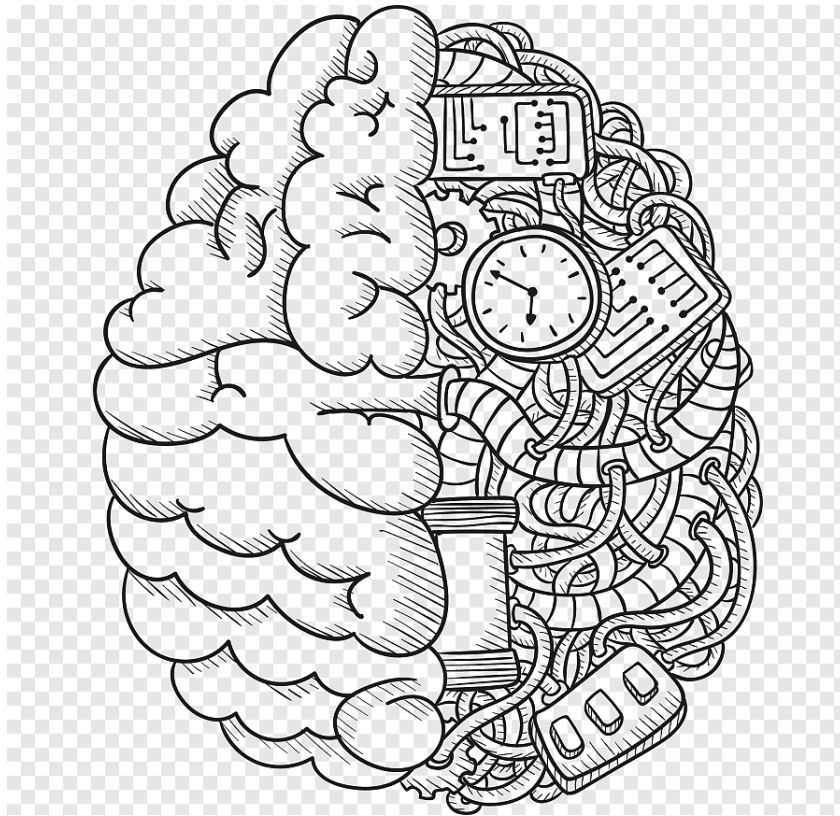


# Αναφορά Εργασίας

Στέργιος Καραγεώργης AM2346 Χρήστος Πανουργιάς AM2405

Εαρινό 2022



## Περιεχόμενα

<b>1</b>	<b>Εισαγωγή</b>	<b>3</b>
1.1	Παρουσίαση προβλήματος	3
1.2	Λόγοι επιλογής του θέματος	3
<b>2</b>	<b>Μεθοδολογία</b>	<b>3</b>
2.1	Πως μπορεί να χρησιμοποιηθεί η επιστήμη των δεδομένων στο θέμα μας	3
2.2	Μέθοδοι	4
<b>3</b>	<b>Σύνολα δεδομένων</b>	<b>4</b>
3.1	Εφαρμογή:	4
<b>4</b>	<b>Προσέγγιση Προβλήματος και Λύσεις</b>	<b>4</b>
4.1	Μοντέλο Arima	4
4.2	Γραμμική Παλινδρόμηση	7
4.3	Νευρωνικό Δίκτυο LSTM	9
4.3.1	Γιατί LSTM ;	9
4.3.2	Διαδικασία του LSTM	9
<b>5</b>	<b>Συμπεράσματα :</b>	<b>11</b>
<b>6</b>	<b>Βιβλιογραφία</b>	<b>15</b>

# 1 Εισαγωγή

## 1.1 Παρουσίαση προβλήματος

Το χρηματιστήριο είναι ένας από τους σημαντικότερους τομείς επενδύσεων. Επομένως η πρόβλεψη της τάσης της των τιμών του χρηματιστηρίου αποτελεί ένα θέμα υψίστης σημασίας για τους ερευνητές του οικονομικού τομέα. Στόχος της παρούσας έρευνας, είναι η δημιουργία ενός μοντέλου πρόβλεψης τιμών που εστιάζει στη βραχυπρόθεσμη πρόβλεψη της τάσης των τιμών. Σύμφωνα με τον Fama [1], η πρόβλεψη οικονομικών χρονοσειρών είναι ένα εξαιρετικά δύσκολο εγχείρημα, λόγω της γενικά αποδεκτής ημι-ισχυρής μορφής αποδοτικότητας της αγοράς και της ύπαρξης υψηλού ποσοστού θορύβου.

[1] Malkiel BG, Fama EF. Efficient capital markets: a review of theory and empirical work. J Finance. 1970;25(2):383–417. Article Google Scholar

## 1.2 Λόγοι επιλογής του θέματος

Η πρόβλεψη τιμών των μετοχών με τη βοήθεια αλγορίθμων μηχανικής μάθησης, βοηθά στον εντοπισμό της μελλοντικής αξίας των μετοχών της εταιρίας καθώς και των υπόλοιπων χρηματοοικονομικών περιουσιακών στοιχείων που συναλλάσσονται σε ένα χρηματιστήριο. Ο κύριος σκοπός της πρόβλεψης των τιμών αφορά τη συνεχή αύξηση των κερδών. Παρόλα αυτά η πρόβλεψη της απόδοσης του χρηματιστηρίου παραμένει ένα δύσκολο έργο, καθώς η πρόβλεψη δύναται να επηρεαστεί από μια σειρά παραγόντων όπως σωματικοί, ψυχολογικοί, ορθολογική ή παρορμητική συμπεριφορά. Όλοι αυτοί οι παράγοντες συνδυάζονται για να κάνουν τις τιμές των μετοχών δυναμικές και ασταθείς, καθιστώντας έτσι δύσκολη την ακριβή πρόβλεψη τους.

# 2 Μεθοδολογία

## 2.1 Πως μπορεί να χρησιμοποιηθεί η επιστήμη των δεδομένων στο θέμα μας

Η επιστήμη των δεδομένων μπορεί να χρησιμοποιηθεί για να μας δώσει μια ξεχωριστή εικόνα στο πως κατανοούμε τη χρηματιστηριακή αγορά και τα οικονομικά δεδομένα. Συγκεκριμένες βασικές αρχές χρησιμοποιούνται κατά τη διάρκεια των συναλλαγών (πώληση, αγορά και διατήρηση) με κύριο γνώμονα την απόκτηση υψηλών κερδών. Διάφορες πλατφόρμες συναλλαγών γίνονται όλο και πιο δημοφιλείς, κάνοντας απαραίτητη την κατανόηση **βασικών εννοιών της Επιστήμης των Δεδομένων**, ώστε να είμαστε σε θέση να αναγνωρίσουμε την αξία επένδυσης σε μια συγκεκριμένη μετοχή καθώς και την ανάλυση της **χρηματιστηριακής αγοράς**.

Η Επιστήμη των Δεδομένων βασίζεται σε μεγάλο βαθμό στην πρόβλεψη

μελλοντικών αποτελεσμάτων και στη μοντελοποίηση δεδομένων. Στο χρηματιστήριο, ένα **μοντέλο χρονοσειρών** χρησιμοποιείται για την πρόβλεψη της ανόδου και της πτώσης των τιμών των μετοχών. Τέλος, τα **μοντέλα ταξινόμησης**, οι **αλγόριθμοι**, η **εκπαίδευση**, οι **δοκιμές** και τα **νευρωνικά δίκτυα** είναι ορισμένες ακόμα έννοιες που χρησιμοποιούνται.

## 2.2 Μέθοδοι

Στο παρόν έργο τα μοντέλα της Επιστήμης των Δεδομένων θα δημιουργηθούν μέσα στο πλαίσιο του **Supervised Learning**. Πιο συγκεκριμένα οι μέθοδοι **ταξινόμησης**, **παλινδρόμησης** και **χρονοσειρών** όπως **Logistic Regression**, το μοντέλο “**ARIMA**” και **Support Vector Machines (SVR)** θα είναι τα κύρια εργαλεία μας. Επίσης στοχεύουμε στη δημιουργία ενός **νευρωνικού δικτύου** όπως το **LSTM (Long-Short-Term-Memory)** το οποίο θα μας προσφέρει περαιτέρω γνώσεις για θέμα.

## 3 Σύνολα δεδομένων

### 3.1 Εφαρμογή:

Τα δεδομένα θα εξαχθούν από την παρακάτω πηγή:

<https://www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs>

Η επίσημη γλώσσα υπολογιστών που θα χρησιμοποιηθεί είναι η **Python 3**. Επιπλέον, θα χρησιμοποιηθούν βιβλιοθήκες για την καλύτερη κατανόηση του προβλήματος, όπως το **keras** για το νευρωνικό δίκτυο, το **numpy** και **scikit-learn** για τις μεθόδους ταξινόμησης και παλινδρόμησης. Τέλος θα χρησιμοποιηθεί η βιβλιοθήκη **matplotlib** για τη σύγκριση των αποτελεσμάτων και των σχεδιασμό τους σε γραφήματα.

## 4 Προσέγγιση Προβλήματος και Λύσεις

### 4.1 Μοντέλο Arima

Η αρχική προσέγγιση για την λύση του προβλήματος μας είναι να εφαρμόσουμε το μοντέλο **ARIMA** το οποίο ονομαστικά σημαίνει: ολοκληρωμένο αυτοπαλινδρομικό μοντέλο κινητού μέσου όρου ( **Auto Regressive Integrated Moving Average** ). Σε αντίθεση με τα ντετερμινιστικά μοντέλα όπως γραμμική παλινδρόμηση, η χρήση των οποίων απαιτεί γνώση των παραγόντων από τις οποίες εξαρτάται το μέγεθος και όπου ο πλήρης εντοπισμός, μέτρηση και πρόβλεψή τους είναι πρακτικά αδύνατος, η εφαρμογή των μοντέλων **ARIMA** βασίζεται στον υπολογισμό της πιθανότητας για την οποία η τιμή του μεγέθους

βρίσκεται εντός κάποιου διαστήματος. Ένα μοντέλο ARIMA έχει τρία χαρακτηριστικά :

- $p$  : Η τάξη του AR όρου
- $d$  : Ο αριθμός διαφοράς όρων έτσι ώστε η χρονοσειρά να έχει σταθερό μέσο όρο
- $q$  : Η τάξη του MA όρου

Αυτό που χρειαζόμαστε έτσι ώστε να χρησιμοποιήσουμε ένα ARIMA μοντέλο είναι μία σταθερή χρονοσειρά ( δηλαδή μία χρονοσειρά όπου η διασπορά του λογαρίθμου των τιμών της αυξάνεται πιο "αργά" από τον ρυθμό ενός τυχαίου περιπάτου  $\Leftrightarrow$  η χρονοσειρά έχει σταθερό μέσο όρο ).

παρακάτω ακολουθεί ο τύπος του μοντέλου ARIMA

$$y'_t = \boxed{c} + \boxed{\varphi_1 y'_{t-1} + \dots + \varphi_p y'_{t-p}} + \boxed{\theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t}$$

intercept
lags  
(AR)
errors  
(MA)

Στην περίπτωση του προβλήματος μας θα χρησιμοποιηθεί το χαρακτηριστικό Close ( δηλαδή η τιμή της μετοχής στο κλείσιμο της ημέρας ).

Αρχικά θα εφαρμόσουμε το Augmented Dickey Fuller (ADF) Test για να ελέγξουμε αν η χρονοσειρά μας είναι σταθερής μορφής ( δηλαδή εάν έχει σταθερό μέσο όρο, σταθερή τυπική απόκλιση και σταθερή αυτοσυσχέτιση ). Προφανώς η χρονοσειρά μίας μετοχής σπάνια θα είναι σταθερής μορφής. Έπειτα για να υλοποιήσουμε το μοντέλο ARIMA πρέπει να προσδιορίσουμε τις παραμέτρους  $p$ ,  $d$ , και  $q$

**Η παράμετρος  $d$  :** Οι τιμές μίας μετοχής στο χρηματιστήριο δεν έχουν σταθερό μέσο όρο, εάν είχαν τότε θα μπορούσαμε να παράξουμε κέρδος εάν πουλούσαμε τις μετοχές όταν ήταν πάνω από τον μέσο όρο και αγοράζαμε τις ίδιες μετοχές όταν ήταν κάτω από τον μέσο όρο. Όμως οι επιστροφές (δηλαδή οι διαφορές) του χρηματιστηρίου τείνουν να έχουν σταθερό μέσο όρο. Εάν οι διαφορές δεν έχουν σταθερό μέσο όρο τότε θα πρέπει να συνεχίσουμε τον έλεγχο για τις διαφορές των διαφορών (  $d = 2$  ), συνεχίζοντας έτσι τους ελέγχους βρίσκουμε το ελάχιστο  $d$  το οποίο θα μας δημιουργήσει μία χρονοσειρά με σταθερό μέσο όρο, αυτό επίσης θα είναι το βέλτιστο  $d$  για το πρόβλημα μας.

Έχοντας βρει το  $d$  η χρονοσειρά μας έχει πλέον σταθερό μέσο όρο, και έτσι μένει να ελέγξουμε εάν χρειάζονται AR ή MA όροι (δηλαδή να προσδιορίσουμε τα  $p$ ,  $q$ ) έτσι ώστε να διορθώσουν οποιαδήποτε υπολοιπόμενη αυτοσυσχέτιση.

**Η παράμετρος  $p$  :** Εξετάζοντας την γραφική παράσταση της συνάρτησης μερικής αυτοσυσχέτισης (PACF) της διαφοροποιημένης σειράς, μπορούμε να προσδιορίσουμε προσωρινά την παράμετρο  $p$ . Η γραφική παράσταση PACF είναι μια γραφική παράσταση των συντελεστών μερικής συσχέτισης μεταξύ της σειράς και των καθυστερήσεων (lags) της ίδιας.

Γενικά, η «μερική» συσχέτιση μεταξύ δύο μεταβλητών είναι η ποσότητα της συσχέτισης μεταξύ τους η οποία δεν εξηγείται από τις αμοιβαίες συσχετίσεις τους με ένα καθορισμένο σύνολο άλλων μεταβλητών. Για παράδειγμα, εάν οπισθοδρομούμε μια μεταβλητή  $Y$  σε άλλες μεταβλητές  $X_1$ ,  $X_2$  και  $X_3$ , η μερική συσχέτιση μεταξύ  $Y$  και  $X_3$  είναι το ποσό συσχέτισης μεταξύ  $Y$  και  $X_3$  που δεν εξηγείται από τις κοινές συσχετίσεις τους με τις  $X_1$  και  $X_2$ . Αυτή η μερική συσχέτιση μπορεί να υπολογιστεί ως η τετραγωνική ρίζα της μείωσης της διακύμανσης που επιτυγχάνεται με την προσθήκη  $X_3$  στην παλινδρόμηση του  $Y$  στα  $X_1$  και  $X_2$ .

**Η παράμετρος  $q$  :** Παρόμοια, εξετάζοντας την γραφική παράσταση της συνάρτησης αυτοσυσχέτισης (ACF) της διαφοροποιημένης σειράς, μπορούμε να προσδιορίσουμε προσωρινά την παράμετρο  $q$ .

Τέλος έχοντας προσδιορίσει τις παραμέτρους  $p$ ,  $d$ ,  $q$  μπορούμε να υλοποιήσουμε το μοντέλο ARIMA και έπειτα να το αξιολογήσουμε σύμφωνα με το πόσο καλά προσεγγίζει τις αληθινές τιμές και με το αν αυτές οι τιμές βρίσκονται εντός των τιμών των διαστημάτων εμπιστοσύνης.

Παρακάτω βρίσκεται η υλοποίηση του μοντέλου ARIMA σε python 3 χρησιμοποιώντας το colab notebook.

```
# uploading file to colab
from google.colab import files
uploaded = files.upload()
```

Choose Files **tsla.us.txt**

- **tsla.us.txt**(text/plain) - 87266 bytes, last modified: 9/21/2019 - 100% done  
Saving tsla.us.txt to tsla.us (2).txt

```
!pip install statsmodels # installing stats models to use the ARIMA model and ADF
import pandas as pd # importing pandas for data handling
import matplotlib.pyplot as plt # this will be used for visualizing our data
from statsmodels.tsa.stattools import adfuller # for checking if price series is st
from statsmodels.graphics.tsaplots import plot_acf #(autocorrelation plot) the plot
from statsmodels.graphics.tsaplots import plot_pacf #(partial autocorrelation plot)
# ( it tells us how many times we have to difference in order to remove any autocor
from statsmodels.tsa.arima_model import ARIMA # this is the arima model
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-w>  
 Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: scipy>=1.3 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages  
 Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (

```
df = pd.read_csv("tsla.us.txt") # reading the stock
df.set_index(pd.DatetimeIndex(df['Date']), inplace=True) # Reindex data using a Dat
df = df[["Close"]] # getting only the Close price
```

```
# splitting the data into 70 percent for training and 30 for testing
train_len = int(len(df) * 0.7)
train = df[["Close"]][:train_len]
test = df[["Close"]][train_len:]
```

----- Detrmining the d term -----

## STATIONARITY

we will subtract the previous value from the current value  $y_t - y_{t-1}$

if we just difference once we might not get a stationary series, so we might need to do that multiple times  $y_t - y_{t-n}$

The minimum number of differencing operations needed to make the series stationary will be

## ▼ ADF TEST

We will use the Augmented Dickey Fuller test to check if the price series is stationary.

The null hypothesis of the ADF test is that the time series is non-stationary. So if the p-value of the test is less than the significance level (0.05) then we can reject the null hypothesis and conclude that the time series is stationary

And so for our problem, if p-value > 0.05 then we will need to find the order of differencing

```
adf = adfuller(train.Close.dropna()) # doing the adf test using the Close feature a
print("ADF Statistic : {}".format(adf[0]) )
print("p-value : {}".format(adf[1]))
```

```
ADF Statistic : -0.5776367039822343
p-value : 0.8759555720937171
```

the p-value is of course > 0.05 and so, since the returns tend to be stationary in the stock market, we need to compute the differences ( the returns ) in order to create a time series that is stationary.

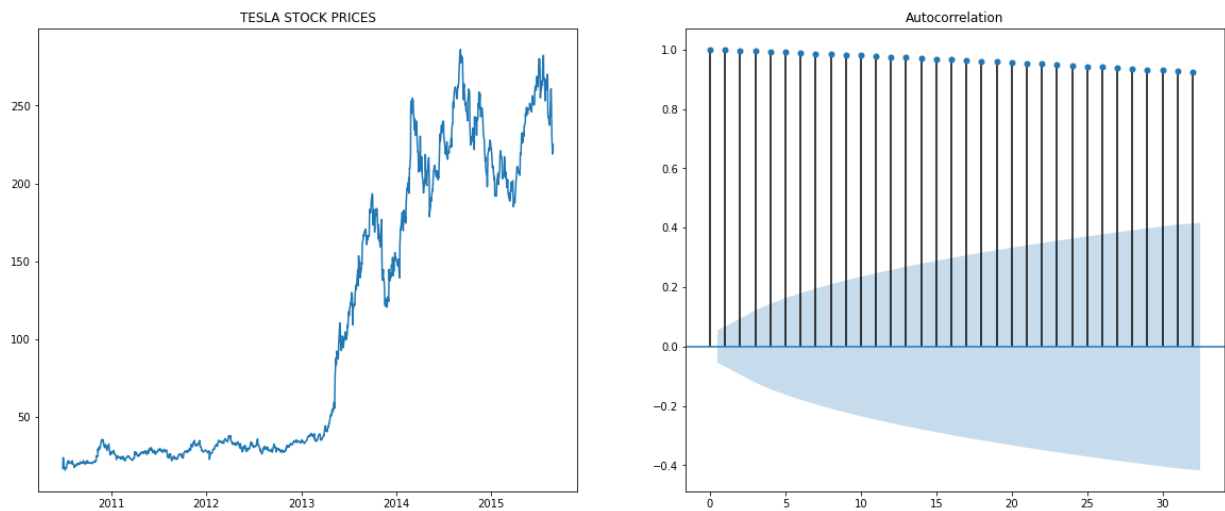
NOTE: if we difference once it might not be stationary so we might have to difference more than one times to make it stationary

that is the role that the d term will play in the ARIMA model.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8)) # creating two spaces for plo

ax1.plot(train) # plotting the tesla stock prices
ax1.set_title("TESLA STOCK PRICES") # titling the first plot
plot_acf(train, ax=ax2) # using the plot_acf to determine differences (the term d)
plt.show()
```

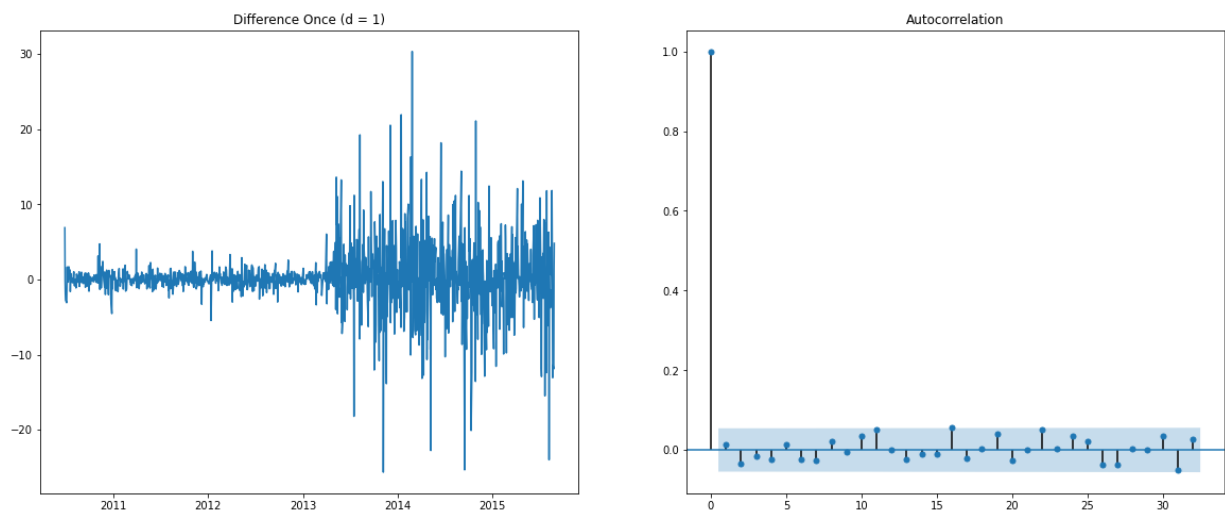




```
diff = train.Close.diff().dropna() # differencing by one term and dropping the NaN

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8)) # creating two spaces for plo

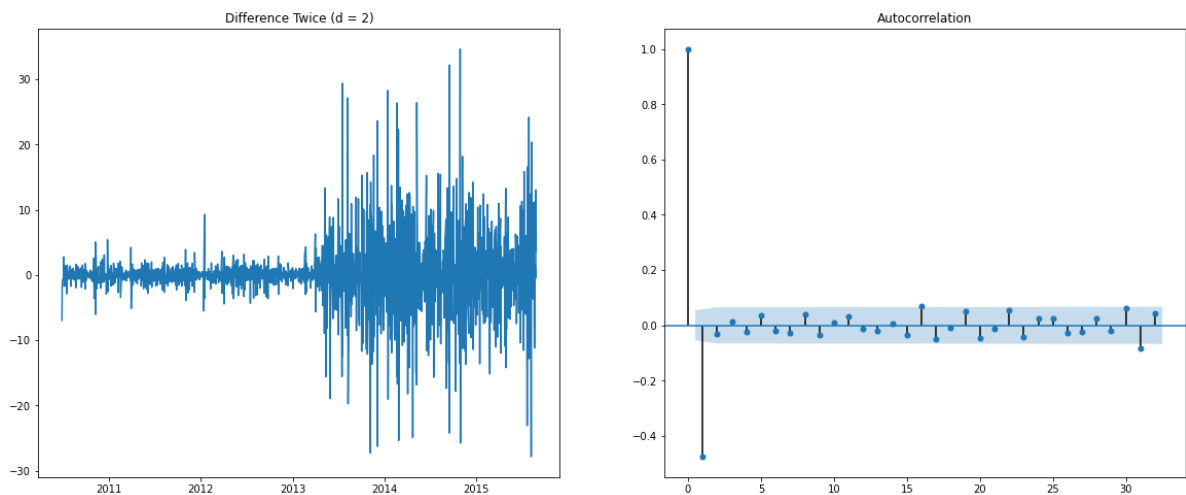
ax1.plot(diff) # plotting the difference once
ax1.set_title("Difference Once (d = 1)") # titling the first plot
plot_acf(diff, ax=ax2) # using the plot_acf to visualize autocorellation and statio
plt.show()
```



```
diff = train.Close.diff().diff().dropna() # differencing by one term and dropping t

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8)) # creating two spaces for plo
```

```
ax1.plot(diff) # plotting the difference once
ax1.set_title("Difference Twice (d = 2)") # titling the first plot
plot_acf(diff, ax=ax2) # using the plot_acf to visualize autocorellation and statio
plt.show()
```



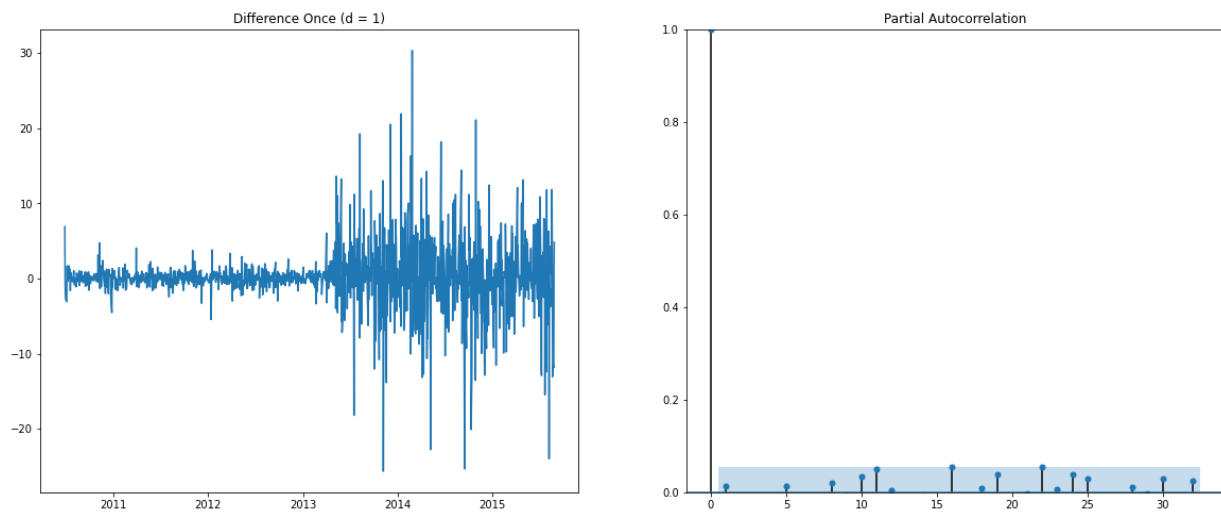
We have reduced the autocorrelation and  $d = 1$  seems to be a lot similar to  $d = 2$  but we notice that when  $d = 2$  the lag goes to the far negative which is an indicator that we have overdifferenced. Therefore the optimal choice for  $d$  is 1

----- Term  $d$  is Determined -----

▼ ----- Determining the  $p$  term -----

```
diff = train.Close.diff().dropna() # differencing by one term and dropping the NaN
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8)) # creating two spaces for plo

ax1.plot(diff) # plotting the difference once
ax1.set_title("Difference Once (d = 1)") # titling the first plot
ax2.set_ylim(0, 1) # setting limitations to the y axis from 0 to 1
plot_pacf(diff, ax=ax2) # using the plot_acf to visualize autocorellation and stati
plt.show()
```



From the graph we can observe that the PACF lag 11 is as close to be significant as it is on of the highest, therefore we are choosing  $p = 11$

----- Term  $p$  is Determined -----

▼ ----- Determining the  $q$  term -----

the term  $q$  is the order of the moving average (MA) term . It refers to the number of lagged forecast errors that should go into the ARIMA model.

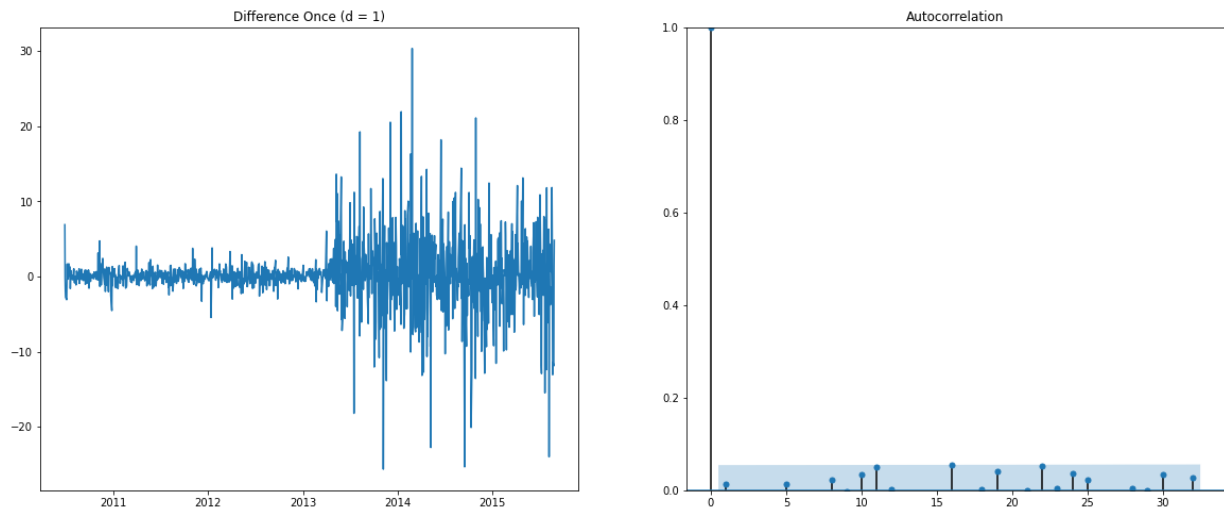
We can look at the ACF plot for the number of MA terms

```
diff = train.Close.diff().dropna() # differencing by one term and dropping the NaN

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8)) # creating two spaces for plo

ax1.plot(diff) # plotting the difference once
ax1.set_title("Difference Once (d = 1)") # titling the first plot
ax2.set_ylim(0, 1)
```

```
plot_acf(diff, ax=ax2) # using the plot_acf to visualize autocorellation and statio
plt.show()
```



From the ACF plot we can see that the term 11 is suitable for the selection of  $q$

And so we conclude  $q = 11$

## ▼ ----- Term $q$ is Determined -----

Now we are ready to create our ARIMA model with  $d = 1$ ,  $p = 11$ ,  $q = 11$

```
# implementing the arima model with d = 1, p = 16 and q = 11
# creating our ARIMA model
model = ARIMA(train, order=(11,1,11))
result = model.fit()
```

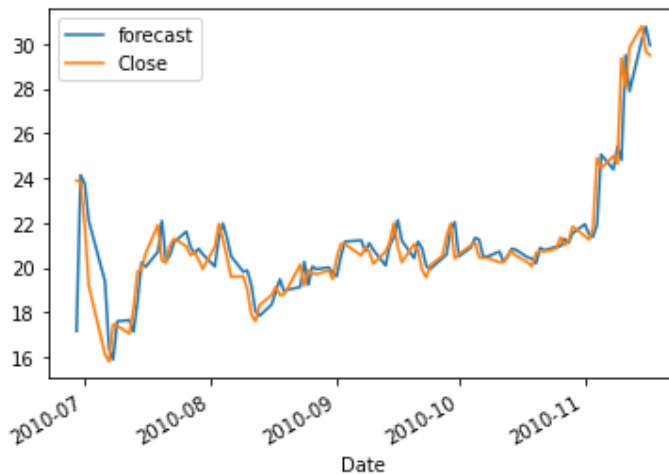
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
the underlying index is a RangeIndex or an integral index.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
the underlying index is a RangeIndex or an integral index.
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:492: HessianI
interval : int
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: Converge
```

```
print(result.summary()) # printing a summary of the results
```

ARIMA Model Results					
=====					
Dep. Variable:	D.Close	No. Observations:			
Model:	ARIMA(11, 1, 11)	Log Likelihood		-3651	
Method:	css-mle	S.D. of innovations		3	
Date:	Tue, 31 May 2022	AIC		7350	
Time:	20:32:16	BIC		7474	
Sample:	1	HQIC		7396	
=====					
	coef	std err	z	P> z	[0.025
-----					
const	0.1622	0.118	1.375	0.169	-0.069
ar.L1.D.Close	-0.3719	0.544	-0.684	0.494	-1.438
ar.L2.D.Close	0.4723	0.384	1.230	0.219	-0.280
ar.L3.D.Close	0.2919	0.146	1.993	0.046	0.005
ar.L4.D.Close	0.2728	0.197	1.382	0.167	-0.114
ar.L5.D.Close	0.0145	0.208	0.070	0.944	-0.393
ar.L6.D.Close	0.3638	0.079	4.597	0.000	0.209
ar.L7.D.Close	0.2865	0.230	1.246	0.213	-0.164
ar.L8.D.Close	0.2600	0.244	1.066	0.287	-0.218
ar.L9.D.Close	-0.6900	0.243	-2.841	0.005	-1.166
ar.L10.D.Close	-0.7348	0.300	-2.448	0.014	-1.323
ar.L11.D.Close	0.2826	0.483	0.585	0.559	-0.664
ma.L1.D.Close	0.4002	0.555	0.721	0.471	-0.688
ma.L2.D.Close	-0.4961	0.391	-1.269	0.205	-1.262
ma.L3.D.Close	-0.3177	0.165	-1.925	0.054	-0.641
ma.L4.D.Close	-0.2801	0.217	-1.293	0.196	-0.704
ma.L5.D.Close	0.0001	0.215	0.001	1.000	-0.421
ma.L6.D.Close	-0.3820	0.066	-5.802	0.000	-0.511
ma.L7.D.Close	-0.3348	0.237	-1.411	0.158	-0.800
ma.L8.D.Close	-0.2515	0.267	-0.943	0.346	-0.774
ma.L9.D.Close	0.7193	0.237	3.039	0.002	0.255
ma.L10.D.Close	0.8030	0.327	2.453	0.014	0.161
ma.L11.D.Close	-0.2723	0.530	-0.514	0.607	-1.310
Roots					
=====					
	Real	Imaginary	Modulus	Freque	
-----					
AR.1	-1.0075	-0.1085j	1.0133	-0.	
AR.2	-1.0075	+0.1085j	1.0133	0.	
AR.3	-0.7005	-0.7139j	1.0002	-0.	
AR.4	-0.7005	+0.7139j	1.0002	0.	
AR.5	-0.1659	-0.9881j	1.0020	-0.	
AR.6	-0.1659	+0.9881j	1.0020	0.	
AR.7	0.5510	-0.8420j	1.0063	-0.	
AR.8	0.5510	+0.8420j	1.0063	0.	
AR.9	1.0103	-0.1730j	1.0250	-0.	
AR.10	1.0103	+0.1730j	1.0250	0.	
AR.11	3.2249	-0.0000j	3.2249	-0.	
MA.1	-0.9937	-0.1130j	1.0001	-0.	
MA.2	-0.9937	+0.1130j	1.0001	0.	
MA.3	-0.6994	-0.7147j	1.0000	-0.	
MA.4	-0.6994	+0.7147j	1.0000	0.	
MA.5	-0.1709	-0.9853j	1.0000	-0.	
MA.6	-0.1709	+0.9853j	1.0000	0.	
MA.7	0.5500	-0.8346j	1.0000	0.	

We can see that our coefs are almost all good for evaluating our model since they are far from 0 and only 4 of them have p-value > 0.5

```
# plotting the predictions vs the actual
result.plot_predict( start=1, end=100, dynamic=False) # plotting the first 100 step
plt.show()
```

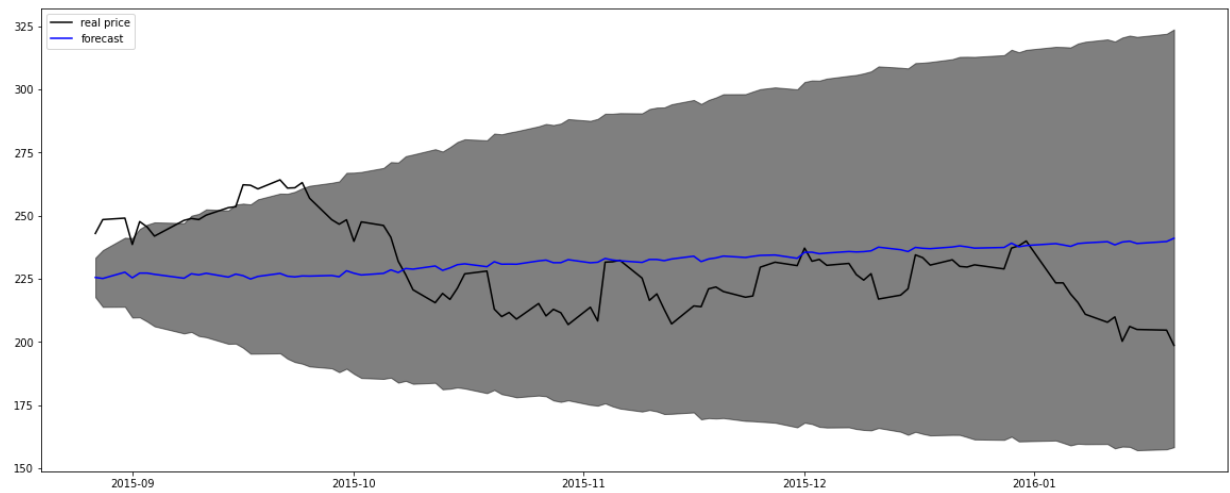


```
# forecasting 100 steps
step = 100
forecast, std_error, confidence = result.forecast(step)
```

we have our forecast, our standard error and the confidence level. We will use these to evaluate our model.

```
forecast = pd.Series(forecast, test[:step].index) # forecast of the next 100 days
lower = pd.Series(confidence[:, 0], test[:step].index) # this will be the lower lim
upper = pd.Series(confidence[:, 1], test[:step].index) # this will be the upper lim
```

```
plt.figure(figsize=(20,8))
plt.plot(test[:step], label="real price", color = "k") # plotting the real price
plt.plot(forecast, label="forecast", color = "b") # plotting the forecast
plt.fill_between(lower.index, lower, upper, color="k", alpha=0.5) # filling with a
plt.legend(loc="upper left")
plt.show()
```



The actual price is in the confidence interval which means that theoretically we could find an optimal arima model to predict stock prices.

But it is still not a profitable predictor

✓ 0s completed at 00:04

● ✕

## 4.2 Γραμμική Παλινδρόμηση

Η δεύτερη προσέγγιση για την λύση του προβλήματος μας είναι να χρησιμοποιήσουμε Γραμμική παλινδρόμηση.

Ο στόχος ( target ) θα είναι το :

- Close : Η τιμή της μετοχής στο κλείσιμο του χρηματιστηρίου.

Έπειτα θα χρησιμοποιήσουμε αρκετά χαρακτηριστικά ( features ) τα οποία θα πρέπει να δημιουργήσουμε από τα στοιχεία που μας δίνονται στο σύνολο δεδομένων μας.

Τα ήδη υπάρχοντα χαρακτηριστικά που θα χρησιμοποιήσουμε είναι :

- Open : Η τιμή της μετοχής στο άνοιγμα του χρηματιστηρίου.
- Prev.Close : Η τιμή της μετοχής στο κλείσιμο του χρηματιστηρίου την προηγούμενη μέρα.
- Prev.Volume : Το άθροισμα των ενεργών μετοχών την προηγούμενη ημέρα.

Στην συνέχεια κατασκευάζουμε κάποια χρήσιμα χαρακτηριστικά χρησιμοποιώντας τα παραπάνω :

- Για να εκπαιδεύσουμε το μοντέλο μας έτσι ώστε να καταλάβει όσο το δυνατόν περισσότερο την τάση του χρηματιστηρίου προσθέτουμε ως χαρακτηριστικά τους απλούς κινητούς μέσους (SMA : Simple Moving Average ) των 5, 10, 20, 50, 100 και 200 ημερών τους οποίους τους υπολογίζουμε μέσω του χαρακτηριστικού Prev.Close.
- Έπειτα δημιουργούμε έναν βραχυπρόθεσμο και έναν μακροπρόθεσμο εκθετικό κινητό μέσο ( EMA exponential moving average ) από τους οποίους θα χρησιμοποιήσουμε ως χαρακτηριστικό την **διαφορά** τους η οποία εκφράζει έναν δείκτη τάσης των τιμών (Moving Average Convergence Divergence (MACD)).
- Στην συνέχεια χρησιμοποιούμε ως χαρακτηριστικό την τάση του δείκτη MACD υπολογίζοντας τον εκθετικό του κινητό μέσο, αυτό ονομάζεται **σήμα** ( signal ). Αυτό εκφράζει έναν πιο κανονικοποιημένο δείκτη τάσης.
- Με σκοπό να εκπαιδεύσουμε το μοντέλο μας να υπολογίζει την τιμή σε μία απλοποιημένη μορφή υπολογίζουμε τον δείκτη σχετικής δύναμης (RSI Relative Strength Index). Όπου

$$RSI = 100 - \frac{100}{1 + RS}$$



, με  $RS = \frac{EMA(1\{\delta > 0\}\delta)}{EMA(-1\{\delta < 0\}\delta)}$  και  $\delta = Prev\_Close_t - Prev\_Close_{t-1}$ .

Με την ίδια λογική υπολογίζουμε το RSI για το Volume

- Θέλουμε να ξέρουμε πότε η μετοχή φτάνει σε ακραίες τιμές έτσι ώστε το μοντέλο μας να μπορεί να προβλέψει την πιθανότητα του ενδεχομένου επιστροφής της μετοχής στην προηγούμενη τιμή. Αυτό μπορούμε να το πετύχουμε προσθέτοντας Bollinger Bands, όπου αυτά στην ουσία είναι δύο δεσμίδες σε απόσταση διπλάσια της τυπικής απόκλισης από την μέση τιμή.
- Τέλος προσθέτουμε ως χαρακτηριστικά τις ποσοστιαίες μεταβολές που έχουν γίνει σε ένα παράθυρο χρόνου  $x$ . Συγκεκριμένα επιλέγουμε τα  $x = 1, 2, 5, 10$

Έχοντας όλα τα παραπάνω χαρακτηριστικά, εκπαιδεύουμε το μοντέλο γραμμικής παλινδρόμησης για να υπολογίσουμε τα βάρη (parameters/weights)  $\mathbf{w} = (w_0, w_1, \dots, w_N)$ . Μέσα στο διάνυσμα αυτό έχουμε ενσωματώσει και την σταθερά (bias)  $w_0 = b$ . Εάν τα χαρακτηριστικά μας είναι  $x_0, x_1, x_2, \dots, x_N$ , όπου  $x_0 = 1$ , τότε έχουμε ότι ισχύει:

$$Close_{prediction} = \sum_{k=0}^N w_k \cdot x_k$$

Τα βάρη (weights) και η σταθερά (bias) θα υπολογισθούν βάσει του αλγορίθμου Gradient Decent :

$$\mathbf{w} = \mathbf{w} - \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

οπού

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^N (Close_{prediction} - Close)^2$$

Παρακάτω βρίσκεται η υλοποίηση της Γραμμικής παλινδρόμησης με την χρήση της python 3 χρησιμοποιώντας το notebook colab

## ▼ STOCK PREDICTION USING LINEAR REGRESSION

```
# uploading file to colab
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed

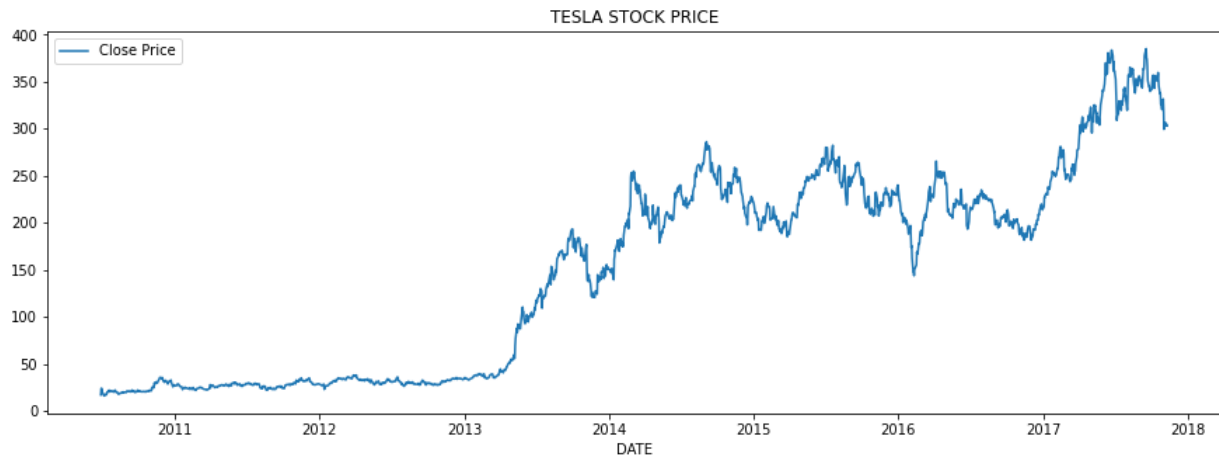
Saving tsla.us.txt to tsla.us (2).txt  
 Saving tslf.us.txt to tslf.us.txt  
 Saving tslx.us.txt to tslx.us.txt  
 Saving tsm.us.txt to tsm.us.txt  
 Saving tsn.us.txt to tsn.us.txt  
 Saving tsq.us.txt to tsq.us.txt  
 Saving tsri.us.txt to tsri.us.txt  
 Saving tsro.us.txt to tsro.us.txt

```
!pip install pandas_ta
from sklearn.linear_model import LinearRegression # importing for linear reg model
from sklearn import metrics # importing for evaluating models
from sklearn.model_selection import train_test_split # importing for splitting the
import matplotlib.pyplot as plt # for plots
import numpy as np # for general math calculations
from numpy import linalg as la # for linear algebra calculations
import pandas as pd # for handling datasets
import pandas_ta # for adding moving average indicators
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-v>  
 Requirement already satisfied: pandas\_ta in /usr/local/lib/python3.7/dist-pack  
 Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-package  
 Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-  
 Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p  
 Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/pythor  
 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa

```
df = pd.read_csv("tsla.us.txt") # reading the csv file
df.set_index(pd.DatetimeIndex(df['Date']), inplace=True) # Reindex data using a Dat
df = df[["Open", "Close", "Volume"]] # holding only Open Close and Volume
# plotting the stock price movement
plt.figure(figsize=(15,5)) # changing the size of the figure
plt.plot(df["Close"], label="Close Price") # plotting the Close of the stock
plt.title("TESLA STOCK PRICE") # titling the graph
plt.xlabel("DATE") # putting the label DATE in the x axis
plt.legend() # adding the labels "Stock Price" and "EMA 10" in the graph
plt.show() # showing the graph
```





```
# Adding some new columns
df["Prev_Close"] = df.loc[:, "Close"].shift(1) # adding the previous day close
df["Prev_Volume"] = df.loc[:, "Volume"].shift(1) # adding the previous day volume

# this function will compare moving averages to get data out of them
def calc_macd(data, len1, len2, len3):
    shortEMA = data.ewm(span=len1, adjust=False).mean()
    longEMA = data.ewm(span=len2, adjust=False).mean()

    MACD = shortEMA - longEMA # this will indicate the trend of the stock by comparin
    signal = MACD.ewm(span=len3, adjust=False).mean() # ema of the macd

    return MACD, signal

# this is the relative stock index and it says if a stock is overbought or oversold
def calc_rsi(data, period):
    delta = data.diff() # the difference of this day and the previous one
    # we want to know when the diffrence is above 0 and when it is below 0
    up = delta.clip(lower=0) # all positive points stay the same and all negative are
    down = -1*delta.clip(upper=0) # the opposite

    # now to see the trend we will calculate the ema
    ema_up = up.ewm(com=period, adjust=False).mean()
    ema_down = down.ewm(com=period, adjust=False).mean()

    rs = ema_up / ema_down # that is the relative strength number
    rsi = 100 - (100/(1+rs))

    return rsi

# this function creates two bands around the mean of the stock that are at distance
# and therefore if the stock reaches is one then there is a high possibility to bou
def calc_bollinger(data, period):
    mean = data.rolling(period).mean()
    std = data.rolling(period).std()

    upper_band = np.array(mean) + 2 * np.array(std)
    lower_band = np.array(mean) - 2 * np.array(std)
```

```

return upper_band, lower_band

# Adding SMA's to the df short term and long term
df["5SMA"] = df["Prev_Close"].rolling(5).mean()
df["10SMA"] = df["Prev_Close"].rolling(10).mean()
df["20SMA"] = df["Prev_Close"].rolling(20).mean()
df["50SMA"] = df["Prev_Close"].rolling(50).mean()
df["100SMA"] = df["Prev_Close"].rolling(100).mean()
df["200SMA"] = df["Prev_Close"].rolling(200).mean()

MACD, signal = calc_macd(df["Prev_Close"], 12, 26, 9) # calculating the macd(differ
df["MACD"] = MACD # adding macd
df["MACD_signal"] = signal # adding signal

df["RSI"] = calc_rsi(df["Prev_Close"], 13)
df["RSI_Volume"] = calc_rsi(df["Prev_Volume"], 13)

upper, lower = calc_bollinger(df["Prev_Close"], 20)
df["Upper_Band"] = upper
df["Lower_Band"] = lower

# calculating the percent changes through #period days and adding it to the df
labels = ["Prev_Close", "Prev_Volume", "200SMA", "5SMA", "10SMA", "20SMA", "50SM
          "100SMA", "MACD", "MACD_signal", "RSI", "RSI_Volume", "Upper_Band", "Lo

period = 1
new_labels = [str(period) + "d_" + label for label in labels]
df[new_labels] = df[labels].pct_change(period, fill_method="ffill")

period = 2
new_labels = [str(period) + "d_" + label for label in labels]
df[new_labels] = df[labels].pct_change(period, fill_method="ffill")

period = 5
new_labels = [str(period) + "d_" + label for label in labels]
df[new_labels] = df[labels].pct_change(period, fill_method="ffill")

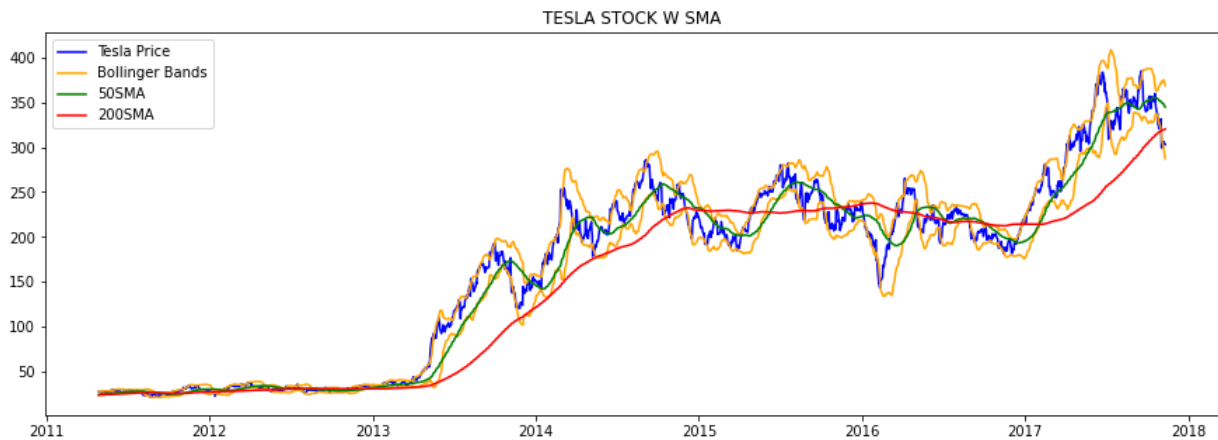
period = 10
new_labels = [str(period) + "d_" + label for label in labels]
df[new_labels] = df[labels].pct_change(period, fill_method="ffill")

# dropping the NaN
df = df.replace(np.inf, np.nan).dropna()

# visualizing some simple averages
plt.figure(figsize=(15,5)) # adjusting figsize
plt.title("TESLA STOCK W SMA")
plt.plot(df["Close"], color="blue", label="Tesla Price")
plt.plot(df["Upper_Band"], color="orange", label="Bollinger Bands")
plt.plot(df["Lower_Band"], color="orange")
plt.plot(df["50SMA"], color="green", label="50SMA")
plt.plot(df["200SMA"], color="red", label="200SMA")

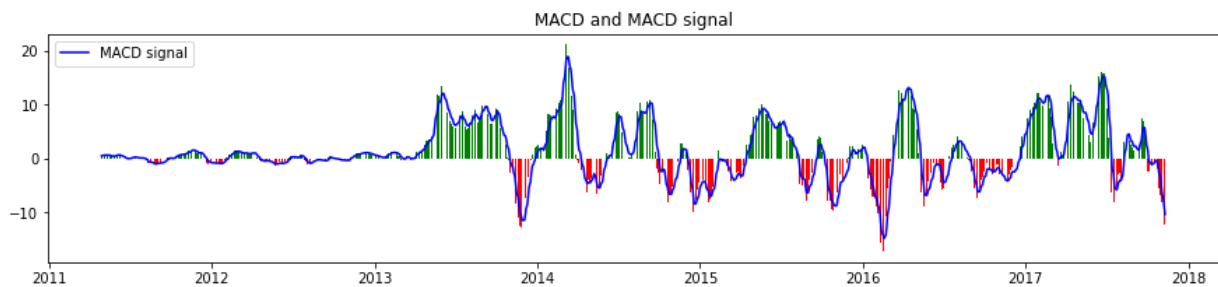
```

```
plt.legend(loc="upper left")
plt.show()
```



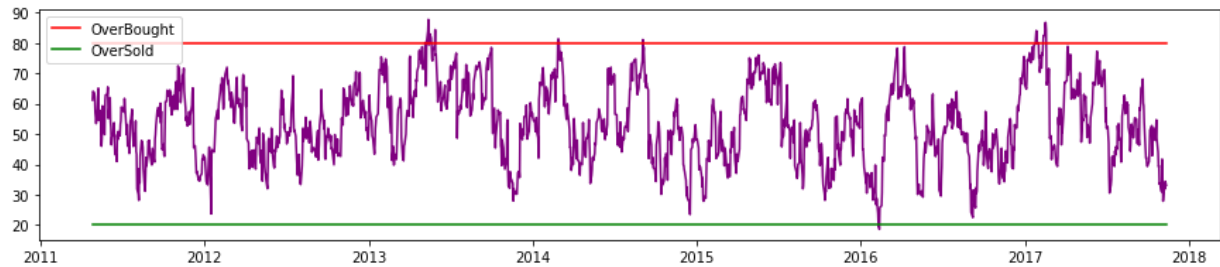
```
# plotting the MACD and the signal
plt.figure(figsize=(15,3))
# if > 0 green if < 0 red
colors = np.array(["green"] * len(df["MACD"])) # making an array of greens
colors[df["MACD"] < 0] = "red"

plt.title("MACD and MACD signal")
plt.bar(df.index.values,df["MACD"], color=colors)
plt.plot(df["MACD_signal"], color="blue", label="MACD signal")
plt.legend(loc="upper left")
plt.show()
```



```
# this graph shows if a stock is overbought or oversold ( if it touches the green t
plt.figure(figsize=(15,3))
plt.plot(df["RSI"], color="purple")
plt.plot( [df.index.values[0], df.index.values[-1] ] , [80,80], color="red", label=
plt.plot( [df.index.values[0], df.index.values[-1] ] , [20,20], color="green", label
```

```
plt.legend(loc="upper left")
plt.show()
```



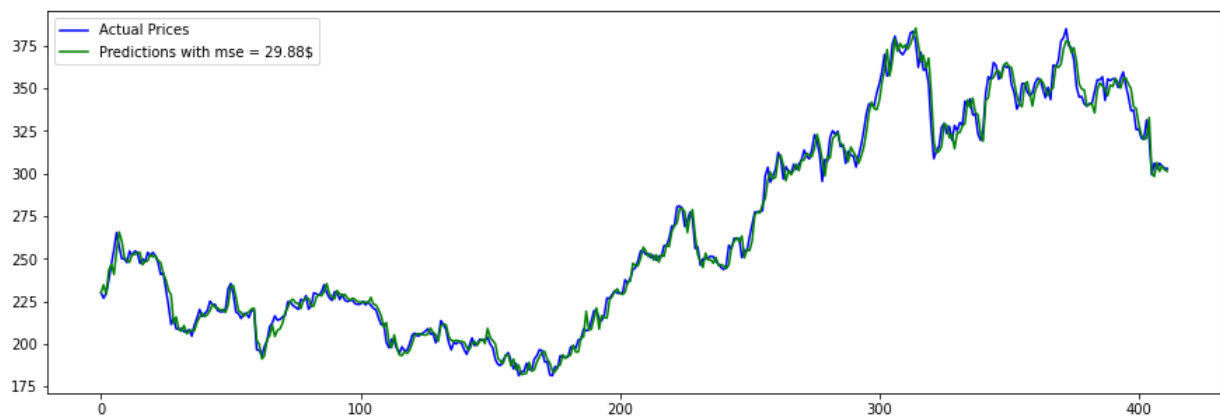
```
y = df["Close"]
X = df.drop(["Close", "Volume"], axis=1).values

# splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, shuffle=F)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = metrics.mean_squared_error(y_test, y_pred)

plt.figure(figsize=(15,5))
plt.plot(range(len(y_test)), y_test, color="blue", label="Actual Prices")
plt.plot(range(len(y_pred)), y_pred, color="green", label="Predictions with mse = {
plt.legend(loc="upper left")
plt.show()
```



```
# Creating a virtual account too see if we would win money with those preds
def VR_trade(opens, closes, preds, start_acc=1000, thresh=0):
    acc = start_acc
    changes = []
```

```

for i in range(len(preds)):
    if (preds[i] - opens[i])/opens[i] >= thresh:
        acc += acc*(closes[i] - opens[i])/opens[i]
    changes.append(acc)

changes = np.array(changes)

plt.figure(figsize=(15,5))
plt.plot(range(len(changes)), changes)
plt.show()

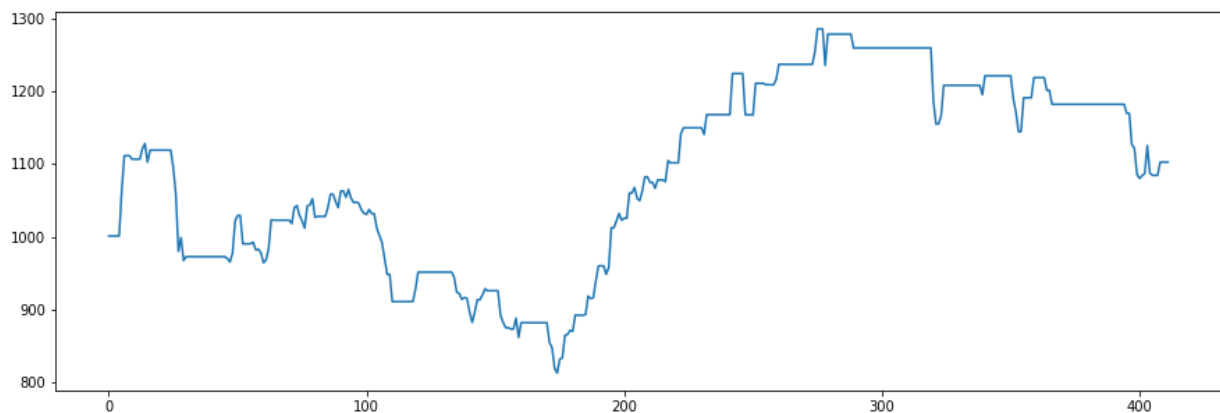
# this is the money we would have won without using the model
invest_total = start_acc + start_acc * ( closes[-1] - opens[0] ) / opens[0]

print("Investing Total : ", round(invest_total,2), \
      str(round((invest_total - start_acc)/start_acc*100,1)) + "%")

print("Algo-Trading Total : ", round(acc,2), \
      str(round((acc - start_acc)/start_acc*100,1)) + "%")

```

```
VR_trade(X_test.T[0], y_test, y_pred, 1000, 0)
```




```

Investing Total : 1317.98 31.8%
Algo-Trading Total : 1102.44 10.2%

```

---

 40s    completed at 09:16



### 4.3 Νευρωνικό Δίκτυο LSTM

Η τρίτη προσέγγιση για την λύση του προβλήματος μας ήταν η χρήση του Επαναλαμβανόμενου Νευρωνικού Δικτύου ( Recurrent Neural Network ) LSTM ( Long-Short-Term-Memory ).

#### 4.3.1 Γιατί LSTM ;

Στην διαδικασία πρόβλεψης των τιμών κάποιας μετοχής του χρηματιστηρίου ένας, ιδιαίτερα σημαντικός, παράγοντας που πρέπει να λάβουμε υπόψιν είναι τα παρελθοντικά δεδομένα της μετοχής αυτής. Ένα οποιοδήποτε άλλο νευρωνικό δίκτυο αποτυγχάνει στην διατήρηση, και έπειτα χρήση, αυτών των δεδομένων. Από την συλλογή των επαναλαμβανόμενων νευρωνικών δικτύων, το LSTM έχει την ικανότητα να διατηρεί , και έπειτα να χρησιμοποιεί, δεδομένα των οποίων το «χρονικό» κενό με το παρόν είναι «μεγάλο». Αυτή η ικανότητα του LSTM είναι διαισθητικά λογικό ότι θα μας φανεί χρήσιμη στην πρόβλεψη των τιμών μίας μετοχής.

#### 4.3.2 Διαδικασία του LSTM

Η διαδικασία που ακολουθεί το LSTM που θα χρησιμοποιήσουμε χωρίζεται σε 4 μέρη (**Σημείωση :** Στην ορολογία των LSTM η μακροπρόθεσμη μνήμη (Long-term Memory) καλείται cell state και έτσι θα χρησιμοποιείται αυτός ο όρος από εδώ και πέρα ).

1. Αρχικά επιλέγουμε τι πληροφορία θα κρατήσουμε (δηλ. τι θα επιλέξουμε να «θυμόμαστε» ) και τι πληροφορία θα αφαιρέσουμε (δηλ. θα επιλέξουμε να «ξεχάσουμε») από το cell state  $C_{t-1}$  . Αυτό θα το επιτύχουμε μέσω ενός layer με σιγμοειδή συνάρτηση ενεργοποίησης (sigmoid activation function) όπου τα ακραία σημεία ως έξοδο τα αντιλαμβανόμαστε ως εξής :

$1 \rightarrow$  το κρατάμε πλήρως

$0 \rightarrow$  το αφαιρούμε πλήρως

2. Στο δεύτερο βήμα αποφασίζουμε εάν υπάρχει κάποια καινούργια πληροφορία που χρειάζεται να αποθηκεύσουμε στο cell state  $C_t$ . Αυτό το επιτυγχάνουμε χρησιμοποιώντας 2 layers, το πρώτο layer έχει σιγμοειδή συνάρτηση ενεργοποίησης και μέσω αυτού επιλέγουμε ποιές τιμές θα ενημερώσουμε. Το δεύτερο layer έχει υπερβολική εφαιπόμενη ως συνάρτηση ενεργοποίησης ( $\tanh$ ) και από αυτό θα δημιουργηθεί ένα διάνυσμα,  $\tilde{C}_t$ , με νέες υποψήφιες τιμές που μπορούν να προστεθούν στο cell state  $C_t$
3. Σε αυτό το σημείο έχουμε ήδη αποφασίσει τι πληροφορία θα κρατήσουμε και τι πληροφορία θα αφαιρέσουμε, και άρα μπορούμε πλέον να υπολογίσουμε ποίο θα είναι το καινούργιο cell state  $C_t$ . Έχουμε :  $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$  όπου

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

,

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

και

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

4. Τέλος, επιλέγουμε την πληροφορία που θα εξάγουμε ( στην περίπτωση μας την τιμή της μετοχής ). Πρώτα θα χρησιμοποιήσουμε ένα layer με σιγμοειδή συνάρτηση ενεργοποίησης το οποίο θα αποφασίσει ποία μέρη του cell state θα εξάγουμε, έπειτα περνάμε το cell state από μία υπερβολική εφαπτομένη  $\tanh$  η οποία κανονικοποιεί την πληροφορία του cell state μεταξύ των τιμών -1 και 1. Πολλαπλασιάζοντας αυτά τα δύο μεγέθη θα έχουμε την συνάρτηση που θα παράγει τις προβλέψεις μας.

Έχουμε :  $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

και άρα

$$h_t = o_t * \tanh(C_t)$$

Για αυτό το νευρωνικό δίκτυο θα χρησιμοποιήσουμε τον βελτιστοποιητή Adam όπου :

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

με  $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ ,  $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

Και ως συνάρτηση απώλειας την Mean Squared Error Όπου

$$MSE = \frac{1}{N} \sum_{i=1}^N (Y_i - \hat{Y}_i)^2$$

Παρακάτω βρίσκεται η υλοποίηση του LSTM με την χρήση της python 3 χρησιμοποιώντας το notebook colab

## ▼ LSTM STOCK PREDICTION

```
# uploading file to colab
from google.colab import files
uploaded = files.upload()
```

Choose Files **tsla.us.txt**

- **tsla.us.txt**(text/plain) - 87266 bytes, last modified: 9/21/2019 - 100% done  
Saving tsla.us.txt to tsla.us (2).txt

```
from sklearn import metrics # importing for evaluating models
from sklearn.model_selection import train_test_split # importing for splitting the
from sklearn.preprocessing import MinMaxScaler # for normalization in the lstm mode
import matplotlib.pyplot as plt # for plots
import numpy as np # for general math calculations
from numpy import linalg as la # for linear algebra calculations
import pandas as pd # for handling datasets
from keras.models import Sequential # for creating a neural network
from keras.layers import Dense, Dropout, LSTM # for creating the LSTM model
```

```
df = pd.read_csv("tsla.us.txt")
```

```
# for the LSTM NN we want to scale our data from 0 to 1 using the min-max scaler
scaler = MinMaxScaler(feature_range=(0, 1)) # creating a minmax scaler object that
data = scaler.fit_transform(df["Close"].values.reshape(-1,1)) # scaling our data fr
train_len = int( len(data) * 0.7 )
```

```
#splitting the data
train_data = data[0: train_len]
test_data = data[train_len::]
```

```
# =====
#                               TRAINING MODEL
# =====
```

```
pred_days = 100 # how many days do we want to look in the past
```

```
x_train_NN = []
y_train_NN = []
```

```
x_test_NN = []
y_test_NN = []
```

```
# creating the train data by putting in training the 60 days prior to the one we wa
for x in range(pred_days, len(train_data)):
    x_train_NN.append(train_data[x - pred_days:x, 0])
    y_train_NN.append(train_data[x, 0])
```

```
for x in range(pred_days, len(test_data)):
```

```

x_test_NN.append(test_data[x - pred_days:x, 0])
y_test_NN.append(test_data[x, 0])

# making them arrays and reshaping them so as to work with nn's
x_train_NN = np.array(x_train_NN)
y_train_NN = np.array(y_train_NN)

x_train_NN = np.reshape(x_train_NN, (x_train_NN.shape[0], x_train_NN.shape[1], 1))

x_test_NN = np.array(x_test_NN)
y_test_NN = np.array(y_test_NN)

x_test_NN = np.reshape(x_test_NN, (x_test_NN.shape[0], x_test_NN.shape[1], 1))

# building the model
model = Sequential()

model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train_NN.shape[1], 1)
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1)) # prediction

model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(x_train_NN, y_train_NN, epochs=200, batch_size=32)

38/38 [=====] - 6s 150ms/step - loss: 8.0511e-04
Epoch 173/200
38/38 [=====] - 6s 150ms/step - loss: 6.3752e-04
Epoch 174/200
38/38 [=====] - 6s 150ms/step - loss: 6.5863e-04
Epoch 175/200
38/38 [=====] - 6s 151ms/step - loss: 6.9762e-04
Epoch 176/200
38/38 [=====] - 6s 151ms/step - loss: 7.4202e-04
Epoch 177/200
38/38 [=====] - 6s 151ms/step - loss: 6.3334e-04
Epoch 178/200
38/38 [=====] - 6s 150ms/step - loss: 6.4882e-04
Epoch 179/200
38/38 [=====] - 6s 150ms/step - loss: 6.5335e-04
Epoch 180/200
38/38 [=====] - 6s 149ms/step - loss: 6.3260e-04
Epoch 181/200
38/38 [=====] - 6s 150ms/step - loss: 6.3055e-04
Epoch 182/200
38/38 [=====] - 6s 150ms/step - loss: 6.8660e-04
Epoch 183/200
38/38 [=====] - 6s 150ms/step - loss: 7.5219e-04
Epoch 184/200
38/38 [=====] - 6s 150ms/step - loss: 6.8137e-04
Epoch 185/200
38/38 [=====] - 6s 151ms/step - loss: 6.1358e-04
Epoch 186/200
38/38 [=====] - 6s 149ms/step - loss: 6.8935e-04
Epoch 187/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 188/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 189/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 190/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 191/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 192/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 193/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 194/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 195/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 196/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 197/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 198/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 199/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04
Epoch 200/200
38/38 [=====] - 6s 149ms/step - loss: 6.8136e-04

```

```

30/30 [-----] - 6s 149ms/step - loss: 0.0130e-04
Epoch 188/200
38/38 [=====] - 6s 150ms/step - loss: 7.7389e-04
Epoch 189/200
38/38 [=====] - 6s 149ms/step - loss: 6.2971e-04
Epoch 190/200
38/38 [=====] - 6s 151ms/step - loss: 5.9317e-04
Epoch 191/200
38/38 [=====] - 6s 149ms/step - loss: 6.9521e-04
Epoch 192/200
38/38 [=====] - 6s 151ms/step - loss: 7.2324e-04
Epoch 193/200
38/38 [=====] - 6s 151ms/step - loss: 6.8996e-04
Epoch 194/200
38/38 [=====] - 6s 149ms/step - loss: 6.9575e-04
Epoch 195/200
38/38 [=====] - 6s 151ms/step - loss: 6.5734e-04
Epoch 196/200
38/38 [=====] - 6s 150ms/step - loss: 6.8523e-04
Epoch 197/200
38/38 [=====] - 6s 150ms/step - loss: 7.6940e-04
Epoch 198/200
38/38 [=====] - 6s 149ms/step - loss: 6.9703e-04
Epoch 199/200
38/38 [=====] - 6s 150ms/step - loss: 6.4809e-04
Epoch 200/200
38/38 [=====] - 6s 149ms/step - loss: 5.6588e-04
<keras.callbacks.History at 0x7f0d9042a490>

```

```

# =====
#                               TESTING MODEL
# =====

```

```

pred_prices = model.predict(x_test_NN)
real_prices = y_test_NN

pred_prices = scaler.inverse_transform(pred_prices) # denormalizing the predicted p
real_prices = real_prices.reshape((458,1))
real_prices = scaler.inverse_transform(real_prices) # denormalizing the real prices

mse = metrics.mean_squared_error(real_prices, pred_prices)

# Plotting the data
plt.figure(figsize=(20,8))

plt.plot(real_prices, label="Real Prices")
plt.plot(pred_prices, label="Predicted Prices with MSE = {0:.2f}".format(mse))
plt.legend()
plt.show()

```



```
print("Mean Squared Error: {0:.2f}$".format(metrics.mean_squared_error(real_prices,
```

```
Mean Squared Error: 231.24$
```

```
# Creating a virtual account too see if we would win money with those preds
```

```
def VR_trade(opens, closes, preds, start_acc=1000, thresh=0):
```

```
    acc = start_acc
```

```
    changes = []
```

```
    for i in range(len(preds)):
```

```
        if (preds[i] - opens[i])/opens[i] >= thresh:
```

```
            acc += acc*(closes[i] - opens[i])/opens[i]
```

```
            changes.append(acc)
```

```
    changes = np.array(changes)
```

```
    plt.figure(figsize=(15,5))
```

```
    plt.plot(range(len(changes)), changes)
```

```
    plt.show()
```

```
# this is the money we would have won without using the model
```

```
invest_total = start_acc + start_acc * ( closes[-1] - opens[0] ) / opens[0]
```

```
print("Investing Total : ", round(invest_total,2), \
```

```
      str(round((invest_total - start_acc)/start_acc*100,1)) + "%")
```

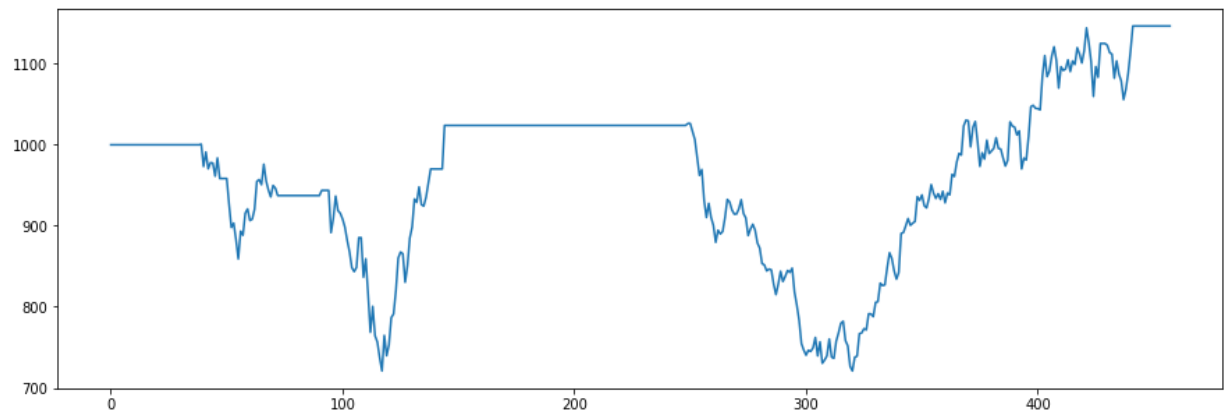
```
print("Algo-Trading Total : ", round(acc,2), \
```

```
      str(round((acc - start_acc)/start_acc*100,1)) + "%")
```

```
opens = df["Open"][train_len:].values
```

```
closes = df["Close"][train_len:].values
```

```
VR_trade(opens, closes, pred_prices)
```



Investing Total : 1311.65 31.2%  
Algo-Trading Total : 1146.63 14.7%

✓ 0s completed at 09:32

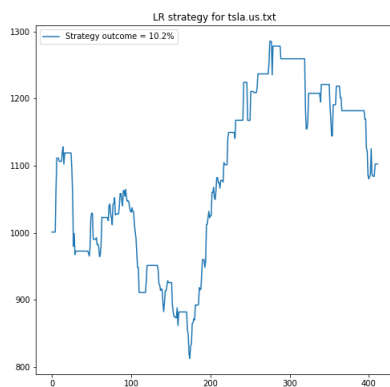
● ✕

## 5 Συμπεράσματα :

Από ότι είδαμε, η γραμμική παλινδρόμηση και το νευρωνικό δίκτυο LSTM φαίνεται να εκπαιδεύονται αρκετά ικανοποιητικά. Όμως αυτό που μας ενδιαφέρει είναι αν οι στρατηγικές που προκύπτουν είναι πιο κερδοφόρες από το εάν απλώς είχαμε επενδύσει χωρίς να ακολουθήσουμε κάποια στρατηγική. Έτσι φτιάξαμε μία προσομοίωση στην οποία έχουμε έναν υποθετικό λογαριασμό αξίας 1000 δολάρια έτσι ώστε να συγκρίνουμε αυτές τις στρατηγικές

- TESLA :

1. Χωρίς στρατηγική : + 31.2 %
2. Linear Regression : + 10.2 %



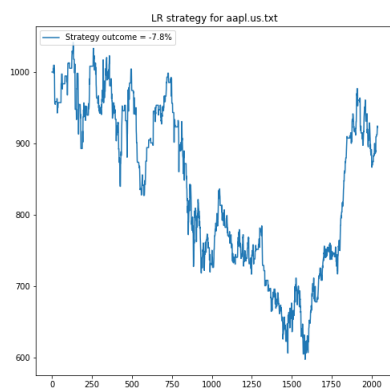
3. LSTM : + 12 %



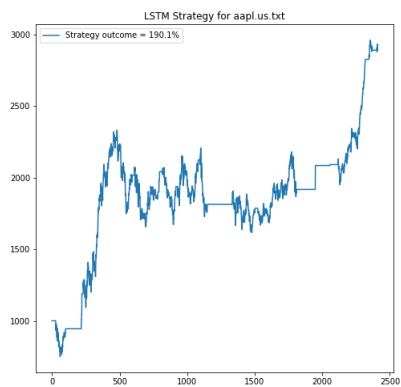


- APPLE :

1. Χωρίς στρατηγική : + 632.5 %
2. Linear Regression : -7.8 %

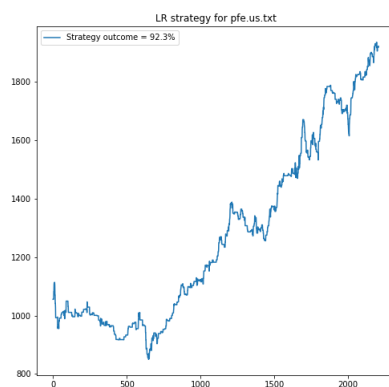


3. LSTM : +190.1 %

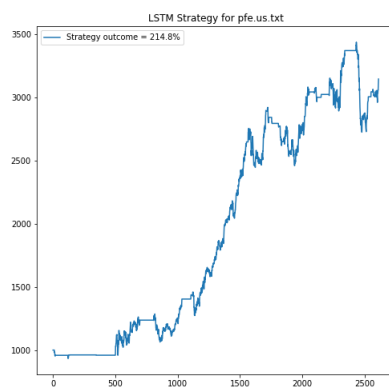


- PFIZER :

1. Χωρίς στρατηγική : +212.2 %
2. Linear Regression : +92.3 %

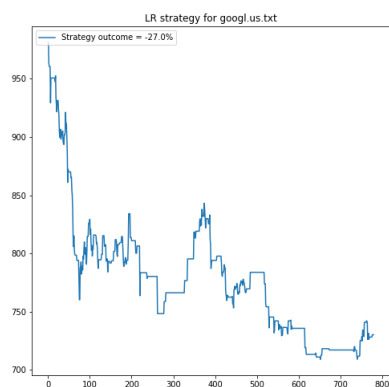


3. LSTM : +214.8 %

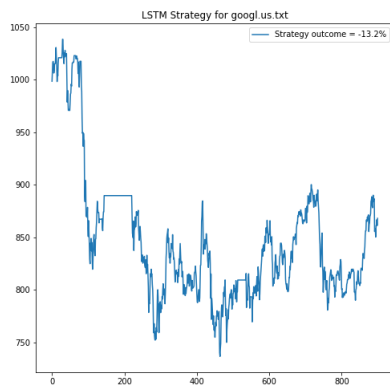


- GOOGLE :

1. Χωρίς στρατηγική : +79.5 %
2. Linear Regression : -27%



3. LSTM : -13.2 %



Παρατηρούμε ότι γενικά η στρατηγική του LSTM έχει καλύτερη απόδοση από την στρατηγική της γραμμικής παλινδρόμησης, και τα δύο όμως έχουν χειρότερη απόδοση από το να μην ακολουθούσαμε στρατηγική ( εκτός της PFIZER )

## 6 Βιβλιογραφία

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow; O'Reilly Media, 2020
- <https://iopscience.iop.org/article/10.1088/1742-6596/2161/1/012065/pdf>
- <https://kesmarag.gitlab.io/courses/mem205/>
- H. L. Siew and M. J. Nordin, "Regression techniques for the prediction of stock price trend," 2012 International Conference on Statistics in Science, Business and Engineering (ICSSBE), Langkawi, 2012, pp. 1-5.
- <https://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR3u8q2ccmPx2DjuO2wsUUQXkLhzbuzUWDs4hdsorLxPWLxsGBiP6OXHQ>
- <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>