Colab link: https://colab.research.google.com/drive/1TilDy1kdyprlLbIXqU5ifQlOoxmBP0Oq

In [0]:

```
#enabling GPU to make this colab run faster
!nvidia-smi
```

```
Thu Nov 28 00:07:09 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   47C    P8    29W / 149W |      0MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

In [0]:

```
#Importing all the neccessary packages
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [0]:

```
#install gdown in order to avoing google drive authorization everytime
import os.path as path
if not path.exists('M3final.zip'):
 !pip install gdown
 !gdown https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
 !unzip M3final.zip
else :
 print('data is in places')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.6/dist-packages (3.6.4)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from g
down) (2.21.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from gdown
) (4.28.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gdown)
(1.12.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packag
es (from requests->gdown) (2019.9.11)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (
from requests->gdown) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
kages (from requests->gdown) (3.0.4)
Downloading...
From: https://drive.google.com/uc?id=1w4Sf7GXSzVs5Gcci9Dx2GqiT96A0OeeU
To: /content/M3final.zip
31.8MB [00:00, 41.5MB/s]
Archive:  M3final.zip
```

```
  inflating: fundamentals.csv
  inflating: __MACOSX/._fundamentals.csv
  inflating: prices-split-adjusted.csv
  inflating: __MACOSX/._prices-split-adjusted.csv
  inflating: prices.csv
  inflating: __MACOSX/._prices.csv
  inflating: securities.csv
  inflating: __MACOSX/._securities.csv
```

In [0]:

```
#loading the data
df = pd.read_csv("prices-split-adjusted.csv")
```

In [0]:

```
#checking the dataframe
df.head()
```

Out[0]:

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

In [0]:

```
#printing the amount of different stocks in our dataframe and looking and the companies s
tocks codes
print('\n Number of stocks: ', len(list(set(df.symbol))))
print('Some stock symbols: ', list(set(df.symbol))[:])
```

```
 Number of stocks:  501
Some stock symbols:  ['APC', 'NOC', 'CVS', 'IP', 'ILMN', 'NFX', 'EQIX', 'SLG', 'ISRG', 'B
WA', 'PCG', 'AIZ', 'IDXX', 'BLL', 'PYPL', 'VAR', 'K', 'AMT', 'SYF', 'KMB', 'COG', 'IRM',
'SWK', 'PHM', 'LNC', 'CNP', 'ESS', 'NVDA', 'FOX', 'CAG', 'GT', 'NWSA', 'FB', 'HBI', 'LEG'
, 'GE', 'ORLY', 'DLPH', 'PAYX', 'NOV', 'AYI', 'M', 'FTI', 'CBS', 'ABC', 'DD', 'GOOGL', 'E
A', 'EIX', 'KSU', 'HIG', 'CMS', 'CSCO', 'UDR', 'SCG', 'OXY', 'AVB', 'PNC', 'KSS', 'EBAY',
'RIG', 'VIAB', 'WYN', 'BAC', 'CVX', 'GS', 'WMB', 'WBA', 'TDG', 'CCL', 'AVGO', 'SLB', 'CSR
A', 'ICE', 'FAST', 'PSA', 'FMC', 'DVA', 'PFG', 'COL', 'TXT', 'FE', 'PRU', 'AMP', 'EMR', '
LUV', 'ABT', 'CME', 'DLTR', 'NFLX', 'NI', 'OKE', 'URI', 'ROST', 'NSC', 'CNC', 'HCA', 'SWK
S', 'ROP', 'ALB', 'SHW', 'HCN', 'KLAC', 'KHC', 'DHR', 'TGNA', 'MPC', 'LB', 'HPQ', 'MCK',
'UNP', 'NTAP', 'ECL', 'DISCK', 'SYK', 'ESRX', 'TWX', 'HAL', 'INTC', 'AKAM', 'XYL', 'EXC',
'DTE', 'SPLS', 'LMT', 'NUE', 'IFF', 'DAL', 'COP', 'HSY', 'TSN', 'ALLE', 'HOG', 'VRSK', 'C
ELG', 'FLIR', 'DISCA', 'BDX', 'GIS', 'PWR', 'ALXN', 'CA', 'MMM', 'NKE', 'JBHT', 'MAA', 'Q
COM', 'XRAY', 'FISV', 'STI', 'HRL', 'CL', 'AES', 'SJM', 'MAS', 'RL', 'LKQ', 'RCL', 'ALL',
'LOW', 'TMK', 'PPL', 'MU', 'CSX', 'MAC', 'WLTW', 'LUK', 'PGR', 'BBBY', 'MNST', 'CMA', 'DE
', 'ENDP', 'MTB', 'PXD', 'CLX', 'JCI', 'UHS', 'PG', 'NEM', 'RAI', 'BCR', 'HAS', 'PVH', 'L
', 'BEN', 'AFL', 'PEP', 'GPS', 'ATVI', 'EXPD', 'KORS', 'ORCL', 'XRX', 'JEC', 'WM', 'EOG',
'MSFT', 'LNT', 'GPC', 'CTSH', 'SE', 'KEY', 'BMY', 'TJX', 'PBI', 'XOM', 'HCP', 'GILD', 'LE
N', 'NWS', 'ARNC', 'TEL', 'ABBV', 'DOV', 'HRB', 'PPG', 'VLO', 'SPG', 'HD', 'CHK', 'BHI',
'WRK', 'FSLR', 'NEE', 'UAL', 'MOS', 'AJG', 'TRV', 'TXN', 'GWW', 'LVLT', 'AAL', 'BLK', 'OM
C', 'UTX', 'AEP', 'VRSN', 'UPS', 'TRIP', 'CHTR', 'ADSK', 'BBY', 'TAP', 'CI', 'DRI', 'XEL'
, 'STX', 'TMO', 'PNW', 'CERN', 'APH', 'F', 'JNPR', 'MO', 'CAT', 'CCI', 'WYNN', 'AN', 'DVN
', 'DPS', 'ALK', 'HBAN', 'VMC', 'AWK', 'AET', 'BBT', 'ITW', 'MRO', 'HON', 'RTN', 'CTAS',
'C', 'VTR', 'WFM', 'ZION', 'COO', 'NLSN', 'KR', 'AON', 'TSO', 'NWL', 'MDT', 'TGT', 'UAA',
'CMCSA', 'COH', 'AAP', 'SWN', 'FOXA', 'SCHW', 'FTR', 'RHT', 'FTV', 'WY', 'APA', 'NAVI', '
ANTM', 'FL', 'GOOG', 'MSI', 'APD', 'LLTC', 'XL', 'QRVO', 'AZO', 'AMG', 'SRE', 'TROW', 'XL
NX', 'MCD', 'GRMN', 'PKI', 'LYB', 'DGX', 'DFS', 'HES', 'BAX', 'BA', 'YHOO', 'NRG', 'SNI',
'LLL', 'DIS', 'HUM', 'MAT', 'DHI', 'MCHP', 'CAH', 'DG', 'FRT', 'MJN', 'PNR', 'IBM', 'EVHC
', 'GPN', 'GLW', 'AIV', 'WHR', 'MTD', 'FIS', 'CRM', 'EQT', 'INTU', 'IVZ', 'CPB', 'KMI',
'MHK', 'HRS', 'DNB', 'PLD', 'SO', 'WEC', 'AXP', 'WAT', 'EQR', 'EXR', 'SNA', 'ADI', 'MMC'
, 'RF', 'AME', 'JPM', 'SEE', 'HPE', 'HP', 'VRTX', 'PDCO', 'ADM', 'COST', 'XEC', 'LLY', 'W
FC', 'MAR', 'O', 'PCAR', 'PBCT', 'USB', 'UNH', 'GD', 'URBN', 'MET', 'PCLN', 'CFG', 'FCX',
'HSIC', 'BSX', 'VNO', 'HST', 'IR', 'SPGI', 'ACN', 'HAR', 'R', 'CMI', 'AAPL', 'EW', 'SRCL'
, 'RHI', 'EXPE', 'CBG', 'PFE', 'MA', 'ADP', 'PX', 'ROK', 'MON', 'CTL', 'AIG', 'UNM', 'EL'
, 'BXP', 'AGN', 'KMX', 'PEG', 'KIM', 'FLR', 'AVY', 'CF', 'ED', 'SYMC', 'SIG', 'ETN', 'FDX
', 'ES', 'YUM', 'MKC', 'CINF', 'PM', 'CXO', 'GM', 'ULTA', 'DLR', 'BK', 'NDAQ', 'SYY', 'RS
G', 'REGN', 'CB', 'D', 'AMGN', 'MRK', 'COF', 'DOW', 'ETR', 'WMT', 'SBUX', 'NBL', 'VZ', 'C
HRW', 'NTRS', 'ADS', 'ETFC', 'ZBH', 'COTY', 'HOLX', 'CTXS', 'STT', 'DUK', 'STZ', 'EMN', '
EFX', 'LH', 'ZTS', 'MLM', 'CMG', 'GGP', 'RRC', 'JWN', 'V', 'WU', 'PSX', 'TSCO', 'ADBE', '
PRGO', 'FITB', 'FBHS', 'PH', 'MNK', 'AEE', 'BIIB', 'A', 'IPG', 'VFC', 'MCO', 'TIF', 'FFIV
', 'WDC', 'MYL', 'MUR', 'AMZN', 'KO', 'AMAT', 'JNJ', 'TDC', 'CHD', 'T', 'MDLZ', 'FLS', 'L
RCX', 'TSS']
```

In [0]:

```
#randomly choosing IBM as a company to predict the stocks, but as well could have chosen
Facebook, Google or Amazon
dfIBM = df[df.symbol == 'IBM']
dfIBM.drop(['symbol'],1,inplace=True)
dfIBM.head()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:4117: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

Out[0]:

	date	open	close	low	high	volume
470	2010-01-04	131.179993	132.449997	130.850006	132.970001	6155300.0
938	2010-01-05	131.679993	130.850006	130.100006	131.850006	6841400.0
1406	2010-01-06	130.679993	130.000000	129.809998	131.490005	5605300.0
1874	2010-01-07	129.869995	129.550003	128.910004	130.250000	5840600.0
2342	2010-01-08	129.070007	130.850006	129.050003	130.919998	4197200.0

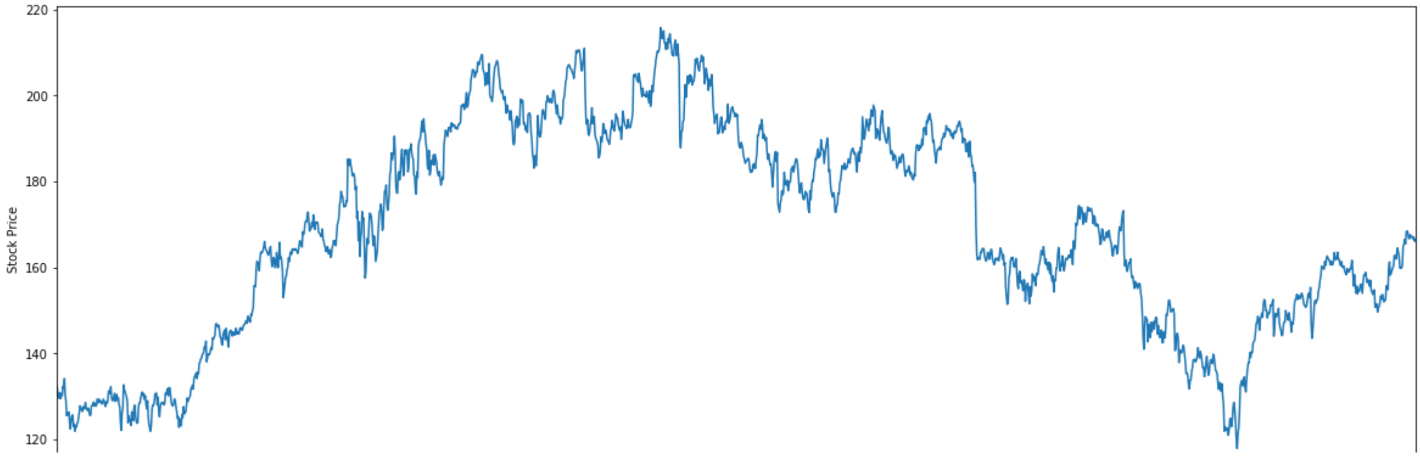
Stock developement

Here we will see how the price of the stocks developed and we will try to get as close to this as possible with our model's prediction

In [0]:

```
#plotting the original stock developement
fig, ax = plt.subplots(figsize=(20,7))
dfIBM.close.plot(ax=ax)
plt.ylabel("Stock Price")
plt.xlabel("Year")

#set ticks every week
ax.xaxis.set_major_locator(mdates.YearLocator())
#set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('%y'))
```



Creating train test

Here we create a train and a test data, we define X and a y.

X is going to be the closing price of the stock each day, which we reshape by using reshape function to get from 1D array a 2D array.

for y we will create a new column called prediction and we will substract the number of predicting days from close column into this. This way we can easily play with the model. The more days we forecast to, the smaller the accuracy score obviously is. For now we leave it for 3 days, but we went all the way from 1 day up to 30 days.

In [0]:

```
forecasted_days = 3
dfIBM["prediction"] = dfIBM[["close"]].shift(-forecasted_days)
X = np.array(dfIBM['close']).reshape(-1,1)
X = X[:-forecasted_days]
y = np.array(dfIBM['prediction'])
y = y[:-forecasted_days]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
```

Using RandomForestRegressor

For our non-neural baseline model we decided to use RandomForestRegressor as it seemed the most appropriate. We are predicting future, therefore some sort of regression is neccessary and RandomForest is a powerful model enough to learn our data well. For 3 days we get the accuracy of 96,7% which is pretty solid.

The MSE is a bit higher than the one in neural net model, but as we studied online, in some cases it happens, that simply from the data you are using and my the choice of model, the MSE cant get lower. Obviously we aim for as low MSE as we can get.

In [0]:

```
reg = RandomForestRegressor(n_estimators = 300, max_depth =300, random_state = 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state =
7)
reg.fit(X_train, y_train)
print("The socre of RandomForest is ", reg.score(X_test, y_test))
```

```
The socre of RandomForest is  0.9670561514856216
```

In [0]:

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, prediction2)
```

Out[0]:

```
19.10229860677159
```

Two predictions

Here we do two predictions. One is for entire dataset and one is for X_testing, therefore only 20% of our data, or in years about one year.

In [0]:

```
prediction = reg.predict(X)
```

In [0]:

```
prediction2 = reg.predict(X_test)
```

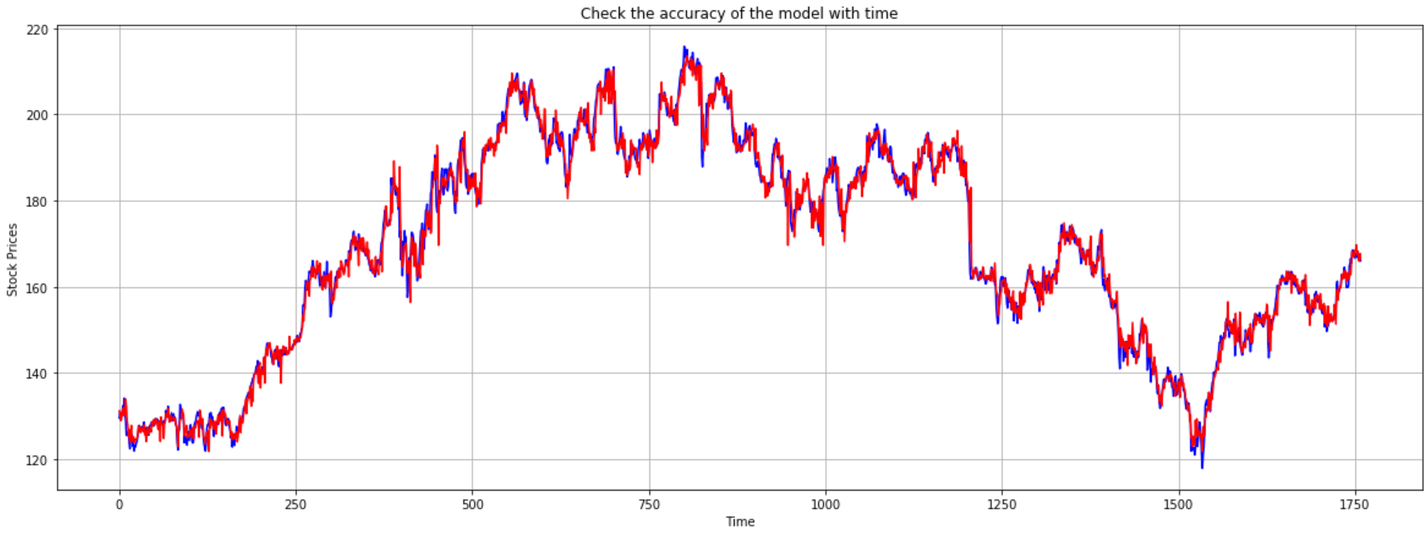
Plotting the entire dataset graph

Here we plot the entire 6 years of stocks developement and the prediction our model would do on them with only 3 days forecasting. We can see how it 96,7% aligns together.

In [0]:

```
print("Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices")
plt.rcParams["figure.figsize"] = (20,7)
plt.plot(y , 'b')
plt.plot(prediction , 'r')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.title('Check the accuracy of the model with time')
plt.grid(True)
plt.show()
```

```
Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices
```



Plotting only the test size graph

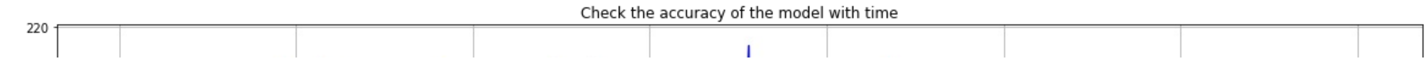
Here we only plot the original stock developement for 20% of our data, ergo approximately one year and the predictions as well. Since its showed on a smaller scale, the numbers tend to be showed more squeezed, therefore the second graph doesnt look as fancy as the first one.

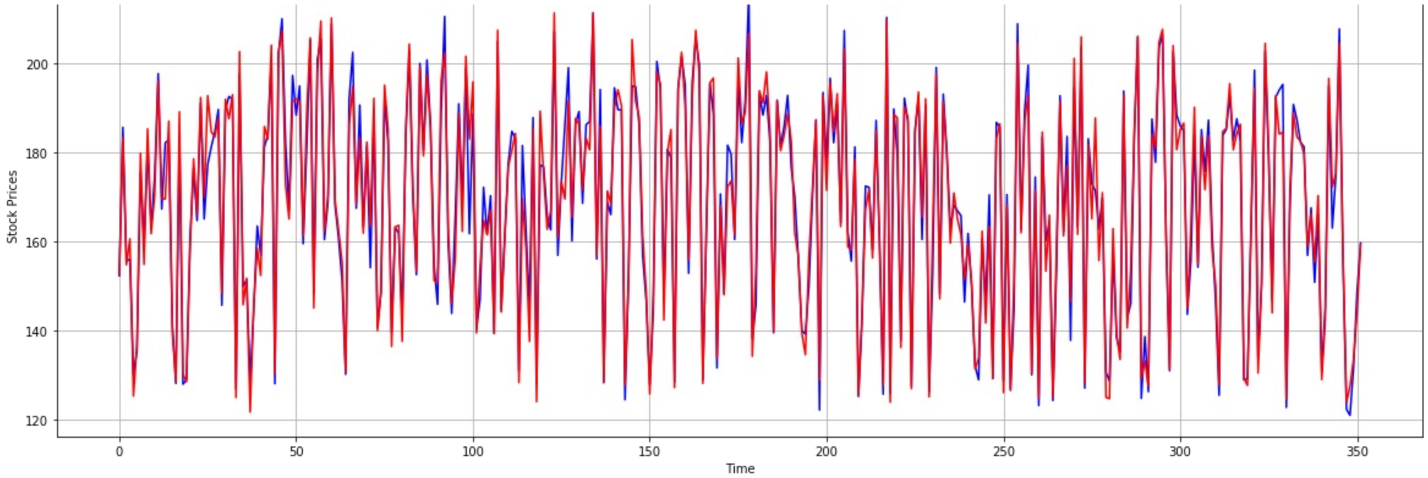
We tried to make it as nice as it gets, but this is how far we got. We could obviously extend it, by using for example 100 width instead of 20, but it wouldnt look as good in PDF or HTML version. You can easily change the number 20 for 100 and see the developement of stocks and predictions much clearer and in more detail :-)

In [0]:

```
print("Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices")
plt.rcParams["figure.figsize"] = (20,7)
plt.plot(y_test , 'b')
plt.plot(prediction2 , 'r')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.title('Check the accuracy of the model with time')
plt.grid(True)
plt.show()
```

```
Red - Predicted Stock Prices  ,  Blue - Actual Stock Prices
```




200
180
160
140
120
Stock Prices
0
50
100
150
200
250
300
350
Time

In [0]: