TASK #1: UNDERSTAND THE PROBLEM STATEMENT & BUSINESS CASE

- **Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) have been transforming finance and investing.**
- **"Artificial intelligence is to trading what fire was to the cavemen"!**
- **Electronic trades account for almost 45% of revenues in cash equities trading" U.K. research firm Coalition Report.**
- **AI powered robo-advisers can perform real-time analysis on massive datasets and trade securities at an extremely faster rate compared to human traders.**
- **AI-powered trading could potentially reduce risk and maximize returns.**
- **Check out the list of companies that leverage AI in trading:**
- https://builtin.com/artificial-intelligence/ai-trading-stock-market-tech

Photo Credit: https://www.pxfuel.com/en/free-photo-qaulj

- **In this project, we will train a ridge regression model and deep neural network model to predict future stock prices.**
- **By accurately predicting stock prices, investors can maximize returns and know when to buy/sell securities.**
- **The AI/ML model will be trained using historical stock price data along with the volume of transactions.**
- **We will use a type of neural nets known as Long Short-Term Memory Networks (LSTM).**
- **Disclaimer: Stock prices are volatile and are generally hard to predict. Invest at your own risk.**

https://builtin.com/artificial-intelligence/ai-trading-stock-market-tech

```
# Environment preparation
import sys
!pip install expectexception
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting expectexception
  Downloading ExpectException-0.1.1-py2.py3-none-any.whl (3.4 kB)
Installing collected packages: expectexception
Successfully installed expectexception-0.1.1
```

```
# Preparing environment
import expectexception

# %%expect_exception TypeError
```

```
!ls sample_data/
```

```
anscombe.json                mnist_test.csv         stocks.csv
california_housing_test.csv  mnist_train_small.csv  stock_volume.csv
california_housing_train.csv README.md
```

TASK #2: IMPORT DATASETS AND LIBRARIES

```
# # Mount Google Drive
# from google.colab import drive
# drive.mount('/content/drive')

import pandas as pd
import plotly.express as px
from copy import copy
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np
import plotly.figure_factory as ff
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from tensorflow import keras

from pandas.core.dtypes.api import is_numeric_dtype

# Set format for float values
pd.options.display.float_format = '{:,.4f}'.format
```

```
# Read stock prices data
stock_price_df = pd.read_csv('sample_data/stocks.csv')

# Sort the data based on Date
stock_price_df.sort_values(by='Date', inplace=True)
stock_price_df.set_index('Date', inplace=True)
stock_price_df
```

Date	AAPL	BA	T	MGM	AMZN	IBM	TSLA	GOOG
2012-01-12	60.1986	75.5100	30.1200	12.1300	175.9300	180.5500	28.2500	313.6444
2012-01-13	59.9729	74.6000	30.0700	12.3500	178.4200	179.1600	22.7900	311.3281
2012-01-17	60.6714	75.2400	30.2500	12.2500	181.6600	180.0000	26.6000	313.1164
2012-01-18	61.3014	75.0600	30.3300	12.7300	189.4400	181.0700	26.8100	315.2733
2012-01-19	61.1071	75.5600	30.4200	12.8000	194.4500	180.5200	26.7600	318.5909
...	...	...	...	...	...	...	...	...
2020-08-05	440.2500	174.2800	29.8500	16.7200	3,205.0300	125.4500	1,485.0200	1,473.6100
2020-08-06	455.6100	172.2000	29.8400	18.4600	3,225.0000	126.1200	1,489.5800	1,500.1000

```
# Read the stocks volume data
stock_vol_df = pd.read_csv("sample_data/stock_volume.csv")

# Sort the volume data based on Date
stock_vol_df.sort_values(by='Date', inplace=True)
stock_vol_df.set_index('Date', inplace=True)
stock_vol_df
```

Date	AAPL	BA	T	MGM	AMZN	IBM	TSLA	GOOG
2012-01-12	53146800	3934500	26511100	17891100	5385800	6881000	729300	3764400
2012-01-13	56505400	4641100	22096800	16621800	4753500	5279200	5500400	4631800
2012-01-17	60724300	3700100	23500200	15480800	5644500	6003400	4651600	3832800
2012-	69197800	4189500	22015000	18387600	7473500	4600600	1260200	5544000

```
# Check if Null values exist in stock prices data
stock_price_df.isnull().sum()
```

```
AAPL     0
BA       0
T        0
MGM      0
AMZN     0
IBM      0
TSLA     0
GOOG     0
sp500    0
dtype: int64
```

```
# Check if Null values exist in stocks volume data
stock_vol_df.isnull().sum()
```

```
AAPL     0
BA       0
T        0
MGM      0
AMZN     0
IBM      0
TSLA     0
GOOG     0
sp500    0
dtype: int64
```

```
# Get stock prices dataframe info
stock_price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2159 entries, 2012-01-12 to 2020-08-11
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AAPL    2159 non-null   float64
 1   BA      2159 non-null   float64
 2   T       2159 non-null   float64
 3   MGM     2159 non-null   float64
 4   AMZN    2159 non-null   float64
```

```
 5   IBM     2159 non-null   float64
 6   TSLA    2159 non-null   float64
 7   GOOG    2159 non-null   float64
 8   sp500   2159 non-null   float64
dtypes: float64(9)
memory usage: 168.7+ KB
```

```
# Get stock volume dataframe info
stock_vol_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2159 entries, 2012-01-12 to 2020-08-11
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   AAPL    2159 non-null   int64
 1   BA      2159 non-null   int64
 2   T       2159 non-null   int64
 3   MGM     2159 non-null   int64
 4   AMZN    2159 non-null   int64
 5   IBM     2159 non-null   int64
 6   TSLA    2159 non-null   int64
 7   GOOG    2159 non-null   int64
 8   sp500   2159 non-null   int64
dtypes: int64(9)
memory usage: 168.7+ KB
```

MINI CHALLENGE #1:

- **What is the average trading volume for Apple stock?**
- **What is the maximum trading volume for sp500?**
- **Which security is traded the most? comment on your answer**
- **What is the average stock price of the S&P500 over the specified time period?**
- **What is the maximum price of Tesla Stock?**

```
print(f'The average trading volume for Apple stocks is {stock_vol_df.A
print(f'The maximum trading volume for sp500 is {stock_vol_df.sp500.ma
print(f'''
The most traded security is {stock_vol_df.mean().idxmax()}
{stock_vol_df.mean().sort_values(ascending=False)}
{stock_vol_df.describe()}
''')
print(f'The average stock price of the sp500 over the specified time p
print(f'The maximum price of Tesla stock is {stock_price_df.TSLA.max()
```

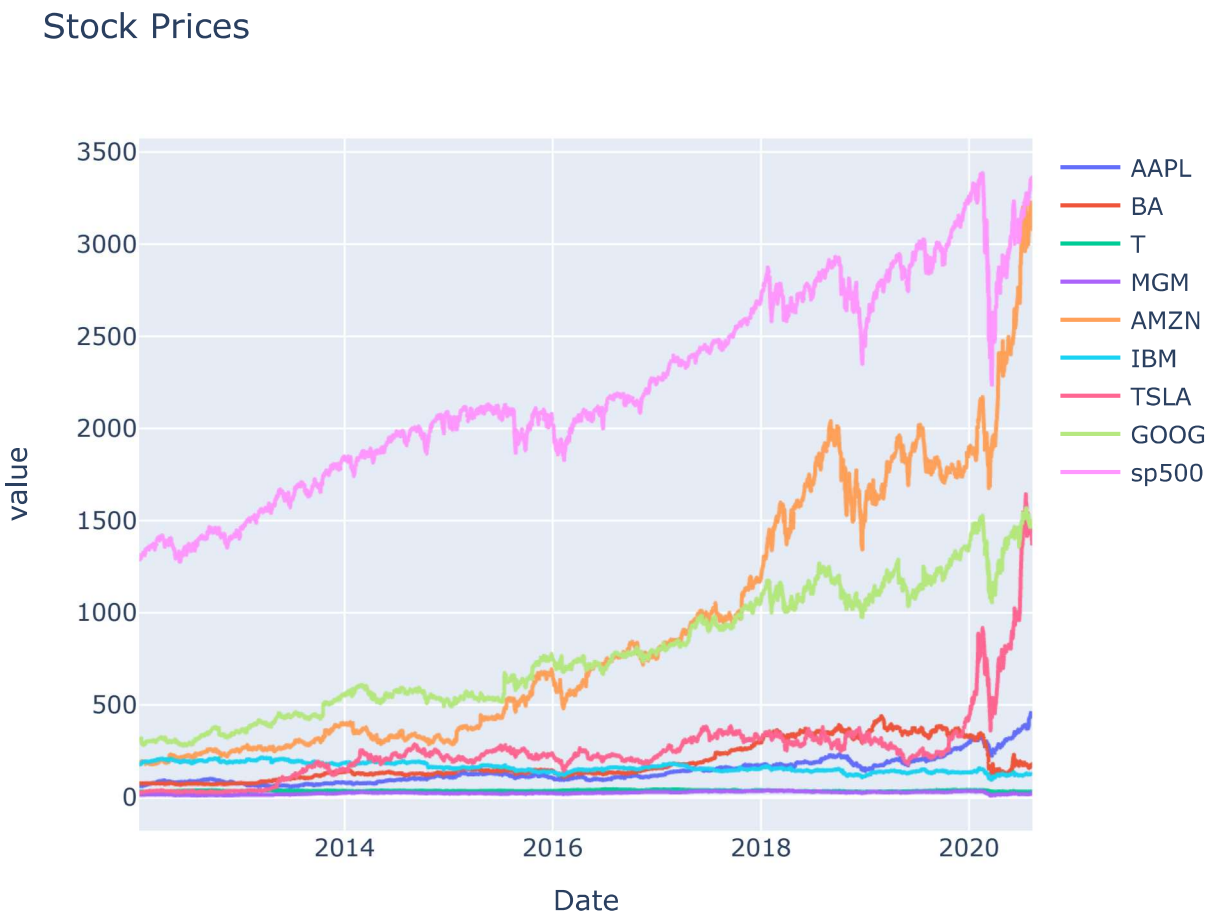
```
The average trading volume for Apple stocks is 58,203,317.42
The maximum trading volume for sp500 is 9,044,690,000.00

The most traded security is sp500
sp500   3,680,732,468.7355
AAPL       58,203,317.4155
T          28,321,313.5711
```

```
MGM         9,845,581.7045
TSLA        7,001,302.2696
BA          6,419,915.9333
IBM         4,453,089.5322
AMZN        4,102,672.9041
GOOG        2,498,238.2585
dtype: float64
                  AAPL               BA                 T              MGM  \
count        2,159.0000        2,159.0000        2,159.0000       2,159.0000
mean    58,203,317.4155    6,419,915.9333   28,321,313.5711   9,845,581.7045
std     45,681,411.9012    9,711,873.1540   14,289,105.8976   7,295,752.6435
min     11,362,000.0000      788,900.0000    6,862,400.0000     950,700.0000
25%     27,699,300.0000    3,031,850.0000   20,021,500.0000   5,796,450.0000
50%     42,094,200.0000    3,991,000.0000   24,859,300.0000   7,899,800.0000
75%     71,824,800.0000    5,325,900.0000   32,105,650.0000  11,040,550.0000
max    376,530,000.0000  103,212,800.0000  195,082,700.0000  90,098,200.0000

                 AMZN             IBM             TSLA             GOOG  \
count       2,159.0000       2,159.0000       2,159.0000       2,159.0000
mean    4,102,672.9041   4,453,089.5322   7,001,302.2696   2,498,238.2585
std     2,290,722.3372   2,462,811.4478   5,781,207.8365   1,928,407.2520
min       881,300.0000   1,193,000.0000     364,900.0000       7,900.0000
25%     2,675,700.0000   3,111,250.0000   3,433,450.0000   1,325,400.0000
50%     3,494,800.0000   3,825,000.0000   5,581,100.0000   1,813,900.0000
75%     4,768,150.0000   4,937,300.0000   8,619,550.0000   3,245,350.0000
max    23,856,100.0000  30,490,200.0000  60,938,800.0000  24,977,900.0000

                   sp500
count          2,159.0000
mean   3,680,732,468.7355
std      862,271,696.8442
min    1,248,960,000.0000
25%    3,211,890,000.0000
50%    3,526,890,000.0000
75%    3,933,290,000.0000
max    9,044,690,000.0000

The average stock price of the sp500 over the specified time period is 2,218.75
The maximum price of Tesla stock is 1643.0
```

TASK #3: PERFORM EXPLORATORY DATA ANALYSIS AND VISUALIZATION

```
# Function to normalize stock prices based on their initial price
def normalize(df):
  return df.apply(lambda col: col/col[0] if is_numeric_dtype(col) else

# Function to plot interactive plots using Plotly Express
def interactive_plot(df, title):
  fig = px.line(df, title = title)
```

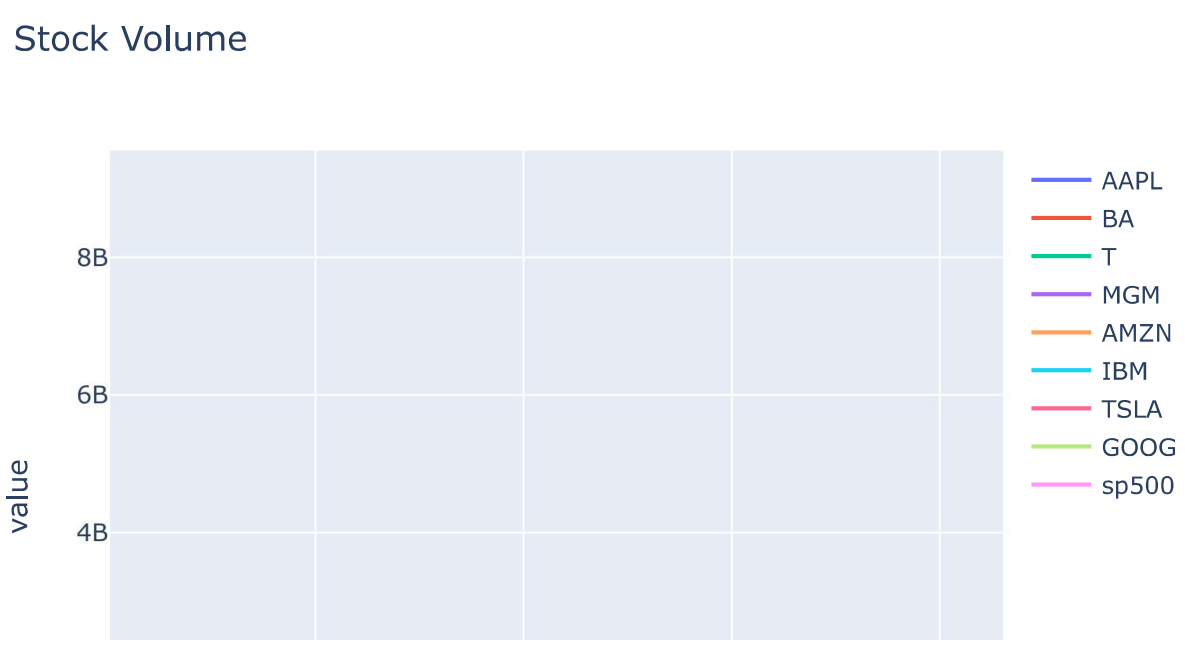
```
    fig.update_layout(legend_title_text = None)
    fig.show()
# plot interactive chart for stocks data
interactive_plot(stock_price_df, 'Stock Prices')
```



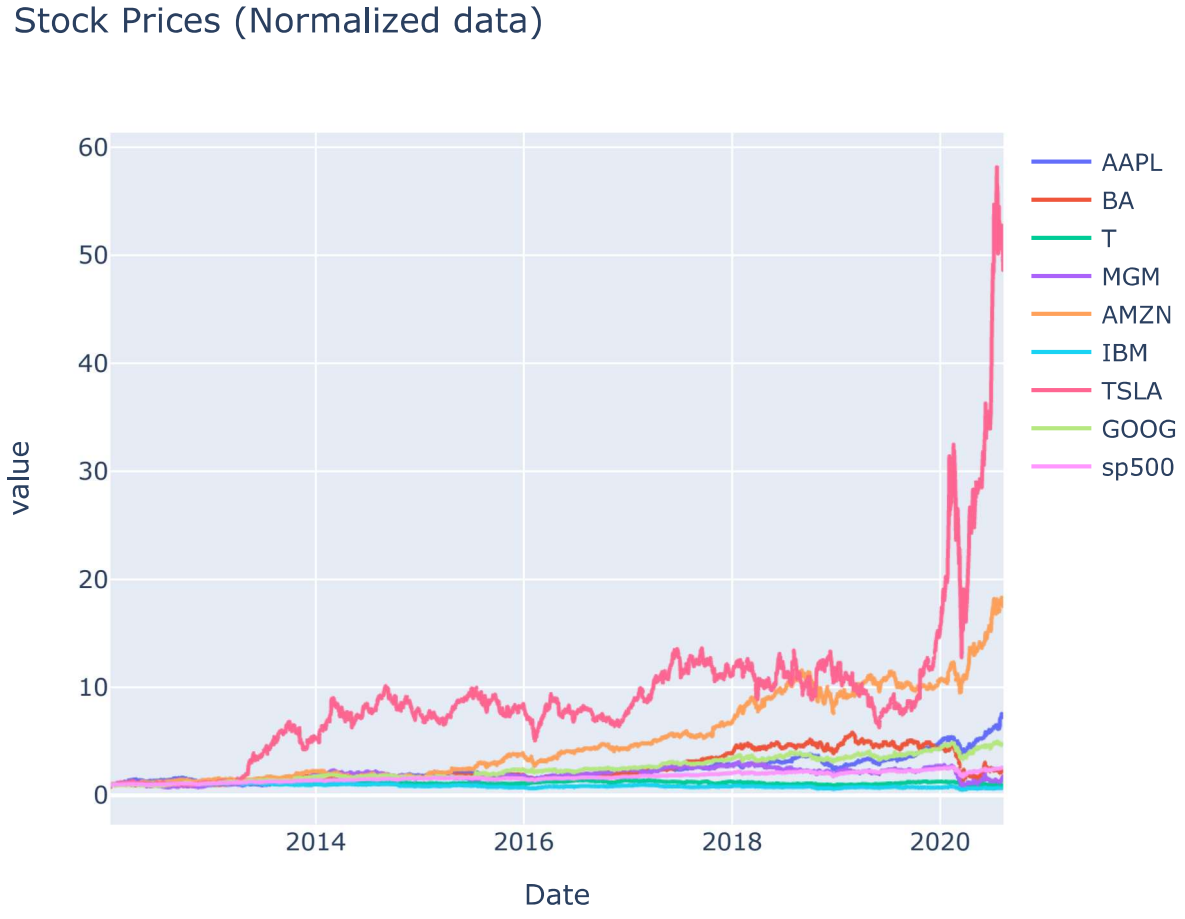
MINI CHALLENGE #2:

- **Plot the volume dataset for all stocks, list any observations you might see.**
- **Plot the normalized stock prices and volume dataset.**

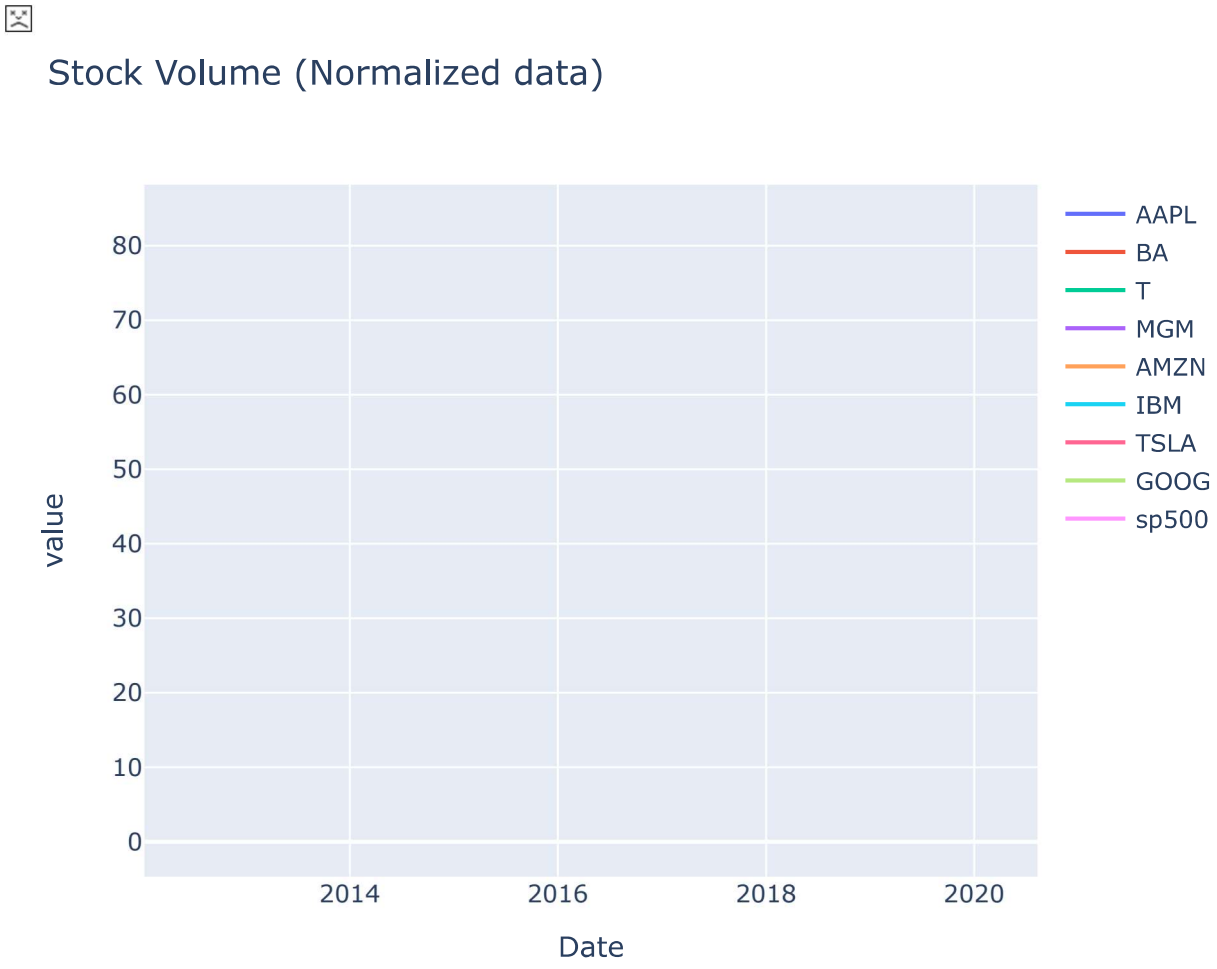
```
# Plot the volume dataset for all stocks, list any observations you mi
interactive_plot(stock_vol_df, 'Stock Volume')
```




```
# Plot the normalized stock prices.
interactive_plot(normalize(stock_price_df), 'Stock Prices (Normalized
```




```
# Plot the normalized volume dataset.
interactive_plot(normalize(stock_vol_df), 'Stock Volume (Normalized da
```



TASK #4: PREPARE THE DATA BEFORE TRAINING THE AI/ML MODEL

TRAINING & TESTING DATA SPLIT

- **Data set is divided into 75% for training and 25% for testing.**
  - **Training set: used for model training.**
  - **Testing set: used for testing trained model. Make sure that the testing dataset has never been seen by the trained model before.**

```
# Function to concatenate the date, stock price, and volume in one dat
def individual_stock(price_df, vol_df, name):
    return pd.DataFrame(data={'Close': price_df[name], 'Volume': vol_c
                        index=price_df.index)

# Function to return the input/output (target) data for AI/ML Model
# Note that our goal is to predict the future stock price
# Target stock price today will be tomorrow's price
# Function to return the input/output (target) data for AI/ML Model
# Note that our goal is to predict the future stock price
# Target stock price today will be tomorrow's price
def trading_window(df, window_period = 1):
  # Create a column containing the prices for the next 1 days
  df['Target'] = df[['Close']].shift(-window_period)

  # Remove the last row as it will be a null value
  df.dropna(inplace=True)

  # return the new dataset
  return df


# Function to scale and split the data
from sklearn.preprocessing import MinMaxScaler

def scaling_and_splitting_data(df, split_percent):
```

```
    sc = MinMaxScaler(feature_range = (0, 1))
    data = sc.fit_transform(df)

    # Creating Feature and Target
    X = data[:,:2]
    y = data[:,2]

    # Spliting the data this way, since order is important in time-serie
    # Note that we did not use train test split with it's default settir
    split = int(split_percent * len(X))
    X_train = X[:split]
    y_train = y[:split]
    X_test = X[split:]
    y_test = y[split:]

    return X, y, X_train, y_train, X_test, y_test


# Define a data plotting function
def show_plot(data, title, legend=['Close Price', 'Volume']):
    plt.figure(figsize = (13, 5))
    plt.plot(data, linewidth = 3)
    plt.title(title)
    plt.legend(legend)
    plt.grid()


# Let's test the functions and get individual stock prices and volumes
stock_name = 'AAPL'
price_volume_target_df = trading_window(individual_stock(stock_price_d
price_volume_target_df
```

Date	Close	Volume	Target
2012-01-12	60.1986	53146800	59.9729
2012-01-13	59.9729	56505400	60.6714

```
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr
(X.shape, y.shape), (X_train.shape, y_train.shape), (X_test.shape, y_t
```

```
(((2158, 2), (2158,)), ((1402, 2), (1402,)), ((756, 2), (756,)))
```

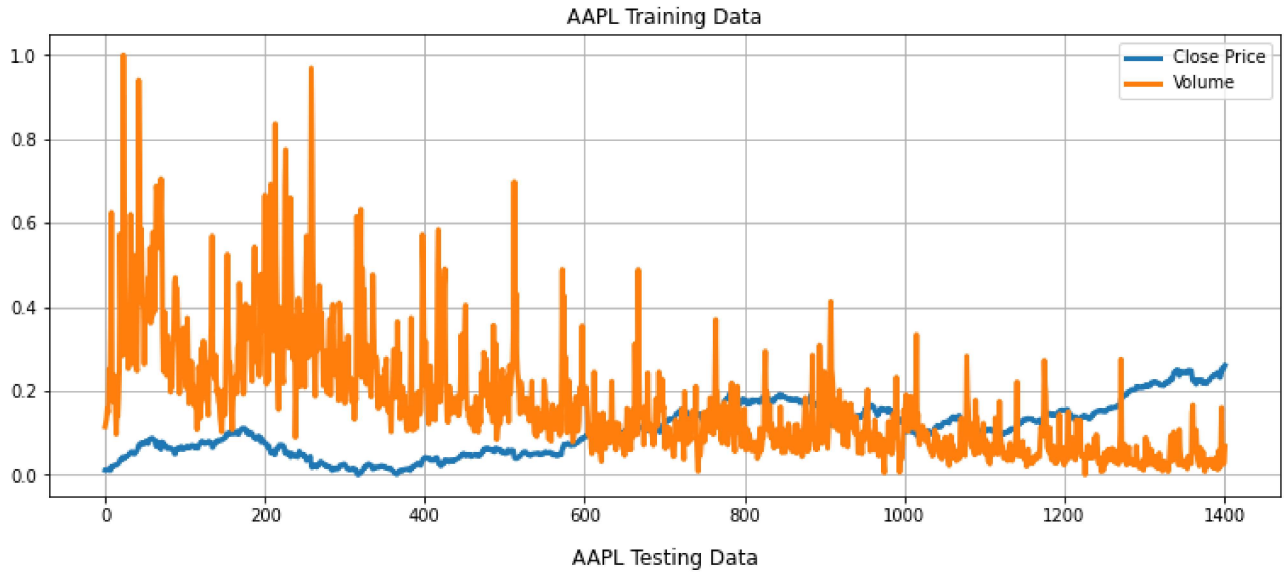
```
# Train data
X_train, y_train
```

```
(array([[0.01102638, 0.11442624],
        [0.01046185, 0.12362365],
        [0.01220906, 0.13517696],
        ...,
        [0.25161323, 0.02518813],
        [0.25766595, 0.02877662],
        [0.26084239, 0.06803417]]),
 array([0.01046185, 0.01220906, 0.01378478, ..., 0.25766595, 0.26084239,
        0.26329349]))
```

```
# # Test data
# X_test, y_test
```

```
%matplotlib inline
```

```
show_plot(X_train, f'{stock_name} Training Data')
show_plot(X_test, f'{stock_name} Testing Data')
```

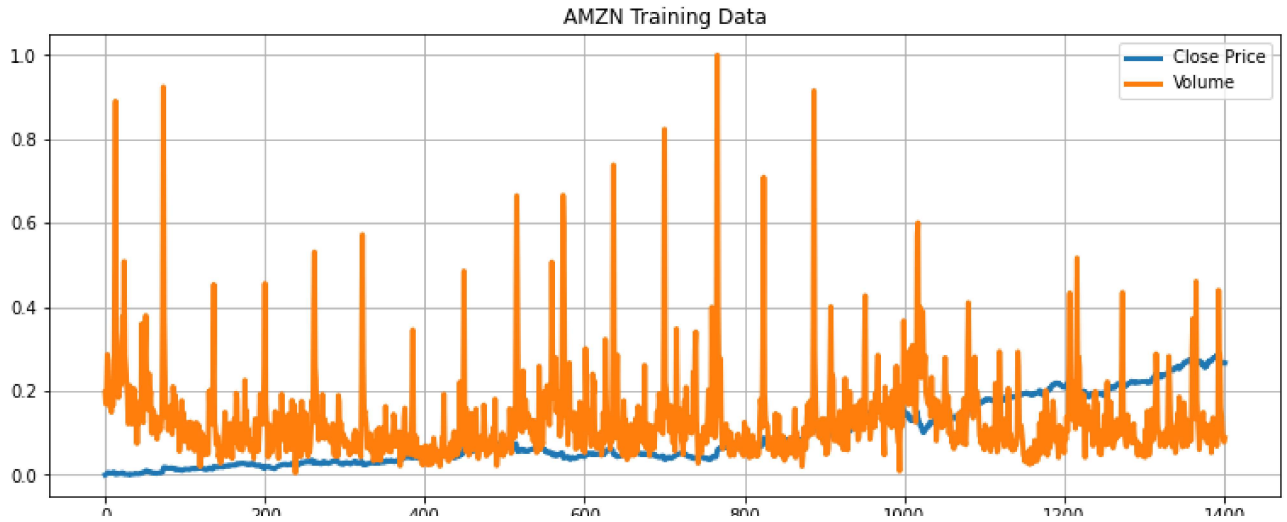


MINI CHALLENGE #3:

- **Test the created pipeline with S&P500 and Amazon datasets**

```
%matplotlib inline
# Test the created pipeline with S&P500

stock_name = 'sp500'
# (1) Get individual stock prices and volumes for stock
price_volume_target_df = trading_window(individual_stock(stock_price_c

# (2) Let's scale and split the data
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# (3) Let's show the training and testing data
show_plot(X_train, f'{stock_name} Training Data')
show_plot(X_test, f'{stock_name} Testing Data')
```

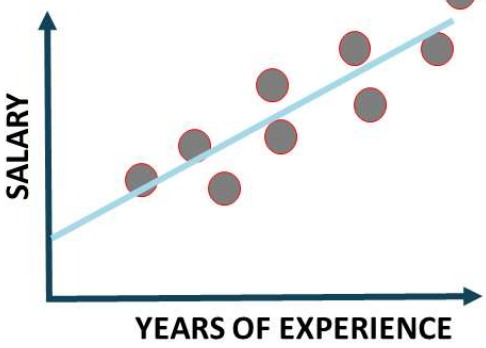

```
%matplotlib inline
# Test the created pipeline with AMZN

stock_name = 'AMZN'
# (1) Get individual stock prices and volumes for stock
price_volume_target_df = trading_window(individual_stock(stock_price_c

# (2) Let's scale and split the data
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# (3) Let's show the training and testing data
show_plot(X_train, f'{stock_name} Training Data')
show_plot(X_test, f'{stock_name} Testing Data')
```



TASK #5: UNDERSTAND THE THEORY AND INTUITION BEHIND REGRESSION

SIMPLE LINEAR REGRESSION: INTUITION

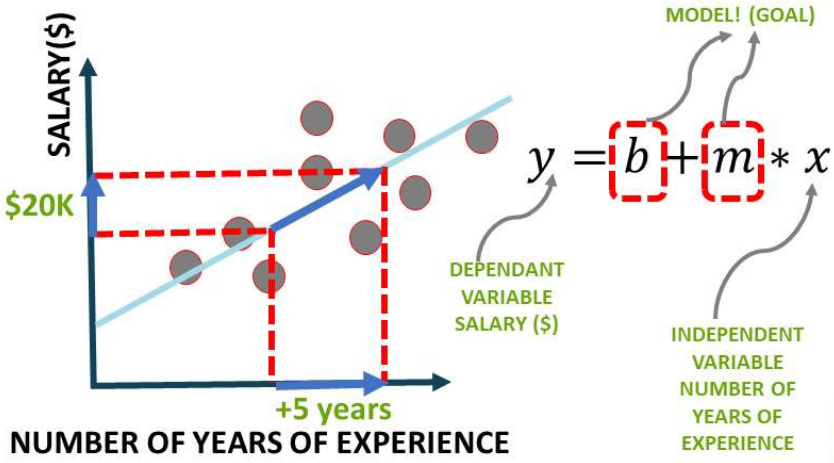
- In simple linear regression, we predict the value of one variable Y based on another variable X.
- X is called the independent variable and Y is called the dependant variable.
- Why simple? Because it examines relationship between two variables only.
- Why linear? when the independent variable increases (or decreases), the dependent variable increases (or decreases) in a linear fashion.



	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111

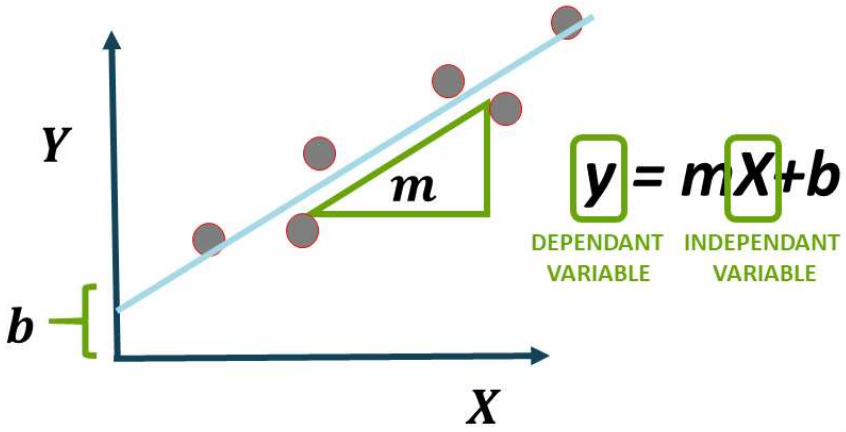
SIMPLE LINEAR REGRESSION: SOME MATH!

- Goal is to obtain a relationship (model) between employee salary and number of years of experience.



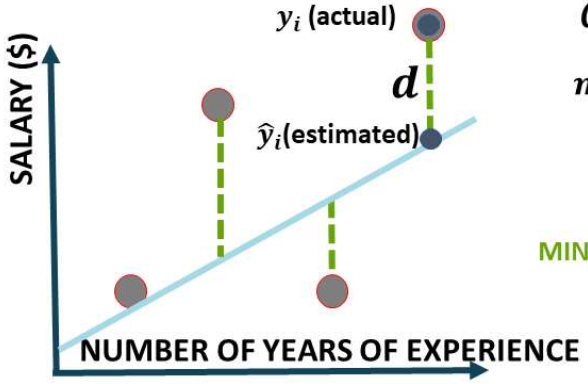
WHAT'S M AND B?

- Once the coefficients m and b are obtained, you have obtained a simple linear regression model!
- This "trained" model can be later used to predict any salary based on the number of years of experience.



SIMPLE LINEAR REGRESSION: HOW TO OBTAIN MODEL PARAMETERS? LEAST SUM OF SQUARES

- Least squares fitting is a way to find the best fit curve or line for a set of points.
- The sum of the squares of the offsets (residuals) are used to estimate the best fit curve or line.
- Least squares method is used to obtain the coefficients m and b.



$$d = \hat{y}_i - y_i$$

$$min \sum (\hat{y}_i - y_i)^2$$

MINIMUM (LEAST) SUM OF SQUARES

TASK #6: UNDERSTAND THE CONCEPT OF REGULARIZATION & RIDGE REGRESSION

REGULARIZATION: INTUITION

- Regularization techniques are used to avoid networks overfitting
- Overfitting occurs when the model provide great results on the training data but performs poorly on testing dataset.
- Overfitting occurs when the model learns all the patterns of the training dataset but fails to generalize.
- Overfitted models generally provide high accuracy on training dataset but low accuracy on testing and validation (evaluation) datasets

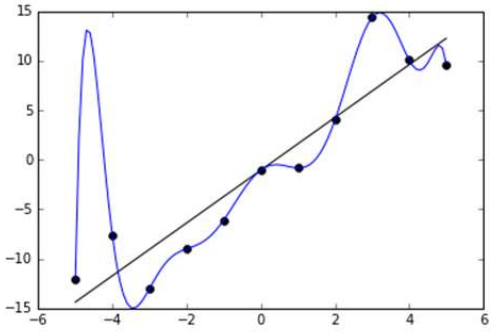
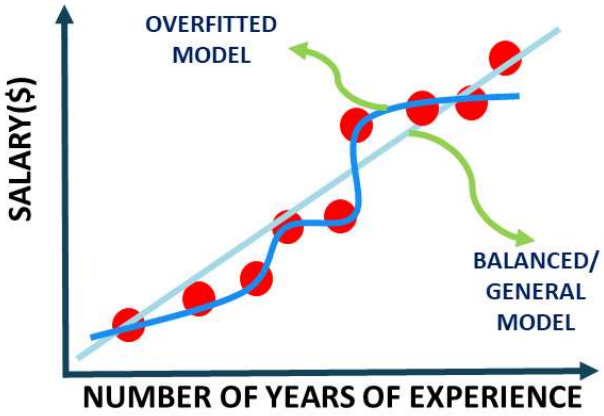


Photo Credit: https://commons.wikimedia.org/wiki/File:Overfitted_Data.png

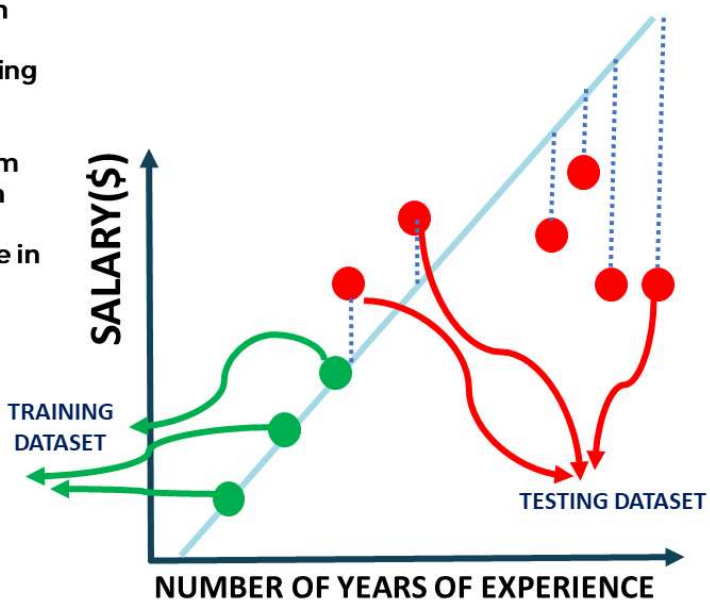
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

- Ridge regression advantage is to avoid overfitting.
- Our ultimate model is the one that could generalize patterns; i.e.: works best on the training and testing dataset
- Overfitting occurs when the trained model performs well on the training data and performs poorly on the testing datasets
- Ridge regression works by applying a penalizing term (reducing the weights and biases) to overcome overfitting.



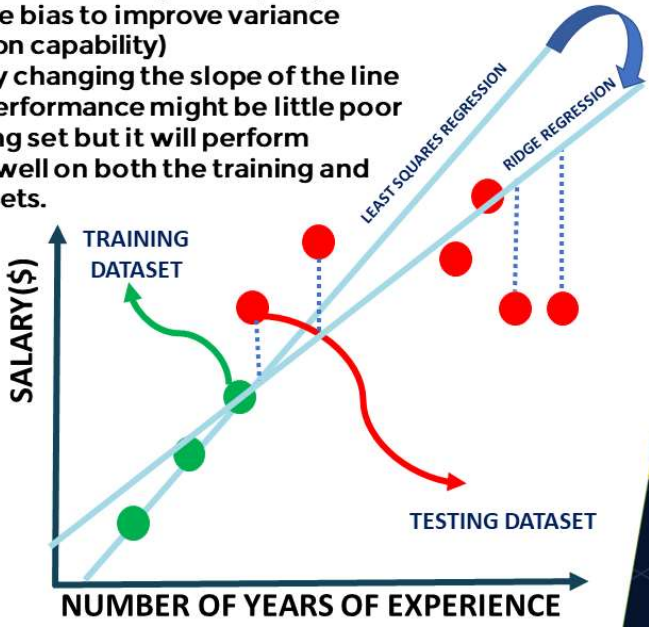
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

- Least sum of squares is applied to obtain the best fit line
- Since the line passes through the 3 training dataset points, the sum of squared residuals = 0
- However, for the testing dataset, the sum of residuals is large so the line has a high variance.
- Variance means that there is a difference in fit (or variability) between the training dataset and the testing dataset.
- This regression model is overfitting the training dataset



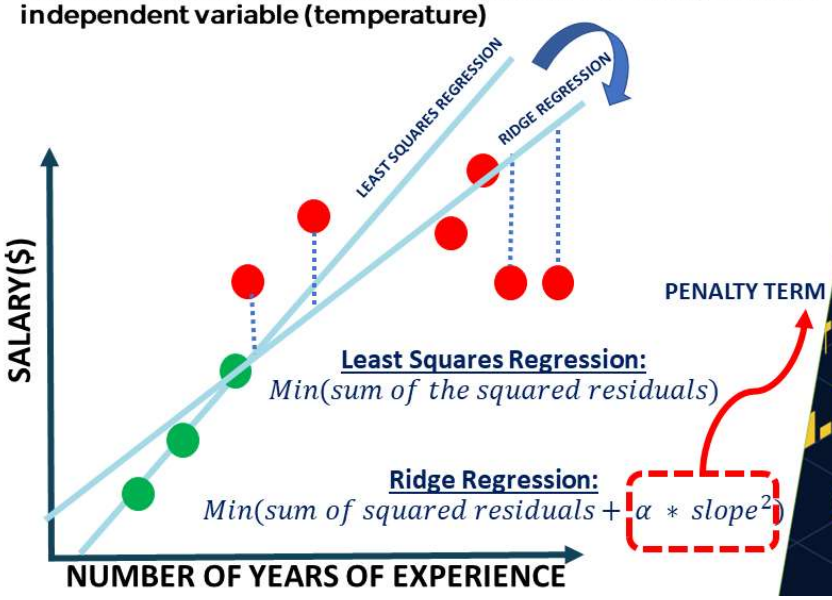
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

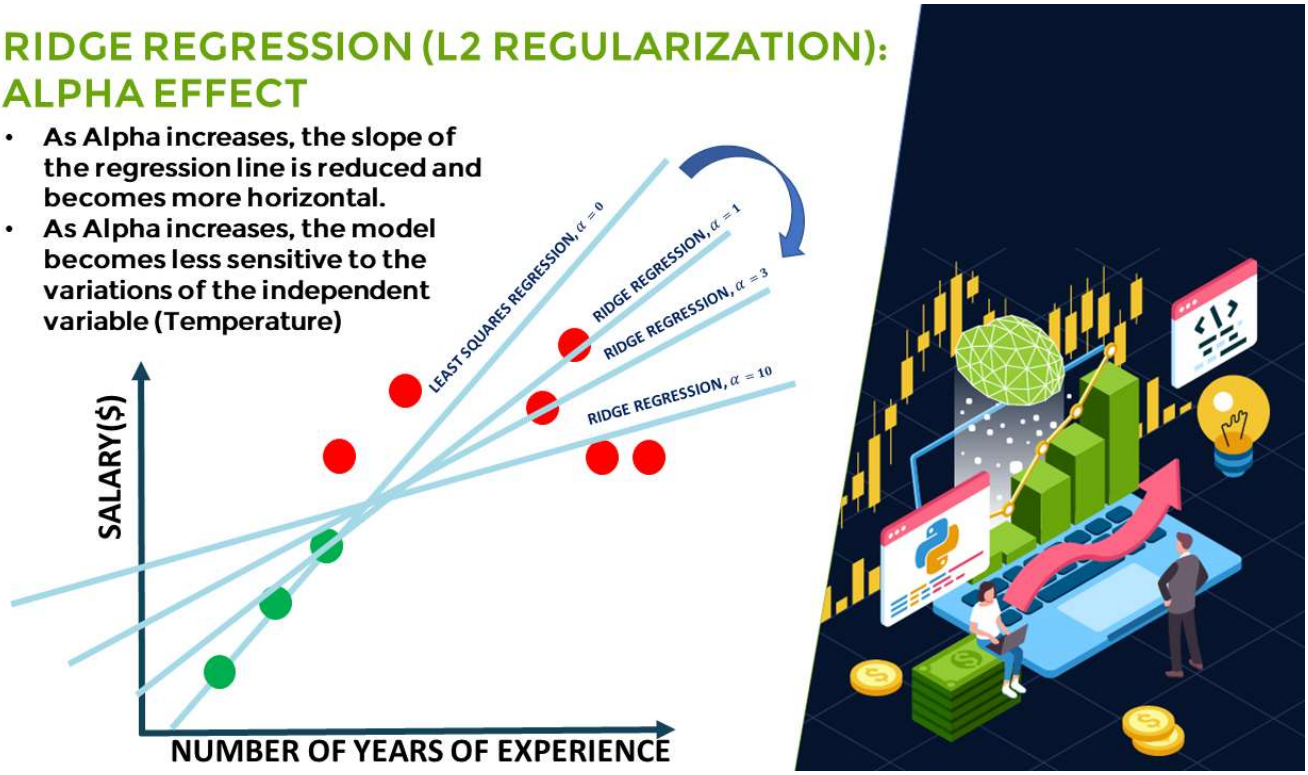
- Ridge regression works by attempting at increasing the bias to improve variance (generalization capability)
- This works by changing the slope of the line
- The model performance might be little poor on the training set but it will perform consistently well on both the training and testing datasets.



RIDGE REGRESSION (L2 REGULARIZATION): MATH

- Slope has been reduced with ridge regression penalty and therefore the model becomes less sensitive to changes in the independent variable (temperature)





TASK #7: BUILD AND TRAIN A RIDGE LINEAR REGRESSION MODEL

```
from sklearn.linear_model import Ridge

# Let's create a function with the Ridge algorith to make predictions
def prices_ridge_predictions(X_train, y_train, X_test, y_test, X_pred,
  # Note that Ridge regression performs linear least squares with L2 r
  # Create and train the Ridge Linear Regression  Model
  model = Ridge(alpha=alpha, random_state=seed)
  model.fit(X_train, y_train)

  # Test the model and calculate its accuracy
  lr_accuracy = model.score(X_test, y_test)
  if verbose:
    print(f"Linear Regression Score (\u03B1 = {alpha}): {lr_accuracy}"

  # Make Prediction
  predicted_prices = model.predict(X_pred)

  # Create a dataframe based on the dates in the individual stock data
  df_predicted = pd.DataFrame(data={'Close': X_pred[:,0],
                                    'Prediction': predicted_prices},
```

```
                                        index=df_index)
  return model, lr_accuracy, predicted_prices, df_predicted
```

```
%matplotlib inline
# Test the created pipeline with S&P500

stock_name = 'AAPL'
# (1) Get individual stock prices and volumes for stock
price_volume_target_df = trading_window(individual_stock(stock_price_d

# (2) Let's scale and split the data
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# # (3) Let's show the training and testing data
# show_plot(X_train, f'{stock_name} Training Data')
# show_plot(X_test, f'{stock_name} Testing Data')

stock_name
```

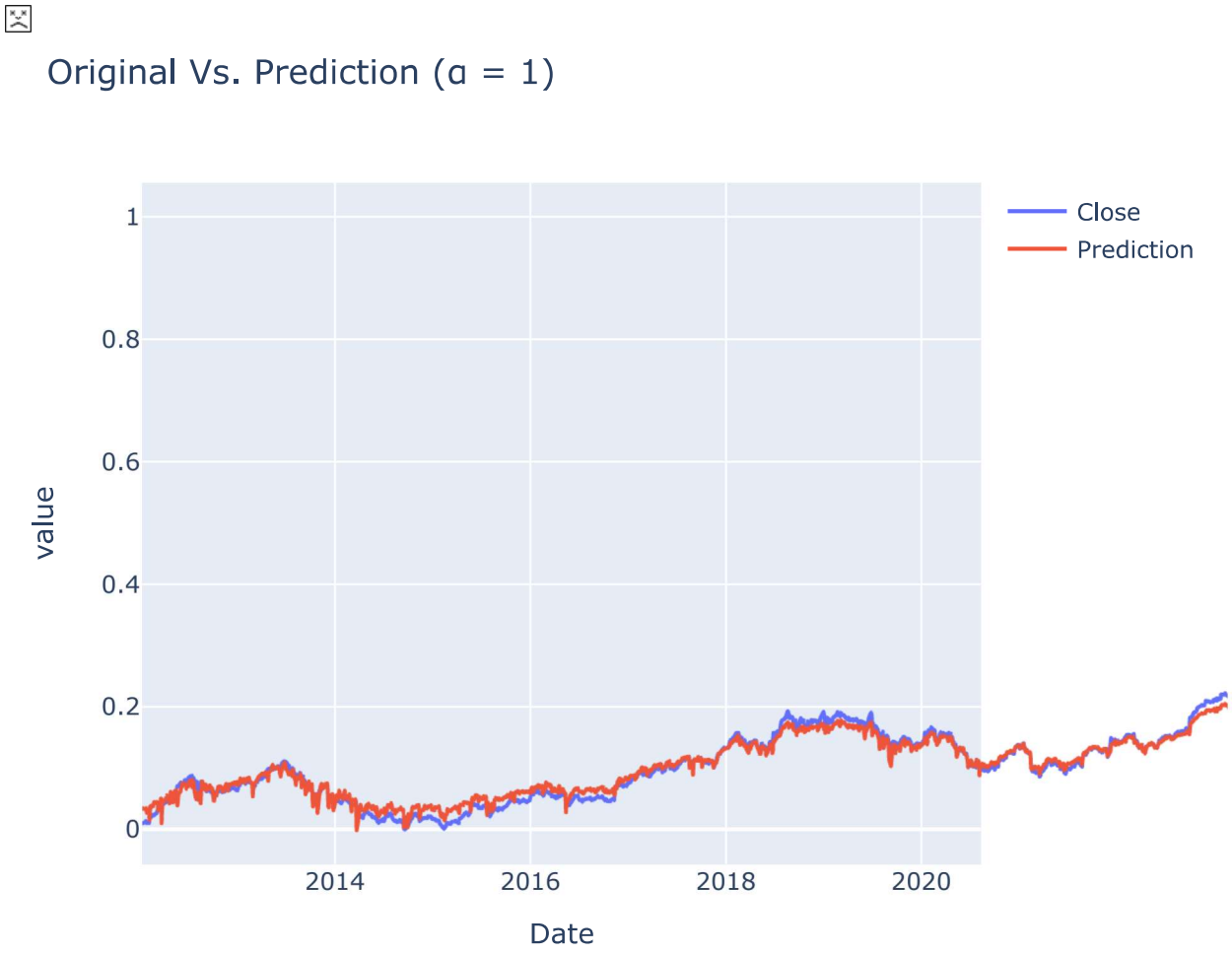
```
    'AAPL'
```

```
regression_model, lr_accuracy, predicted_prices, df_predicted = prices
```

```
df_predicted
```

```
Linear Regression Score (α = 1): 0.7950028030821767
```

```
# Plot the results
interactive_plot(df_predicted, "Original Vs. Prediction (\u03B1 = 1)")
```



MINI CHALLENGE #4:

- **Experiment with various regularization values for alpha**
- **What is the impact of increasing alpha?**
- **Note: default value for alpha is = 1**

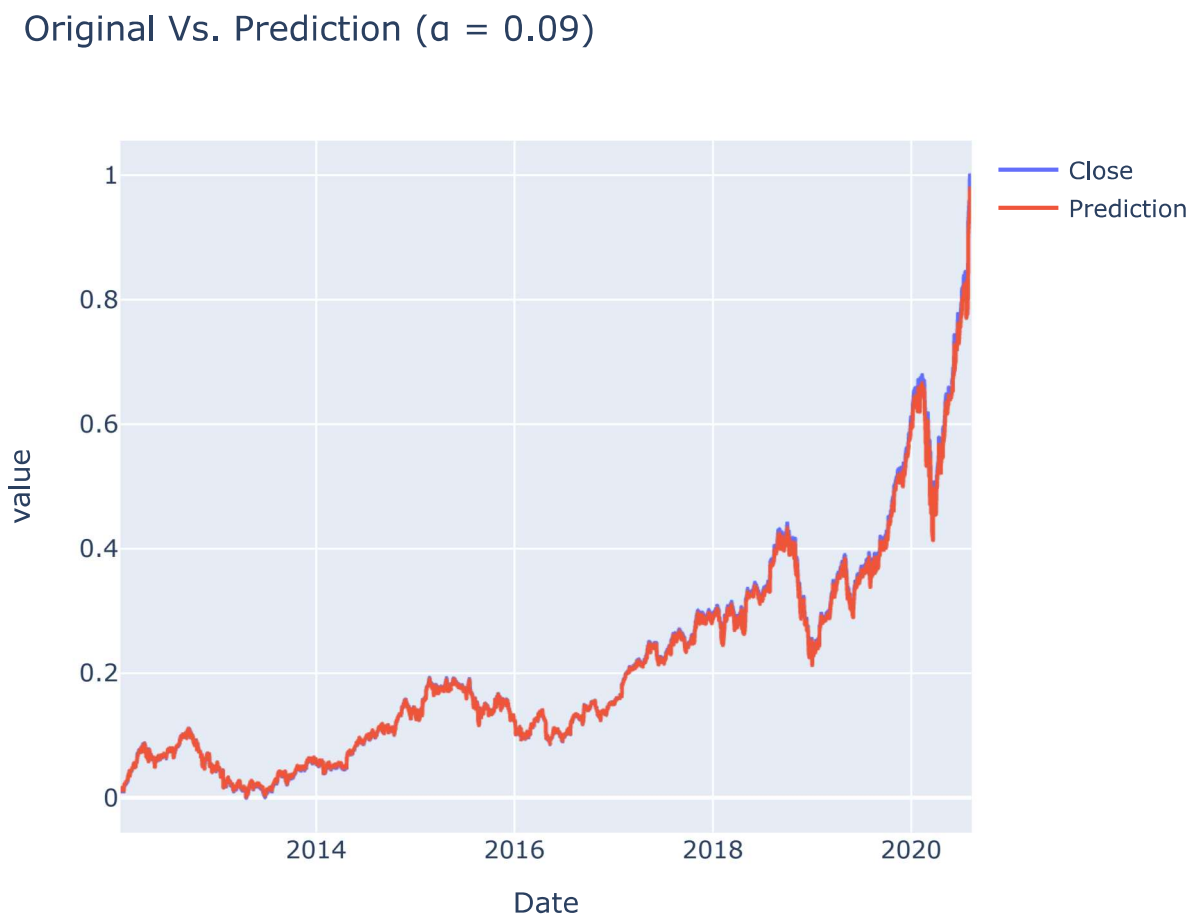
```
for alpha in [1.75, 1.5, 1.25, 1, 0.75, 0.5, 0.4, 0.3, 0.2, 0.1, 0.09]
  regression_model, lr_accuracy, predicted_prices, df_predicted = pric
```

```
  # interactive_plot(df_predicted, f"Original Vs. Prediction (\u03B1 =
```

```
Linear Regression Score (α = 1.75): 0.55339851961466
Linear Regression Score (α = 1.5): 0.6358248780491251
Linear Regression Score (α = 1.25): 0.7170036207928734
```

```
Linear Regression Score (α = 1): 0.7950028030821767
Linear Regression Score (α = 0.75): 0.8669780090009542
Linear Regression Score (α = 0.5): 0.9287228990823156
Linear Regression Score (α = 0.4): 0.949242879243803
Linear Regression Score (α = 0.3): 0.9666324235194903
Linear Regression Score (α = 0.2): 0.9802717865145065
Linear Regression Score (α = 0.1): 0.9894269495470439
Linear Regression Score (α = 0.09): 0.9900636069676167
```

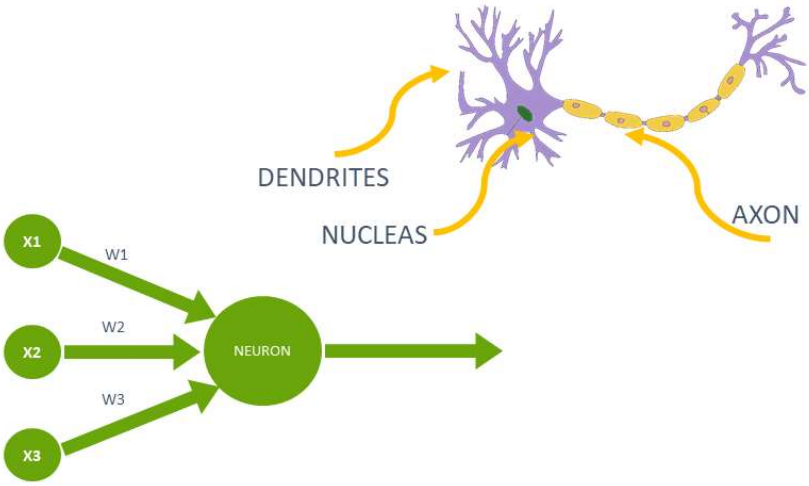
```
interactive_plot(df_predicted, f"Original Vs. Prediction (\u03B1 = {al
```



TASK #8: UNDERSTAND THE THEORY AND INTUITION BEHIND NEURAL NETWORKS

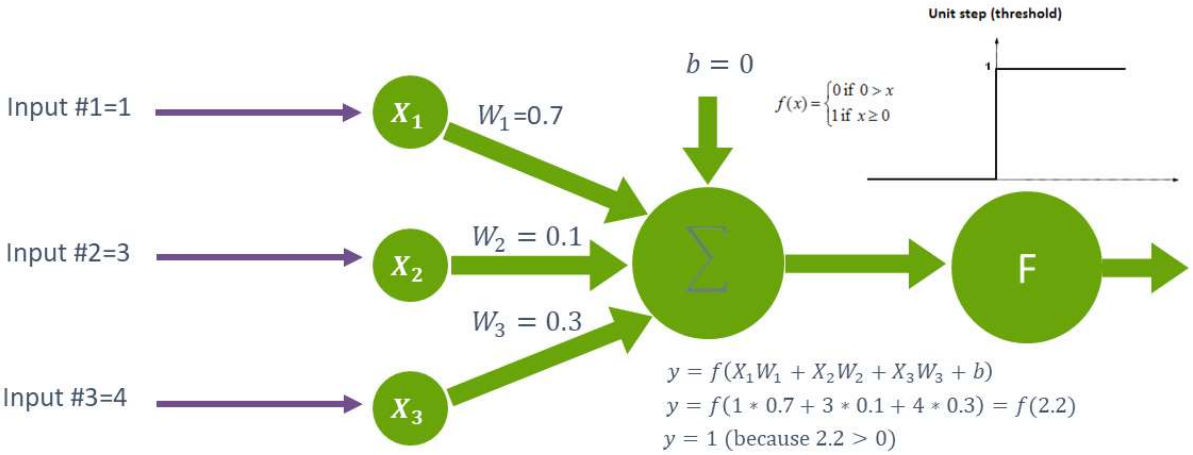
NEURON MATHEMATICAL MODEL

- **Artificial Neural Networks are information processing models that are inspired by the human brain.**
- **The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called axon.**



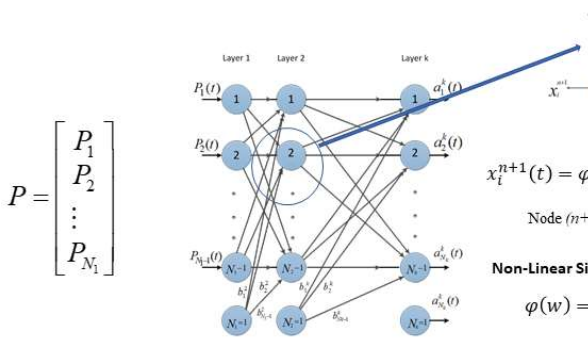
SINGLE NEURON MODEL IN ACTION!

- **Let's assume an activation function of Unit Step.**
- **The activation functions is used to map the input between (0, 1).**



MULTI-LAYER PERCEPTRON NETWORK

- Let's connect multiple of these neurons in a multi-layer fashion.
- The more hidden layers, the more "deep" the network will get.



$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$

$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node $(n+1, i)$ representation

Non-Linear Sigmoid Activation function

$$\varphi(w) = \frac{1}{1+e^{-w}}$$

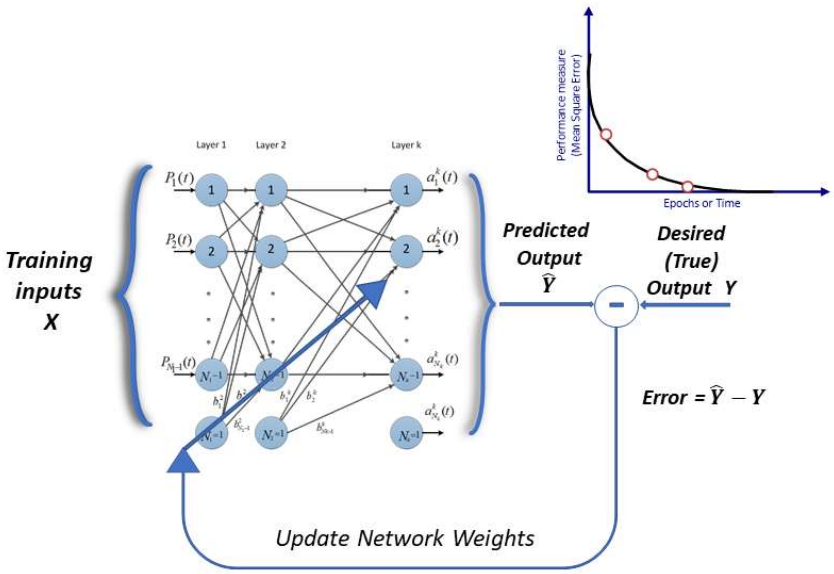
$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$

m: number of neurons in the hidden layer

N_1*: number of inputs*

TASK #9: UNDERSTAND HOW DO ARTFICIAL NEURAL NETWORKS TRAIN

HOW DO ANN TRAIN? PROCESS OVERVIEW



HOW DO ANN TRAIN? GRADIENT DESCENT

- Gradient descent is an optimization algorithm used to obtain the optimized network weight and bias values.
- It works by iteratively trying to minimize the cost function.
- It works by calculating the gradient of the cost function and moving in the negative direction until the local/global minimum is achieved.
- The size of the steps taken are called the learning rate
- If learning rate increases, the area covered in the search space will increase so we might reach global minimum faster
- However, we can overshoot the target
- For small learning rates, training will take much longer to reach optimized weight values

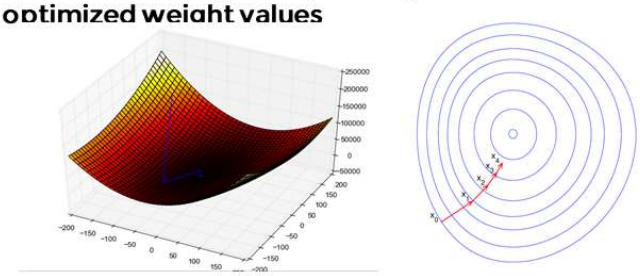


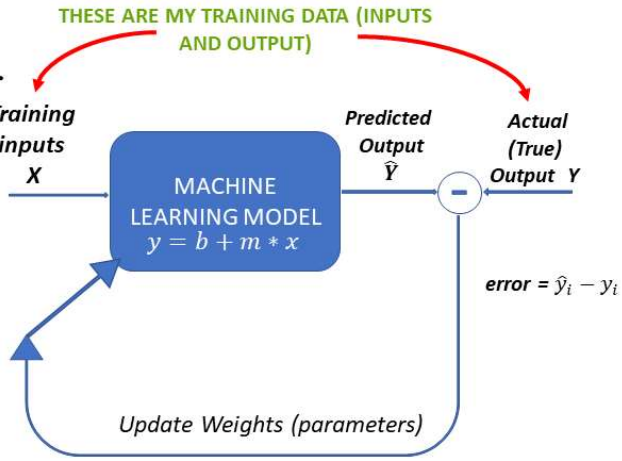
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

HOW DO ANN TRAIN? GRADIENT DESCENT

- Let's assume that we want to obtain the optimal values for parameters 'm' and 'b'.

$$y = b + m * x$$

GOAL IS TO FIND BEST PARAMETERS

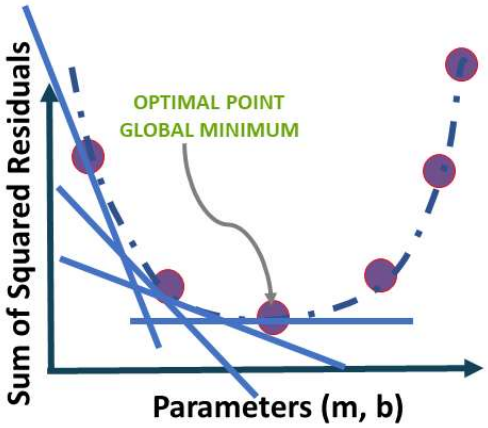


- We need to first formulate a loss function as follows:

$$Cost\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(error)^2 = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

HOW DO ANN TRAIN? GRADIENT DESCENT

$$Loss\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



GRADIENT DESCENT WORKS AS FOLLOWS:

1. Calculate the gradient (derivative) of the Loss function $\frac{\partial loss}{\partial w}$
2. Pick random values for weights (m, b) and substitute
3. Calculate the step size (how much are we going to update the parameters?)

$$Step\ size\ = learning\ rate * gradient = \alpha * \frac{\partial loss}{\partial w}$$

4. Update the parameters and repeat

$$new\ weight\ = old\ weight - step\ size$$

$$w_{new} = w_{old} - \alpha * \frac{\partial loss}{\partial w}$$

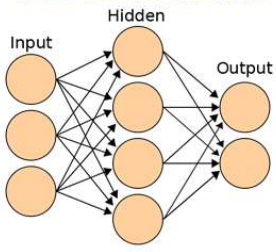
**Note: in reality, this graph is 3D and has three axes, one for m, b and sum of squared residuals*

TASK #10: UNDERSTAND THE THEORY AND INTUITION BEHIND RECURRENT NEURAL NETWORKS

RECURRENT NEURAL NETWORKS (RNN): WHAT ARE THEY?

- Feedforward Neural Networks (vanilla networks) map a fixed size input (such as image) to a fixed size output (classes or probabilities).
- A drawback in Feedforward networks is that they do not have any time dependency or memory effect.
- A RNN is a type of ANN that is designed to take temporal dimension into consideration by having a memory (internal state) (feedback loop).

FEED FORWARD ANN



RECURRENT NEURAL NETWORK

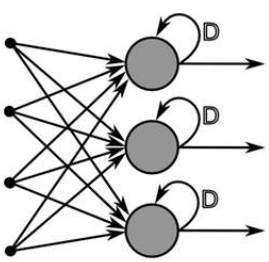
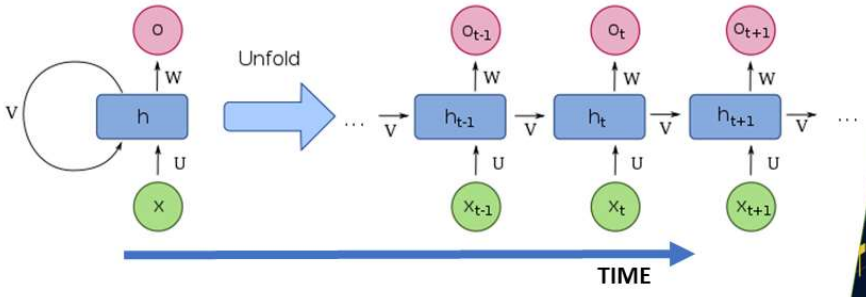


Photo Credit: https://commons.wikimedia.org/wiki/File:RecurrentLayerNeuralNetwork_english.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

RNN ARCHITECTURE

- A RNN contains a temporal loop in which the hidden layer not only gives an output but it feeds itself as well.
- An extra dimension is added which is time!
- RNN can recall what happened in the previous time stamp so it works great with sequence of text.



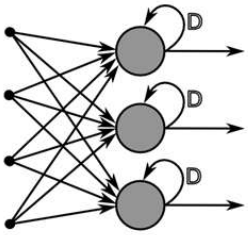
"We'll train RNNs to generate text character by character and ponder the question "how is that even possible?"

Source: The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

WHAT MAKES RNNs SO SPECIAL?

- Feedforward ANNs are so constrained with their fixed number of input and outputs.
- For example, a CNN will have fixed size image (28x28) and generates a fixed output (class or probabilities).
- Feedforward ANN have a fixed configuration, i.e.: same number of hidden layers and weights.
- Recurrent Neural Networks offer huge advantage over feedforward ANN and they are much more fun!
- RNN allow us to work with a sequence of vectors:
  - Sequence in inputs
  - Sequence in outputs
  - Sequence in both!



Source: The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Photo Credit: https://commons.wikimedia.org/wiki/File:RecurrentLayerNeuralNetwork_english.png

WHAT MAKES RNNs SO SPECIAL?

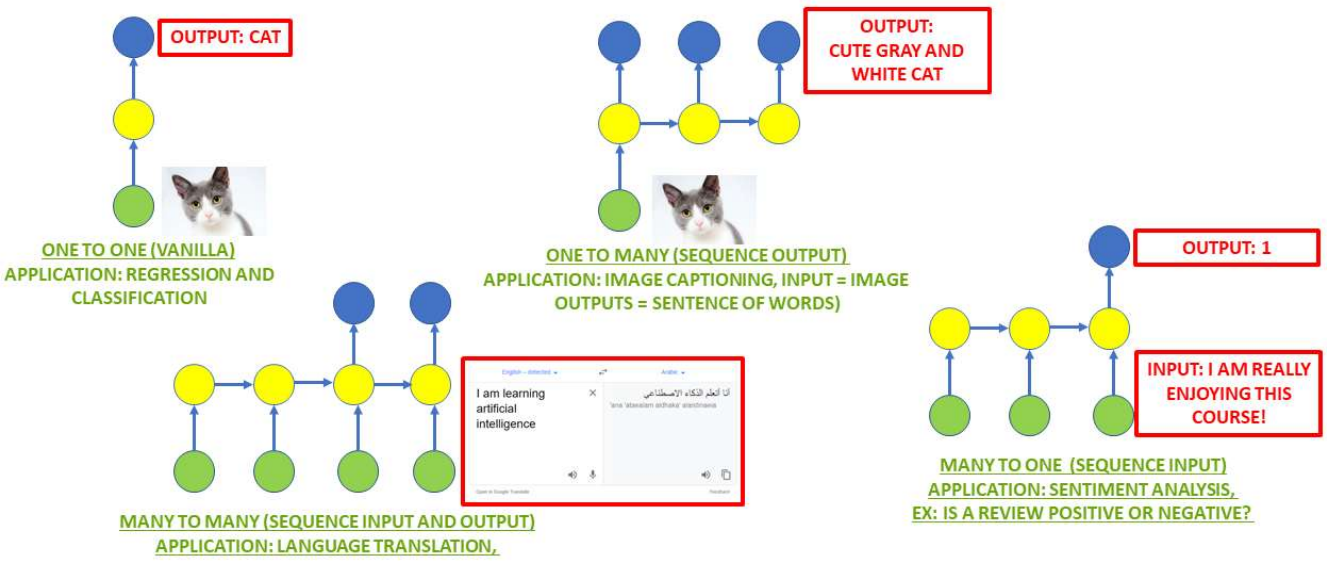


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

RNN MATH

- A RNN accepts an input x and generate an output o.
- The output o does not depend on the input x alone, however, it depends on the entire history of the inputs that have been fed to the network in previous time steps.
- Two equations that govern the RNN are as follows:
  - **INTERNAL STATE UPDATE:**
    $$h_t = \tanh(X_t * U + h_{t-1} * V)$$
  - **OUTPUT UPDATE:**
    $$o_t = softmax(W * h_t)$$

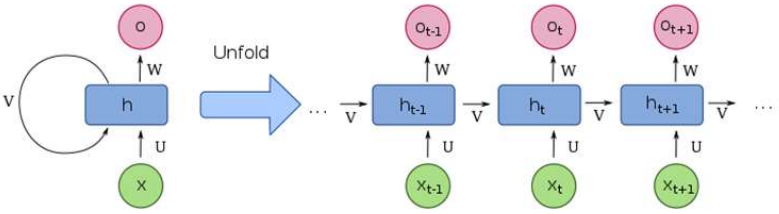


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

LET'S WATCH THIS MOVIE WRITTEN BY AN RNN!

- Let's watch a movie written by AI!
https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/
- The movie was written by an LSTM recurrent neural network
- The LSTM network was trained with a corpus of dozens of sci-fi screenplays from movies from the 1980s and 90s.

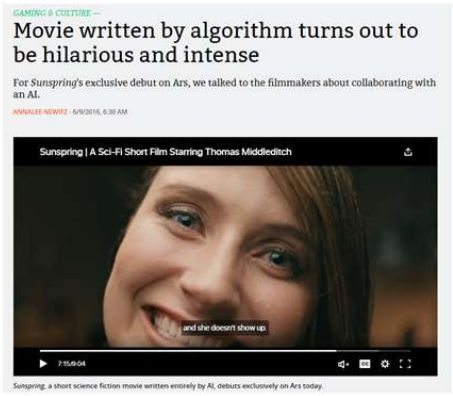


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

TASK #11: UNDERSTAND THE THEORY AND INTUITION BEHIND LONG SHORT TERM MEMORY NETWORKS

VANISHING GRADIENT PROBLEM

- LSTM networks work much better compared to vanilla RNN since they overcome the vanishing gradient problem.
- The error has to propagate through all the previous layers resulting in a vanishing gradient.
- As the gradient goes smaller, the network weights are no longer updated.
- As more layers are added, the gradients of the loss function approaches zero, making the network hard to train.

EACH LAYER DEPENDS ON THE OUTPUT FROM THE PREVIOUS LAYERS, THE "V" IS MULTIPLIED SEVERAL TIMES RESULTING IN VANISHING GRADIENT

0.1 * 0.1 * 0.1 *.....*0.1 =1e-10

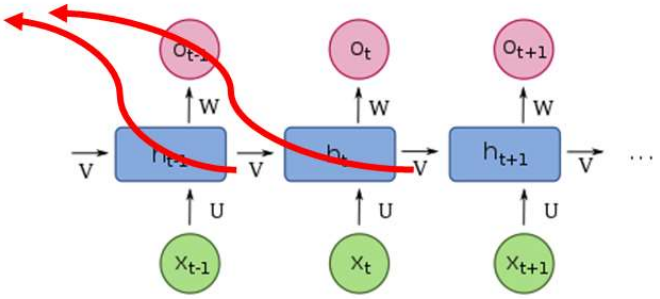
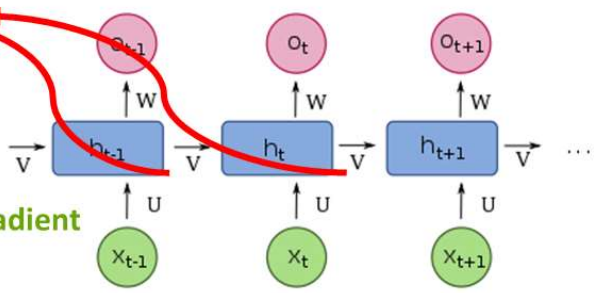


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

VANISHING GRADIENT PROBLEM

- ANN gradients are calculated during backpropagation.
- In backpropagation, we calculate the derivatives of the network by moving from the outermost layer (close to output) back to the initial layers (close to inputs).
- The chain rule is used during this calculation in which the derivatives from the final layers are multiplied by the derivatives from early layers.
- The gradients keeps diminishing exponentially and therefore the weights and biases are no longer being updated.

EACH LAYER DEPENDS ON THE OUTPUT FROM THE PREVIOUS LAYERS, THE "V" IS MULTIPLIED SEVERAL TIMES RESULTING IN VANISHING GRADIENT, (ex: 0.1 * 0.1 * 0.1 *.....*0.1 =1e-10)



New Weight = Old Weight – Learning rate * gradient

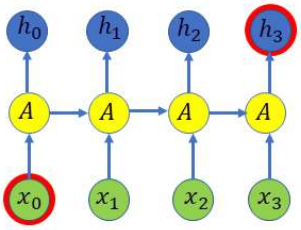
9.09999 = 10.1 – 1 * 0.001

Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

LSTM INTUITION

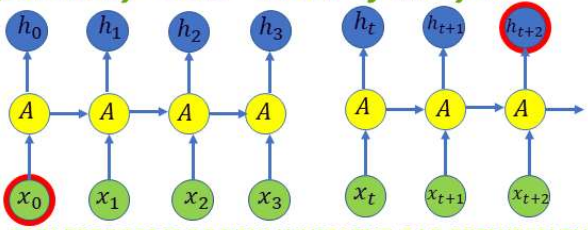
- LSTM networks work better compared to vanilla RNN since they overcome vanishing gradient problem.
- In practice, RNN fail to establish long term dependencies.
- Reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The tree color is "green"



RNN PERFORMS WELL SINCE THE GAP BETWEEN THE PREDICTION "GREEN" AND THE NECESSARY CONTEXT INFORMATION "TREE" IS SMALL

I live in Quebec in Northern Canada.....where I live, the weather is generally "cold" most of the year

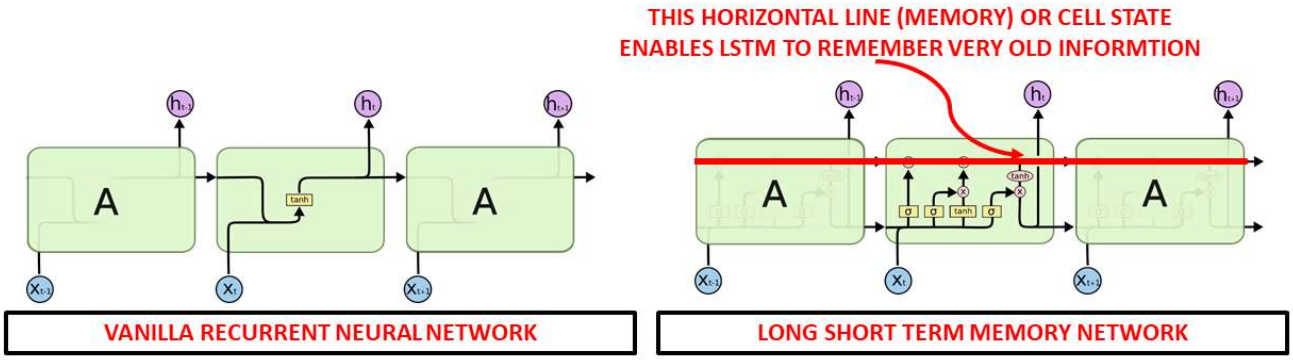


RNN PERFORMS POORLY WHEN THE GAP BETWEEN THE PREDICTION "COLD" AND THE NECESSARY CONTEXT INFORMATION "CANADA" IS LARGE

Reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM INTUITION

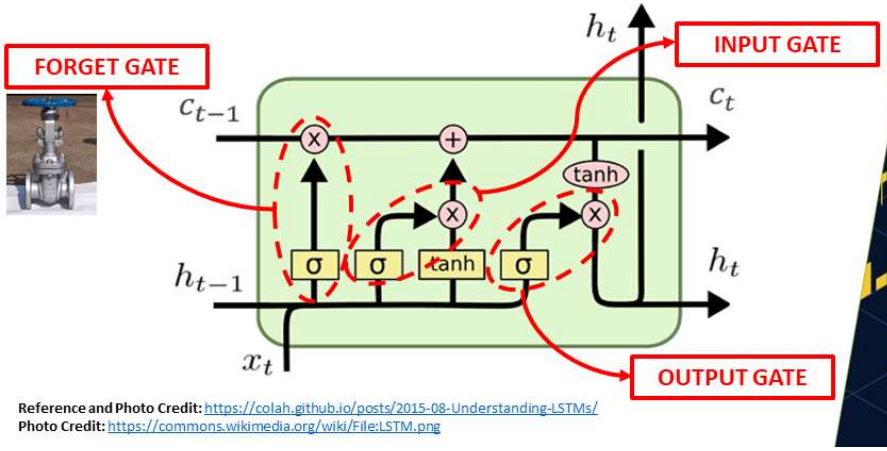
- LSTM networks are type of RNN that are designed to remember long term dependencies by default.
- LSTM can remember and recall information for a prolonged period of time.
- Recall that each line represents a full vector.



Reference and Photo Credit:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM INTUITION – GATES

- LSTM contains gates that can allow or block information from passing by.
- Gates consist of a sigmoid neural net layer along with a pointwise multiplication operation.
- Sigmoid output ranges from 0 to 1:
  0 = Don't allow any data to flow
  1 = Allow everything to flow!



Reference and Photo Credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Photo Credit: https://commons.wikimedia.org/wiki/File:LSTM.png

TASK #12: TRAIN AN LSTM TIME SERIES MODEL

http://playground.tensorflow.org/

```
# Source: https://stackoverflow.com/questions/32419510/how-to-get-repr
import os
```

```
import random
from sklearn.metrics import r2_score
import tensorflow as tf

# Let's create a function with keras to make predictions
def prices_keras_predictions(X_train, y_train, X_test, y_test, X_eval,
                             lstm=150, metrics=None, seed=101, verbose
  # Set the seed for random
  os.environ['PYTHONHASHSEED']=str(seed)
  np.random.seed(seed)
  random.seed(seed)
  tf.random.set_seed(seed)

  # Let´s prepare the Dataframe
  df_predicted = pd.DataFrame(data={'Close': X_eval[:,0]}, index=df_ir

  # Reshape the 1D arrays to 3D arrays to feed in the model
  X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1
  X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
  X_eval = np.reshape(X_eval, (X_eval.shape[0], X_eval.shape[1], 1))

  # Create the model
  inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2
  layer = keras.layers.LSTM(lstm, return_sequences= True)(inputs)
  layer = keras.layers.Dropout(0.3)(layer)
  layer = keras.layers.LSTM(lstm, return_sequences=True)(layer)
  layer = keras.layers.Dropout(0.3)(layer)
  layer = keras.layers.LSTM(lstm)(layer)
  outputs = keras.layers.Dense(1, activation='linear')(layer)

  model = keras.Model(inputs=inputs, outputs=outputs)
  model.compile(optimizer='adam', loss="mean_absolute_error", metrics=

  if verbose==2:
    print(model.summary())

  # Train the model
  history = model.fit(X_train, y_train, epochs = 20,
                      batch_size = 32, validation_split = 0.2,
                      verbose=verbose)

  # Test the model and calculate its accuracy
  # score = model.evaluate(X_test, y_test, verbose=verbose==2)
  y_pred = model.predict(X_test, verbose=verbose)
  score = r2_score(y_test, y_pred.flatten())
```

```
    print(f"Keras model r2 score:", score)

    # Make Prediction
    predicted_prices = model.predict(X_eval, verbose=verbose)

    # Complete the dataframe with the predictions
    df_predicted['Prediction'] = predicted_prices

    return model, score, predicted_prices, df_predicted, history


# Let's get individual stock prices and volumes for sp500
stock_name = 'sp500'

# (1) Get individual stock prices and volumes for stock
price_volume_target_df = trading_window(individual_stock(stock_price_c

# (2) Let's scale and split the data
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# # (3) Let's show the training and testing data
# show_plot(X_train, f'{stock_name} Training Data')
# show_plot(X_test, f'{stock_name} Testing Data')

#X, np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))


# Create the model
metrics = None # ['accuracy', 'mae','mse','mape']
model, score, predicted_prices, df_predicted, history = prices_keras_p
```

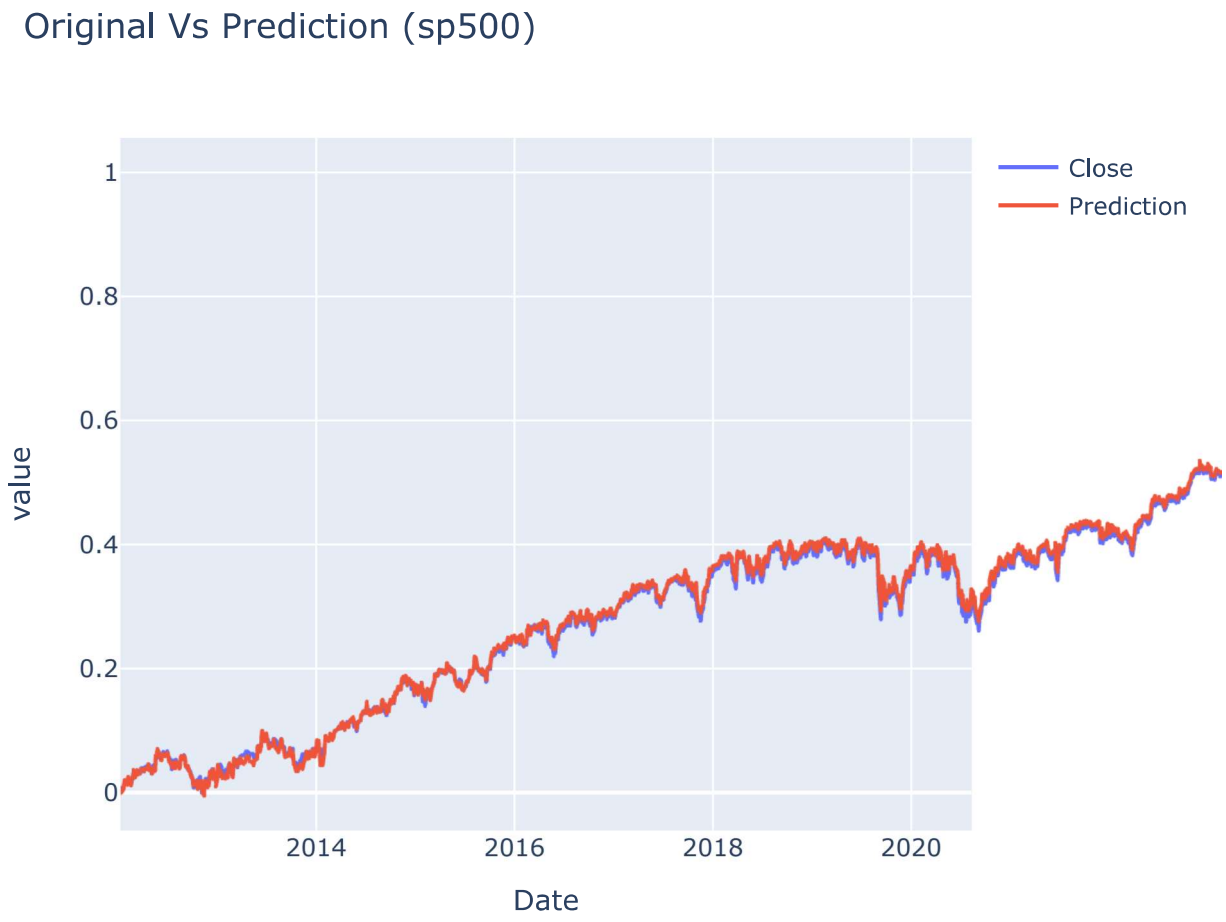
```
 lstm_1 (LSTM)               (None, 2, 150)            180600

 dropout_1 (Dropout)         (None, 2, 150)            0

 lstm_2 (LSTM)               (None, 150)               180600

 dense (Dense)               (None, 1)                 151

=================================================================
Total params: 452,551
Trainable params: 452,551
Non-trainable params: 0
_________________________________________________________________
None
```

```
None
Epoch 1/20
38/38 - 8s - loss: 0.1234 - val_loss: 0.1328 - 8s/epoch - 222ms/step
Epoch 2/20
38/38 - 1s - loss: 0.0421 - val_loss: 0.0313 - 742ms/epoch - 20ms/step
Epoch 3/20
38/38 - 1s - loss: 0.0243 - val_loss: 0.0349 - 736ms/epoch - 19ms/step
Epoch 4/20
38/38 - 1s - loss: 0.0200 - val_loss: 0.0357 - 728ms/epoch - 19ms/step
Epoch 5/20
38/38 - 1s - loss: 0.0171 - val_loss: 0.0356 - 714ms/epoch - 19ms/step
Epoch 6/20
38/38 - 1s - loss: 0.0151 - val_loss: 0.0260 - 734ms/epoch - 19ms/step
Epoch 7/20
38/38 - 1s - loss: 0.0156 - val_loss: 0.0354 - 752ms/epoch - 20ms/step
Epoch 8/20
38/38 - 1s - loss: 0.0143 - val_loss: 0.0261 - 717ms/epoch - 19ms/step
Epoch 9/20
38/38 - 1s - loss: 0.0143 - val_loss: 0.0585 - 728ms/epoch - 19ms/step
Epoch 10/20
38/38 - 1s - loss: 0.0154 - val_loss: 0.0385 - 782ms/epoch - 21ms/step
Epoch 11/20
38/38 - 1s - loss: 0.0145 - val_loss: 0.0367 - 725ms/epoch - 19ms/step
Epoch 12/20
38/38 - 1s - loss: 0.0127 - val_loss: 0.0377 - 715ms/epoch - 19ms/step
Epoch 13/20
38/38 - 1s - loss: 0.0132 - val_loss: 0.0376 - 727ms/epoch - 19ms/step
Epoch 14/20
38/38 - 1s - loss: 0.0124 - val_loss: 0.0221 - 749ms/epoch - 20ms/step
Epoch 15/20
38/38 - 1s - loss: 0.0113 - val_loss: 0.0125 - 823ms/epoch - 22ms/step
Epoch 16/20
38/38 - 1s - loss: 0.0108 - val_loss: 0.0134 - 751ms/epoch - 20ms/step
Epoch 17/20
38/38 - 1s - loss: 0.0104 - val_loss: 0.0161 - 748ms/epoch - 20ms/step
Epoch 18/20
38/38 - 1s - loss: 0.0107 - val_loss: 0.0269 - 726ms/epoch - 19ms/step
Epoch 19/20
38/38 - 1s - loss: 0.0123 - val_loss: 0.0053 - 737ms/epoch - 19ms/step
Epoch 20/20
38/38 - 1s - loss: 0.0106 - val_loss: 0.0052 - 723ms/epoch - 19ms/step
21/21 - 1s - 1s/epoch - 69ms/step
Keras model r2 score: 0.9206530261897211
68/68 - 0s - 457ms/epoch - 7ms/step
```

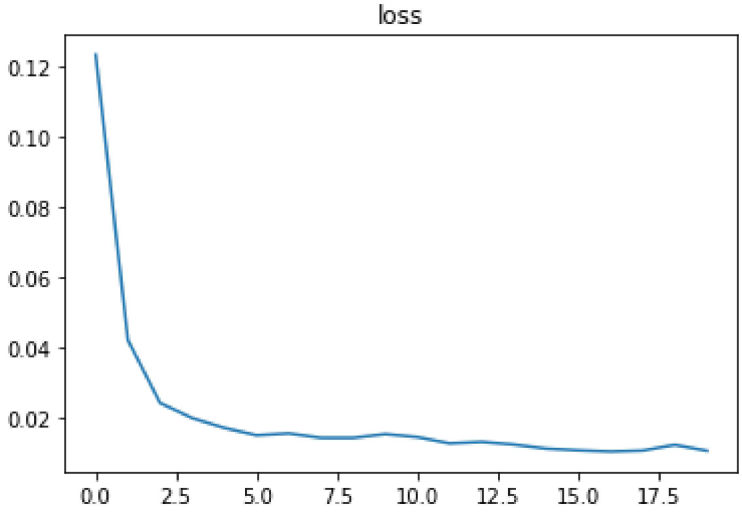
```
# Let's review the predictions
df_predicted
```

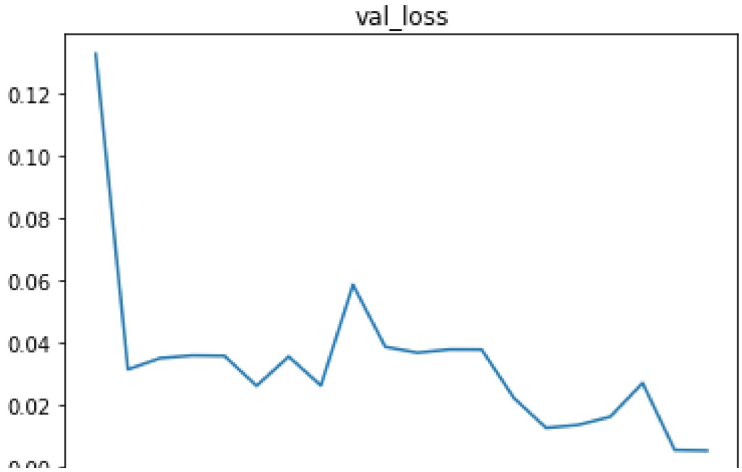
Date	Close	Prediction
2012-01-12	0.0083	0.0066
2012-01-13	0.0052	0.0000
2012-01-17	0.0074	0.0056
2012-01-18	0.0142	0.0136
2012-01-19	0.0173	0.0204
...	...	...

```
# Plot the data
interactive_plot(df_predicted, f"Original Vs Prediction ({stock_name})
```




```
for key in history.history.keys():
  plt.plot(history.history[key])
  plt.title(key)
  plt.show();
```





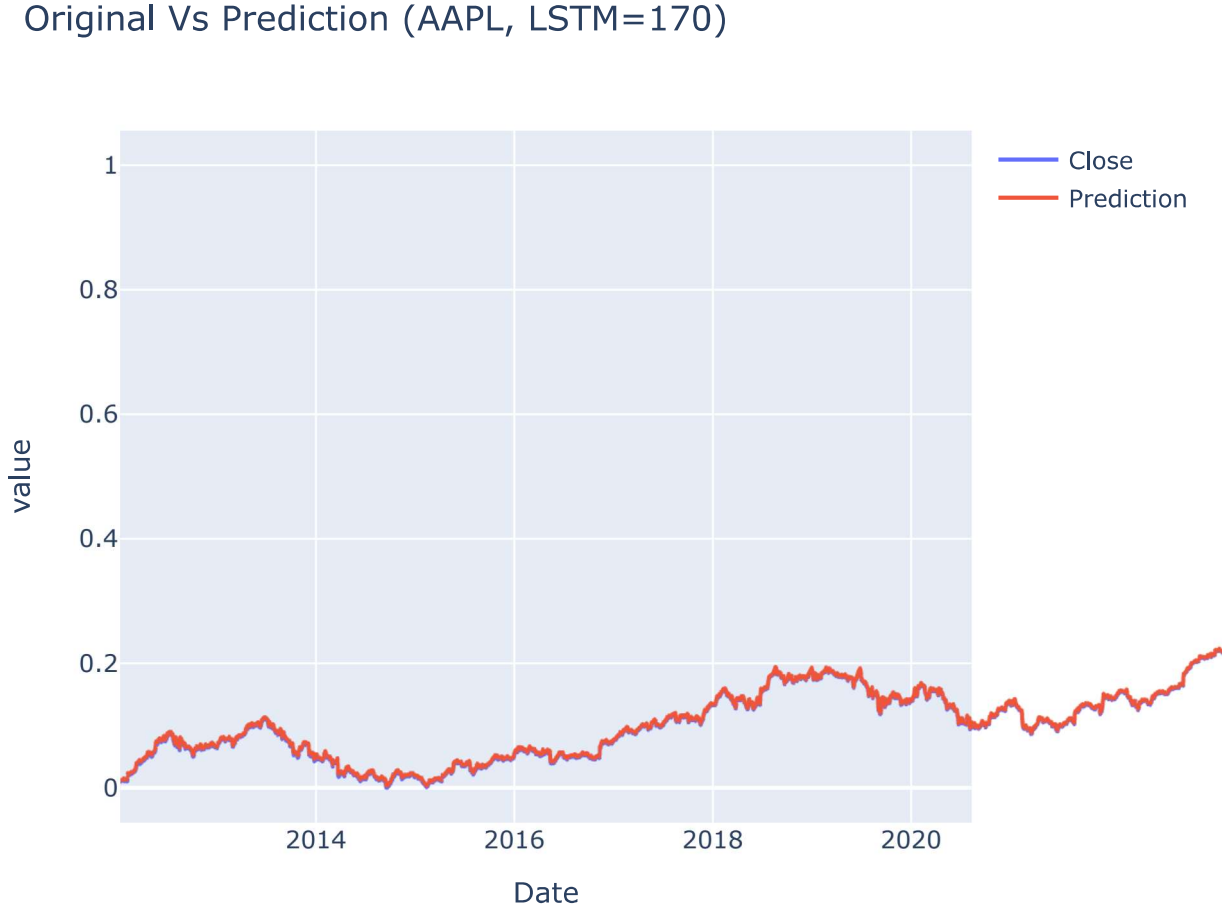
MINI CHALLENGE #5:

- **Test the pipeline with at least 3 other stocks**
- **Experiment with various LSTM model parameters (Ex: Use 150 units instead of 50), print out the model summary and retrain the model**

```
stock_name = 'AAPL'
# Let's prepare the data
price_volume_target_df = trading_window(individual_stock(stock_price_c
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# Let's find the best model
metrics = None # ['accuracy', 'mae','mse','mape']
best_score = 0
best_df_predicted = best_lstm = None
for lstm in [130, 140, 150, 160, 170]:
  model, score, predicted_prices, df_predicted, history = prices_keras




  if score > best_score:
```

```
    best_score, best_df_predicted, best_lstm = score, df_predicted, ls

# Let's present the best model
interactive_plot(best_df_predicted, f"Original Vs Prediction ({stock_r
```

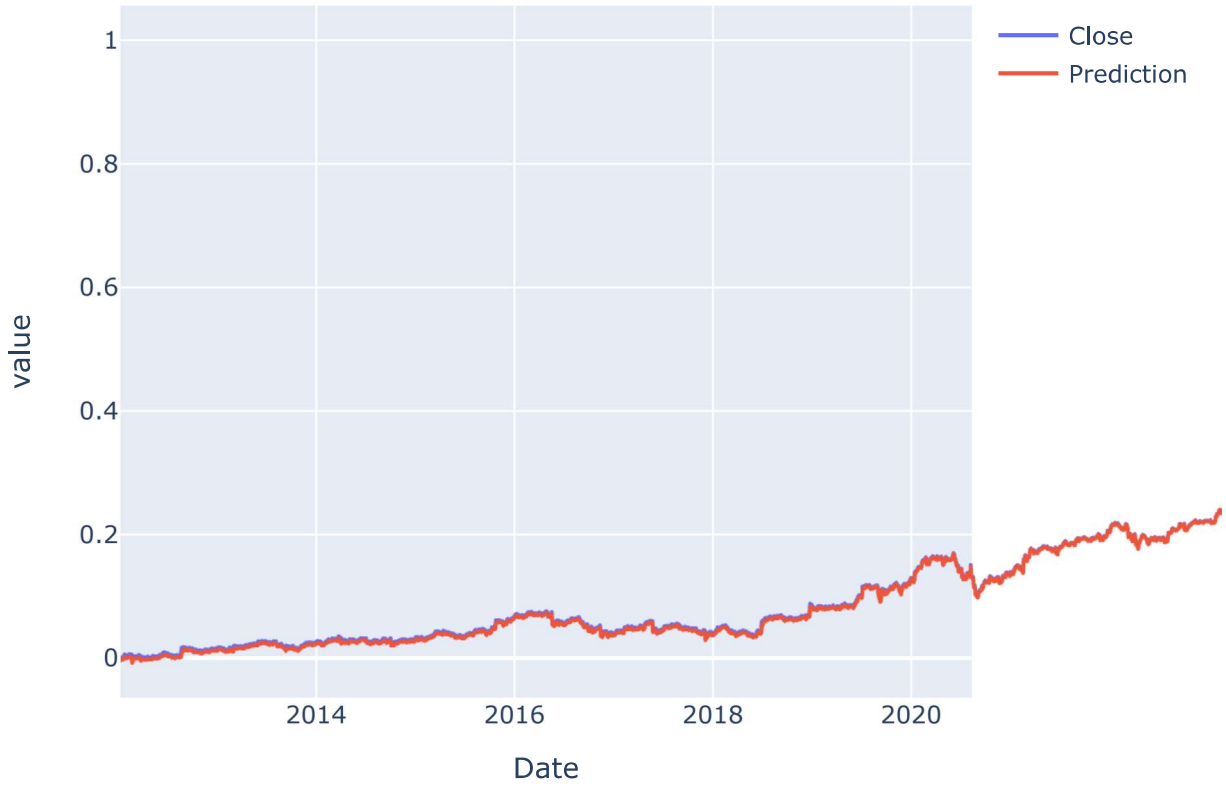
```
Keras model r2 score: 0.9522721722131753
Keras model r2 score: 0.9335375121151852
Keras model r2 score: 0.8919529336857859
Keras model r2 score: 0.9463822011442395
Keras model r2 score: 0.963256233373001
```




```
stock_name = 'AMZN'
# Let's prepare the data
price_volume_target_df = trading_window(individual_stock(stock_price_c
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# Let's find the best model
metrics = None # ['accuracy', 'mae','mse','mape']
best_score = 0
best_df_predicted = best_lstm = None
for lstm in [130, 140, 150, 160, 170]:
  model, score, predicted_prices, df_predicted, history = prices_keras
```

```
  if score > best_score:
    best_score, best_df_predicted, best_lstm = score, df_predicted, ls

# Let's present the best model
interactive_plot(best_df_predicted, f"Original Vs Prediction ({stock_r
```

```
Keras model r2 score: 0.8666343160648566
Keras model r2 score: 0.8436734931251033
Keras model r2 score: 0.9027172936322028
Keras model r2 score: 0.7261179548759333
Keras model r2 score: 0.9496712110188226
```

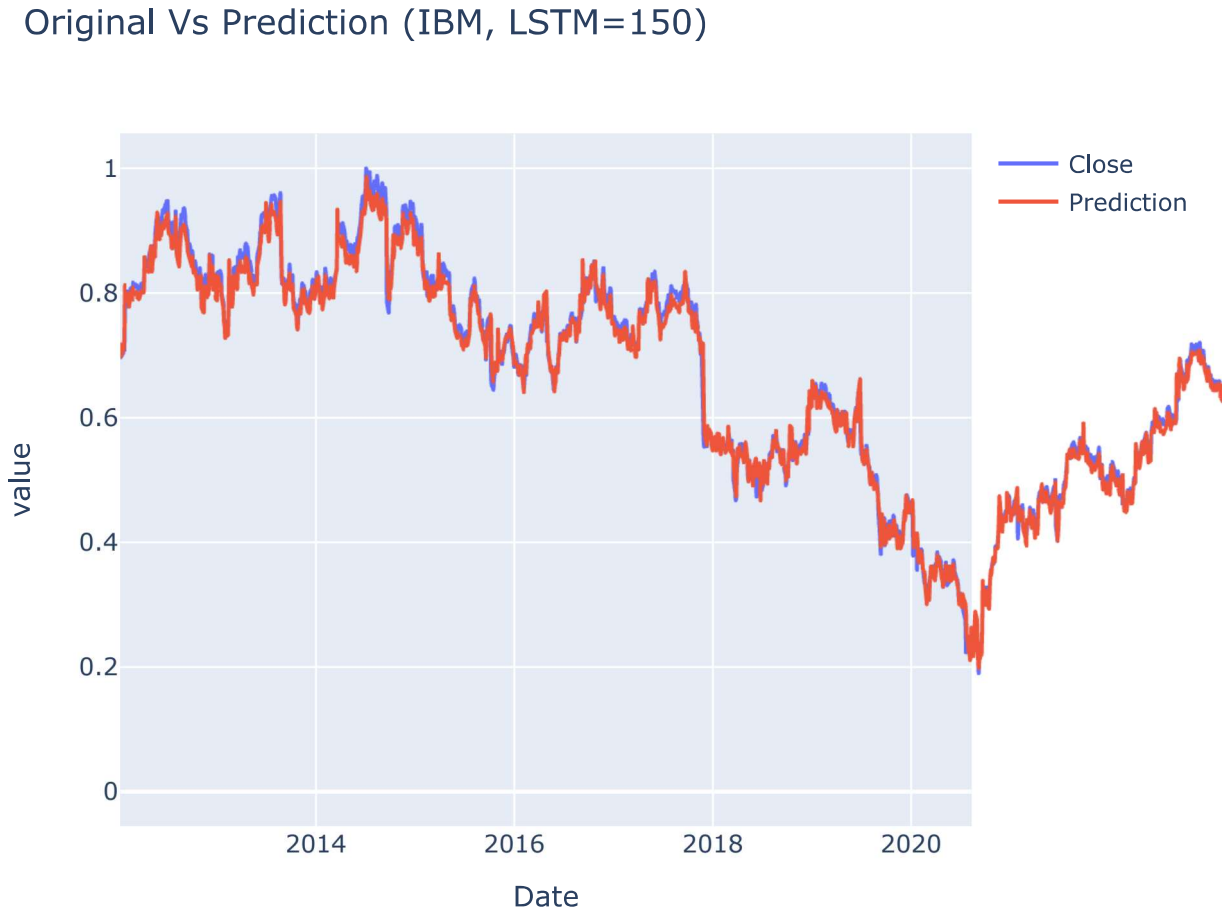



```
stock_name = 'IBM'
# Let's prepare the data
price_volume_target_df = trading_window(individual_stock(stock_price_c
X, y, X_train, y_train, X_test, y_test = scaling_and_splitting_data(pr

# Let's find the best model
metrics = None # ['accuracy', 'mae','mse','mape']
best_score = 0
```

```
best_df_predicted = best_lstm = None
for lstm in [130, 140, 150, 160, 170]:
  model, score, predicted_prices, df_predicted, history = prices_keras
```

```
  if score > best_score:
    best_score, best_df_predicted, best_lstm = score, df_predicted, ls

# Let's present the best model
interactive_plot(best_df_predicted, f"Original Vs Prediction ({stock_r
```

```
Keras model r2 score: 0.9257328323606243
Keras model r2 score: 0.9135125093778195
Keras model r2 score: 0.9313880495412741
Keras model r2 score: 0.9236042764605994
Keras model r2 score: 0.9211226807060822
```



EXCELLENT JOB! YOU SHOULD BE TRULY PROUD WITH THE NEWLY ACQUIRED SKILLS

MINI CHALLENGE SOLUTIONS

MINI CHALLENGE #1 SOLUTION:

- **What is the average trading volume for Apple stock?**
- **What is the maximum trading volume for sp500?**
- **Which security is traded the most? comment on your answer**
- **What is the average stock price of the S&P500 over the specified time period?**
- **What is the maximum price of Tesla Stock?**

```
# Get the statistical data for the stocks volume dataframe
# Average trading volume for Apple stock is 2.498238e+06
# Average trading volume for S&P500 is 3.680732e+09

# Why S&P500 is the most traded of all? Check out this article:
# https://www.investopedia.com/articles/personal-finance/022216/put-10

# Text from the article above:
# "The S&P 500 index is a broad-based measure of large corporations tr
# passively holding the index often produces better results than activ
# Over long-time horizons, the index typically produces better returns

stock_vol_df.describe()
```

	AAPL	BA	T	MGM	Al
count	2,159.0000	2,159.0000	2,159.0000	2,159.0000	2,159.0(
mean	58,203,317.4155	6,419,915.9333	28,321,313.5711	9,845,581.7045	4,102,672.9(
std	45,681,411.9012	9,711,873.1540	14,289,105.8976	7,295,752.6435	2,290,722.3:
min	11,362,000.0000	788,900.0000	6,862,400.0000	950,700.0000	881,300.0(
25%	27,699,300.0000	3,031,850.0000	20,021,500.0000	5,796,450.0000	2,675,700.0(
50%	42,094,200.0000	3,991,000.0000	24,859,300.0000	7,899,800.0000	3,494,800.0(
75%	71,824,800.0000	5,325,900.0000	32,105,650.0000	11,040,550.0000	4,768,150.0(
max	376,530,000.0000	103,212,800.0000	195,082,700.0000	90,098,200.0000	23,856,100.0(

```
# Get the statistical data for the prices dataframe
stock_price_df.describe()
```

	AAPL	BA	T	MGM	AMZN	IBM	TSLA
count	2,159.0000	2,159.0000	2,159.0000	2,159.0000	2,159.0000	2,159.0000	2,159.0000
mean	140.8198	189.9427	35.1629	23.1057	915.6657	161.8530	259.6008
std	70.8276	103.6786	3.2075	6.9638	697.8389	25.5619	210.9880
min	55.7900	67.2400	26.7700	7.1400	175.9300	94.7700	22.7900
25%	89.1657	124.0150	33.0400	18.5450	316.4900	142.7700	184.5950
50%	116.6000	142.4200	34.9300	23.7800	676.0100	156.9500	231.9600
75%	175.0200	297.0450	37.4200	28.4300	1,593.6450	185.9750	307.3500
max	455.6100	440.6200	43.4700	38.0300	3,225.0000	215.8000	1,643.0000

```
# Average price for S&P500 = 2218.749554
# Maximum Tesla Price = 1643.000000
```

MINI CHALLENGE #2 SOLUTION:

- **Plot the normalized stock prices and volume dataset.**

```
# # Plot interactive chart for volume data
# # Notice that S&P500 trading is orders of magnitude compared to indi
# interactive_plot(stock_vol_df, 'Stocks Volume')


# # plot interactive chart for normalized stocks prices data
# interactive_plot(normalize(stock_price_df), 'Stock Prices')

# # Let's normalize the data and re-plot interactive chart for volume
# interactive_plot(normalize(stock_vol_df), 'Normalized Volume')
```

MINI CHALLENGE #3 SOLUTION:

- **Test the pipeline with S&P500 and AMZN datasets instead of AAPL**

```
# Let's test the functions and get individual stock prices and volumes
price_volume_df = individual_stock(stock_price_df, stock_vol_df, 'sp50
price_volume_df
```

Date	Close	Volume
2012-01-12	1,295.5000	4019890000
2012-01-13	1,289.0900	3692370000
2012-01-17	1,293.6700	4010490000
2012-01-18	1,308.0400	4096160000
2012-01-19	1,314.5000	4465890000
...	...	...
2020-08-05	3,327.7700	4732220000
2020-08-06	3,349.1599	4267490000

```
# Let's test the functions and get individual stock prices and volumes
price_volume_df = individual_stock(stock_price_df, stock_vol_df, 'AMZN
price_volume_df
```

Date	Close	Volume
2012-01-12	175.9300	5385800
2012-01-13	178.4200	4753500
2012-01-17	181.6600	5644500
2012-01-18	189.4400	7473500
2012-01-19	194.4500	7096000
...	...	...
2020-08-05	3,205.0300	3930000
2020-08-06	3,225.0000	3940600
2020-08-07	3,167.4600	3929600
2020-08-10	3,148.1599	3167300
2020-08-11	3,080.6699	3706600

2159 rows × 2 columns

MINI CHALLENGE #4 SOLUTION:

- **Experiment with various regularization value for alpha**
- **What is the impact of increasing alpha?**
- **Note: default value for alpha is = 1**

```
from sklearn.linear_model import Ridge
# Note that Ridge regression performs linear least squares with L2 reg
# Create and train the Ridge Linear Regression  Model
regression_model = Ridge(alpha = 2)
regression_model.fit(X_train, y_train)
```

```
Ridge(alpha=2)
```

MINI CHALLENGE #5 SOLUTION:

- **Test the pipeline with at least 3 other stocks**
- **Experiment with various LSTM model parameters (Ex: Use 150 units instead of 50), print out the model summary and retrain the model**

```
# Create the model
# Reshape the 1D arrays to 3D arrays to feed in the model
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
x = keras.layers.LSTM(150, return_sequences= True)(inputs)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150, return_sequences=True)(x)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150)(x)
outputs = keras.layers.Dense(1, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss="mse")
model.summary()
```

```
Model: "model_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_17 (InputLayer)       [(None, 2, 1)]            0

 lstm_48 (LSTM)              (None, 2, 150)            91200

 dropout_32 (Dropout)        (None, 2, 150)            0

 lstm_49 (LSTM)              (None, 2, 150)            180600

 dropout_33 (Dropout)        (None, 2, 150)            0

 lstm_50 (LSTM)              (None, 150)               180600

 dense_16 (Dense)            (None, 1)                 151

=================================================================
Total params: 452,551
```

```
Trainable params: 452,551
Non-trainable params: 0
```
