# CS620-DataProject

# 2023

# *Final Project Report*

**Predicting the price of crypto currencies data using web scrapping with Scikit-Learn Machine Learning VS Deep Neural Networks Models.**

**Name:** Ashish Verma

**Email:** [averm004@odu.edu](mailto:averm004@odu.edu)

**UIN:** 01266453

**Cell:** 341-209-7315

**Portfolio**: [https://ashishodu2023.github.io/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fashishodu2023.github.io%2F)

# Content[\_](#_Toc152110261)

[*Final Project Report* 1](#_Toc152110263)

[Content 2](#_Toc152110265)

[Abstract 3](#_Toc152110266)

[Bitcoin History 4](#_Toc152110267)

[Problem Statement 5](#_Toc152110268)

[High Level Design 6](#_Toc152110269)

[Data Sources 7](#_Toc152110270)

[Python Packages 7](#_Toc152110271)

[Data Acquisition 9](#_Toc152110272)

[Data Preprocessing 10](#_Toc152110273)

[Exploratory Data Analysis 13](#_Toc152110274)

[Features Engineering 14](#_Toc152110275)

[Feature Selection Process 15](#_Toc152110276)

[Train and Test Split 17](#_Toc152110277)

[Model Selection and Why? 17](#_Toc152110278)

[Deep Learning Model with Pytorch 20](#_Toc152110279)

[Using lagged observations as features 21](#_Toc152110280)

[Generating features from timestamps 21](#_Toc152110281)

[Generating cyclical time features 22](#_Toc152110282)

[Model Configuration 23](#_Toc152110283)

[Model Architecture 24](#_Toc152110284)

[Pytorch Training 25](#_Toc152110285)

[Loss Plot for Out of Fold Vs Actual Data 26](#_Toc152110286)

[ Pytorch-widedeep -> https://github.com/jrzaurin/pytorch-widedeep 26](#_Toc152110287)

[ Pytorch tabular -> https://pytorch-tabular.readthedocs.io/en/latest/ 26](#_Toc152110288)

[ Raytune -> https://docs.ray.io/en/latest/tune/index.html 26](#_Toc152110289)

[ Optuna -> https://optuna.org/ 26](#_Toc152110290)

[ AutoMl -> https://www.automl.org/automl/ 26](#_Toc152110291)

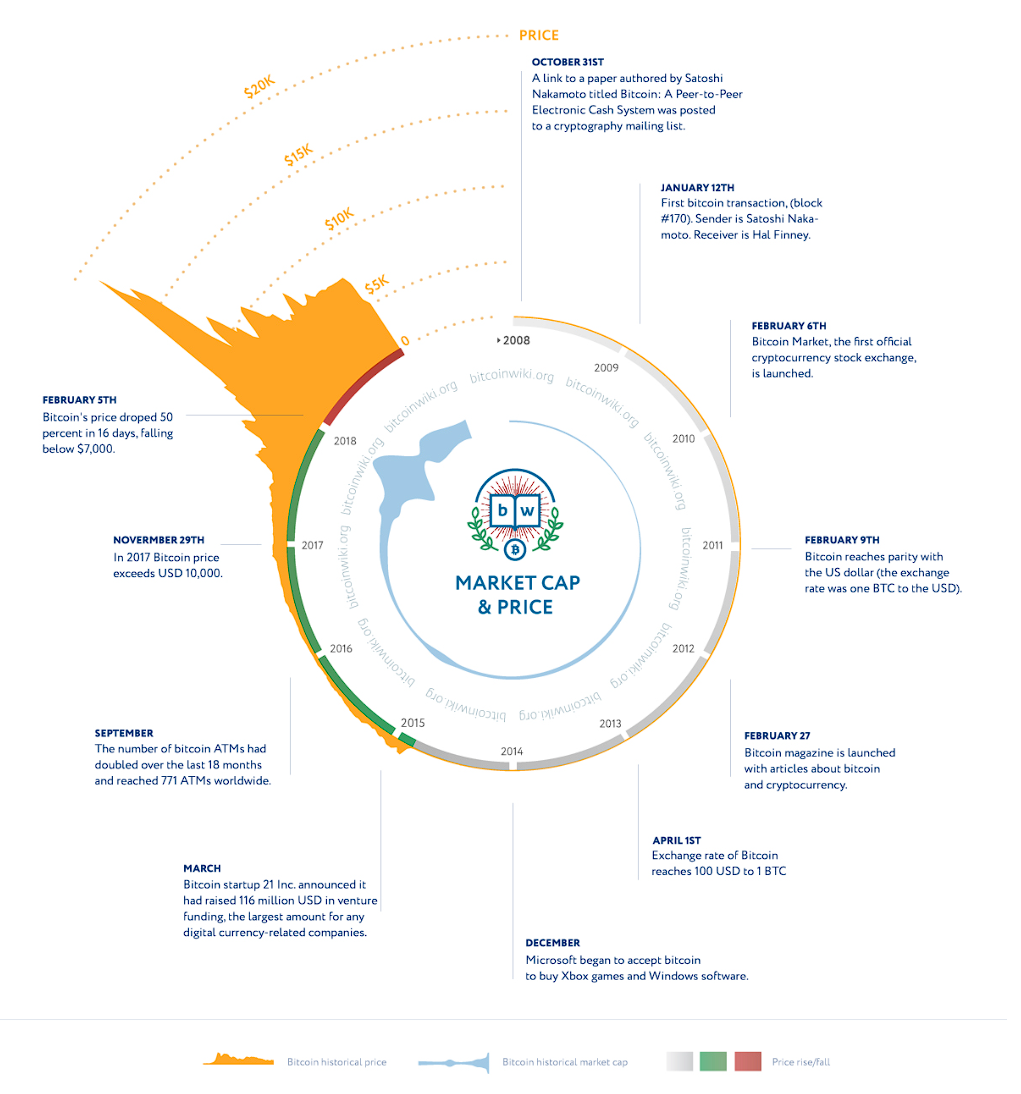
[Analysis Between Machine Learning and Deep Neural Network modelling 28](#_Toc152110295)

[Reference and Motivation 29](#_Toc152110297)

# Abstract

Traders and investors are interested in accurately predicting cryptocurrency prices to increase returns and minimize risk. However, due to their uncertainty, volatility, and dynamism, forecasting crypto prices is a challenging time series analysis task. Researchers have proposed predictors based on statistical, machine learning (ML), and deep learning (DL) approaches, but the literature is limited. With the help of this data project, we compare the performances of widely used statistical, ML, and DL approaches in the literature for predicting the price of the popular cryptocurrencies that is ***Bitcoin***.

# Bitcoin History

****

**https://en.bitcoinwiki.org/upload/en/images/c/cf/Bitcoin\_history\_price.jpg**

# Problem Statement

One of the main problems with crypto currencies is price volatility, which indicates the need for studying the underlying price model. This case study is based on Time-series forecasting of various crypto currencies prices mainly ***Bitcoin*** It proposed as regression problem. For regression the paper was predicting prices. It was performed to predict next day, 30th day & 90th day price. In this case study, we attempt to predict next day prices based on features of previous day using machine learning & deep learning regression algorithm

# High Level Design

***Identifying the correct data source and data volume.***

***Data acquisition and Web scrapping.***

***Data Pre-processing***

***Exploratory Data Analysis***

***Outlier Detection***

***Feature Engineering***

***Architecting Machine Learning Model using Scikit learn***

***Architecting Deep Learning Model using Pytorch***

***Saving and Loading Models for Deployment***

***Making Predictions***

***Metric Evaluation***

# Data Sources

**Website 1 :** [https://coinmarketcap.com](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fcoinmarketcap.com)

**Website 2:** [https://bitinfocharts.com](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fbitinfocharts.com)

**Time Period:** 01/01/2015 to current timestamp

**Historical Data Api:** [https://api.coinmarketcap.com/data-api/v3/cryptocurrency/historical](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fapi.coinmarketcap.com%2Fdata-api%2Fv3%2Fcryptocurrency%2Fhistorical)

**Yahoo Finance Api:**<https://finance.yahoo.com/quote/BTC-USD/history?period1=1420070400&period2=1697846400&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true>

# Python Packages

# Data Extraction and Manipulation

from  datetime import datetime,timedelta

import requests

import time

import os

from bs4 import BeautifulSoup

import pandas as pd

import json

from tqdm import tqdm

import numpy as np

import warnings

import quandl

import random

import gc

import yfinance as yf

import pickle

#Plots

import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib as mpl

import matplotlib.lines as mlines

import plotly.express as px

import plotly.graph\_objects as go

warnings.filterwarnings("ignore")

#Google Drive

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

#EDA

import pandas\_ta as ta

#Machine Learning

import sklearn.metrics as metrics

from tqdm.notebook import tqdm

from sklearn.dummy import DummyRegressor

from sklearn.svm import SVR

from sklearn.linear\_model import SGDRegressor

import xgboost as xgb

from sklearn.linear\_model import HuberRegressor

from sklearn.linear\_model import QuantileRegressor

from sklearn.model\_selection import TimeSeriesSplit

#Pytorch

import torch

from torch.optim.lr\_scheduler import ReduceLROnPlateau

from torch.utils.data import TensorDataset, DataLoader

#Tensorflow

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.layers import LSTM

# Data Acquisition

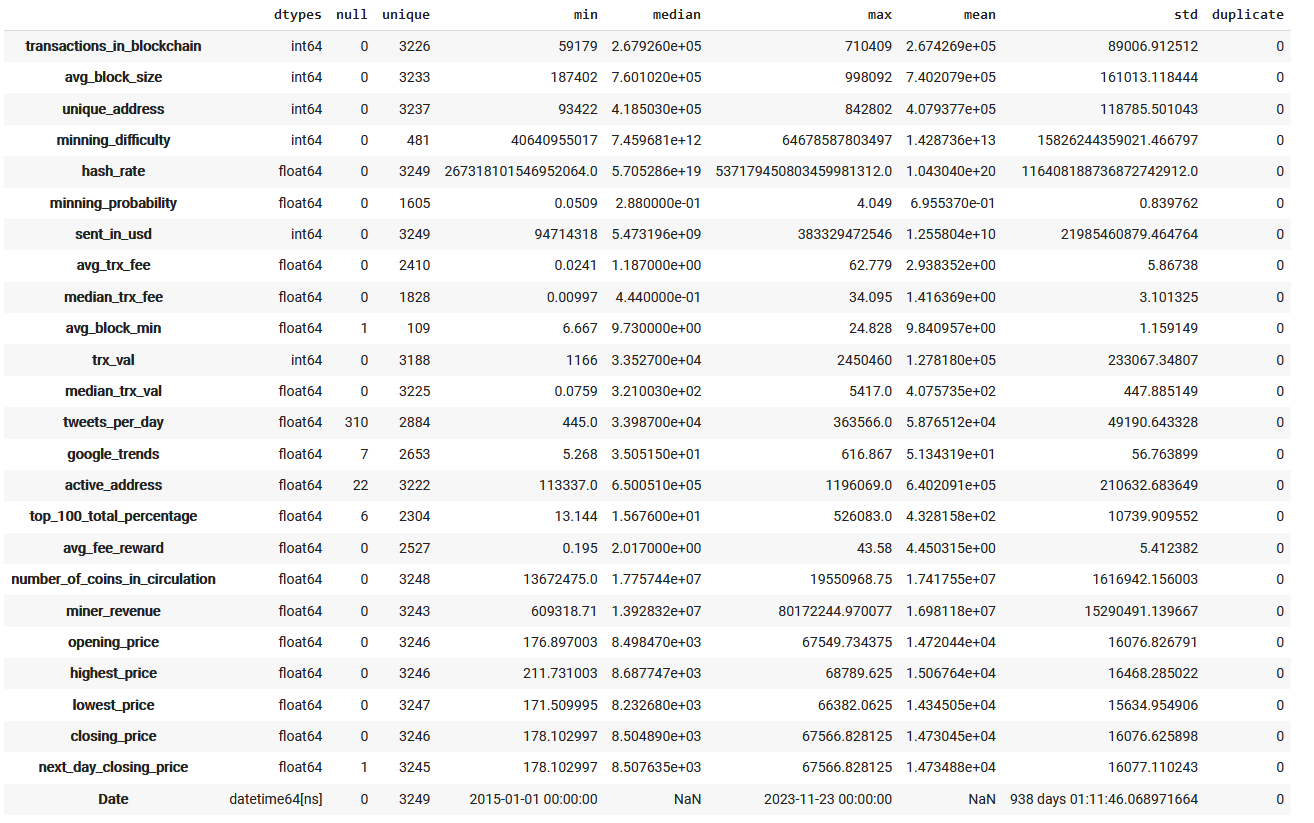
The **class Acquisition** mainly deals with acquiring two type data sources using two different techniques. The historical data and incremental data of ***Bitcoin.*** The historical data is extracted from yahoo finance Api and saved in the google drive since 01/01/2015 and incremental data is extracted with the help of ***web scrapping*** techniques using beautiful soap python package from bitinfocharts.com and coinmarketcap.com which provides free Api to get ***Bitcoin*** data to be used for student projects. The historical dataset contains fields like Date ,Open,High,Low,Close,Adj-Close and Volume. Overall, there are 24 fields in raw dataset extracted from Api as well as web scrapping.< 'Date', 'transactions\_in\_blockchain', 'avg\_block\_size', 'unique\_address', 'minning\_difficulty', 'hash\_rate', 'minning\_probability', 'sent\_in\_usd', 'avg\_trx\_fee', 'median\_trx\_fee', 'avg\_block\_min', 'trx\_val', 'median\_trx\_val', 'tweets\_per\_day', 'google\_trends', 'active\_address', 'top\_100\_total\_percentage', 'avg\_fee\_reward', 'number\_of\_coins\_in\_circulation', 'miner\_revenue', 'opening\_price', 'highest\_price', 'lowest\_price', 'closing\_price'>.***The challenging part the data acquisition layer was to identity the tags and labels from scrapping and converting them into pandas’ data frame.Earlier I had plan to use “investpy” python package but it no working anymore to I went for alternate route.***

# Data Preprocessing

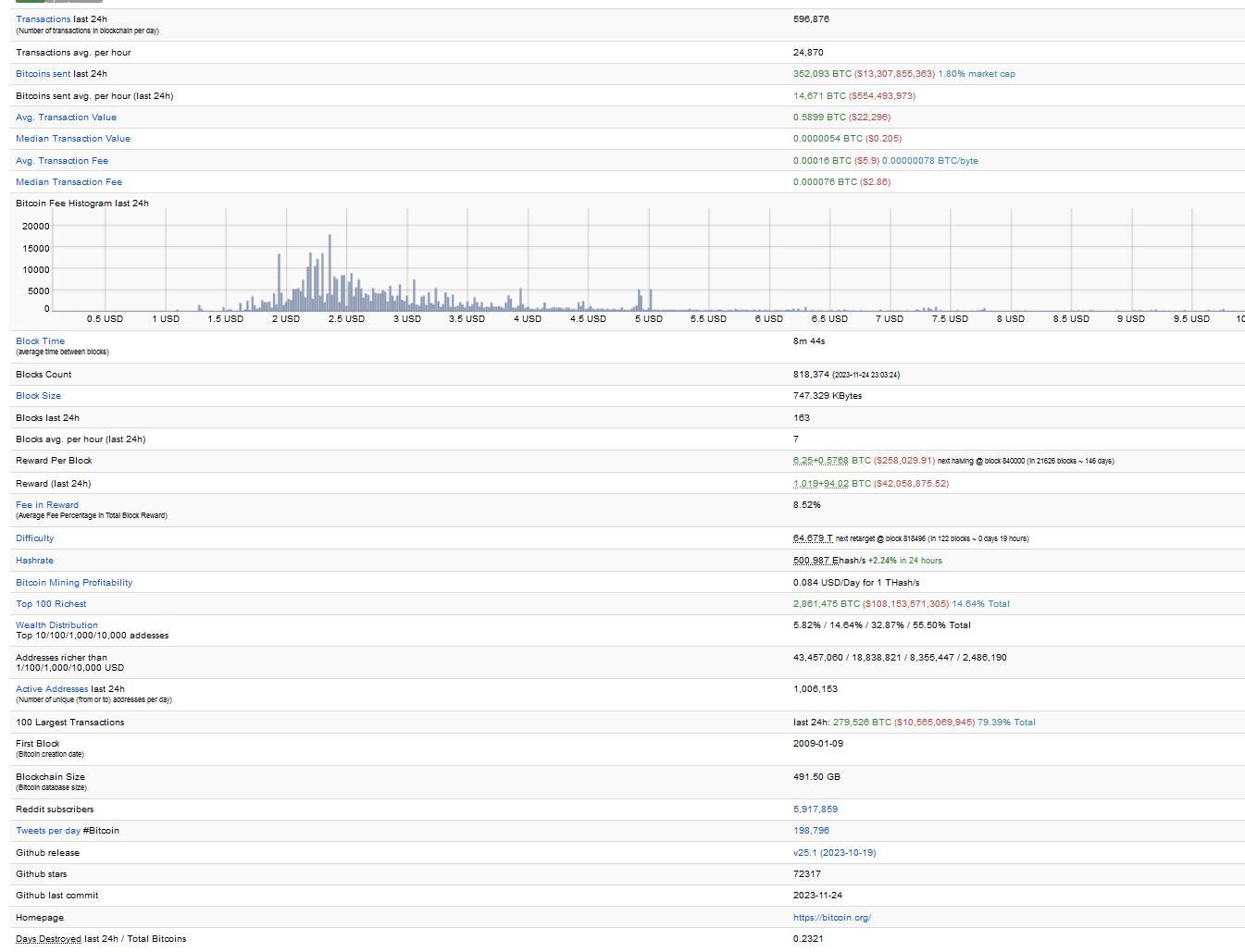
After we have collected the raw dataset using pandas data frame next step is to clean the datasets, any machine learning algorithms don’t like dirty data, there were many data cleansing and preprocessing techniques are show cased in the current implementation. Here comes **class Preprocessing** which helps us to get clean data frame for further processing and feature extraction. The main activities performed by preprocessing class are as follows:

1. ***Generate the target label(next\_day\_closing\_price)***
2. ***Replace NULL values with NANs in the data frame.***
3. ***Convert object data types to float datatypes.***
4. ***Get data frame summary information which includes(null,unique,min,median,max,mean,std and duplicate).***
5. ***Fill missing values with mean values for better results.***
6. ***Remove null and inf values from data frame.***
7. ***Finally remove NANs from data frame.***

Example of Summary Data frame is shown below:



Fields from Bitinofcharts.com which are part of ***clean\_raw\_data*** data frame



# Exploratory Data Analysis

After cleansing the data its time to perform data analysis on numerical fields. There are close to 25 numerical fields which can affect the price of ***Bitcoin*** on any given day.

**Motivations**

* **Prices of bitcoin**
* **Number of transactions in blockchain per day.**
* **Average block size(Kb)**

### **Number of sent by addresses and number of active addresses**

### **Average mining difficulty per day**

### **Average hash rate per day**

### **Mining Profitability USD/Day for 1 Hash/s**

### **Sent coins in USD per day**

### **Average & Median transaction fee, USD**

### **Average block time (minutes)**

### **Avg. Transaction Value in USD**

### **Median Transaction Value in USD**

### **Number of coins in circulation**

### **Miner Revenue**

### **Tweets & Google Trends per day**

### **Top 100 Richest Addresses**

### **Fee Percentage in Total Block**

With each of the above fields in the note book => <https://colab.research.google.com/drive/10r9OvSWv380ZgICUJnxLil8vSkRug3xw#scrollTo=JdZ7iE8RzxQb> we have tried to address the following

* Co-relation of the particular feature with other features in dataset
* Date trends vs the particular feature.

Dynamic Plotly Graph:



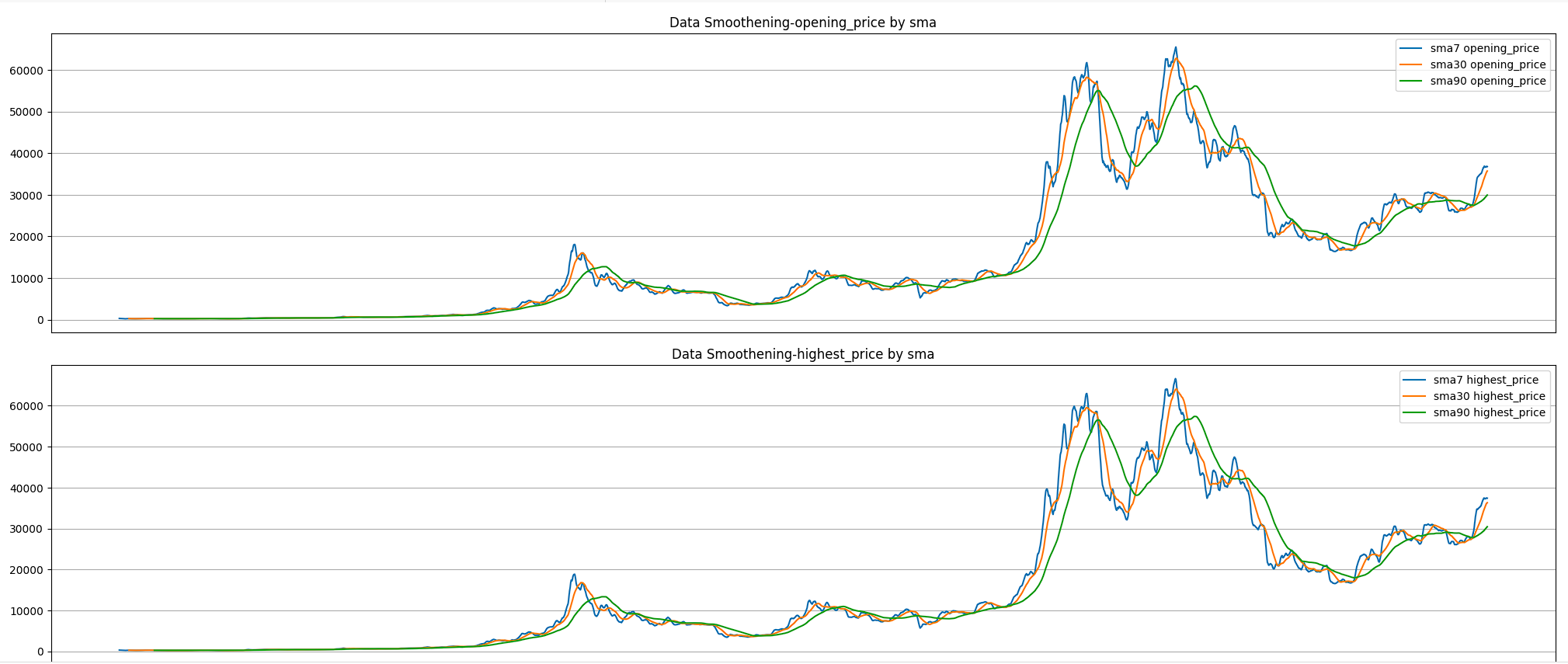
All the observations are very detailed mentioned under each section of above notebook.

# Features Engineering

Data smoothing is done by using an algorithm to remove noise from a data set. This allows important patterns to more clearly stand out. Feature engineering is the process of more meaningful fields using existing datasets which will help the machine learning model to predict close results. With the given time series data, the feature engineering section is divided into two parts. These features are specifically chosen to analyze the trends in market and may help in checking over bought or over sold stocks.

* Calculations of Moving Averages.
  1. Simple Moving Average.
  2. Weighted Moving Average.
  3. Exponential Moving Average.
* Statistical Analysis
  1. Variance
  2. Standard Deviation
  3. Relative strength index
  4. Receiver operating characteristic.
  5. Double Exponential Moving Average.
  6. Triple Exponential Moving Average
  7. Moving Average Convergence/Divergence indicator

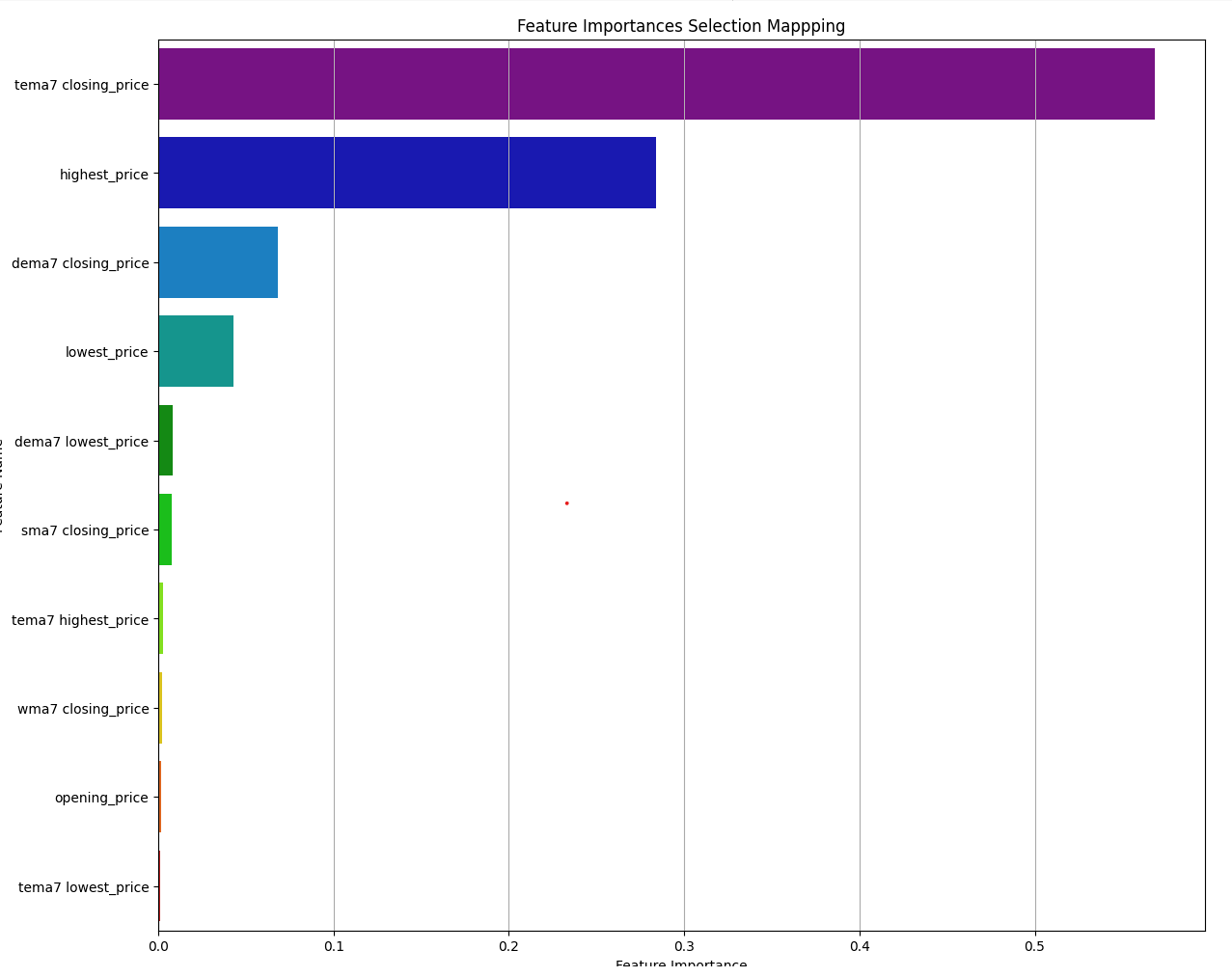
Sample SMA graph shown below

****

# Feature Selection Process

The most important process for any machine learning process is the feature selection to have best possible predictions, in this project we have utilize sklearn famous RandomForestRegressor to run on the engineered features and come up with top 10 most effective features.

rf = RandomForestRegressor(n\_estimators=100,n\_jobs=-1,bootstrap=True,verbose=5,random\_state=1). Since random forest perform best in predicting the price or trend because Random Forest Regression is preferred over linear regression when predicting numerical values because it offers greater ***accuracy*** and ***prediction stability.***

****

The top selected features are

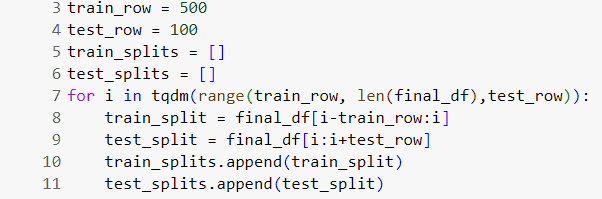
‘tema7 closing\_price','highest\_price','lowest\_price','dema7 closing\_price','dema7 lowest\_price','sma7 closing\_price','tema7 highest\_price','opening\_price’.

# Train and Test Split

In order to test the model accuracy, we need to split our datasets into train and test datasets, train dataset will be used to train the model and test dataset will be used to make predictions and validations.



We have data starting from 01/01/2015 to current date as show in above diagram. The train data is colored in red in color and test data is in green color. These datasets will be passed as mini batches to the regression models.



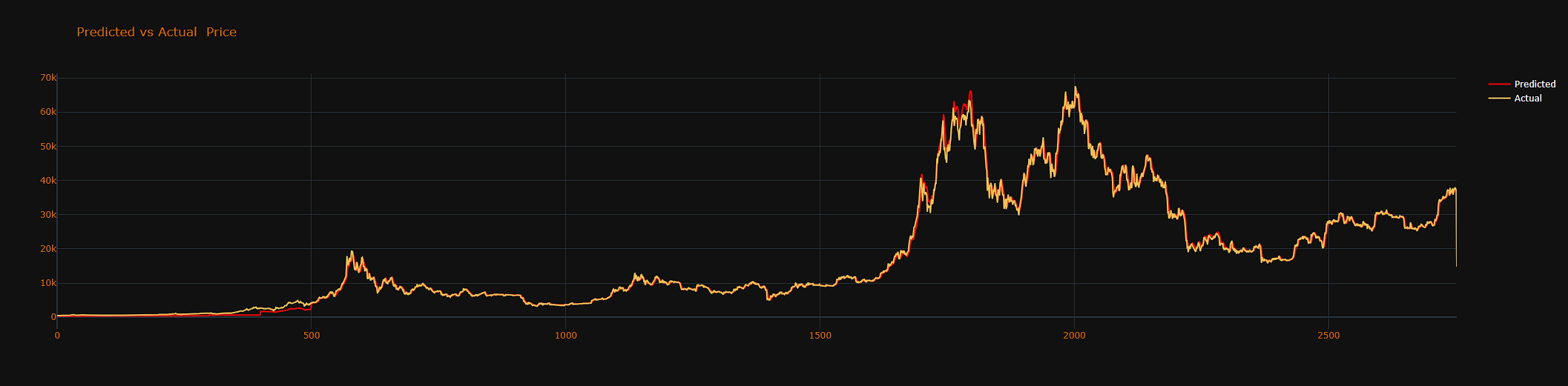
# Model Selection and Why?

There are various regression models available today to evaluate and make predictions, I have covered some relevant and basic model to understand and later these can be scaled and mixed up with model selection techniques like stacking, bagging etc.

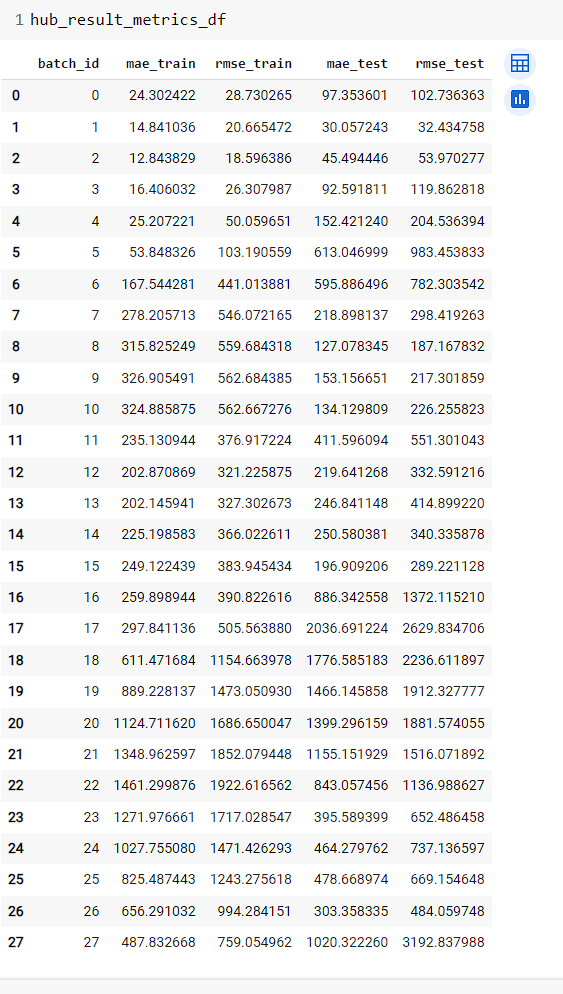
Below is the list of regression model covered in the note book:

* **Dummy Regressor** – Dummy Regressor is a regressor that makes predictions using simple rules. This regressor is useful as a simple baseline to compare with other (real) regressors.
* **Support Vector Regressor** -- SVR can handle non-linear relationships between the input variables and the target variable by using a kernel function to map the data to a higher-dimensional space. This makes it a powerful tool for regression tasks where there may be complex relationships between the input variables and the target variable.
* **Linear Regression** --Linear regression is commonly used in many fields, including economics, finance, and social sciences, to analyze and predict trends in data. It can also be extended to multiple linear regression, where there are multiple independent variables.
* **Xgboost Regressor** -- XGBoost is used for these two reasons: execution speed and model performance. Execution speed is crucial because it's essential to working with large datasets.
* **Huber Regressor** --L2-regularized linear regression model that is robust to outliers better for making price predictions.
* **Quantile Regressor** -- Quantile regression provides greater flexibility than other regression methods to identify differing relationships at different parts of the distribution of the dependent variable.

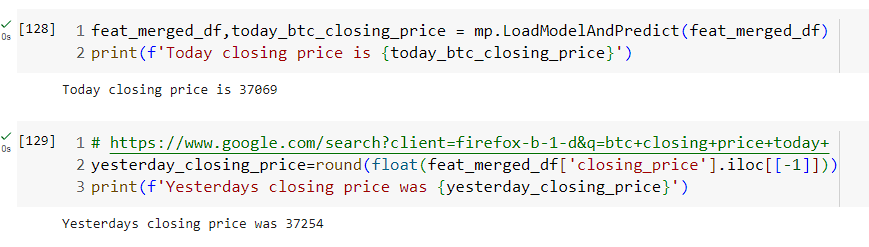
With the given dataset we could see the that Huber Regressor perform better and make better predictions. The loss function used for regression are Mean Absolute Error, Mean Squared Error, Root Mean Squared Error and Root Mean Absolute Error best for regression type mode. The predictions are made for the historical data plus next day predictions. As we can see the yellow and red lines are very closely overlapping, make the model good choice for making predictions.



Sample Loss Metrics for test and train data is given below



Comparing the price for today and yesterday the data is very close



The above screen shot depicts data from comparison for 11/27 and 11/26 data and number are very close.

# Deep Learning Model with Pytorch

A time series is a sequence of data points that are ordered in time and represent some phenomenon or process that changes over time. For example***, the stock prices***, the weather, the electricity demand, or the heart rate of a patient are all examples of time series. Time series analysis and forecasting aim to understand the patterns, trends, and seasonality of the data, and to predict the future values based on the past and present observations.

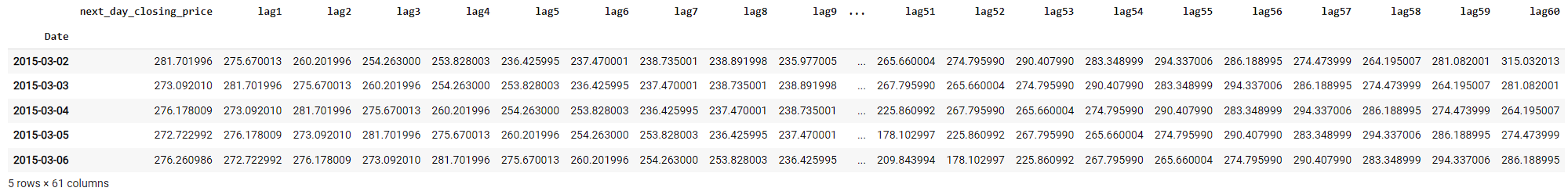
RNNs are well-suited for time series because they can exploit the sequential nature of the data and learn from the temporal dependencies. Unlike traditional feedforward neural networks, RNNs have a recurrent connection that allows them to store and update a hidden state that acts as a memory of the previous inputs. This way, RNNs can capture the long-term and short-term relationships among the data points and use them to make predictions. RNNs can also handle variable-length inputs and outputs, which is useful for time series that have different frequencies or horizons.

In this project we have used a variant of RNN as LSTM(Long Short-Term Memory) deep learning model. LSTM and GRU are two types of RNNs that have been designed to overcome some of the limitations of the basic RNNs, such as ***the vanishing or exploding gradient problem***. This problem occurs when the gradients of the error function become too small or too large during the backpropagation, making it difficult to train the network and update the weights. LSTM and GRU solve this problem by introducing gates that control the flow of information in and out of the hidden state, and allow the network to learn what to remember and what to forget. LSTM has three gates: input, output, and forget; while GRU has two gates: reset and update.

LSTM and GRU have several advantages over the basic RNNs for time series applications. First, they can capture long-term dependencies better than RNNs, which tend to forget the distant past inputs. This is important for time series that have long-term cycles or trends, such as climate data or economic indicators. Second, they can avoid overfitting by using dropout or regularization techniques, which reduce the complexity of the network and prevent overfitting to the noise or outliers in the data. Third, they can deal with missing values or irregular intervals in the data by using masking or interpolation methods, which prevent the network from learning false dependencies or losing information.

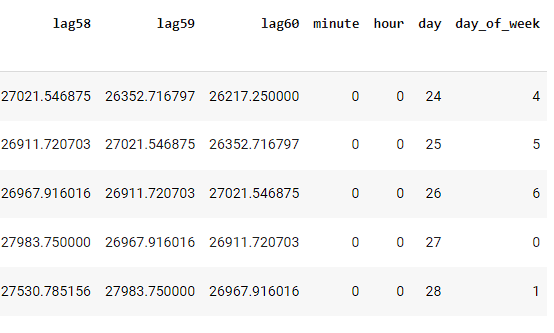
# Using lagged observations as features

Let’s start with using time steps as features. In other words, we’re trying to predict the next value, X(t+n), from the previous n observations Xt, X+1, …, and X(t+n-1). Then, what we need to do is simply create n columns with the preceding observations. Luckily, Pandas provides the method shift() to shift the values in a column. So, we can write a for loop to create such lagged observations by shifting the values in a column by n times and removing the first n columns. Lagging is simple yet a good starting point, especially if you don’t have many features to work with at the start.

******

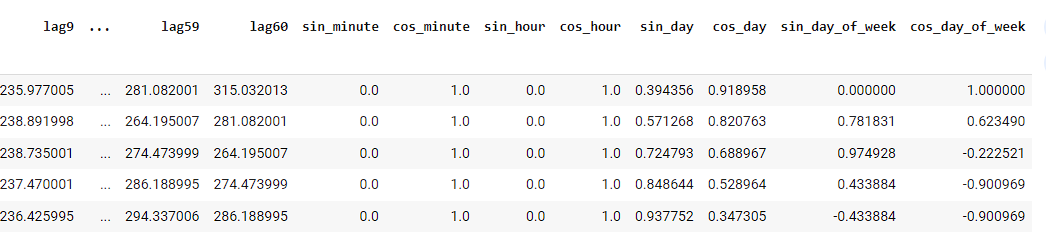
# Generating features from timestamps

Having a univariate time-series dataset, it seems logical to generate date and time features. As we have already converted its index into Pandas’ DatetimeIndex type, a series of Date Time objects, we can easily create new features from the index values, like the hour of the day, the day of the month, the month, the day of the week and the week of the year, as follows.

******

# Generating cyclical time features

The gist is to create two new cyclical features, calculating sine and cosine transform of the given DateTime feature, say the hour of the day. Instead of using the hour’s original value, the model then uses its sine transform, preserving its cyclicality. To see how and why it works, feel free to refer to Pierre-Louis’or David’s blog post on the matter, which explains the concept more in detail.

******

# Model Configuration

There are large number of parameters required to setup any sensitive deep learning model, for the time series prediction I have below configuration for LSTM model

class Configuration:

    BATCH\_SIZE\_TRAIN = 16 # Mini batch size for trainset

    BATCH\_SIZE\_VALIDATION = 8 # Mini batch size for validation set

    BATCH\_SIZE\_TEST = 8 # Mini batch size for test dataset

    DROPOUT = 0.2 #Drop out for fully connect layers

    EPOCHS = 50 = #Forward, backward and update weight cycles

    FOLDS = 10 # Iteration for each cycle

    HIDDEN\_DIM = 64 # Number of hidden layers

    LAYER\_DIM = 3

    LEARNING\_RATE = 1e-3 # Value for multiplier for error

    LR\_FACTOR = 0.4  # By how much learning rate decreasing

    LR\_PATIENCE = 1

    OUTPUT\_DIM = 1 # Output dimension

    WEIGHT\_DECAY = 1e-6 # Regularization

# Model Architecture

class LSTMModel(torch.nn.Module):

    def \_\_init\_\_(self, input\_dim, hidden\_dim, layer\_dim, output\_dim, dropout\_prob, device):

        super(LSTMModel, self).\_\_init\_\_()

        # Defining the number of layers and the nodes in each layer

        self.hidden\_dim = hidden\_dim

        self.layer\_dim = layer\_dim

        # LSTM layers

        self.lstm = torch.nn.LSTM(

            input\_dim, hidden\_dim, layer\_dim, batch\_first=True, dropout=dropout\_prob

        )

        # Fully connected layer

        self.fc = torch.nn.Linear(hidden\_dim, output\_dim)

        self.device = device

    def forward(self, x):

        # Initializing hidden state for first input with zeros

        h0 = torch.zeros(self.layer\_dim, x.size(0), self.hidden\_dim).requires\_grad\_().to(device)

        # Initializing cell state for first input with zeros

        c0 = torch.zeros(self.layer\_dim, x.size(0), self.hidden\_dim).requires\_grad\_().to(device)

        # We need to detach as we are doing truncated backpropagation through time (BPTT)

        # If we don’t, we’ll backprop all the way to the start even after going through another batch

        # Forward propagation by passing in the input, hidden state, and cell state into the model

        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

        # Reshaping the outputs in the shape of (batch\_size, seq\_length, hidden\_size)

        # so that it can fit into the fully connected layer

        out = out[:, -1, :]

        # Convert the final state to our desired output shape (batch\_size, output\_dim)

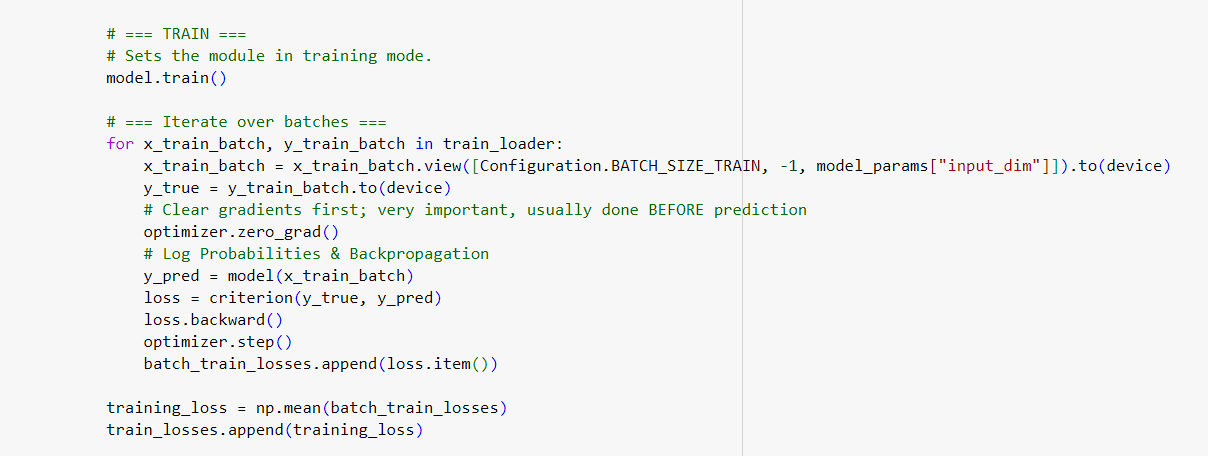
        out = self.fc(out)

        return out

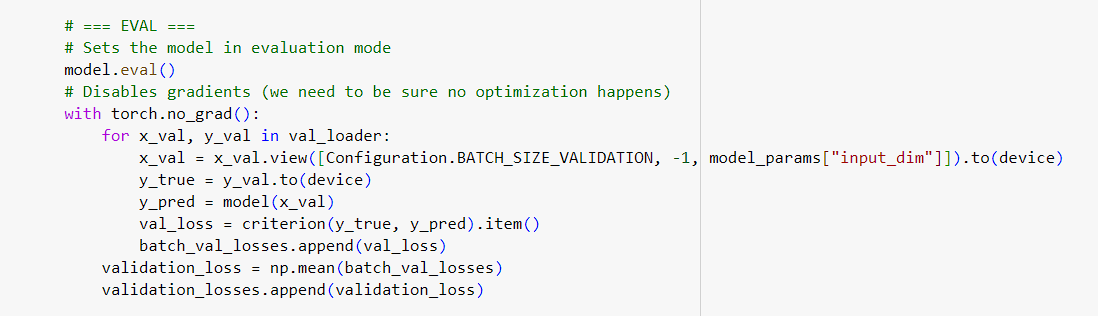
# Pytorch Training

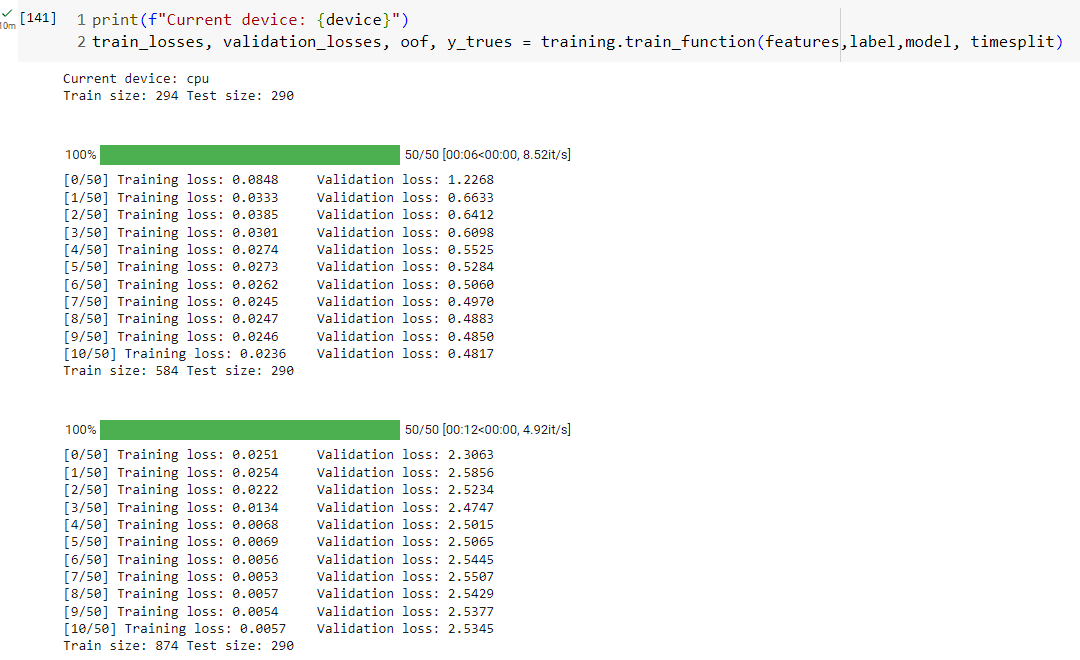
Since Pytorch module don’t have built in FIT method to train the neural network model and its more pythonic implementation we have to write our own training methods. These training methods can be written in multiple ways, I have tried here to write simple training method that have following –

* Model.train() mode



* Model.Eval() mode





# Loss Plot for Out of Fold Vs Actual Data

As we can see the model performance with current settings perform poorly for tabular data with Pytorch module which is also clearly reflected in Out of fold(predicted values) vs Ground Truth(Actual data plot). The loss parameter can be increased by optimizing hyper parameter settings or by using Auto ML module. There other hyper parameter optimization module like Ray Tune, Optuna to name a few can be used to have better predictions on tabular data and minimize the losses. Another module which can be used in place of vanilla Pytorch are as follow:

# Pytorch-widedeep -> <https://github.com/jrzaurin/pytorch-widedeep>

# Pytorch tabular -> <https://pytorch-tabular.readthedocs.io/en/latest/>

# Raytune -> <https://docs.ray.io/en/latest/tune/index.html>

# Optuna -> <https://optuna.org/>

# AutoMl -> https://www.automl.org/automl/

# 

# *Since this will make project more complex hence, I have used vanilla Pytorch module to make time series prediction.*

# 

# ****Analysis Between Machine Learning and Deep Neural Network modelling****

There is some limited and indirect support in the literature for this hypothesis:

According to [1], the manifold hypothesis [2] states that all natural data lies in a lower-dimensional space (a manifold) which is embedded in the higher dimensional feature space and locally behaves like an Euclidean space. And deep learning models can learn these manifolds which is the reason why they actually work so well. However, the author does not explicitly define what "natural data" is but he provides some examples like human faces, MNIST digits, natural language and human voices. In line with that, [3] provides images as an example for natural data and [4] states that it has been shown that some image and video data, in fact, form a manifold. ([4] also provides examples of other, non-neural, manifold learning algorithms.)

In summary, I infer that these authors do not refer to tabular data when they speak about natural data. And, hence, neural nets might not work that well because tabular data does not form a manifold.

At least this is the case for the way we usually encode tabular data, i.e. it might be possible to present tabular data in way that does embed a manifold. But that is just speculation. (There are a few examples which go into this direction already: Some applications of CNNs operate on data visualizations/plots and another example are transformers which are able to learn arithmetic operations from natural language to a limited extend.)

Moreover, [1] raises the point that neural nets work well because the inductive biases of their architecture mirror the data (e.g. CNNs have a very special structures which works particularly well on image data). Again I am speculating, but maybe we will develop architectures going forward which provide inductive biases suitable for tabular data - or some special types of tabular data. But currently we do not have these which is another reason why neural nets lack performance on tabular data.

References:

[1] Chollet, Francois; "Deep Learning with Python"; Second Edition 2nd Edition; 2021

[2] [https://en.wikipedia.org/wiki/Manifold\_hypothesis](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FManifold_hypothesis)

[3] [https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fcolah.github.io%2Fposts%2F2014-03-NN-Manifolds-Topology%2F)

[4] Cayton, Lawrence; "Algorithms for manifold learning"; 2005; [https://www.lcayton.com/resexam.pdf](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.lcayton.com%2Fresexam.pdf)

[5] [https://towardsdatascience.com/the-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Ftowardsdatascience.com%2Fthe-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33)

# 

# ****Reference and Motivation****

Web Scrapping

[https://en.bitcoinwiki.org/wiki/Main\_Page](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fen.bitcoinwiki.org%2Fwiki%2FMain_Page)

[https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7256561/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.ncbi.nlm.nih.gov%2Fpmc%2Farticles%2FPMC7256561%2F)

[https://www.mdpi.com/2571-9394/5/1/10](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.mdpi.com%2F2571-9394%2F5%2F1%2F10)

[https://www.coindesk.com/markets/2016/12/31/not-just-bitcoin-the-top-7-cryptocurrencies-all-gained-in-2016/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.coindesk.com%2Fmarkets%2F2016%2F12%2F31%2Fnot-just-bitcoin-the-top-7-cryptocurrencies-all-gained-in-2016%2F)

[https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fpandas.pydata.org%2Fdocs%2Freference%2Fapi%2Fpandas.DataFrame.plot.html)

[https://medium.com/@HalderNilimesh/applied-data-science-for-beginners-how-to-calculate-moving-average-using-pandas-dataframe-in-7e5f7944c07a](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fmedium.com%2F%40HalderNilimesh%2Fapplied-data-science-for-beginners-how-to-calculate-moving-average-using-pandas-dataframe-in-7e5f7944c07a)

[https://stackoverflow.com/a/43807666](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fstackoverflow.com%2Fa%2F43807666)

Machine Learning

[https://www.geeksforgeeks.org/dummy-regressor/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.geeksforgeeks.org%2Fdummy-regressor%2F)

[https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn-ensemble-randomforestregressor](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fscikit-learn.org%2Fstable%2Fmodules%2Fgenerated%2Fsklearn.ensemble.RandomForestRegressor.html%23sklearn-ensemble-randomforestregressor)

[https://coinmarketcap.com/api/documentation/v1/#section/Standards-and-Conventions](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fcoinmarketcap.com%2Fapi%2Fdocumentation%2Fv1%2F%23section%2FStandards-and-Conventions)

[https://finance.yahoo.com/quote/BTC-USD/history?period1=1420070400&period2=1697846400&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Ffinance.yahoo.com%2Fquote%2FBTC-USD%2Fhistory%3Fperiod1%3D1420070400%26period2%3D1697846400%26interval%3D1d%26filter%3Dhistory%26frequency%3D1d%26includeAdjustedClose%3Dtrue)

[https://www.kaggle.com/code/medyasun/crypto-stock-prediction-with-python-doge-coin](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fcode%2Fmedyasun%2Fcrypto-stock-prediction-with-python-doge-coin)

[https://plotly.com/python/horizontal-vertical-shapes/](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fplotly.com%2Fpython%2Fhorizontal-vertical-shapes%2F)

Pytorch Deep Learning

[https://www.javatpoint.com/pytorch-deep-learning](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.javatpoint.com%2Fpytorch-deep-learning)

[https://towardsdatascience.com/the-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Ftowardsdatascience.com%2Fthe-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33)

[https://towardsdatascience.com/exploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Ftowardsdatascience.com%2Fexploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf)