After the detailed analysis, applying different algorithms to improve the performance of the final metrics.

```
# install necessaries libraries
! pip install ta
! pip install seglearn
```

# Load and transform data

```
# Connecting to the google drive
from google.colab import drive
drive.mount('/content/drive')
from IPython.display import clear_output
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
# import libraries
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.model_selection import RandomizedSearchCV

#picking models for prediction.
from sklearn.svm import SVC
```

```
# file path
folder_path = '/content/drive/MyDrive/MADS_23_DL_final_project'
daily = pd.read_csv(folder_path + '/data/crypto_data_daily_cleaned_v1.csv')
```

```
# view the first rows of dataset
daily.head()
```

| | Open Time | Open | High | Low | Close | Volume | train_test | Crypto |
|---|---|---|---|---|---|---|---|---|

## Create Train / Test Dataset

```
# check the percentage of train / test set
daily['train_test'].value_counts(normalize=True)
```

```
Train    0.82546
Test     0.17454
Name: train_test, dtype: float64
```

```
# create train & test datasets
df = daily.copy()

train_df = df[df['train_test']=='Train']
test_df = df[df['train_test']=='Test']
```

```
train_df.columns
```

```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto'],
      dtype='object')
```

```
train_df.head()
```

| | Open Time | Open | High | Low | Close | Volume | train_test | Crypto |
|---|---|---|---|---|---|---|---|---|
| **0** | 2013-04-01 | 93.155 | 105.90 | 93.155 | 104.750 | 11008.524 | Train | BTC |
| **1** | 2013-04-02 | 104.720 | 127.00 | 99.000 | 123.016 | 24187.398 | Train | BTC |
| **2** | 2013-04-03 | 123.001 | 146.88 | 101.511 | 125.500 | 31681.780 | Train | BTC |
| **3** | 2013-04-04 | 125.500 | 143.00 | 125.500 | 135.632 | 15035.206 | Train | BTC |
| **4** | 2013-04-05 | 136.000 | 145.00 | 135.119 | 142.990 | 11697.741 | Train | BTC |

## Data Normalization

```
from sklearn.preprocessing import MinMaxScaler
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

def minmax_scale(df_x, normalizers=None):


    #Try to pass the column values as a parameter from outside
    features_to_minmax = ['Open','High', 'Low','Close','Volume']
```

```
    if not normalizers:
        normalizers = {}

    for feat in features_to_minmax:
        if feat not in normalizers:
            normalizers[feat] = MinMaxScaler()
            normalizers[feat].fit(df_x[feat].values.reshape(-1, 1))

        df_x[feat] = normalizers[feat].transform(df_x[feat].values.reshape(-1, 1))

    # series_y=normalizers["pct_change_2hour"].transform(series_y.values.reshape(-1, 1))

    return df_x , normalizers
```

```
# since the model performs better without transformation, we do not apply this step
# train_df, train_normalizers = minmax_scale(train_df)
# test_df, test_normalizers = minmax_scale(test_df)
```

```
train_df.head()
```

| | Open Time | Open | High | Low | Close | Volume | train_test | Crypto |
|---|---|---|---|---|---|---|---|---|
| **0** | 2013-04-01 | 93.155 | 105.90 | 93.155 | 104.750 | 11008.524 | Train | BTC |
| **1** | 2013-04-02 | 104.720 | 127.00 | 99.000 | 123.016 | 24187.398 | Train | BTC |
| **2** | 2013-04-03 | 123.001 | 146.88 | 101.511 | 125.500 | 31681.780 | Train | BTC |
| **3** | 2013-04-04 | 125.500 | 143.00 | 125.500 | 135.632 | 15035.206 | Train | BTC |
| **4** | 2013-04-05 | 136.000 | 145.00 | 135.119 | 142.990 | 11697.741 | Train | BTC |

## Calculate percentage change

```
# create function to calculate daily percentage change
def calculate_pct_change(df):
  coins = df.Crypto.unique()
  df_pct_change = pd.DataFrame()
  for coin in coins:
    x = df[df['Crypto']==coin]
    x['pct_change_1day'] = x['Close'].pct_change(1)
    df_pct_change = pd.concat([df_pct_change,x])
  return df_pct_change
```

```
train_df = calculate_pct_change(train_df)
test_df = calculate_pct_change(test_df)
```

```
test_df.head()
```

| | Open Time | Open | High | Low | Close | Volume | train_test | Crypto | pct |
|---|---|---|---|---|---|---|---|---|---|
| **3098** | 2021-10-01 | 43828.89 | 48500.00 | 43287.44 | 48165.76 | 38375.517 | Test | BTC | |
| **3099** | 2021-10-02 | 48185.61 | 48361.83 | 47438.00 | 47657.69 | 12310.011 | Test | BTC | |
| | 2021- | | | | | | | | |

# Generate new features

## Lag features, moving average, exponential moving average and market cap

```
def create_market_volumn_features(df):

  # calculate value of each cryto at certain time points
  df['Total_Value'] = df['Close']*df['Volume']
  # the sum of values at each time point
  sum_at_timepoints = df.groupby('Open Time').sum()['Total_Value']
  merged = df.merge(sum_at_timepoints, how='left',
                            on='Open Time', suffixes=('','_market'))
  merged['Value_Weight'] = merged['Total_Value']/merged['Total_Value_market']

  return merged


# function to create shift features
def create_shift_features(df, col = 'pct_change_1day'):
    df['1_d_lag'] = df[col].shift(periods=1)
    return df


#list to collect all relevant lags
from ta import add_all_ta_features
def create_analysis_colums(df):

  master_df = pd.DataFrame()
  crypto_coins = df['Crypto'].unique()

  for coin in crypto_coins:

    temp_df = df[df['Crypto']==coin]
    temp_df['pct_change_1day'] = temp_df['Close'].pct_change()

    # temp_df = create_shift_features(temp_df.copy(),col = 'pct_change_1day',lags=5, freq=
    # temp_df = create_shift_features(temp_df.copy(),col = 'pct_change_1day',lags=2, freq=
    # temp_df = create_shift_features(temp_df.copy(),col = 'pct_change_1day',lags=4, freq=
    temp_df = create_shift_features(temp_df.copy(),col = 'pct_change_1day')
    temp_df = add_all_ta_features(temp_df.copy(), open="Open", high="High", low="Low", clo
    if master_df.empty :
```

```
      master_df = temp_df
    else:
      master_df = pd.concat([master_df, temp_df])
  return master_df

# create analysis columns for train and test dataset
train_df = create_analysis_colums(train_df)
test_df = create_analysis_colums(test_df)


# check the columns of train dataset
train_df.columns
```

```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto', 'pct_change_1day', '1_d_lag', 'volume_adi', 'volume_obv',
       'volume_cmf', 'volume_fi', 'volume_em', 'volume_sma_em', 'volume_vpt',
       'volume_vwap', 'volume_mfi', 'volume_nvi', 'volatility_bbm',
       'volatility_bbh', 'volatility_bbl', 'volatility_bbw', 'volatility_bbp',
       'volatility_bbhi', 'volatility_bbli', 'volatility_kcc',
       'volatility_kch', 'volatility_kcl', 'volatility_kcw', 'volatility_kcp',
       'volatility_kchi', 'volatility_kcli', 'volatility_dcl',
       'volatility_dch', 'volatility_dcm', 'volatility_dcw', 'volatility_dcp',
       'volatility_atr', 'volatility_ui', 'trend_macd', 'trend_macd_signal',
       'trend_macd_diff', 'trend_sma_fast', 'trend_sma_slow', 'trend_ema_fast',
       'trend_ema_slow', 'trend_vortex_ind_pos', 'trend_vortex_ind_neg',
       'trend_vortex_ind_diff', 'trend_trix', 'trend_mass_index', 'trend_dpo',
       'trend_kst', 'trend_kst_sig', 'trend_kst_diff', 'trend_ichimoku_conv',
       'trend_ichimoku_base', 'trend_ichimoku_a', 'trend_ichimoku_b',
       'trend_stc', 'trend_adx', 'trend_adx_pos', 'trend_adx_neg', 'trend_cci',
       'trend_visual_ichimoku_a', 'trend_visual_ichimoku_b', 'trend_aroon_up',
       'trend_aroon_down', 'trend_aroon_ind', 'trend_psar_up',
       'trend_psar_down', 'trend_psar_up_indicator',
       'trend_psar_down_indicator', 'momentum_rsi', 'momentum_stoch_rsi',
       'momentum_stoch_rsi_k', 'momentum_stoch_rsi_d', 'momentum_tsi',
       'momentum_uo', 'momentum_stoch', 'momentum_stoch_signal', 'momentum_wr',
       'momentum_ao', 'momentum_roc', 'momentum_ppo', 'momentum_ppo_signal',
       'momentum_ppo_hist', 'momentum_pvo', 'momentum_pvo_signal',
       'momentum_pvo_hist', 'momentum_kama', 'others_dr', 'others_dlr',
       'others_cr'],
      dtype='object')
```

```
# create market volume features
train_df = create_market_volumn_features(train_df.copy())
test_df = create_market_volumn_features(test_df.copy())


# check the columns of train_df
train_df.columns[:50]
```

```
Index(['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto', 'pct_change_1day', '1_d_lag', 'volume_adi', 'volume_obv',
       'volume_cmf', 'volume_fi', 'volume_em', 'volume_sma_em', 'volume_vpt',
       'volume_vwap', 'volume_mfi', 'volume_nvi', 'volatility_bbm',
       'volatility_bbh', 'volatility_bbl', 'volatility_bbw', 'volatility_bbp',
       'volatility_bbhi', 'volatility_bbli', 'volatility_kcc',
       'volatility_kch', 'volatility_kcl', 'volatility_kcw', 'volatility_kcp',
       'volatility_kchi', 'volatility_kcli', 'volatility_dcl',
       'volatility_dch', 'volatility_dcm', 'volatility_dcw', 'volatility_dcp',
```

```
            'volatility_atr', 'volatility_ui', 'trend_macd', 'trend_macd_signal',
            'trend_macd_diff', 'trend_sma_fast', 'trend_sma_slow', 'trend_ema_fast',
            'trend_ema_slow', 'trend_vortex_ind_pos', 'trend_vortex_ind_neg'],
           dtype='object')
```

```
# choose features to keep for model training
feat_to_keep = ['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'train_test',
       'Crypto', 'pct_change_1day'] #,'volatility_kcw','trend_cci','volume_adi','momentum_
            #  'volatility_dcw','volume_vpt','volatility_bbw','Total_Value', 'Total_Value_
lag_cols = [col for col in train_df.columns if 'lag' in col]

feat_to_keep.extend(lag_cols)
```

```
# check number of features to keep
len(feat_to_keep)
```

```
10
```

```
# filter train and set with only keep feature
train_df = train_df[feat_to_keep]
test_df = test_df[feat_to_keep]
```

```
len(train_df.columns)
```

```
10
```

```
# create function to calculate shift volume
def shift_vol(df):
  impo_feat = ['volatility_kcw','trend_cci','volume_adi','momentum_ppo_hist','momentum_sto
               'volatility_dcw','volume_vpt','volatility_bbw']
  master_df = pd.DataFrame()
  crypto_coins = df['Crypto'].unique()


  for coin in crypto_coins:
    temp_df = df[df['Crypto']==coin]
    for feat in impo_feat:

      temp_df[feat] = temp_df[feat].shift(1)
    if master_df.empty :
      master_df = temp_df
    else:
      master_df = pd.concat([master_df, temp_df])
  return master_df
```

# Extract year, month, day, hour and weekday from time stamp

## Encoding of ordinals

```
# feature to encode ordinals
def encode_cyclicals(df_x):
    #"month","day","hour", "minute", "dayofweek"

    df_x['month_sin'] = np.sin(2*np.pi*df_x.month/12)
    df_x['month_cos'] = np.cos(2*np.pi*df_x.month/12)
    df_x.drop('month', axis=1, inplace=True)

    df_x['day_sin'] = np.sin(2*np.pi*df_x.day/31)
    df_x['day_cos'] = np.cos(2*np.pi*df_x.day/31)
    df_x.drop('day', axis=1, inplace=True)

    df_x['dayofweek_sin'] = np.sin(2*np.pi*df_x.weekday/7)
    df_x['dayofweek_cos'] = np.cos(2*np.pi*df_x.weekday/7)
    df_x.drop('weekday', axis=1, inplace=True)

    df_x['hour_sin'] = np.sin(2*np.pi*df_x.hour/24)
    df_x['hour_cos'] = np.cos(2*np.pi*df_x.hour/24)
    df_x.drop('hour', axis=1, inplace=True)

    df_x['hour_sin'] = np.sin(2*np.pi*df_x.minute/60)
    df_x['hour_cos'] = np.cos(2*np.pi*df_x.minute/60)
    df_x.drop('minute', axis=1, inplace=True)

    return df_x


# feature to measure date values extraction
def date_values_extraction(new_df):

  df = new_df.copy()
  df['year'] = pd.DatetimeIndex(df['Open Time']).year
  df['month'] = pd.DatetimeIndex(df['Open Time']).month
  df['day'] = pd.DatetimeIndex(df['Open Time']).day
  df['weekday'] = pd.DatetimeIndex(df['Open Time']).dayofweek


  df['Open Time'] = pd.to_datetime(df['Open Time'])
  df['minute'] = df['Open Time'].dt.minute
  df['hour'] = df['Open Time'].dt.hour
  df = encode_cyclicals(df.copy())
  return df


# apply features on train and test dataset
train_df = date_values_extraction(train_df)
test_df = date_values_extraction(test_df)
```

## One hot coding the coins

```
# choose time window and forecast distance to the future
TIME_WINDOW = 5
FORECAST_DISTANCE = 1
```

```
# function to one hot encoding on Crypto Coin
def crypto_one_hot_encoding(df):
  y_dummies = pd.get_dummies(df['Crypto'], prefix='Crypto', drop_first= False)
  # df = pd.concat([df, y_dummies], axis=1)
  # df.drop(['Crypto'], axis=1, inplace=True)
  # creating a additional column if the model is used for new coin.
  y_dummies['other_crypto'] =0
  return y_dummies
```

```
# crete rolling data
def rolling_hot_data(train_df):
  train_hot = pd.DataFrame()
  for col in train_df.columns:
    if train_hot.empty:
        train_hot = train_df[train_df[col]==1][:-TIME_WINDOW]
    else:
      train_hot = pd.concat([train_hot, train_df[train_df[col]==1][:-TIME_WINDOW]], axis=0
  return train_hot
```

```
crypto_one_hot_encoding(test_df).shape
```

```
(3621, 11)
```

```
train_onehot_data = crypto_one_hot_encoding(train_df)
test_onehot_data = crypto_one_hot_encoding(test_df)
```

```
train_onehot_data.describe()
```

| | Crypto_ADA | Crypto_BTC | Crypto_ETC | Crypto_ETH | Crypto_LINK | Crypto_ |
|---|---|---|---|---|---|---|
| **count** | 17125.000000 | 17125.000000 | 17125.000000 | 17125.000000 | 17125.000000 | 17125.000 |
| **mean** | 0.076496 | 0.180905 | 0.101255 | 0.118131 | 0.048058 | 0.101 |
| **std** | 0.265798 | 0.384951 | 0.301676 | 0.322773 | 0.213896 | 0.301 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
| **75%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000 |

```
train_onehot_data = rolling_hot_data(train_onehot_data)
```

```
test_onehot_data = rolling_hot_data(test_onehot_data)
```

```
train_onehot_data.shape, test_onehot_data.shape
```

```
((17075, 11), (3571, 11))
```

```
train_df['pct_change_1day'].describe()
```

```
count    17115.000000
mean         0.005533
std          0.162083
min         -0.564847
25%         -0.025641
50%          0.000000
75%          0.028824
max         19.058824
Name: pct_change_1day, dtype: float64
```

```
test_df['pct_change_1day'].describe()
```

```
count    3611.000000
mean       -0.001452
std         0.046840
min        -0.204696
25%        -0.026274
50%         0.000000
75%         0.024091
max         0.344702
Name: pct_change_1day, dtype: float64
```

```
train_df = train_df.replace(np.NaN,0)
test_df = test_df.replace(np.NaN,0)
```

# Defining the Target Variable.

We want to follow the classification approach and hence based on the "pct_change_2hour" we are creating 2 classes one class '0' when the returns are negative and '1' When the retruns are postive.

```
# create taraget variable for model
def create_target(df, prob='Classification'):
  if prob == 'Classification':
    market_RoR = 26.89
    market_RoR_1d = market_RoR/365
    df['Target'] = np.where(df['pct_change_1day']>0, 1,0)
    df['Target'] = np.where(df['pct_change_1day']>market_RoR_1d, 2,1)
    df['Target'][df['Target']==1] = np.where(df['pct_change_1day'][df['Target']==1]>=0, 1,
  elif prob == 'Regression':
    df['Target'] = df['pct_change_1day']
  return df
```

```
# create target variable for regression model
train_df = create_target(train_df, 'Regression')
test_df = create_target(test_df, 'Regression')
```

```
# check target values of train_df
train_df['Target'].value_counts(normalize=True)
```

```
 0.000000    0.059445
 0.071429    0.001635
 0.045455    0.001460
 0.043478    0.001343
 0.066667    0.001343
               ...
 0.002939    0.000058
-0.003848    0.000058
 0.030492    0.000058
 0.015801    0.000058
 0.033752    0.000058
Name: Target, Length: 13919, dtype: float64
```

```
# check target values of test_df
test_df['Target'].value_counts(normalize=True)
```

```
 0.000000    0.032035
 0.015873    0.002209
 0.015385    0.001657
 0.016949    0.001657
-0.016667    0.001381
               ...
-0.070367    0.000276
 0.001066    0.000276
-0.048456    0.000276
 0.043649    0.000276
-0.002159    0.000276
Name: Target, Length: 3319, dtype: float64
```

```
# check pct_change_1day value
test_df.drop(['pct_change_1day'], axis=1, inplace=True) # droppping the column as we alrea
train_df.drop(['pct_change_1day'], axis=1, inplace=True) # droppping the column as we alre
```

```
# create target variable
target = train_df['Target']
test_target = test_df['Target']
```

# Drop columns

```
# drop unesccesary columns
train_df.drop(['Target','Open Time','train_test',],axis=1,inplace=True)
test_df.drop(['Target','Open Time','train_test',],axis=1,inplace=True)
```

```
# dropping the listof the columns
drop_columns = ['Open','Close']

if drop_columns:
  norm_train_df = train_df.drop(drop_columns,axis=1)
  norm_test_df = test_df.drop(drop_columns,axis=1)
else:
  norm_train_df = train_df
  norm_test_df = test_df
```

```
norm_train_df.columns
```

```
Index(['High', 'Low', 'Volume', 'Crypto', '1_d_lag', 'year', 'month_sin',
       'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin', 'dayofweek_cos',
       'hour_sin', 'hour_cos'],
      dtype='object')
```

```
norm_train_df.shape
```

```
(17125, 14)
```

```
target.shape
```

```
(17125,)
```

# Creating target (y) and "windows" (X) for modeling

By default we use the next 24 hour value of "pm2.5" for prediction, that is, I would like to predict what the pm2.5 will be like **at this hour 24 hours from now.**

We use the quite handy **seglearn** package for this.

Because of computational reasons, we **use the window of 100 hours** to predict. Classical models would have hard time to accommodate substantially (like 5-10x) context windows, LSTM-s would suffer from the challenge of long term memory. After a basic run of modeling the next big challenge would be to investigate PACF structure more and use eg. stateful LSTM modeling to try to accommodate the large "lookback".

```
# function to create rolling dataset
from seglearn.transform import FeatureRep, SegmentXYForecast, last

def Segment_multi(train_df, target):
  master_df_x = pd.DataFrame()
  master_df_y = pd.DataFrame()

  crypto_coins = df['Crypto'].unique()

  segmenter = SegmentXYForecast(width=TIME_WINDOW, step=1, y_func=last, forecast=FORECAST_
```

```
  for coin in crypto_coins:

    coin_index = train_df[train_df['Crypto']==coin].index
    X_train_rolled, y_train_rolled,_=segmenter.fit_transform([train_df.iloc[coin_index].dr

    if coin == 'BTC' :
      master_df_x = X_train_rolled
      master_df_y = y_train_rolled
    else:
      master_df_x = np.concatenate([master_df_x, X_train_rolled], axis=0)

      master_df_y = np.concatenate([master_df_y, y_train_rolled], axis=0)

  return master_df_x, master_df_y
```

```
norm_train_df.shape, norm_test_df.shape
```

```
((17125, 14), (3621, 14))
```

```
# create train and test rolling dataset
train_roll, y_roll =  Segment_multi(norm_train_df.fillna(0), target)
test_roll, y_test_roll =  Segment_multi(norm_test_df.fillna(0), test_target)
```

```
train_roll.shape
```

```
(17075, 5, 13)
```

```
norm_train_df.columns
```

```
Index(['High', 'Low', 'Volume', 'Crypto', '1_d_lag', 'year', 'month_sin',
       'month_cos', 'day_sin', 'day_cos', 'dayofweek_sin', 'dayofweek_cos',
       'hour_sin', 'hour_cos'],
      dtype='object')
```

```
train_roll.shape, y_roll.shape
```

```
((17075, 5, 13), (17075,))
```

# Evaluation function

```
# !pip install tensorflow-addons
# !pip install keras-tuner --upgrade
```

```
from kerastuner.tuners import RandomSearch
import keras_tuner

from tensorflow import keras
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional, BatchNormalizatio
```

```
from tensorflow.keras.models import Model
from tensorflow.keras import backend as be
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
train_roll.shape
```

```
(17075, 5, 13)
```

```
shape = (train_roll.shape[1],train_roll.shape[2] )
```

```
shape
```

```
(5, 13)
```

```
train_roll.shape ,train_onehot_data.shape, y_roll.shape
```

```
((17075, 5, 13), (17075, 11), (17075,))
```

```
from tensorflow.keras import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, LSTM, Input
from tensorflow.keras import backend as be
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStoppi
```

```
from keras.callbacks import Callback

max_len = 20

class ResetStatesCallback(Callback):
    def __init__(self):
        self.counter = 0

    def on_batch_begin(self, batch, logs={}):
        if self.counter % max_len == 0:
            self.model.reset_states()
        self.counter += 1
```

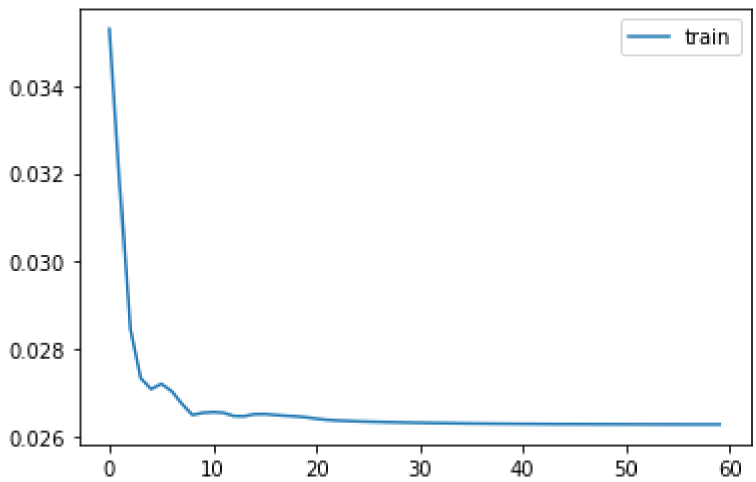
```
# list of learning rate scheduler, early stopping and checkpoints for callbacks
import keras_tuner as kt

def scheduler(epoch, lr):
  if epoch < 10:
    return lr
  else:
    return lr * np.exp(-0.1)

stop_early = EarlyStopping(monitor='loss', patience=50)

model_saver = ModelCheckpoint(
    filepath='/content/drive/MyDrive/database',
```

```
    save_weights_only=True,
    monitor='val_loss',
    mode='max',
    save_best_only=True)

callback = LearningRateScheduler(scheduler)

my_callbacks = [model_saver, callback] #ResetStatesCallback()]


# create LSTM model with 2 LSTM layers, dropout and last dense linear layer
float_input = Input(shape=(train_roll.shape[1],train_roll.shape[2] ))
one_hot_input = Input(shape=(11,) )

first_lstm = LSTM(256,return_sequences=True,stateful=False)(float_input)
dropout_layer = Dropout(rate=0.005)(first_lstm)
second_lstm = LSTM(128)(dropout_layer)
third_dense = Dense(64)(one_hot_input)
merge_one = concatenate([second_lstm, third_dense])
dense_inner = Dense(10)(merge_one)
dense_output = Dense(1, activation='linear')(dense_inner )

model = Model(inputs=[float_input, one_hot_input], outputs=dense_output)
model.compile(loss='mean_squared_error',
              optimizer='Adam')
model.summary()

history = model.fit([train_roll,train_onehot_data], y_roll, epochs=60, batch_size=64, verb
```

```
267/267 [==============================] - 2s 6ms/step - loss: 0.0264 - lr: 3.0119
Epoch 23/60
263/267 [============================>.] - ETA: 0s - loss: 0.0265WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0264 - lr: 2.7253
Epoch 24/60
267/267 [==============================] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0264 - lr: 2.4660
Epoch 25/60
265/267 [============================>.] - ETA: 0s - loss: 0.0265WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 2.2313
Epoch 26/60
259/267 [============================>.] - ETA: 0s - loss: 0.0268WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 2.0190
Epoch 27/60
263/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.8268
Epoch 28/60
264/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.6530
Epoch 29/60
262/267 [============================>.] - ETA: 0s - loss: 0.0265WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.4957
Epoch 30/60
260/267 [============================>.] - ETA: 0s - loss: 0.0267WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.3534
Epoch 31/60
267/267 [==============================] - ETA: 0s - loss: 0.0263WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.2246
```

```
Epoch 32/60
263/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 1.1080
Epoch 33/60
266/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 7ms/step - loss: 0.0263 - lr: 1.0026
Epoch 34/60
261/267 [============================>.] - ETA: 0s - loss: 0.0266WARNING:tensorflo
267/267 [==============================] - 2s 8ms/step - loss: 0.0263 - lr: 9.0718
Epoch 35/60
266/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 7ms/step - loss: 0.0263 - lr: 8.2085
Epoch 36/60
266/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 7.4274
Epoch 37/60
260/267 [============================>.] - ETA: 0s - loss: 0.0267WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 6.7206
Epoch 38/60
266/267 [============================>.] - ETA: 0s - loss: 0.0264WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 6.0810
Epoch 39/60
262/267 [============================>.] - ETA: 0s - loss: 0.0265WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 5.5023
Epoch 40/60
262/267 [============================>.] - ETA: 0s - loss: 0.0265WARNING:tensorflo
267/267 [==============================] - 2s 6ms/step - loss: 0.0263 - lr: 4.9787
Epoch 41/60
264/267 [============================>.] - ETA: 0s - loss: 0.0263WARNING:tensorflo
```

```
plt.plot(history.history['loss'], label='train')
plt.legend()
plt.show()
```




```
from sklearn.metrics import mean_squared_error
from math import sqrt

predictions = model.predict([test_roll,test_onehot_data],batch_size=5)

RMSE = sqrt(mean_squared_error(y_test_roll, predictions))
```

```
print(f'{RMSE:.5f}')
```

```
715/715 [==============================] - 4s 5ms/step
0.47520
```

## End of notebook

```
from pip._internal.utils.misc import get_installed_distributions
import sys
#import numpy as np # imported to test whether numpy shows up, which it does!

def get_imported_packages():
    p = get_installed_distributions()
    p = {package.key:package.version for package in p}

    imported_modules = set(sys.modules.keys())

    imported_modules.remove('pip')

    modules = [(m, p[m]) for m in imported_modules if p.get(m, False)]

    return modules


def generate_requirements(filepath:str, modules):
    with open(filepath, 'w') as f:
        for module, version in modules:
            f.write(f"{module}=={version}")


generate_requirements('requirements.txt', get_imported_packages())
```