In [1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn import preprocessing
from sklearn import ensemble
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import plotly.offline  as pyo
import datetime as dt
import ipywidgets as widgest
from IPython.display import display
from ipywidgets import interact, interact_manual
import pandas.plotting as pp
from pandas.plotting import autocorrelation_plot
#import lazypredict
from sklearn.utils import deprecated
import datetime


# from sklearn.utils._testing import ignore_warnings

from sklearn.preprocessing import StandardScaler,MinMaxScaler,LabelEncoder, Normalizer
from sklearn.model_selection import train_test_split
```

In [19]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [20]:
```
import statsmodels.api as sm
from pylab import rcParams
import scipy.stats as stats
from scipy.stats import lognorm
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.filters.hp_filter import hpfilter
from statsmodels.tsa.arima_model import ARIMA

#from arch import arch_model
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [21]:
```
from sklearn.metrics import (
    mean_absolute_error as mae,
    r2_score as r2,
    mean_absolute_percentage_error as mape)
```

In [22]:
```
import random
from collections import deque
from sklearn import preprocessing
```

In [23]:
```
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
```

In [24]:
```
start = dt.datetime(2020,1,1)
end =dt.datetime.now()
interval = '1d'
```

```
coins = ['ADA-USD',
 'ALGO-USD',
 'ANKR-USD',
 'ATOM-USD',
 'BAT-USD',
 'BCH-USD',
 'BNB-USD',
 'CHZ-USD',
 'CRO-USD',
 'DASH-USD',
 'DCR-USD',
 'DOGE-USD',
 'ENJ-USD',
 'EOS-USD',
 'ETC-USD',
 'FIL-USD',
 'FTM-USD',
 'FTT-USD',
 'HBAR-USD',
 'KAVA-USD',
 'LINK-USD',
 'LRC-USD',
 'LTC-USD',
 'MANA-USD',
 'MIOTA-USD',
 'MKR-USD',
 'NEO-USD',
 'RUNE-USD',
 'RVN-USD',
 'SNX-USD',
 'THETA-USD',
 'TRX-USD',
 'TUSD-USD',
 'VET-USD',
 'XEM-USD',
 'XLM-USD',
 'XMR-USD',
 'XRP-USD',
 'ZEC-USD']
df = yf.download(coins, start =start, end =end, interval = interval)
```

```
[**********************100%**********************]  39 of 39 completed
```

In [25]:
```
df
```

Out[25]:

Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD	CRO-USD
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654	0.033973
2020-01-02	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654	0.032858
2020-01-03	0.034180	0.228098	0.001416	4.247897	0.187701	222.412979	13.660452	0.007224	0.034666
2020-01-04	0.034595	0.236382	0.001430	4.286356	0.189891	226.018692	13.891512	0.007601	0.034689
2020-	0.034721	0.231657	0.001418	4.231877	0.188898	224.096527	14.111019	0.007661	0.034618

01-05									
...	...	...	...	...	...	...	...	...	...
2023-04-14	0.438330	0.227652	0.036314	12.262309	0.286126	132.494904	329.173859	0.134127	0.070591
2023-04-15	0.453280	0.232241	0.037160	12.394587	0.285833	132.805786	333.407288	0.133416	0.071222
2023-04-16	0.451755	0.234859	0.037015	12.697115	0.288967	134.453751	348.220917	0.138526	0.072558
2023-04-17	0.434167	0.220693	0.036002	12.341851	0.278806	131.615753	339.994110	0.135578	0.069947
2023-04-18	0.437954	0.219850	0.036813	12.437926	0.279735	132.076385	341.808136	0.135561	0.071929

1204 rows × 234 columns

создадим функцию которая добавляет индикаторы для монеты

In [26]:
```
def get_technical_indicators(data, column):
    data['MA7', column] = data['Adj Close', column].rolling(window=7).mean()
    data.loc[data['MA7', column].isna(), ('MA7', column)] = data.loc[data['MA7', column]

    data['MA21', column] = data['Adj Close', column].rolling(window=21).mean()
    data.loc[data['MA21', column].isna(), ('MA21', column)] = data.loc[data['MA21', colu

    data['MACD', column] = data['Adj Close', column].ewm(span=26).mean() - data['Adj Clo
    data.loc[data['MACD', column].isna(), ('MACD', column)] = data.loc[data['MACD', colu

    data['20SD', column] = data['Adj Close', column].rolling(20).std()
#     data.loc[data['20SD', column].isna(), ('20SD', column)] = data.loc[data['20SD', co

    data['upper_band', column] = data['MA21', column] + (data['20SD', column] * 2)
    data['lower_band', column] = data['MA21', column] - (data['20SD', column] * 2)

    data['EMA', column] = data['Adj Close', column].ewm(com=0.5).mean()
    data.loc[data['EMA', column].isna(), ('EMA', column)] = data.loc[data['EMA', column]

    data['logmomentum', column] = np.log(data['Adj Close', column] + 0.001)

    return data
```

In [27]:
```
get_technical_indicators(df, 'ADA-USD').head()
```

Out[27]:

	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD	CRO-USD	
Date										
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654	0.033973	4
2020-01-02	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654	0.032858	4
2020-01-03	0.034180	0.228098	0.001416	4.247897	0.187701	222.412979	13.660452	0.007224	0.034666	4
2020-	0.034595	0.236382	0.001430	4.286356	0.189891	226.018692	13.891512	0.007601	0.034689	4

2020-01-04										
2020-01-05	0.034721	0.231657	0.001418	4.231877	0.188898	224.096527	14.111019	0.007661	0.034618	5

5 rows × 242 columns

In [28]:
```
# Видим как алгоритм добавления сработал для одной монеты,
# теперь добавим колонки с индиакторами для всех монет. Будем делать это в цикле

for coin in df['Adj Close'].columns:
    df = get_technical_indicators(df, coin).copy()
```

In [29]:
```
df.head()
```

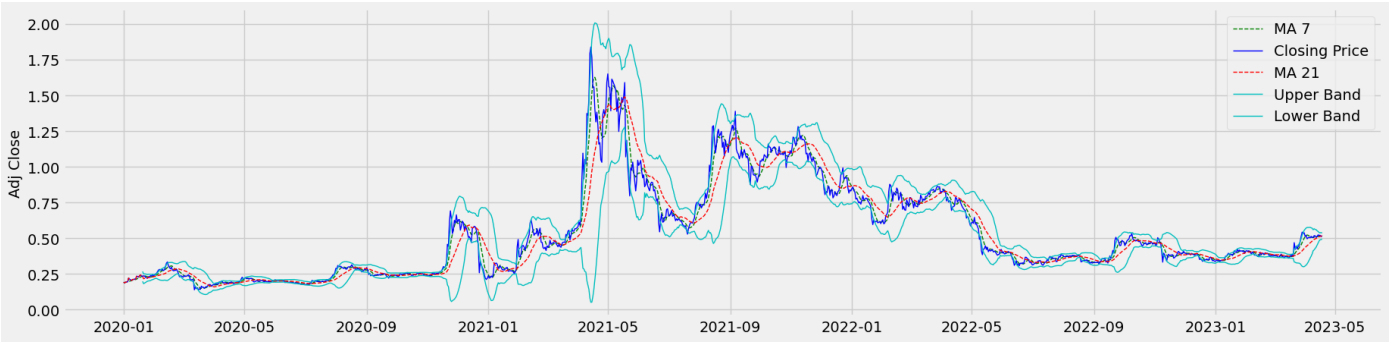
Out[29]:

Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD	CRO-USD	
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654	0.033973	4
2020-01-02	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654	0.032858	4
2020-01-03	0.034180	0.228098	0.001416	4.247897	0.187701	222.412979	13.660452	0.007224	0.034666	4
2020-01-04	0.034595	0.236382	0.001430	4.286356	0.189891	226.018692	13.891512	0.007601	0.034689	4
2020-01-05	0.034721	0.231657	0.001418	4.231877	0.188898	224.096527	14.111019	0.007661	0.034618	5

5 rows × 546 columns

In [31]:
```
 # Plot first subplot
plt.figure(figsize=(20, 5))
plt.plot(df['MA7', 'XRP-USD'], label='MA 7', color='g', linestyle='--', linewidth=1.0)
plt.plot(df['Adj Close', 'XRP-USD'], label='Closing Price', color='b', linewidth=1.0)
plt.plot(df['MA21', 'XRP-USD'], label='MA 21', color='r', linestyle='--', linewidth=1.0)
plt.plot(df['upper_band', 'XRP-USD'], label='Upper Band', color='c', linewidth=1.0)
plt.plot(df['lower_band', 'XRP-USD'], label='Lower Band', color='c', linewidth=1.0)
plt.ylabel('Adj Close')
plt.legend()
```

Out[31]:
```
<matplotlib.legend.Legend at 0x7fdc5e7b6f80>
```



In [78]:
```
data = df.loc[:, ['Adj Close','Volume', 'MA7', 'MA21', 'MACD', '20SD', 'EMA', 'logmoment
#levels = df.columns.get_level_values(0)[cols]
data.shape
```

Out[78]:
```
(1204, 312)
```

In [79]:
```
SEQ_LEN = 60  # how long of a preceeding sequence to collect for RNN, using the past 60
FUTURE_PERIOD_PREDICT = 10 # days, how far into the future are we trying to predict?
COIN_TO_PREDICT =  'ADA-USD'
```

In [80]:
```
data['future'] = data['Adj Close', COIN_TO_PREDICT].shift(-FUTURE_PERIOD_PREDICT)
data
```

Out[80]:

Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD	CRO-USD
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654	0.033973
2020-01-02	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654	0.032858
2020-01-03	0.034180	0.228098	0.001416	4.247897	0.187701	222.412979	13.660452	0.007224	0.034666
2020-01-04	0.034595	0.236382	0.001430	4.286356	0.189891	226.018692	13.891512	0.007601	0.034689
2020-01-05	0.034721	0.231657	0.001418	4.231877	0.188898	224.096527	14.111019	0.007661	0.034618
...	...	...	...	...	...	...	...	...	...
2023-04-14	0.438330	0.227652	0.036314	12.262309	0.286126	132.494904	329.173859	0.134127	0.070591
2023-04-15	0.453280	0.232241	0.037160	12.394587	0.285833	132.805786	333.407288	0.133416	0.071222
2023-04-16	0.451755	0.234859	0.037015	12.697115	0.288967	134.453751	348.220917	0.138526	0.072558
2023-04-17	0.434167	0.220693	0.036002	12.341851	0.278806	131.615753	339.994110	0.135578	0.069947
2023-04-18	0.439820	0.220725	0.036835	12.461351	0.279565	132.450699	341.799561	0.136012	0.072042

1204 rows × 313 columns

In [81]:
```
dataset = data.values
```

In [82]:
```
y= np.array(data['future'])[:-FUTURE_PERIOD_PREDICT]
len(y)
```

Out[82]:
```
1194
```

In [83]:
```
X_1=data.drop(['future'],axis =1)
X= np.array(X_1[X_1.columns])
X= X[:len(X_1)-FUTURE_PERIOD_PREDICT]
X.shape
```

Out[83]:
```
(1194, 312)
```

In [84]:
```
import math
training_data_len = math.ceil(len(dataset)*.8)
```

```
X_train = X[:training_data_len]
X_test = X[training_data_len:]
len(X_test)
```

Out[84]:
```
230
```

In [85]:
```
len(X_train)
```

Out[85]:
```
964
```

In [86]:
```
#scale the data
min_max_scaler = preprocessing.MinMaxScaler()
X_train_sc= min_max_scaler.fit_transform(X_train)

X_test_sc=min_max_scaler.fit_transform(X_test)
```

In [87]:
```
y_train = y[:training_data_len]
y_test = y[training_data_len:]
len(y_test)
```

Out[87]:
```
230
```

In [88]:
```
scaler = MinMaxScaler()
y_train_sc = scaler.fit_transform(y_train.reshape(-1, 1))
y_test_sc = scaler.fit_transform(y_test.reshape(-1, 1))
```

In [89]:
```
def report_metrics(model, X_train_sc, X_test_sc, y_train_sc, y_test_sc, label):
    print(f'Train MAE ({label}):', round(mae(y_train_sc, model.predict(X_train_sc)), 4))
    print(f'Test MAE ({label}) :', round(mae(y_test_sc, model.predict(X_test_sc)), 4), '

    print(f'Train R^2 ({label}):', round(r2(y_train_sc, model.predict(X_train_sc)), 4))
    print(f'Test R^2 ({label}) :', round(r2(y_test_sc, model.predict(X_test_sc)), 4), '\
```

In [90]:
```
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.dummy import DummyRegressor
```

In [36]:
```
baseline = DummyRegressor(strategy='mean').fit(X_train_sc, y_train_sc)
report_metrics(baseline, X_train_sc, X_test_sc, y_train_sc, y_test_sc, 'baseline')
```

In [34]:
```
LR_baseline = LinearRegression().fit(X_train_sc, y_train_sc)
report_metrics(LR_baseline, X_train_sc, X_test_sc, y_train_sc, y_test_sc, 'LR_baseline')
```

In [94]:
```
from catboost import CatBoostRegressor
# from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split, KFold, cross_val_score
```

In [35]:
```
list_of_models = [
        RandomForestRegressor(),
        XGBRegressor(),
        KNeighborsRegressor(),
#         LGBMRegressor(),
        SVR()]



list_of_model_names = [type(x).__name__ for x in list_of_models]
```

```
cv_results = pd.DataFrame(
        data=0.0,
        index=list_of_model_names,
        columns=['fit_time', 'score_time',  'test_neg_mean_absolute_error','test_r2','ne



    # обучение всех моделей из списка
for model in list_of_models:

        cv_result = cross_validate(
            estimator=model,
            X=X_train_sc,
            y=y_train_sc,
            scoring=['neg_mean_absolute_error','r2','neg_mean_squared_error'],
            cv=50,
            n_jobs=-1)


        cv_results.loc[type(model).__name__] = list(map(np.mean, cv_result.values()))

print(cv_results)
```

In [ ]:
```
list_of_models = [
        CatBoostRegressor()]



list_of_model_names = [type(x).__name__ for x in list_of_models]
cv_results = pd.DataFrame(
        data=0.0,
        index=list_of_model_names,
        columns=['fit_time', 'score_time', 'test_neg_mean_absolute_error','test_r2','neg



    # обучение всех моделей из списка
for model in list_of_models:

        cv_result = cross_validate(
            estimator=model,
            X=X_train_sc,
            y=y_train_sc,
            scoring=['neg_mean_absolute_error','r2','neg_mean_squared_error'],
            cv=50,
            n_jobs=-1,)


        cv_results.loc[type(model).__name__] = list(map(np.mean, cv_result.values()))

print(cv_results)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: