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 [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
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 [4]:
```
from sklearn.metrics import (
    mean_absolute_error as mae,
    r2_score as r2,
    mean_absolute_percentage_error as mape)
```

In [5]:
```
import random
from collections import deque
from sklearn import preprocessing
```

In [6]:
```
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
```

In [7]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import TimeDistributed,Dense, Dropout, LSTM,BatchNormalizat
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ModelCheckpoint
import time
```

```
2023-04-18 21:28:08.537759: I tensorflow/core/platform/cpu_feature_guard.cc:193] This Te
nsorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler fla
gs.
```

In [8]:

```
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 [9]:

```
df
```

Out[9]:

	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
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	0.438529	0.220074	0.036843	12.451017	0.280037	132.164276	341.941895	0.135617	0.071959

1204 rows × 234 columns

In [10]:

```
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 [11]:

```
 # 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 [12]:

```
df.head()
```

Out[12]:

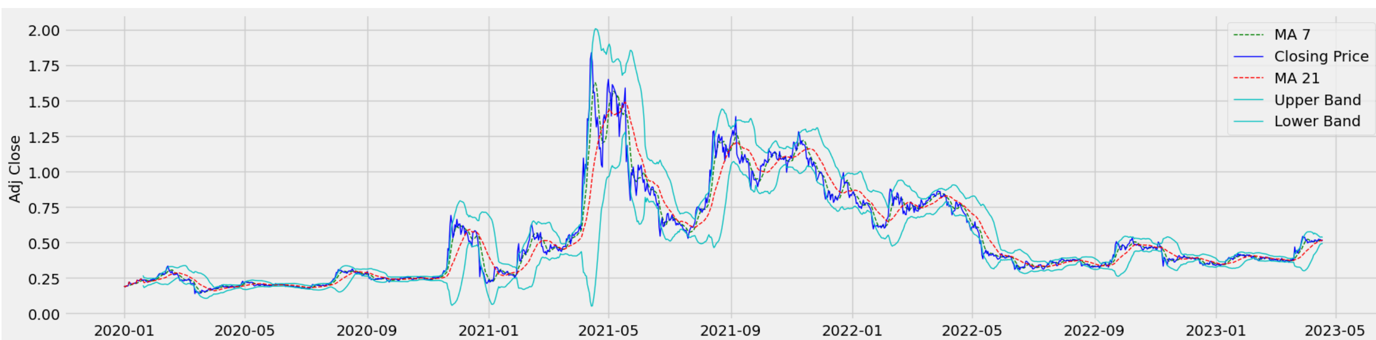
Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD	CRO-USD	
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654	0.033973	4
2020-01-02	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654	0.032858	4
2020-01-03	0.034180	0.228098	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 [13]:

```
# 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[13]: <matplotlib.legend.Legend at 0x7fa58938f520>



In [13]:

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

Out[13]: (1204, 78)

In [14]:

```
SEQ_LEN = 30  # 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 [15]:

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

Out[15]:

Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD
2020-01-01	0.033458	0.219938	0.001446	4.380158	0.196129	204.397537	13.689083	0.006654

00:00:00+00:00								
2020-01-02 00:00:00+00:00	0.032751	0.213518	0.001397	4.091817	0.183821	195.698563	13.027011	0.006654
2020-01-03 00:00:00+00:00	0.034180	0.228098	0.001416	4.247897	0.187701	222.412979	13.660452	0.007224
2020-01-04 00:00:00+00:00	0.034595	0.236382	0.001430	4.286356	0.189891	226.018692	13.891512	0.007601
2020-01-05 00:00:00+00:00	0.034721	0.231657	0.001418	4.231877	0.188898	224.096527	14.111019	0.007661
...	...	...	...	...	...	...	...	...
2023-04-14 00:00:00+00:00	0.438330	0.227652	0.036314	12.262309	0.286126	132.494904	329.173859	0.134127
2023-04-15 00:00:00+00:00	0.453280	0.232241	0.037160	12.394587	0.285833	132.805786	333.407288	0.133416
2023-04-16 00:00:00+00:00	0.451755	0.234859	0.037015	12.697115	0.288967	134.453751	348.220917	0.138526
2023-04-17 00:00:00+00:00	0.434167	0.220693	0.036002	12.341851	0.278806	131.615753	339.994110	0.135578
2023-04-18 00:00:00+00:00	0.446870	0.225227	0.037674	12.745592	0.285334	133.828247	346.186279	0.138460

1204 rows × 79 columns

In [16]:

```
import math
dataset = data.values
training_data_len = math.ceil(len(dataset)*.8)
train_data = data[:training_data_len]
```

In [17]:

```
train_data = data[:training_data_len]
test_data = data[training_data_len:]
test_data
```

Out[17]:

Date	ADA-USD	ALGO-USD	ANKR-USD	ATOM-USD	BAT-USD	BCH-USD	BNB-USD	CHZ-USD
2022-08-22 00:00:00+00:00	0.461446	0.303030	0.037094	11.084868	0.362567	122.857780	300.559113	0.221774
2022-08-23 00:00:00+00:00	0.465207	0.307729	0.037227	12.117525	0.371596	133.494919	299.029938	0.249499
2022-08-24 00:00:00+00:00	0.458109	0.302357	0.040271	13.016421	0.368381	131.212296	296.449677	0.242247
2022-08-25 00:00:00+00:00	0.464999	0.307252	0.041060	12.869333	0.373507	130.680695	301.583649	0.224285
2022-08-26 00:00:00+00:00	0.430863	0.282159	0.036941	11.499005	0.336370	116.286339	279.598175	0.207100
...	...	...	...	...	...	...	...	...
2023-04-14 00:00:00+00:00	0.438330	0.227652	0.036314	12.262309	0.286126	132.494904	329.173859	0.134127
2023-04-15 00:00:00+00:00	0.453280	0.232241	0.037160	12.394587	0.285833	132.805786	333.407288	0.133416

2023-04-16 00:00:00+00:00	0.451755	0.234859	0.037015	12.697115	0.288967	134.453751	348.220917	0.138526
2023-04-17 00:00:00+00:00	0.434167	0.220693	0.036002	12.341851	0.278806	131.615753	339.994110	0.135578
2023-04-18 00:00:00+00:00	0.446870	0.225227	0.037674	12.745592	0.285334	133.828247	346.186279	0.138460

240 rows × 79 columns

In [18]:
```
X_train= train_data.drop("future",axis= 1)
Y_train =train_data["future"]
```

In [19]:
```
X_test= test_data.drop("future",axis= 1)
Y_test =test_data["future"]
print(Y_test)
```

```
Date
2022-08-22 00:00:00+00:00    0.457481
2022-08-23 00:00:00+00:00    0.454559
2022-08-24 00:00:00+00:00    0.480387
2022-08-25 00:00:00+00:00    0.503084
2022-08-26 00:00:00+00:00    0.499121
                               ...
2023-04-14 00:00:00+00:00         NaN
2023-04-15 00:00:00+00:00         NaN
2023-04-16 00:00:00+00:00         NaN
2023-04-17 00:00:00+00:00         NaN
2023-04-18 00:00:00+00:00         NaN
Name: future, Length: 240, dtype: float64
```

In [20]:
```
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 [21]:
```
scaler = MinMaxScaler()
Y_train_sc = scaler.fit_transform(Y_train.values.reshape(-1, 1))
Y_test_sc = scaler.fit_transform(Y_test.values.reshape(-1, 1))
#Y_train_sc
# convert the scaled data back to a Pandas Series
#scaled_series = pd.Series(Y_train_sc.reshape(-1))
```

In [22]:
```
X_train =[]
y_train =[]
for i in range(SEQ_LEN,len(X_train_sc)):
    X_train.append(X_train_sc[i-SEQ_LEN:i])
    y_train.append(Y_train_sc[i][0])
X_train,y_train = np.array(X_train),np.array(y_train)
```

In [23]:
```
X_train,y_train = np.array(X_train),np.array(y_train)
```

In [24]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(934, 30, 78)
(934,)
```

In [25]:
```
X_test =[]
y_test =[]

for i in range(SEQ_LEN,len(X_test_sc)):
```

```
    X_test.append(X_test_sc[i-SEQ_LEN:i])
    y_test.append(Y_test_sc[i][0])
X_test = np.array(X_test)
```

In [26]:
```
X_train_shape = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],X_train.shape[2]))
X_train_shape.shape
```

Out[26]:
```
(934, 30, 78)
```

In [ ]:
```
model = Sequential()
model.add(LSTM(units=512, activation = 'softplus',return_sequences=True, input_shape=(X_
model.add(Dropout(0.2))
model.add(LSTM(units=256,activation = 'softplus', return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(units=64))
model.add(Dropout(0.5))


model.add(Dense(units=1, activation = 'linear'))
opt =tf.keras.optimizers.Adam(
    learning_rate=0.001,

    name='Adam')
model.compile(optimizer=opt, loss='mean_squared_error', metrics=['mae'])
history = model.fit(X_train_shape, y_train, epochs=20, batch_size=16,validation_split  =
```

```
Epoch 1/20
47/47 [==============================] - 26s 486ms/step - loss: 0.1585 - mae: 0.3138 - v
al_loss: 0.0102 - val_mae: 0.0682
Epoch 2/20
47/47 [==============================] - 22s 476ms/step - loss: 0.1080 - mae: 0.2735 - v
al_loss: 0.0080 - val_mae: 0.0805
Epoch 3/20
47/47 [==============================] - 22s 478ms/step - loss: 0.1035 - mae: 0.2701 - v
al_loss: 0.0178 - val_mae: 0.1183
Epoch 4/20
47/47 [==============================] - 23s 484ms/step - loss: 0.0907 - mae: 0.2511 - v
al_loss: 0.0403 - val_mae: 0.1851
Epoch 5/20
47/47 [==============================] - 24s 504ms/step - loss: 0.0956 - mae: 0.2610 - v
al_loss: 0.0253 - val_mae: 0.1432
Epoch 6/20
47/47 [==============================] - 23s 490ms/step - loss: 0.0909 - mae: 0.2569 - v
al_loss: 0.0304 - val_mae: 0.1580
Epoch 7/20
47/47 [==============================] - 23s 485ms/step - loss: 0.0883 - mae: 0.2535 - v
al_loss: 0.0372 - val_mae: 0.1766
Epoch 8/20
47/47 [==============================] - 23s 485ms/step - loss: 0.0849 - mae: 0.2463 - v
al_loss: 0.0227 - val_mae: 0.1349
Epoch 9/20
47/47 [==============================] - 23s 498ms/step - loss: 0.0846 - mae: 0.2526 - v
al_loss: 0.0072 - val_mae: 0.0773
Epoch 10/20
47/47 [==============================] - 23s 489ms/step - loss: 0.0819 - mae: 0.2475 - v
al_loss: 0.0259 - val_mae: 0.1448
Epoch 11/20
47/47 [==============================] - 23s 496ms/step - loss: 0.0787 - mae: 0.2448 - v
al_loss: 0.0076 - val_mae: 0.0789
Epoch 12/20
47/47 [==============================] - 23s 492ms/step - loss: 0.0825 - mae: 0.2513 - v
al_loss: 0.0076 - val_mae: 0.0788
Epoch 13/20
47/47 [==============================] - 23s 487ms/step - loss: 0.0794 - mae: 0.2474 - v
al_loss: 0.0094 - val_mae: 0.0861
```

```
Epoch 14/20
47/47 [==============================] - 24s 511ms/step - loss: 0.0773 - mae: 0.2442 - v
al_loss: 0.0152 - val_mae: 0.1083
Epoch 15/20
47/47 [==============================] - 24s 511ms/step - loss: 0.0815 - mae: 0.2517 - v
al_loss: 0.0068 - val_mae: 0.0758
Epoch 16/20
47/47 [==============================] - 23s 491ms/step - loss: 0.0782 - mae: 0.2460 - v
al_loss: 0.0121 - val_mae: 0.0964
Epoch 17/20
47/47 [==============================] - 23s 483ms/step - loss: 0.0791 - mae: 0.2489 - v
al_loss: 0.0080 - val_mae: 0.0803
Epoch 18/20
39/47 [=======================>......] - ETA: 3s - loss: 0.0792 - mae: 0.2479
```

In [36]:

```
print(history.history['mae'][-10:]), print(history.history['val_mae'][-10:])
```

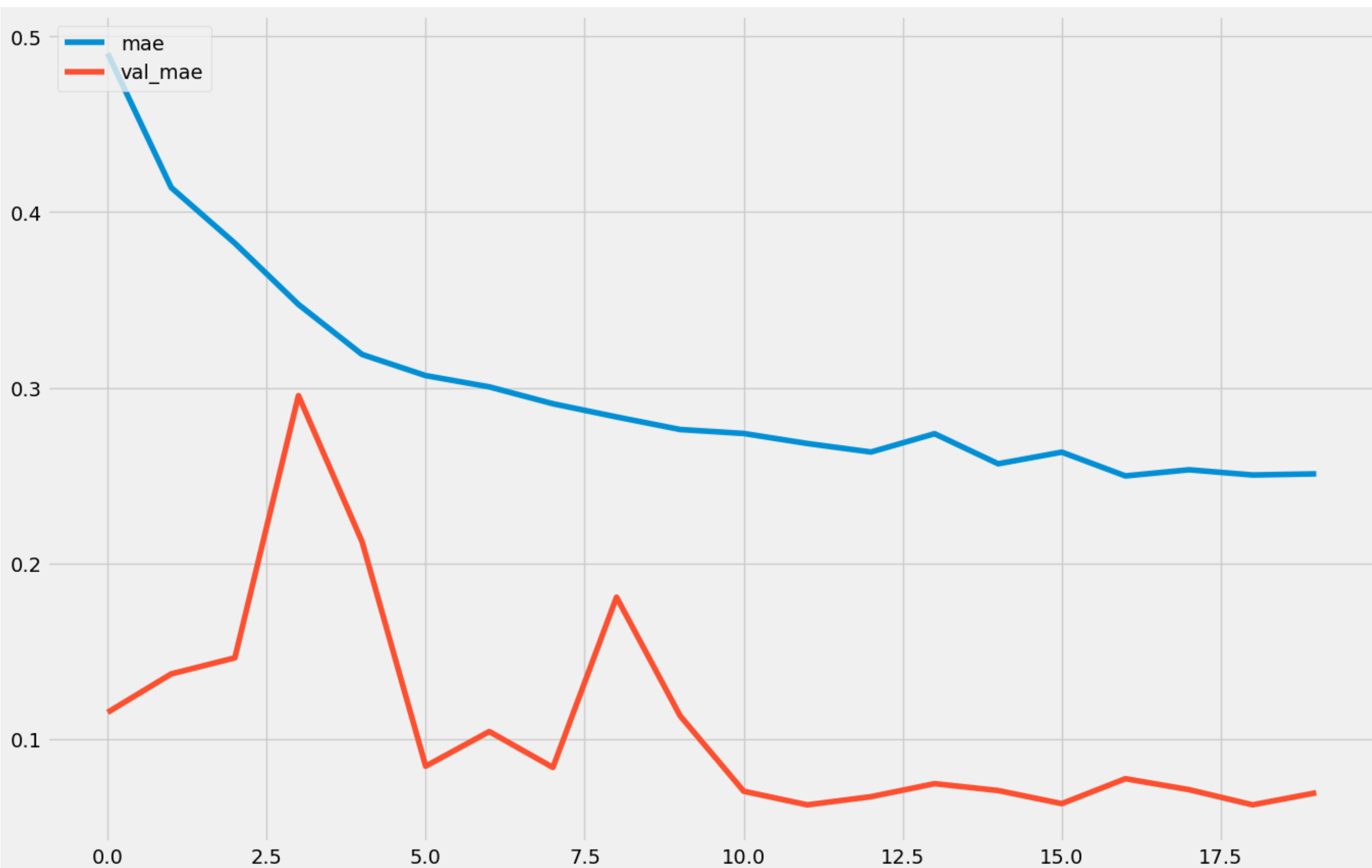
```
[0.2739524245262146, 0.2682483196258545, 0.26340439915657043, 0.27381446957588196, 0.256
6968500614166, 0.2633240222930908, 0.24978433549404144, 0.25328880548477173, 0.250311762
0944977, 0.25096043944358826]
[0.07028141617774963, 0.06253711879253387, 0.06718610227108002, 0.07460053265094757, 0.0
7068581134080887, 0.06322547048330307, 0.07742375135421753, 0.07118966430425644, 0.06255
412101745605, 0.06943517923355103]
```

Out[36]:

```
(None, None)
```

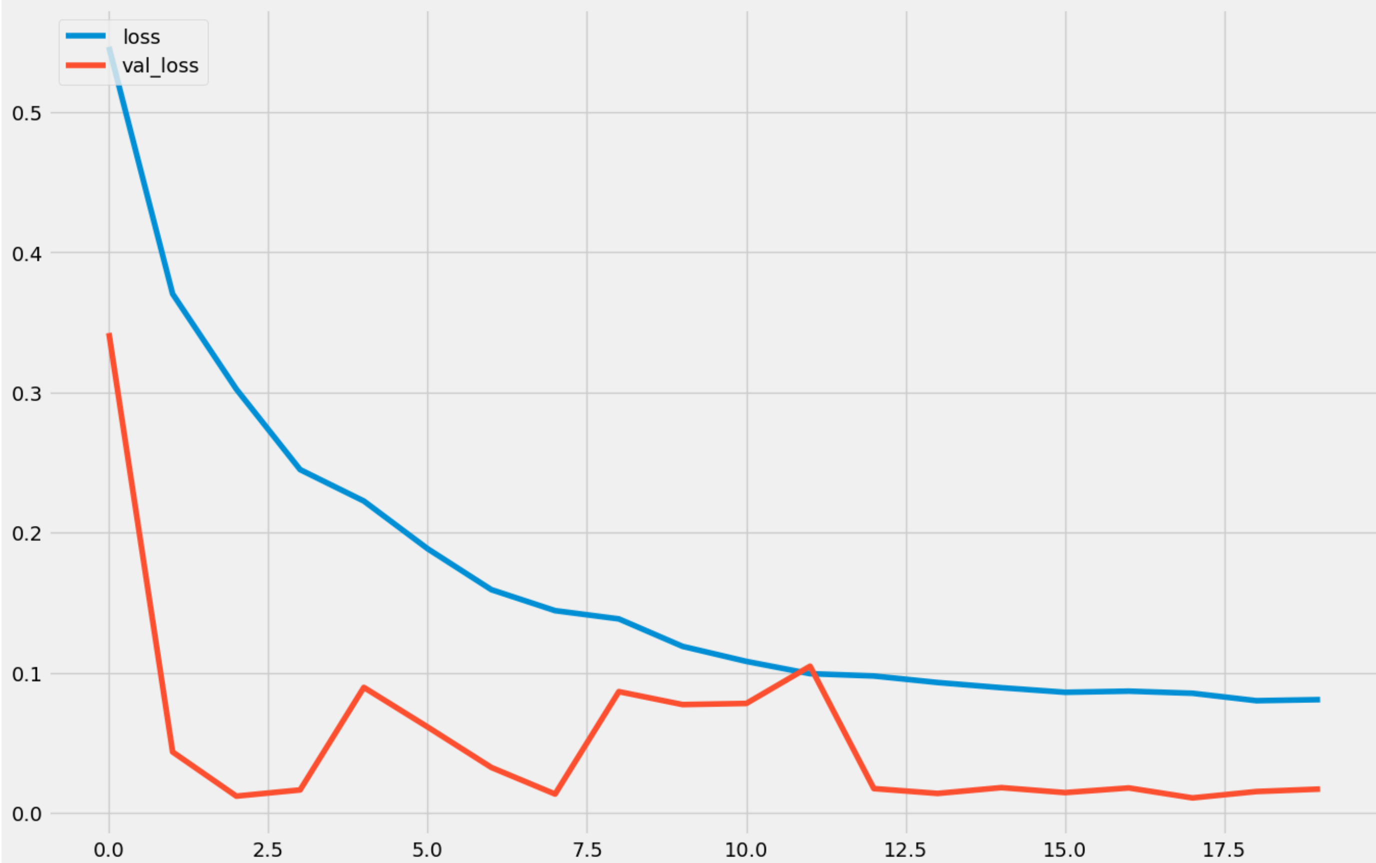
In [37]:

```
plt.figure(figsize =(15,10))
plt.plot(history.history['mae'],label= 'mae')
plt.plot(history.history['val_mae'], label='val_mae')
plt.legend(loc="upper left")
plt.show()
```



In [34]:

```
plt.figure(figsize =(15,10))
plt.plot(history.history['loss'],label= 'loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend(loc="upper left")
plt.show()
```



In [ ]:

In [ ]:

In [221…

```
import math
dataset = data.values
training_data_len = math.ceil(len(dataset)*.8)
train_data = data[:training_data_len]
```

In [222…

```
y_pred = model.predict(X_test)
```

```
7/7 [==============================] - 2s 222ms/step
```

In [223…

```
print(r2(y_test[:-10], y_pred[:-10]))
```

```
-0.07192073984242042
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```