# Aim: To predict the prices of stainless-steel (Column B: StainlessSteelPrice)

Tasks:

1. Conduct exploratory data analysis and report the insights Answer:Exploratory Data Analysis has been done. Following steps performed for this purpose #1. Missing Values if any #2. Distribution of the Numerical Variables #3. Relationship between independent and dependent feature(StainlessSteelPrice) #4. Relationship in between independent features #5. Removed high correlated independent features #6. Analysed and displayed the trend, seasonality etc.

2. Forecast the prices of stainless-steel for the given time period(s) using the following methods

3. Statistical Models (at least one type): Answer: Following methods has been used: #1. Prophet Statistical method #2. SARIMA

4. Machine Learning (at least one type) Answer: Linear Regresion, LASSO, Ridge Regression, Gradient Boosting Method(xgboost), Neural Network,

3.Deep Learning (at least one type) Answer: LSTM and deep neural network have been used for this purpose

Select the features - Column C (Steel_Inventory_M_USD) through column T (Copper_Global_USD) – that help to improve the accuracy Answer: I have used some features for price forcasting

1. Provide insights on how your model made the prediction Answer: There are two ways to learn the time searies data:
2. Use the time or date with other independent fearture. For this purpose the previous prices as well as independent features have been taken into account to learn this time series data. Here I have used the prophet, SARIMA, LSTM approaches to forcast the stainless steel prices.
3. Take into account the time or effect as a new variable and build the machine learning model( Here I have created a new variable number of days from today to that date, it can be taken as future date as well to take the effect of future as well). Neural Network, Gradient boosting and Random forest approach have been used to learn these data.

4. Use the following evaluation criteria for model selection Mean Absolute Percentage Error (MAPE) Directional Symmetry Answer: Both methods have been implemented an have been used to evaluate the models

Conclusion (Summary): Among all models prophet model is perfoming best with Mean Absolute Percentage Error(MAPE) 4.25. Crossvalidation has been used to evaluate this model. For each year 2 data points has been used to test the model with total 13 independent features.

Libraries used: Pytoech, SKlearn, Keras, AutoFeat, Facebook Prophet

In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import xgboost
import csv as csv
from xgboost import plot_importance
from matplotlib import pyplot
from sklearn.model_selection import cross_val_score,KFold
from sklearn.model_selection import  train_test_split,GridSearchCV
from sklearn.metrics import mean_absolute_error
from scipy.stats import skew
from collections import OrderedDict
from sklearn.preprocessing import StandardScaler
from fbprophet import Prophet
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pandas.plotting import register_matplotlib_converters
import torch
import torch.nn as nn
from pandas import DataFrame
import itertools
import os
import sys
import warnings
from sklearn.datasets import load_boston, load_diabetes
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.seasonal import seasonal_decompose
from autofeat import AutoFeatRegressor
from fbprophet.plot import plot_cross_validation_metric
from fbprophet.diagnostics import performance_metrics
from fbprophet.diagnostics import cross_validation
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import make_scorer
%matplotlib inline
%load_ext autoreload
%autoreload 2
register_matplotlib_converters()
# Comment this if the data visualisations doesn't work on your side
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:

pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
```

In [2]:

```
data=pd.read_csv('https://raw.githubusercontent.com/sureshkuc/chemovator./main/Stainless-Steel-Prices-Foreca
```

In [3]:

```
data
```

Out[3]:

|  | Date | StainlessSteelPrice | Steel_Inventory_M_USD | Steel_Orders_M_USD | IronOre_Global_USD | CoalAustralia_Global_USD | Energy_I |
|---|---|---|---|---|---|---|---|
| **0** | 7/1/2013 | 0.827352 | 20575 | 10568 | 127.191304 | 82.222360 | 227.02 |
| **1** | 8/1/2013 | 0.760221 | 20609 | 10821 | 137.055000 | 82.221429 | 232.23 |
| **2** | 9/1/2013 | 0.784715 | 20761 | 10774 | 134.185714 | 83.295918 | 234.33 |
| **3** | 10/1/2013 | 0.769293 | 20735 | 11291 | 132.572727 | 85.443634 | 229.39 |
| **4** | 11/1/2013 | 0.750242 | 20854 | 11559 | 136.323810 | 88.363265 | 225.43 |
| **...** | ... | ... | ... | ... | ... | ... |  |
| **79** | 2/1/2020 | 0.893123 | 19753 | 9914 | 87.625000 | 71.050179 | 110.31 |
| **80** | 3/1/2020 | 0.829167 | 19945 | 9072 | 88.659091 | 70.204870 | 73.68 |
| **81** | 4/1/2020 | 0.789704 | 19824 | 7800 | 83.750000 | 64.620000 | 52.77 |
| **82** | 5/1/2020 | 0.762035 | 19441 | 8908 | 91.333333 | 54.148308 | 66.91 |
| **83** | 6/1/2020 | 0.811477 | 18998 | 8979 | 103.340909 | 55.002760 | 82.41 |

84 rows × 20 columns

In [4]:

```
data.describe()
```

Out[4]:

|  | StainlessSteelPrice | Steel_Inventory_M_USD | Steel_Orders_M_USD | IronOre_Global_USD | CoalAustralia_Global_USD | Energy_Index | N |
|---|---|---|---|---|---|---|---|
| **count** | 84.000000 | 84.000000 | 84.000000 | 84.000000 | 84.000000 | 84.000000 |  |
| **mean** | 0.810715 | 20070.559524 | 9831.595238 | 79.962940 | 81.844151 | 143.408951 |  |
| **std** | 0.143444 | 1764.033210 | 1240.333905 | 23.874776 | 19.443360 | 47.136073 |  |
| **min** | 0.499859 | 16688.000000 | 7650.000000 | 40.882609 | 53.428929 | 52.771306 |  |
| **25%** | 0.743778 | 18980.000000 | 8904.000000 | 61.484190 | 67.111352 | 111.607197 |  |
| **50%** | 0.812157 | 20190.000000 | 9947.000000 | 74.280682 | 78.677679 | 131.088857 |  |
| **75%** | 0.893690 | 21458.500000 | 11062.500000 | 92.409524 | 96.985695 | 163.923895 |  |
| **max** | 1.075014 | 23464.000000 | 11825.000000 | 137.055000 | 125.085877 | 237.208789 |  |

In [5]:

```
#Exploratory Data Analysis:
#In Data Analysis We will Analyze To Find out the below stuff
#1. Missing Values if any
#2. Distribution of the Numerical Variables
#3. Relationship between independent and dependent feature(StainlessSteelPrice)
#4. Relationship in between independent features
```

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84 entries, 0 to 83
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Date                      84 non-null     object 
 1   StainlessSteelPrice       84 non-null     float64
 2   Steel_Inventory_M_USD     84 non-null     int64  
 3   Steel_Orders_M_USD        84 non-null     int64  
 4   IronOre_Global_USD        84 non-null     float64
 5   CoalAustralia_Global_USD  84 non-null     float64
 6   Energy_Index              84 non-null     float64
 7   Nickel_Global_USD         84 non-null     float64
 8   Zinc_Global_USD           84 non-null     float64
 9   Freight_Transport_Index   84 non-null     float64
 10  WTI_Crude_Global_USD      84 non-null     float64
 11  Baltic_Dry_Index          84 non-null     int64  
 12  BCI_China                 84 non-null     float64
 13  BCI_Europe                84 non-null     float64
 14  BCI_US                    84 non-null     float64
 15  CLI_China                 84 non-null     float64
 16  CLI_Europe                84 non-null     float64
 17  CLI_US                    84 non-null     float64
 18  PMI_US_Manufacturing      84 non-null     float64
 19  Copper_Global_USD         84 non-null     float64
dtypes: float64(16), int64(3), object(1)
memory usage: 13.2+ KB
```

# Lets take a look at how the StainlessSteel Price is distributed

In [8]:

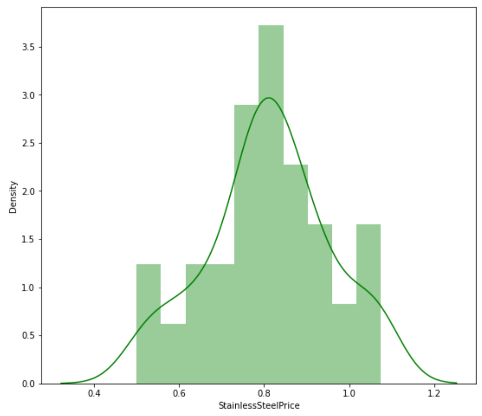
```
print(data['StainlessSteelPrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(data['StainlessSteelPrice'], color='g', bins=10, hist_kws={'alpha': 0.4})
```

```
count    84.000000
mean      0.810715
std       0.143444
min       0.499859
25%       0.743778
50%       0.812157
75%       0.893690
max       1.075014
Name: StainlessSteelPrice, dtype: float64
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use e
ither `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for
histograms).
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff6baadf208>
```



# New Variable Creation for date to take into account the time component into model( machine Learning models)

In [9]:

```
from datetime import date
data['dd'],data['mm'],data['yyyy']=data['Date'].str.split("/", expand=True)[0],data['Date'].str.split("/", e
today = date.today()
number_of_days=[]
dates=[]
for yy, mm, dd in zip(data['yyyy'],data['mm'],data['dd']):
  d1 = date(int(yy),int(mm),int(dd))
  dates.append(d1)
  delta = today - d1
  number_of_days.append(delta.days)
data['Date']=dates
```

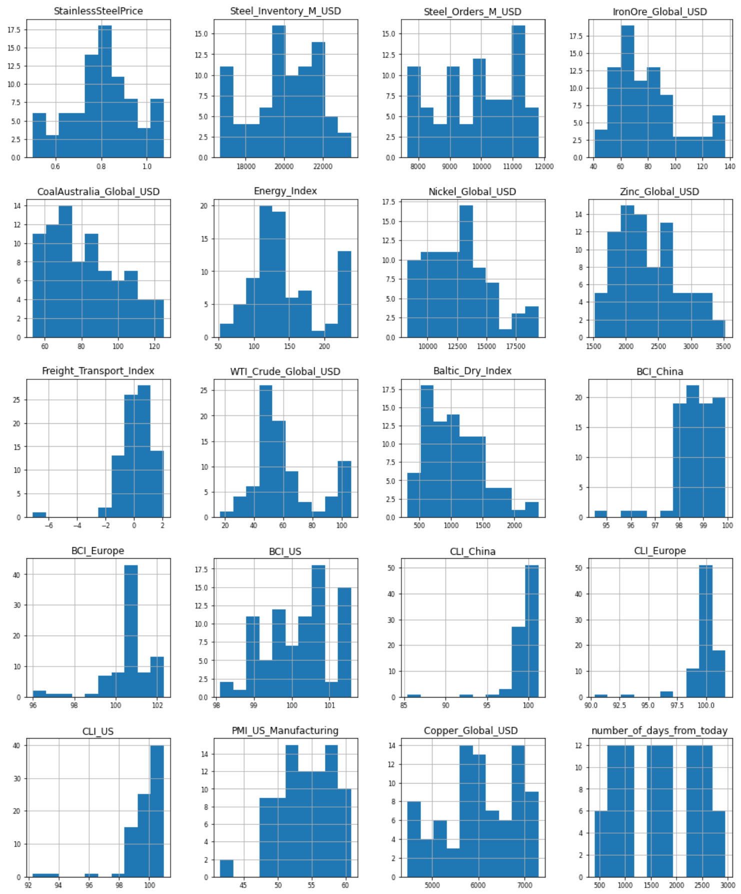
## Distribution of Independent Features

In [10]:

```
data['number_of_days_from_today']=number_of_days
data.hist(figsize=(16, 20), bins=10, xlabelsize=8, ylabelsize=8)
```

Out[10]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9adf518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b962bd30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b965cf98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9619240>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b95cd4a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9581710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9535978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9568ba8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9568c18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b94de0f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9493358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b94465c0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b93f8828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b93ada90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b93e1cf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9396f60>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9adf128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9529278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b9405ef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7ff6b92db9b0>]],
      dtype=object)
```



## Correlation

Now we'll try to find which features are strongly correlated. Featurres which has correlation 0.9 we will store them and will remove later bofore the model building process. We found 4 fearture which has more than 0.5 correlation with StainlessSteel Price dependent variable, which are as follows: 1.Steel_Inventory_M_USD 2. Steel_Orders_M_USD 3. Copper_Global_USD 5. Nickel_Global_USD

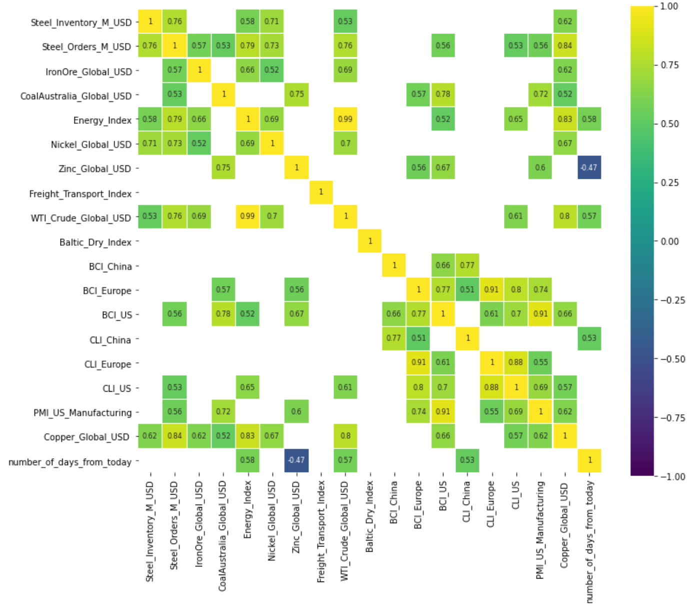
In [13]:

```
correlatedcolumns=list()
for i in [col for col in data.columns if col not in ['Date','dd','mm','yyyy','StainlessSteelPrice']]:
  df_num_corr = data.corr()[i][1:] # 1 because the first row is StainlessSteelPrice
  removable_features_list = df_num_corr[abs(df_num_corr) > 0.9  ].sort_values(ascending=False)
  for key, value in dict(removable_features_list).items():
    if key != i:
      correlatedcolumns.append(key)
  print("There is {} strongly correlated values with {}:\n{}".format(len(removable_features_list), i, remova
columns_need_to_remove=list(set(correlatedcolumns))
```

```
There is 1 strongly correlated values with Steel_Inventory_M_USD:
Steel_Inventory_M_USD    1.0
Name: Steel_Inventory_M_USD, dtype: float64
There is 1 strongly correlated values with Steel_Orders_M_USD:
Steel_Orders_M_USD    1.0
Name: Steel_Orders_M_USD, dtype: float64
There is 1 strongly correlated values with IronOre_Global_USD:
IronOre_Global_USD    1.0
Name: IronOre_Global_USD, dtype: float64
There is 1 strongly correlated values with CoalAustralia_Global_USD:
CoalAustralia_Global_USD    1.0
Name: CoalAustralia_Global_USD, dtype: float64
There is 2 strongly correlated values with Energy_Index:
Energy_Index            1.000000
WTI_Crude_Global_USD    0.988651
Name: Energy_Index, dtype: float64
There is 1 strongly correlated values with Nickel_Global_USD:
Nickel_Global_USD    1.0
Name: Nickel_Global_USD, dtype: float64
There is 1 strongly correlated values with Zinc_Global_USD:
Zinc_Global_USD    1.0
Name: Zinc_Global_USD, dtype: float64
There is 1 strongly correlated values with Freight_Transport_Index:
Freight_Transport_Index    1.0
Name: Freight_Transport_Index, dtype: float64
There is 2 strongly correlated values with WTI_Crude_Global_USD:
WTI_Crude_Global_USD    1.000000
Energy_Index            0.988651
Name: WTI_Crude_Global_USD, dtype: float64
There is 1 strongly correlated values with Baltic_Dry_Index:
Baltic_Dry_Index    1.0
Name: Baltic_Dry_Index, dtype: float64
There is 1 strongly correlated values with BCI_China:
BCI_China    1.0
Name: BCI_China, dtype: float64
There is 2 strongly correlated values with BCI_Europe:
BCI_Europe    1.000000
CLI_Europe    0.907737
Name: BCI_Europe, dtype: float64
There is 2 strongly correlated values with BCI_US:
BCI_US                  1.000000
PMI_US_Manufacturing    0.908061
Name: BCI_US, dtype: float64
There is 1 strongly correlated values with CLI_China:
CLI_China    1.0
Name: CLI_China, dtype: float64
There is 2 strongly correlated values with CLI_Europe:
CLI_Europe    1.000000
BCI_Europe    0.907737
Name: CLI_Europe, dtype: float64
There is 1 strongly correlated values with CLI_US:
CLI_US    1.0
Name: CLI_US, dtype: float64
There is 2 strongly correlated values with PMI_US_Manufacturing:
PMI_US_Manufacturing    1.000000
BCI_US                  0.908061
Name: PMI_US_Manufacturing, dtype: float64
There is 1 strongly correlated values with Copper_Global_USD :
Copper_Global_USD     1.0
Name: Copper_Global_USD , dtype: float64
There is 1 strongly correlated values with number_of_days_from_today:
number_of_days_from_today    1.0
Name: number_of_days_from_today, dtype: float64
```

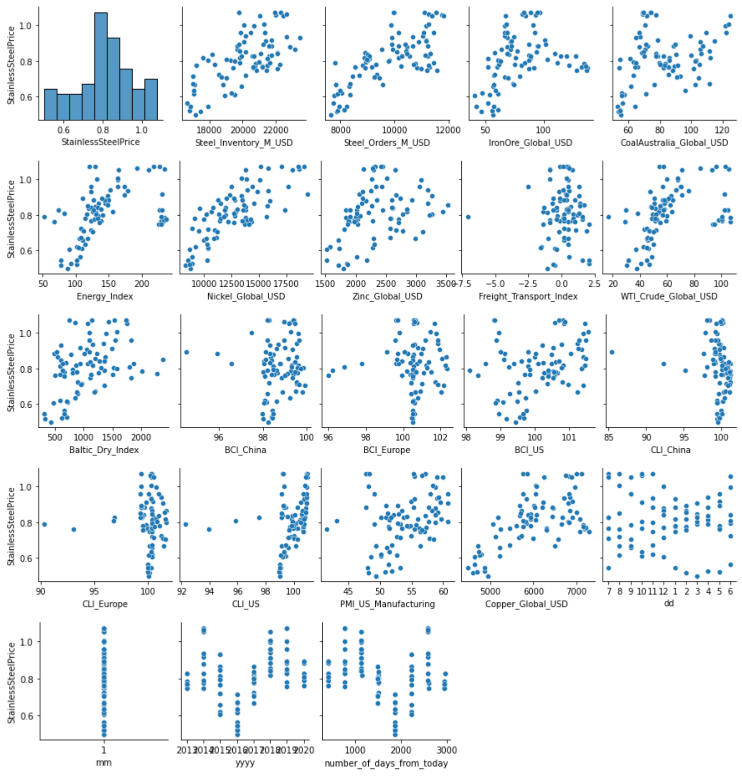
In [16]:

```
#heatmap
corr = data.drop('StainlessSteelPrice', axis=1).corr()
plt.figure(figsize=(12, 10))

sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True);
```



```
In [14]: #we now have a list of strongly correlated values but this list is incomplete as we know that correlation is

         #Plot the numerical features and see which ones have very few or explainable outliers
         #Remove the outliers from these features and see which one can have a good correlation without their outlier
         #Btw, correlation by itself does not always explain the relationship between data so ploting them could even

         #For example, relationships such as curvilinear relationship cannot be guessed just by looking at the correl
```

```
In [15]: for i in range(1, len(data.columns), 5):
             sns.pairplot(data=data,
                          x_vars=data.columns[i:i+5],
                          y_vars=['StainlessSteelPrice'])
```



## Price vs Year plot

```
In [17]: data.groupby(['yyyy'])['StainlessSteelPrice'].median().plot()
         plt.xlabel('Year')
         plt.ylabel('Median  Price')
         plt.title("Price vs Year")
```

```
Out[17]: Text(0.5, 1.0, 'Price vs Year')
```

```
In [18]: every_column_except_y= [col for col in data.columns if col not in ['Date','dd','mm','yyyy','StainlessSteelPr
```

## Evaluation Metric

```
In [19]: #Mean Absolute Percentage Error
         def mean_absolute_percentage_error(y_true, y_pred):
           y_true, y_pred = np.array(y_true), np.array(y_pred)
           return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
         #Directional Symmetry Statistic
         def directional_symmetry(y_true, y_pred):
           y_true, y_pred = np.array(y_true), np.array(y_pred)
           return (1/(len(y_true)-1))*100*np.sum(np.where((y_true[1:]-y_true[:-1])*(y_pred[1:]-y_pred[:-1])>0,1,0))
```

## Statistical Model1: Prophet model

```
In [21]: df_tmp=data.copy()
         # split into train and test
         # Split the data into X & y
         X = df_tmp.drop(columns=['dd','mm','yyyy']+columns_need_to_remove,axis=1)
         data_train=X
         data_train.rename(columns={"Date": "ds", "StainlessSteelPrice": "y"}, inplace=True)
         #data_test.rename(columns={"Date": "ds", "StainlessSteelPrice": "y"}, inplace=True)
         #y_pred = m.predict(data_test)
         #y_pred
         #y_hat=np.array(y_pred[ ['yhat']]).reshape(2)
         #print("MAPE on test data:", mean_absolute_percentage_error(data_test['y'], y_hat))
         #print("Directional Symmetry Statistic on test data:", directional_symmetry(data_test['y'], y_hat))
```

```
In [22]: param_grid = {
             'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5],
             'seasonality_prior_scale': [0.01, 0.1, 1.0, 10.0],
         }

         # Generate all combinations of parameters
         all_params = [dict(zip(param_grid.keys(), v)) for v in itertools.product(*param_grid.values())]
         mape = []  # Store the RMSEs for each params here

         # Use cross validation to evaluate all parameters
         for params in all_params:
             m = Prophet(**params) # Fit model with given params
             m = Prophet()
             for i in data_train.columns:
               if i not in ['ds','y']:
                 m.add_regressor(i)
             m.fit(data_train)
             #df_cv = cross_validation(m, cutoffs=cutoffs, horizon='30 days', )
             cutoffs = pd.to_datetime(['2013-01-10', '2014-01-09', '2015-01-09','2016-01-09','2017-01-09','2018-01-09
             df_cv = cross_validation(m, cutoffs=cutoffs, horizon='2 days',parallel="processes")
             #df_p = performance_metrics(df_cv)
             df_p = performance_metrics(df_cv, rolling_window=1)
             mape.append(df_p['mape'].values[0])

         # Find the best parameters
         tuning_results = pd.DataFrame(all_params)
         tuning_results['mape'] = mape
         print(tuning_results)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6aeb
cccf8>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6aeb
cc128>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae3
25320>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6aeb
cc7b8>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6aeb
cc828>
```

```
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6b74
a4400>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6b74
a4048>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d0940>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d0dd8>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d00f0>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d0a90>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6b8d
a6208>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6b8d
a6550>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d0a20>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6ae7
d0cc0>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
INFO:fbprophet:Applying in parallel with <concurrent.futures.process.ProcessPoolExecutor object at 0x7ff6b8d
a6a58>
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
```

```
    changepoint_prior_scale  seasonality_prior_scale      mape
0                     0.001                     0.01  4.248742
1                     0.001                     0.10  4.248742
2                     0.001                     1.00  4.248742
3                     0.001                    10.00  4.248742
4                     0.010                     0.01  4.248742
5                     0.010                     0.10  4.248742
6                     0.010                     1.00  4.248742
7                     0.010                    10.00  4.248742
8                     0.100                     0.01  4.248742
9                     0.100                     0.10  4.248742
10                    0.100                     1.00  4.248742
11                    0.100                    10.00  4.248742
12                    0.500                     0.01  4.248742
13                    0.500                     0.10  4.248742
14                    0.500                     1.00  4.248742
15                    0.500                    10.00  4.248742
```

In [23]:

```
tuning_results['mape'] = mape
print(tuning_results)
best_params = all_params[np.argmin(mape)]
print(best_params)
```

```
    changepoint_prior_scale  seasonality_prior_scale      mape
0                     0.001                     0.01  4.248742
1                     0.001                     0.10  4.248742
2                     0.001                     1.00  4.248742
3                     0.001                    10.00  4.248742
4                     0.010                     0.01  4.248742
5                     0.010                     0.10  4.248742
6                     0.010                     1.00  4.248742
7                     0.010                    10.00  4.248742
8                     0.100                     0.01  4.248742
9                     0.100                     0.10  4.248742
10                    0.100                     1.00  4.248742
11                    0.100                    10.00  4.248742
12                    0.500                     0.01  4.248742
13                    0.500                     0.10  4.248742
14                    0.500                     1.00  4.248742
15                    0.500                    10.00  4.248742
{'changepoint_prior_scale': 0.001, 'seasonality_prior_scale': 0.01}
```

In [24]:

```
m = Prophet(changepoint_prior_scale=0.001, seasonality_prior_scale=0.01)
for i in data_train.columns:
  if i not in ['ds','y']:
    m.add_regressor(i)
m.fit(data_train)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

Out[24]:

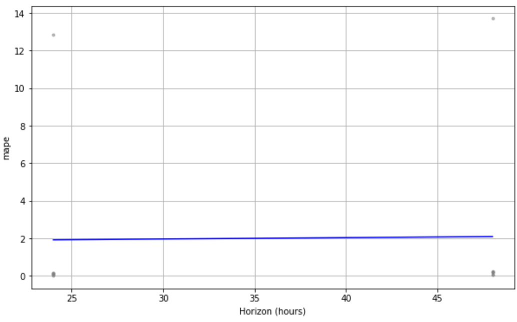
```
<fbprophet.forecaster.Prophet at 0x7ff6ae398dd8>
```

In [25]:

```
cutoffs = pd.to_datetime(['2013-01-10', '2014-01-09', '2015-01-09','2016-01-09','2017-01-09','2018-01-09','2
df_cv = cross_validation(m, cutoffs=cutoffs, horizon='2 days')
df_p = performance_metrics(df_cv)
df_p.head()
fig = plot_cross_validation_metric(df_cv, metric='mape')
```

```
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than initial window. Consider increa
sing initial.
```

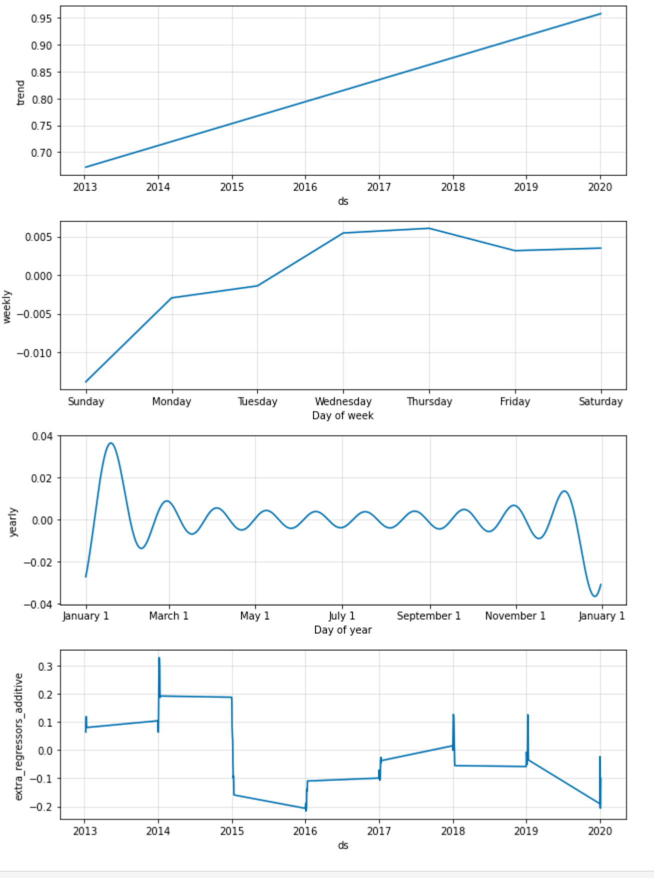
```
INFO:fbprophet:n_changepoints greater than number of observations. Using 2.
INFO:fbprophet:n_changepoints greater than number of observations. Using 11.
INFO:fbprophet:n_changepoints greater than number of observations. Using 20.
```

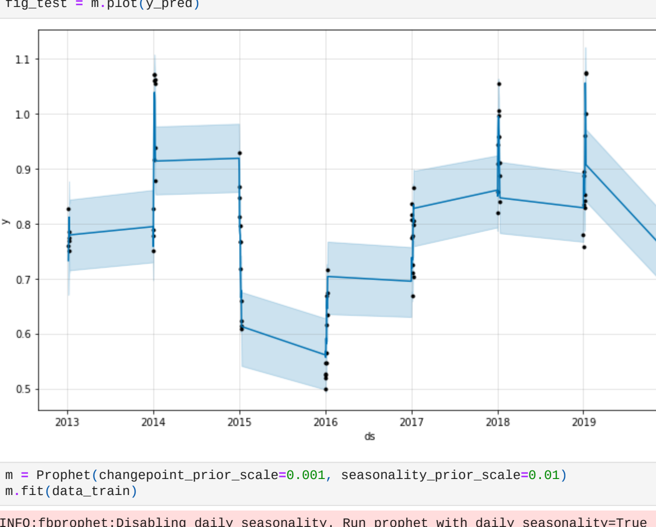


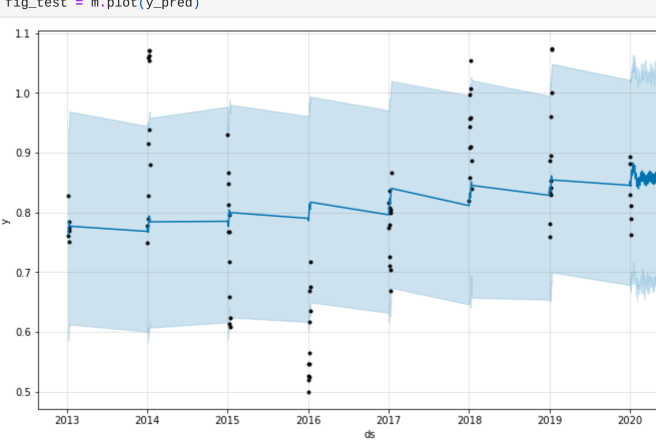
In [26]:

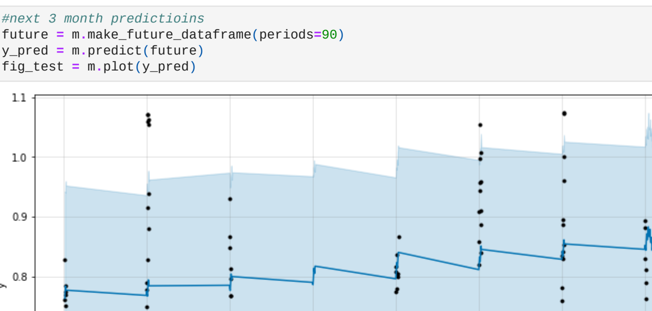
```
y_pred = m.predict(data_train)
print("MAPE Mean Absolute Percentage Error:", mean_absolute_percentage_error(data_train['y'], np.array(y_pre
print("Directional Symmetry Statistic:", directional_symmetry(data_train['y'], np.array(y_pred[['yhat']]).re
fig2 = m.plot_components(y_pred)
```

```
MAPE Mean Absolute Percentage Error: 5.173354120322739
Directional Symmetry Statistic: 60.24096385542169
```

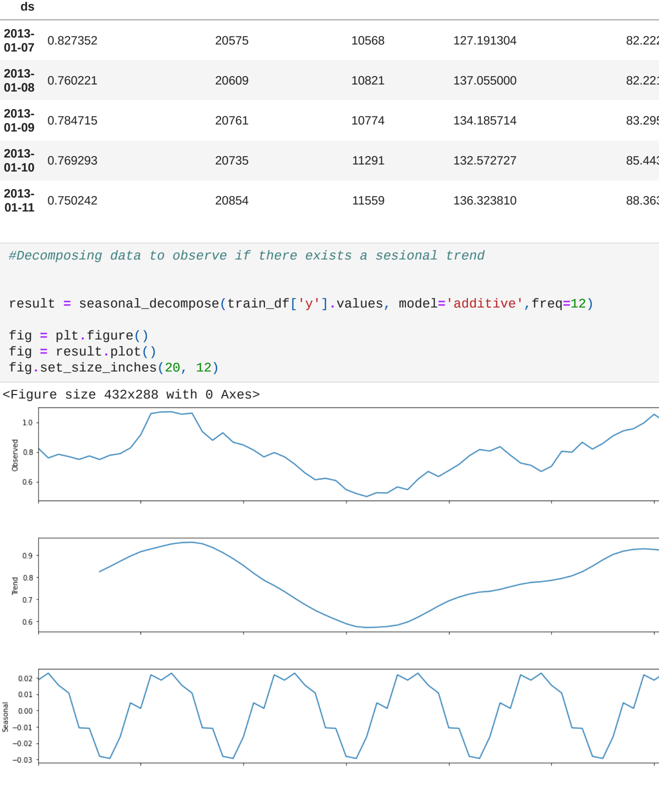


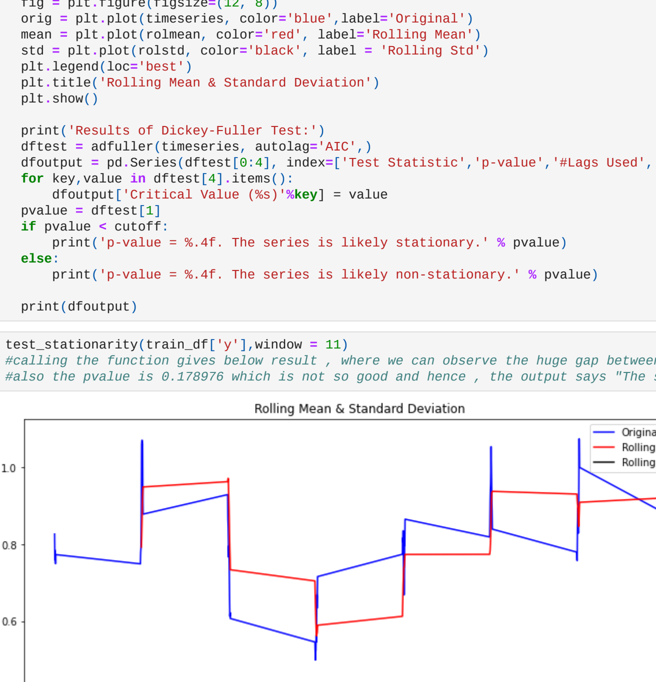


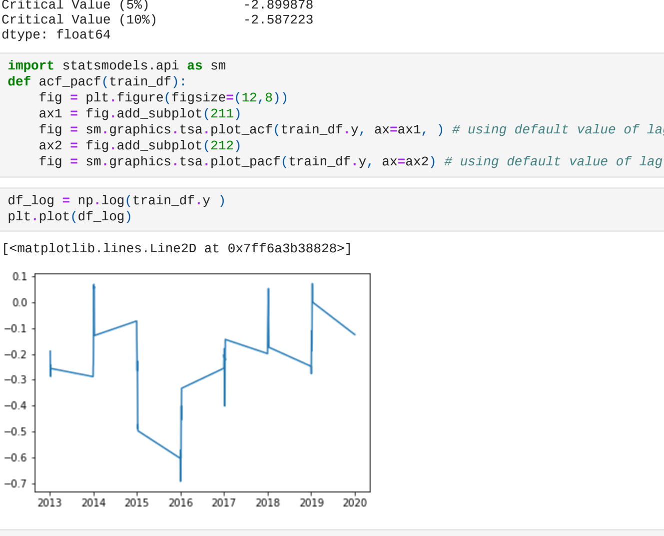




## Statistical Model2: SARIMA

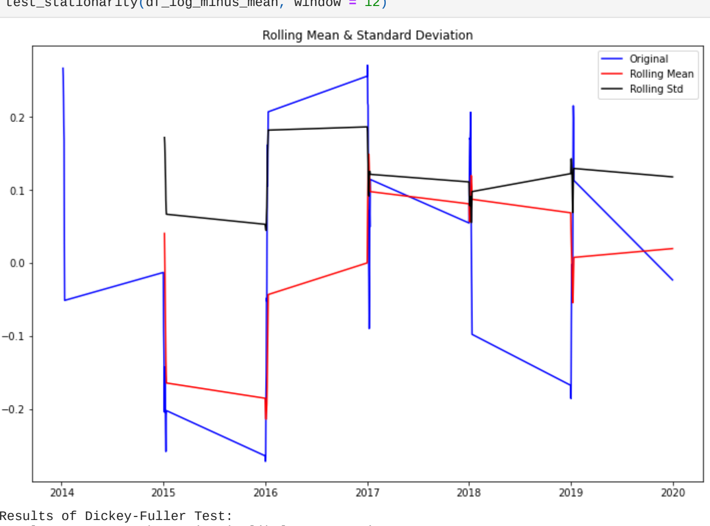






In [38]:
```
rolling_mean = df_log.rolling(window=12).mean()
df_log_minus_mean = df_log - rolling_mean
df_log_minus_mean.dropna(inplace=True)
```

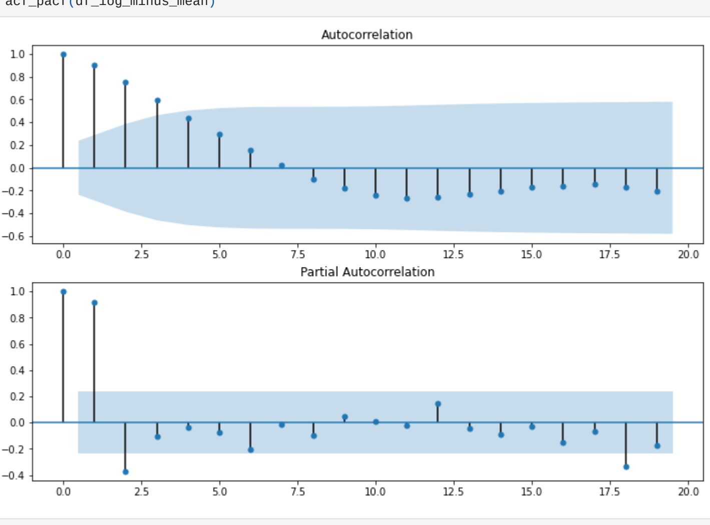
In [39]:
```
test_stationarity(df_log_minus_mean, window = 12)
```



```
Results of Dickey-Fuller Test:
p-value = 0.0670. The series is likely non-stationary.
Test Statistic                  -2.742101
p-value                          0.067032
#Lags Used                       1.000000
Number of Observations Used     66.000000
Critical Value (1%)             -3.533560
Critical Value (5%)             -2.906444
Critical Value (10%)            -2.590724
dtype: float64
```

In [40]:
```
df_log_minus_mean = pd.DataFrame({'ds':df_log_minus_mean.index, 'y':df_log_minus_mean.values})
```

In [41]:
```
acf_pacf(df_log_minus_mean)
```

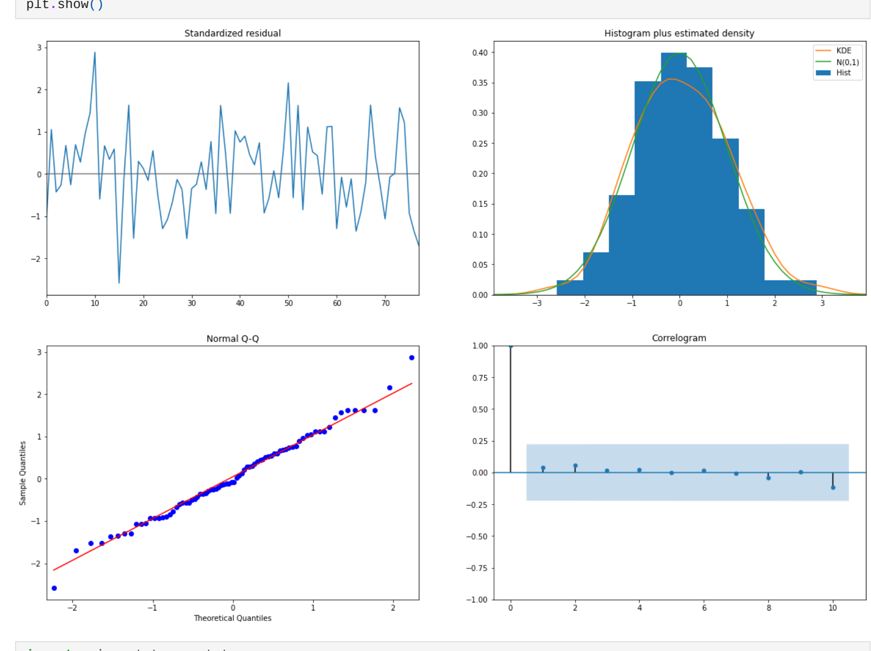


In [42]:
```
sarimax_mod = sm.tsa.statespace.SARIMAX(train_df.y, order=(11, 1, 0), seasonal_order=(0, 0, 0, 0), trend='ct
print(sarimax_mod.summary())
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWarning:

A date index has been provided, but it has no associated frequency information and so will be ignored when
e.g. forecasting.
```

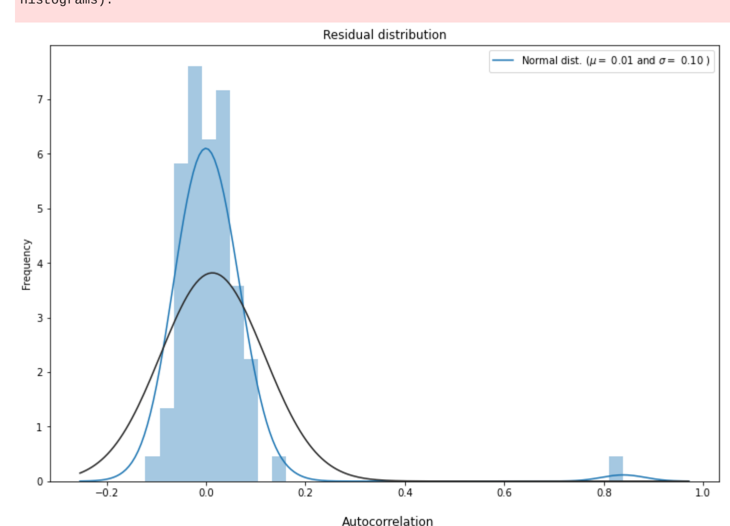
```
                           Statespace Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   79
Model:               SARIMAX(11, 1, 0)   Log Likelihood                 128.031
Date:                Sun, 14 Feb 2021   AIC                           -228.061
Time:                        02:28:30   BIC                           -195.067
Sample:                             0   HQIC                          -214.853
                                 - 79
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0146      0.014     -1.011      0.312      -0.043       0.014
drift          0.0003      0.000      1.029      0.304      -0.000       0.001
ar.L1          0.3044      0.150      2.029      0.042       0.010       0.598
ar.L2         -0.0646      0.157     -0.411      0.681      -0.372       0.243
ar.L3         -0.0917      0.146     -0.628      0.530      -0.378       0.195
ar.L4         -0.1358      0.164     -0.826      0.409      -0.458       0.186
ar.L5          0.1298      0.159      0.818      0.413      -0.181       0.441
ar.L6         -0.0753      0.181     -0.417      0.677      -0.429       0.279
ar.L7         -0.0038      0.135     -0.028      0.977      -0.269       0.261
ar.L8          0.0257      0.148      0.173      0.862      -0.265       0.316
ar.L9         -0.1338      0.149     -0.897      0.369      -0.426       0.158
ar.L10        -0.0219      0.169     -0.130      0.897      -0.353       0.309
ar.L11        -0.1503      0.181     -0.831      0.406      -0.505       0.204
sigma2         0.0022      0.001      4.366      0.000       0.001       0.003
===================================================================================
Ljung-Box (Q):                       27.21   Jarque-Bera (JB):                 0.32
Prob(Q):                              0.94   Prob(JB):                         0.85
Heteroskedasticity (H):               0.84   Skew:                             0.16
Prob(H) (two-sided):                  0.66   Kurtosis:                         3.02
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

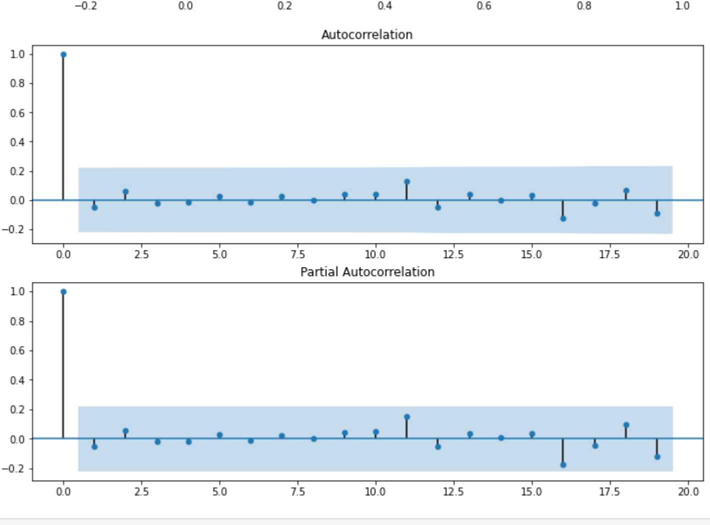
In [43]:
```
sarimax_mod.plot_diagnostics(figsize=(20, 14))
plt.show()
```



In [44]:
```
import scipy.stats as stats
import seaborn as sns # informative statistical graphics.
import statsmodels.api as sm #for ARIMA and SARIMAX
resid = sarimax_mod.resid #gives residual degree of freedom (mu, sigma, pvalue ... )

fig = plt.figure(figsize=(12,8))
ax0 = fig.add_subplot(111)

sns.distplot(resid ,fit = stats.norm, ax = ax0) # need to import scipy.stats

# Get the fitted parameters used by the function
(mu, sigma) = stats.norm.fit(resid)

#Now plot the distribution using
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')
plt.title('Residual distribution')


# ACF and PACF
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(sarimax_mod.resid, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(sarimax_mod.resid, ax=ax2)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use e
ither `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for
histograms).
```





In [45]:
```
future_predict=sarimax_mod.forecast(steps=5)
future_predict=pd.DataFrame(future_predict)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarning:

No supported index is available. Prediction results will be given with an integer index beginning at `start
`.
```

In [46]:
```
data_test
```

Out[46]:

| | ds | y | Steel_Inventory_M_USD | Steel_Orders_M_USD | IronOre_Global_USD | CoalAustralia_Global_USD | Nickel_Global_USD | Z |
|---|---|---|---|---|---|---|---|---|
| **79** | 2020-01-02 | 0.893123 | 19753 | 9914 | 87.625000 | 71.050179 | 12715.55000 | |
| **80** | 2020-01-03 | 0.829167 | 19945 | 9072 | 88.659091 | 70.204870 | 11846.22727 | |
| **81** | 2020-01-04 | 0.789704 | 19824 | 7800 | 83.750000 | 64.620000 | 11804.00950 | |
| **82** | 2020-01-05 | 0.762035 | 19441 | 8908 | 91.333333 | 54.148308 | 12179.60526 | |
| **83** | 2020-01-06 | 0.811477 | 18998 | 8979 | 103.340909 | 55.002760 | 12727.15227 | |

In [47]:
```
#next 6 month prediction
sarimax_mod.predict(3,180)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:576: ValueWarning:

No supported index is available. Prediction results will be given with an integer index beginning at `start
`.
```

Out[47]:
```
3      0.790306
4      0.762912
5      0.740640
6      0.762103
7      0.744721
         ...
176    2.979352
177    3.013326
178    3.047558
179    3.082046
180    3.116790
Length: 178, dtype: float64
```

In [48]:
```
future_predict['ds']=data_test.reset_index()['ds'].values
future_predict = future_predict.set_index('ds')
data_test = data_test.set_index('ds')
```

In [49]:
```
data_test.reset_index()['ds'].values
```

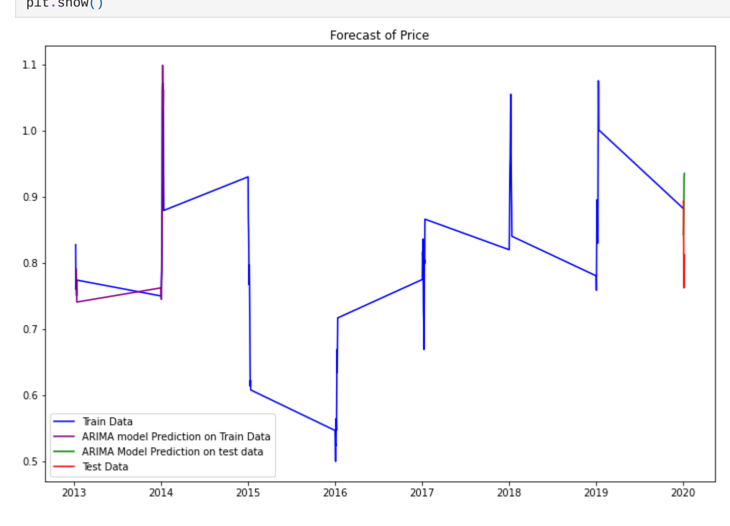
Out[49]:
```
array([datetime.date(2020, 1, 2), datetime.date(2020, 1, 3),
       datetime.date(2020, 1, 4), datetime.date(2020, 1, 5),
       datetime.date(2020, 1, 6)], dtype=object)
```

In [49]:

In [50]:
```
print("MAPE Mean Absolute Percentage Error:", mean_absolute_percentage_error(data_test['y'], future_predict[
print("Directional Symmetry Statistic:", directional_symmetry(data_test['y'], future_predict[0]))
```

```
MAPE Mean Absolute Percentage Error: 11.996464470656324
Directional Symmetry Statistic: 25.0
```

In [51]:
```
#6 month forcasting
figg = plt.figure(figsize=(12, 8))
orig = plt.plot(train_df[['y']], color='blue',label='Train Data')
pred_train=plt.plot(sarimax_mod.predict(3,16),color='purple',label='ARIMA model Prediction on Train Data')
fore = plt.plot(future_predict, color='green', label='ARIMA Model Prediction on test data')
fes = plt.plot(data_test[['y']], color='red', label='Test Data')
plt.legend(loc='best')
plt.title('Forecast of Price')
plt.show()
```



## Deep learning Model : LSTM

In [52]:
```
# Stacked LSTM for price forcasting with memory
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back),1:]
                dataX.append(a)
                dataY.append(dataset[i + look_back,0])
        return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [53]:
```
X
```

Out[53]:

| | ds | y | Steel_Inventory_M_USD | Steel_Orders_M_USD | IronOre_Global_USD | CoalAustralia_Global_USD | Nickel_Global_USD | Z |
|---|---|---|---|---|---|---|---|---|
| **0** | 2013-01-07 | 0.827352 | 20575 | 10568 | 127.191304 | 82.222360 | 13750.31522 | |
| **1** | 2013-01-08 | 0.760221 | 20609 | 10821 | 137.055000 | 82.221429 | 14308.26190 | |
| **2** | 2013-01-09 | 0.784715 | 20761 | 10774 | 134.185714 | 83.295918 | 13801.39286 | |
| **3** | 2013-01-10 | 0.769293 | 20735 | 11291 | 132.572727 | 85.443634 | 14117.65217 | |
| **4** | 2013-01-11 | 0.750242 | 20854 | 11559 | 136.323810 | 88.363265 | 13684.01190 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **79** | 2020-01-02 | 0.893123 | 19753 | 9914 | 87.625000 | 71.050179 | 12715.55000 | |
| **80** | 2020-01-03 | 0.829167 | 19945 | 9072 | 88.659091 | 70.204870 | 11846.22727 | |
| **81** | 2020-01-04 | 0.789704 | 19824 | 7800 | 83.750000 | 64.620000 | 11804.00950 | |
| **82** | 2020-01-05 | 0.762035 | 19441 | 8908 | 91.333333 | 54.148308 | 12179.60526 | |
| **83** | 2020-01-06 | 0.811477 | 18998 | 8979 | 103.340909 | 55.002760 | 12727.15227 | |

84 rows × 15 columns

In [54]:
```
dataset=X.copy()
```

In [55]:
```
dataset = dataset.set_index('ds')
```

In [56]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset.drop(['y'],axis=1))
y=np.array(X['y'])
dataset=np.hstack((y.reshape(y.shape[0],1),dataset))
#split into train and test sets
train_size = int(len(dataset) * 0.90)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

In [57]:
```
# reshape into X=t and Y=t+1
look_back = 1
X_train, y_train= create_dataset(train, look_back)
X_test, y_test = create_dataset(test, look_back)
```

In [58]:
```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[2],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[2],1)
X_tr_t=X_train.copy()
X_tst_t=X_test.copy()
```

In [59]:
```
batch_size = 1
look_back=13
#model = Sequential()
#model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
#model.add(Dense(1))

#Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (X_tr_t.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(loss='mean_absolute_percentage_error', optimizer='adam')
model.fit(X_tr_t , y_train, epochs=1000, batch_size=batch_size, verbose=2, shuffle=False)
```

```
Epoch 1/1000
73/73 - 3s - loss: 16.6686
Epoch 2/1000
73/73 - 1s - loss: 11.1544
Epoch 3/1000
73/73 - 1s - loss: 16.3745
Epoch 4/1000
73/73 - 1s - loss: 15.9255
Epoch 5/1000
73/73 - 1s - loss: 14.8324
Epoch 6/1000
73/73 - 1s - loss: 13.5205
Epoch 7/1000
73/73 - 1s - loss: 14.8882
Epoch 8/1000
73/73 - 1s - loss: 16.9860
Epoch 9/1000
73/73 - 1s - loss: 16.0892
Epoch 10/1000
73/73 - 1s - loss: 16.2650
Epoch 11/1000
73/73 - 1s - loss: 16.0422
Epoch 12/1000
73/73 - 1s - loss: 15.4847
Epoch 13/1000
73/73 - 1s - loss: 15.5258
Epoch 14/1000
73/73 - 1s - loss: 15.6620
Epoch 15/1000
73/73 - 1s - loss: 15.3062
Epoch 16/1000
73/73 - 1s - loss: 14.8814
Epoch 17/1000
73/73 - 1s - loss: 14.9517
Epoch 18/1000
73/73 - 1s - loss: 14.4485
Epoch 19/1000
73/73 - 1s - loss: 14.3254
Epoch 20/1000
73/73 - 1s - loss: 13.4223
Epoch 21/1000
73/73 - 1s - loss: 14.2308
Epoch 22/1000
73/73 - 1s - loss: 11.9834
Epoch 23/1000
73/73 - 1s - loss: 13.2315
Epoch 24/1000
73/73 - 1s - loss: 11.7251
Epoch 25/1000
73/73 - 1s - loss: 13.1293
Epoch 26/1000
73/73 - 1s - loss: 11.9479
Epoch 27/1000
73/73 - 1s - loss: 12.8901
Epoch 28/1000
73/73 - 1s - loss: 10.6560
Epoch 29/1000
73/73 - 1s - loss: 10.5008
Epoch 30/1000
73/73 - 1s - loss: 10.2042
Epoch 31/1000
73/73 - 1s - loss: 10.4297
Epoch 32/1000
73/73 - 1s - loss: 11.7264
Epoch 33/1000
73/73 - 1s - loss: 11.5426
Epoch 34/1000
73/73 - 1s - loss: 14.9979
Epoch 35/1000
73/73 - 1s - loss: 9.4609
Epoch 36/1000
73/73 - 1s - loss: 10.1446
Epoch 37/1000
73/73 - 1s - loss: 11.7384
Epoch 38/1000
73/73 - 1s - loss: 9.6023
Epoch 39/1000
73/73 - 1s - loss: 9.0791
Epoch 40/1000
73/73 - 1s - loss: 9.4246
Epoch 41/1000
73/73 - 1s - loss: 10.8249
Epoch 42/1000
73/73 - 1s - loss: 15.3493
Epoch 43/1000
73/73 - 1s - loss: 10.4777
Epoch 44/1000
73/73 - 1s - loss: 9.2577
Epoch 45/1000
73/73 - 1s - loss: 10.3619
Epoch 46/1000
73/73 - 1s - loss: 9.7255
Epoch 47/1000
73/73 - 1s - loss: 12.4860
Epoch 48/1000
73/73 - 1s - loss: 10.1867
Epoch 49/1000
```

```
73/73 - 1s - loss: 9.1308
Epoch 50/1000
73/73 - 1s - loss: 8.8691
Epoch 51/1000
73/73 - 1s - loss: 10.0651
Epoch 52/1000
73/73 - 1s - loss: 10.6717
Epoch 53/1000
73/73 - 1s - loss: 10.4536
Epoch 54/1000
73/73 - 1s - loss: 11.5227
Epoch 55/1000
73/73 - 1s - loss: 8.8495
Epoch 56/1000
73/73 - 1s - loss: 9.2793
Epoch 57/1000
73/73 - 1s - loss: 9.5151
Epoch 58/1000
73/73 - 1s - loss: 9.1458
Epoch 59/1000
73/73 - 1s - loss: 8.9534
Epoch 60/1000
73/73 - 1s - loss: 9.8947
Epoch 61/1000
73/73 - 1s - loss: 10.5504
Epoch 62/1000
73/73 - 1s - loss: 10.6453
Epoch 63/1000
73/73 - 1s - loss: 8.9155
Epoch 64/1000
73/73 - 1s - loss: 9.9582
Epoch 65/1000
73/73 - 1s - loss: 8.5770
Epoch 66/1000
73/73 - 1s - loss: 9.2922
Epoch 67/1000
73/73 - 1s - loss: 8.9372
Epoch 68/1000
73/73 - 1s - loss: 10.3288
Epoch 69/1000
73/73 - 1s - loss: 8.8458
Epoch 70/1000
73/73 - 1s - loss: 9.1335
Epoch 71/1000
73/73 - 1s - loss: 9.3744
Epoch 72/1000
73/73 - 1s - loss: 8.9494
Epoch 73/1000
73/73 - 1s - loss: 8.8578
Epoch 74/1000
73/73 - 1s - loss: 8.4887
Epoch 75/1000
73/73 - 1s - loss: 10.2073
Epoch 76/1000
73/73 - 1s - loss: 10.5445
Epoch 77/1000
73/73 - 0s - loss: 10.7190
Epoch 78/1000
73/73 - 1s - loss: 9.9779
Epoch 79/1000
73/73 - 1s - loss: 11.1172
Epoch 80/1000
73/73 - 1s - loss: 8.8493
Epoch 81/1000
73/73 - 1s - loss: 9.6659
Epoch 82/1000
73/73 - 1s - loss: 8.9881
Epoch 83/1000
73/73 - 1s - loss: 7.6855
Epoch 84/1000
73/73 - 1s - loss: 8.9661
Epoch 85/1000
73/73 - 1s - loss: 6.4628
Epoch 86/1000
73/73 - 1s - loss: 9.2485
Epoch 87/1000
73/73 - 1s - loss: 8.1486
Epoch 88/1000
73/73 - 1s - loss: 8.0399
Epoch 89/1000
73/73 - 1s - loss: 8.1434
Epoch 90/1000
73/73 - 1s - loss: 8.7628
Epoch 91/1000
73/73 - 1s - loss: 8.8168
Epoch 92/1000
73/73 - 1s - loss: 6.3712
Epoch 93/1000
73/73 - 1s - loss: 9.0326
Epoch 94/1000
73/73 - 1s - loss: 6.9213
Epoch 95/1000
73/73 - 1s - loss: 10.5765
Epoch 96/1000
73/73 - 1s - loss: 9.0353
Epoch 97/1000
73/73 - 1s - loss: 8.0682
Epoch 98/1000
73/73 - 1s - loss: 8.1449
Epoch 99/1000
73/73 - 1s - loss: 11.3947
Epoch 100/1000
73/73 - 0s - loss: 9.5798
Epoch 101/1000
73/73 - 1s - loss: 10.5491
Epoch 102/1000
73/73 - 1s - loss: 9.7190
Epoch 103/1000
73/73 - 1s - loss: 9.0289
Epoch 104/1000
73/73 - 1s - loss: 8.9240
Epoch 105/1000
73/73 - 1s - loss: 8.1087
Epoch 106/1000
73/73 - 1s - loss: 7.9990
Epoch 107/1000
73/73 - 1s - loss: 12.2508
Epoch 108/1000
73/73 - 0s - loss: 10.3048
Epoch 109/1000
73/73 - 1s - loss: 8.7180
Epoch 110/1000
73/73 - 1s - loss: 8.4087
Epoch 111/1000
73/73 - 1s - loss: 7.5814
Epoch 112/1000
73/73 - 1s - loss: 10.2637
Epoch 113/1000
73/73 - 1s - loss: 7.1034
Epoch 114/1000
73/73 - 1s - loss: 6.9976
Epoch 115/1000
73/73 - 1s - loss: 8.7595
Epoch 116/1000
73/73 - 1s - loss: 9.2474
Epoch 117/1000
73/73 - 1s - loss: 7.7509
Epoch 118/1000
73/73 - 1s - loss: 7.1288
Epoch 119/1000
73/73 - 1s - loss: 8.9633
Epoch 120/1000
73/73 - 0s - loss: 7.8855
Epoch 121/1000
73/73 - 1s - loss: 6.7821
Epoch 122/1000
73/73 - 1s - loss: 7.1722
Epoch 123/1000
73/73 - 1s - loss: 6.9314
Epoch 124/1000
73/73 - 1s - loss: 7.1461
Epoch 125/1000
73/73 - 1s - loss: 6.3548
Epoch 126/1000
73/73 - 1s - loss: 8.2978
Epoch 127/1000
73/73 - 1s - loss: 8.1452
Epoch 128/1000
73/73 - 1s - loss: 6.7837
Epoch 129/1000
73/73 - 1s - loss: 10.1932
Epoch 130/1000
73/73 - 1s - loss: 7.4674
Epoch 131/1000
73/73 - 1s - loss: 7.0240
Epoch 132/1000
73/73 - 1s - loss: 7.5148
Epoch 133/1000
73/73 - 1s - loss: 9.0253
Epoch 134/1000
73/73 - 1s - loss: 8.8212
Epoch 135/1000
73/73 - 1s - loss: 9.0542
Epoch 136/1000
73/73 - 1s - loss: 7.0230
Epoch 137/1000
73/73 - 1s - loss: 7.3647
Epoch 138/1000
73/73 - 1s - loss: 7.9580
Epoch 139/1000
73/73 - 1s - loss: 5.6407
Epoch 140/1000
73/73 - 1s - loss: 6.1776
Epoch 141/1000
73/73 - 1s - loss: 6.4797
Epoch 142/1000
73/73 - 1s - loss: 7.9954
Epoch 143/1000
73/73 - 1s - loss: 7.5180
Epoch 144/1000
73/73 - 1s - loss: 7.0060
Epoch 145/1000
73/73 - 1s - loss: 7.6306
Epoch 146/1000
73/73 - 1s - loss: 8.0494
Epoch 147/1000
73/73 - 1s - loss: 6.6793
Epoch 148/1000
73/73 - 1s - loss: 6.2642
Epoch 149/1000
73/73 - 1s - loss: 5.6675
Epoch 150/1000
73/73 - 1s - loss: 5.5136
Epoch 151/1000
73/73 - 1s - loss: 5.5780
Epoch 152/1000
73/73 - 1s - loss: 6.0916
Epoch 153/1000
73/73 - 1s - loss: 6.0337
Epoch 154/1000
73/73 - 1s - loss: 6.4261
Epoch 155/1000
73/73 - 1s - loss: 6.4254
Epoch 156/1000
73/73 - 1s - loss: 6.0397
Epoch 157/1000
73/73 - 1s - loss: 6.6465
Epoch 158/1000
73/73 - 1s - loss: 5.4589
Epoch 159/1000
73/73 - 1s - loss: 5.5915
Epoch 160/1000
73/73 - 1s - loss: 5.9167
Epoch 161/1000
73/73 - 1s - loss: 5.4057
Epoch 162/1000
73/73 - 1s - loss: 6.0261
Epoch 163/1000
73/73 - 1s - loss: 6.1452
Epoch 164/1000
73/73 - 1s - loss: 6.2647
Epoch 165/1000
73/73 - 1s - loss: 7.3043
Epoch 166/1000
73/73 - 1s - loss: 5.6175
Epoch 167/1000
73/73 - 1s - loss: 6.9792
Epoch 168/1000
73/73 - 1s - loss: 4.0903
Epoch 169/1000
73/73 - 1s - loss: 5.4812
Epoch 170/1000
73/73 - 1s - loss: 4.6615
Epoch 171/1000
73/73 - 1s - loss: 6.0311
Epoch 172/1000
73/73 - 1s - loss: 5.3281
Epoch 173/1000
73/73 - 1s - loss: 5.6826
Epoch 174/1000
73/73 - 1s - loss: 6.6753
Epoch 175/1000
73/73 - 1s - loss: 5.2468
Epoch 176/1000
73/73 - 1s - loss: 5.1230
Epoch 177/1000
73/73 - 1s - loss: 6.0360
Epoch 178/1000
73/73 - 1s - loss: 5.0044
Epoch 179/1000
73/73 - 1s - loss: 4.9592
Epoch 180/1000
73/73 - 1s - loss: 4.9567
Epoch 181/1000
73/73 - 1s - loss: 5.1814
Epoch 182/1000
73/73 - 1s - loss: 4.7949
Epoch 183/1000
73/73 - 1s - loss: 5.5160
Epoch 184/1000
73/73 - 1s - loss: 5.6417
Epoch 185/1000
73/73 - 1s - loss: 6.1195
Epoch 186/1000
73/73 - 1s - loss: 5.0800
Epoch 187/1000
73/73 - 1s - loss: 4.1390
Epoch 188/1000
73/73 - 1s - loss: 6.0706
Epoch 189/1000
73/73 - 1s - loss: 5.3510
Epoch 190/1000
73/73 - 1s - loss: 5.0459
Epoch 191/1000
73/73 - 1s - loss: 4.6939
Epoch 192/1000
73/73 - 1s - loss: 5.5062
Epoch 193/1000
73/73 - 1s - loss: 4.9504
Epoch 194/1000
73/73 - 1s - loss: 4.3536
Epoch 195/1000
73/73 - 1s - loss: 4.7736
Epoch 196/1000
73/73 - 1s - loss: 4.9094
Epoch 197/1000
73/73 - 1s - loss: 4.9917
Epoch 198/1000
73/73 - 1s - loss: 4.7891
Epoch 199/1000
73/73 - 1s - loss: 6.4429
Epoch 200/1000
73/73 - 1s - loss: 4.0092
Epoch 201/1000
73/73 - 1s - loss: 4.4292
Epoch 202/1000
73/73 - 1s - loss: 3.6525
Epoch 203/1000
73/73 - 1s - loss: 5.0039
Epoch 204/1000
73/73 - 1s - loss: 4.6706
Epoch 205/1000
73/73 - 1s - loss: 5.5065
Epoch 206/1000
73/73 - 1s - loss: 4.5150
Epoch 207/1000
73/73 - 1s - loss: 5.6358
Epoch 208/1000
73/73 - 1s - loss: 5.5127
Epoch 209/1000
73/73 - 1s - loss: 4.3454
Epoch 210/1000
73/73 - 1s - loss: 4.3184
Epoch 211/1000
73/73 - 1s - loss: 5.3633
Epoch 212/1000
73/73 - 1s - loss: 4.6936
Epoch 213/1000
73/73 - 1s - loss: 4.3023
Epoch 214/1000
73/73 - 1s - loss: 6.5328
Epoch 215/1000
73/73 - 1s - loss: 4.2415
Epoch 216/1000
73/73 - 1s - loss: 4.5099
Epoch 217/1000
73/73 - 1s - loss: 5.0759
Epoch 218/1000
73/73 - 1s - loss: 4.0647
Epoch 219/1000
73/73 - 1s - loss: 4.4033
Epoch 220/1000
73/73 - 1s - loss: 4.8539
Epoch 221/1000
73/73 - 1s - loss: 5.3386
Epoch 222/1000
73/73 - 1s - loss: 6.0118
Epoch 223/1000
73/73 - 1s - loss: 5.0090
Epoch 224/1000
73/73 - 1s - loss: 4.2309
Epoch 225/1000
73/73 - 1s - loss: 4.3016
Epoch 226/1000
73/73 - 1s - loss: 3.9467
Epoch 227/1000
73/73 - 1s - loss: 4.1843
Epoch 228/1000
73/73 - 1s - loss: 4.7292
Epoch 229/1000
73/73 - 1s - loss: 4.5265
Epoch 230/1000
73/73 - 1s - loss: 3.2054
Epoch 231/1000
73/73 - 1s - loss: 4.3324
Epoch 232/1000
73/73 - 1s - loss: 4.9566
Epoch 233/1000
73/73 - 1s - loss: 3.7968
Epoch 234/1000
73/73 - 1s - loss: 4.6773
Epoch 235/1000
73/73 - 1s - loss: 7.3329
Epoch 236/1000
73/73 - 1s - loss: 4.3270
Epoch 237/1000
73/73 - 1s - loss: 4.3825
Epoch 238/1000
73/73 - 1s - loss: 4.7178
Epoch 239/1000
73/73 - 1s - loss: 4.0409
Epoch 240/1000
73/73 - 1s - loss: 4.1218
Epoch 241/1000
73/73 - 1s - loss: 4.6452
Epoch 242/1000
73/73 - 1s - loss: 4.5249
Epoch 243/1000
73/73 - 1s - loss: 4.4292
Epoch 244/1000
73/73 - 1s - loss: 4.4796
Epoch 245/1000
73/73 - 1s - loss: 6.2057
Epoch 246/1000
73/73 - 1s - loss: 5.0482
Epoch 247/1000
73/73 - 1s - loss: 4.2194
Epoch 248/1000
73/73 - 1s - loss: 4.1436
Epoch 249/1000
73/73 - 1s - loss: 4.5430
Epoch 250/1000
73/73 - 1s - loss: 4.3373
Epoch 251/1000
73/73 - 1s - loss: 4.0452
Epoch 252/1000
73/73 - 1s - loss: 4.4716
Epoch 253/1000
73/73 - 1s - loss: 4.0509
Epoch 254/1000
73/73 - 1s - loss: 5.9331
Epoch 255/1000
73/73 - 1s - loss: 4.0590
Epoch 256/1000
73/73 - 1s - loss: 4.2143
Epoch 257/1000
73/73 - 1s - loss: 4.3340
Epoch 258/1000
73/73 - 1s - loss: 3.7505
Epoch 259/1000
73/73 - 1s - loss: 3.6794
Epoch 260/1000
73/73 - 1s - loss: 4.3865
Epoch 261/1000
73/73 - 1s - loss: 3.8349
Epoch 262/1000
73/73 - 1s - loss: 4.0720
Epoch 263/1000
73/73 - 1s - loss: 3.8623
Epoch 264/1000
73/73 - 1s - loss: 3.9303
Epoch 265/1000
73/73 - 1s - loss: 4.4718
Epoch 266/1000
73/73 - 1s - loss: 4.5165
Epoch 267/1000
73/73 - 1s - loss: 4.2990
Epoch 268/1000
73/73 - 1s - loss: 4.2791
Epoch 269/1000
73/73 - 1s - loss: 4.5645
Epoch 270/1000
73/73 - 1s - loss: 4.5841
Epoch 271/1000
73/73 - 1s - loss: 4.2782
Epoch 272/1000
73/73 - 1s - loss: 4.1347
Epoch 273/1000
73/73 - 1s - loss: 4.6087
Epoch 274/1000
73/73 - 1s - loss: 4.1012
Epoch 275/1000
73/73 - 1s - loss: 4.6552
Epoch 276/1000
73/73 - 1s - loss: 4.8587
Epoch 277/1000
73/73 - 1s - loss: 4.8918
Epoch 278/1000
73/73 - 1s - loss: 4.6861
Epoch 279/1000
73/73 - 1s - loss: 4.0640
Epoch 280/1000
73/73 - 1s - loss: 5.7298
Epoch 281/1000
73/73 - 1s - loss: 7.7972
Epoch 282/1000
73/73 - 1s - loss: 3.9075
Epoch 283/1000
73/73 - 1s - loss: 3.2185
Epoch 284/1000
73/73 - 1s - loss: 3.3185
Epoch 285/1000
73/73 - 1s - loss: 3.5942
Epoch 286/1000
73/73 - 1s - loss: 3.6824
Epoch 287/1000
73/73 - 1s - loss: 4.1851
Epoch 288/1000
73/73 - 1s - loss: 3.6090
Epoch 289/1000
73/73 - 1s - loss: 4.1873
Epoch 290/1000
73/73 - 1s - loss: 3.5757
Epoch 291/1000
73/73 - 1s - loss: 3.9410
Epoch 292/1000
73/73 - 1s - loss: 3.9550
Epoch 293/1000
73/73 - 1s - loss: 4.1144
Epoch 294/1000
73/73 - 1s - loss: 4.6876
Epoch 295/1000
73/73 - 1s - loss: 4.0243
Epoch 296/1000
73/73 - 1s - loss: 4.1181
Epoch 297/1000
73/73 - 1s - loss: 5.1588
Epoch 298/1000
73/73 - 1s - loss: 4.5370
Epoch 299/1000
73/73 - 1s - loss: 4.0272
Epoch 300/1000
73/73 - 1s - loss: 3.1719
Epoch 301/1000
73/73 - 1s - loss: 3.5629
Epoch 302/1000
73/73 - 1s - loss: 3.7293
Epoch 303/1000
73/73 - 1s - loss: 4.7323
Epoch 304/1000
73/73 - 1s - loss: 3.5493
Epoch 305/1000
73/73 - 1s - loss: 3.6346
Epoch 306/1000
73/73 - 1s - loss: 3.1490
Epoch 307/1000
73/73 - 1s - loss: 3.3474
Epoch 308/1000
73/73 - 1s - loss: 3.8644
Epoch 309/1000
73/73 - 1s - loss: 3.0451
Epoch 310/1000
73/73 - 1s - loss: 3.5695
Epoch 311/1000
73/73 - 1s - loss: 3.6133
Epoch 312/1000
73/73 - 1s - loss: 3.8244
Epoch 313/1000
73/73 - 1s - loss: 3.5181
Epoch 314/1000
73/73 - 1s - loss: 4.3678
Epoch 315/1000
73/73 - 1s - loss: 3.7504
Epoch 316/1000
73/73 - 1s - loss: 3.0882
Epoch 317/1000
73/73 - 1s - loss: 3.3717
Epoch 318/1000
73/73 - 1s - loss: 3.6470
Epoch 319/1000
73/73 - 1s - loss: 3.4621
Epoch 320/1000
73/73 - 1s - loss: 3.6190
Epoch 321/1000
73/73 - 1s - loss: 4.8518
Epoch 322/1000
73/73 - 1s - loss: 4.0193
Epoch 323/1000
73/73 - 1s - loss: 3.3675
Epoch 324/1000
73/73 - 1s - loss: 3.3378
Epoch 325/1000
73/73 - 1s - loss: 3.1263
Epoch 326/1000
73/73 - 1s - loss: 3.3436
Epoch 327/1000
73/73 - 1s - loss: 3.7983
Epoch 328/1000
73/73 - 1s - loss: 3.6835
Epoch 329/1000
73/73 - 1s - loss: 3.3065
Epoch 330/1000
73/73 - 1s - loss: 3.2025
Epoch 331/1000
73/73 - 1s - loss: 4.3196
Epoch 332/1000
73/73 - 1s - loss: 4.2567
Epoch 333/1000
73/73 - 1s - loss: 3.6483
Epoch 334/1000
73/73 - 1s - loss: 3.9819
Epoch 335/1000
73/73 - 1s - loss: 5.4404
Epoch 336/1000
73/73 - 1s - loss: 3.6912
Epoch 337/1000
73/73 - 1s - loss: 3.3574
Epoch 338/1000
73/73 - 1s - loss: 4.0912
Epoch 339/1000
73/73 - 1s - loss: 3.0018
Epoch 340/1000
73/73 - 1s - loss: 4.1284
Epoch 341/1000
73/73 - 1s - loss: 4.7250
Epoch 342/1000
73/73 - 1s - loss: 4.7311
Epoch 343/1000
73/73 - 1s - loss: 3.7044
Epoch 344/1000
73/73 - 1s - loss: 3.4580
Epoch 345/1000
73/73 - 1s - loss: 4.2420
Epoch 346/1000
73/73 - 1s - loss: 4.6282
Epoch 347/1000
73/73 - 1s - loss: 3.8122
Epoch 348/1000
73/73 - 1s - loss: 3.2817
Epoch 349/1000
73/73 - 1s - loss: 3.5987
Epoch 350/1000
73/73 - 1s - loss: 3.0453
Epoch 351/1000
73/73 - 1s - loss: 4.2223
Epoch 352/1000
73/73 - 1s - loss: 3.3393
Epoch 353/1000
73/73 - 1s - loss: 4.9963
Epoch 354/1000
73/73 - 1s - loss: 5.3256
Epoch 355/1000
73/73 - 1s - loss: 4.0263
Epoch 356/1000
73/73 - 1s - loss: 3.0680
Epoch 357/1000
73/73 - 1s - loss: 4.7158
Epoch 358/1000
73/73 - 1s - loss: 4.0036
Epoch 359/1000
73/73 - 1s - loss: 4.0964
Epoch 360/1000
73/73 - 1s - loss: 3.5782
```

```
Epoch 529/1000
73/73 - 1s - loss: 2.5721
Epoch 530/1000
73/73 - 1s - loss: 2.6424
Epoch 531/1000
73/73 - 1s - loss: 2.5573
Epoch 532/1000
73/73 - 1s - loss: 3.2149
Epoch 533/1000
73/73 - 1s - loss: 3.6569
Epoch 534/1000
73/73 - 1s - loss: 3.0286
Epoch 535/1000
73/73 - 1s - loss: 2.9955
Epoch 536/1000
73/73 - 1s - loss: 3.3490
Epoch 537/1000
73/73 - 1s - loss: 3.0576
Epoch 538/1000
73/73 - 1s - loss: 2.7894
Epoch 539/1000
73/73 - 1s - loss: 3.8379
Epoch 540/1000
73/73 - 1s - loss: 5.3271
Epoch 541/1000
73/73 - 1s - loss: 4.9515
Epoch 542/1000
73/73 - 1s - loss: 3.4813
Epoch 543/1000
73/73 - 1s - loss: 4.3345
Epoch 544/1000
73/73 - 1s - loss: 3.5783
Epoch 545/1000
73/73 - 1s - loss: 3.8092
Epoch 546/1000
73/73 - 1s - loss: 3.6726
Epoch 547/1000
73/73 - 1s - loss: 3.2795
Epoch 548/1000
73/73 - 1s - loss: 4.4244
Epoch 549/1000
73/73 - 1s - loss: 3.8872
Epoch 550/1000
73/73 - 1s - loss: 3.1693
Epoch 551/1000
73/73 - 1s - loss: 4.5071
Epoch 552/1000
73/73 - 1s - loss: 4.0424
Epoch 553/1000
73/73 - 1s - loss: 5.2433
Epoch 554/1000
73/73 - 1s - loss: 2.9551
Epoch 555/1000
73/73 - 1s - loss: 3.2215
Epoch 556/1000
73/73 - 1s - loss: 3.0093
Epoch 557/1000
73/73 - 0s - loss: 3.7318
Epoch 558/1000
73/73 - 1s - loss: 2.5988
Epoch 559/1000
73/73 - 1s - loss: 3.3696
Epoch 560/1000
73/73 - 1s - loss: 4.5759
Epoch 561/1000
73/73 - 1s - loss: 3.8999
Epoch 562/1000
73/73 - 1s - loss: 3.3173
Epoch 563/1000
73/73 - 1s - loss: 3.5667
Epoch 564/1000
73/73 - 1s - loss: 3.6331
Epoch 565/1000
73/73 - 1s - loss: 3.7567
Epoch 566/1000
73/73 - 1s - loss: 2.9429
Epoch 567/1000
73/73 - 1s - loss: 3.0816
Epoch 568/1000
73/73 - 1s - loss: 3.3072
Epoch 569/1000
73/73 - 1s - loss: 2.8774
Epoch 570/1000
73/73 - 1s - loss: 3.5714
Epoch 571/1000
73/73 - 1s - loss: 3.5690
Epoch 572/1000
73/73 - 0s - loss: 2.7704
Epoch 573/1000
73/73 - 1s - loss: 3.7290
Epoch 574/1000
73/73 - 1s - loss: 3.1052
Epoch 575/1000
73/73 - 1s - loss: 4.1633
Epoch 576/1000
73/73 - 1s - loss: 3.3846
Epoch 577/1000
73/73 - 1s - loss: 2.7866
Epoch 578/1000
73/73 - 1s - loss: 2.5189
Epoch 579/1000
73/73 - 1s - loss: 3.5586
Epoch 580/1000
73/73 - 1s - loss: 2.7308
Epoch 581/1000
73/73 - 1s - loss: 2.8117
Epoch 582/1000
73/73 - 1s - loss: 2.6654
Epoch 583/1000
73/73 - 1s - loss: 2.8184
Epoch 584/1000
73/73 - 1s - loss: 3.0408
Epoch 585/1000
73/73 - 1s - loss: 3.6156
Epoch 586/1000
73/73 - 1s - loss: 3.0991
Epoch 587/1000
73/73 - 1s - loss: 2.7895
Epoch 588/1000
73/73 - 1s - loss: 2.8117
Epoch 589/1000
73/73 - 1s - loss: 3.6632
Epoch 590/1000
73/73 - 1s - loss: 3.7646
Epoch 591/1000
73/73 - 1s - loss: 4.5978
Epoch 592/1000
73/73 - 0s - loss: 3.5979
Epoch 593/1000
73/73 - 1s - loss: 2.5711
Epoch 594/1000
73/73 - 1s - loss: 2.8220
Epoch 595/1000
73/73 - 1s - loss: 2.9867
Epoch 596/1000
73/73 - 1s - loss: 2.6530
Epoch 597/1000
73/73 - 1s - loss: 3.0375
Epoch 598/1000
73/73 - 1s - loss: 3.2536
Epoch 599/1000
73/73 - 1s - loss: 4.0089
Epoch 600/1000
73/73 - 1s - loss: 3.6458
Epoch 601/1000
73/73 - 1s - loss: 3.5947
Epoch 602/1000
73/73 - 1s - loss: 3.0223
Epoch 603/1000
73/73 - 0s - loss: 2.6992
Epoch 604/1000
73/73 - 1s - loss: 2.9850
Epoch 605/1000
73/73 - 0s - loss: 3.3746
Epoch 606/1000
73/73 - 1s - loss: 4.4611
Epoch 607/1000
73/73 - 1s - loss: 3.8901
Epoch 608/1000
73/73 - 1s - loss: 3.2199
Epoch 609/1000
73/73 - 1s - loss: 4.5181
Epoch 610/1000
73/73 - 1s - loss: 4.0873
Epoch 611/1000
73/73 - 0s - loss: 2.6213
Epoch 612/1000
73/73 - 1s - loss: 3.3622
Epoch 613/1000
73/73 - 1s - loss: 3.6311
Epoch 614/1000
73/73 - 1s - loss: 2.7781
Epoch 615/1000
73/73 - 1s - loss: 3.4022
Epoch 616/1000
73/73 - 1s - loss: 3.4103
Epoch 617/1000
73/73 - 1s - loss: 2.5208
Epoch 618/1000
73/73 - 1s - loss: 2.4203
Epoch 619/1000
73/73 - 1s - loss: 2.8469
Epoch 620/1000
73/73 - 1s - loss: 2.5934
Epoch 621/1000
73/73 - 1s - loss: 2.5030
Epoch 622/1000
73/73 - 1s - loss: 3.3140
Epoch 623/1000
73/73 - 1s - loss: 2.7942
Epoch 624/1000
73/73 - 1s - loss: 3.4961
Epoch 625/1000
73/73 - 1s - loss: 2.6535
Epoch 626/1000
73/73 - 1s - loss: 2.5942
Epoch 627/1000
73/73 - 1s - loss: 2.5266
Epoch 628/1000
73/73 - 1s - loss: 3.3398
Epoch 629/1000
73/73 - 1s - loss: 2.9191
Epoch 630/1000
73/73 - 1s - loss: 2.7654
Epoch 631/1000
73/73 - 1s - loss: 4.2025
Epoch 632/1000
73/73 - 1s - loss: 3.1695
Epoch 633/1000
73/73 - 0s - loss: 2.8919
Epoch 634/1000
73/73 - 1s - loss: 2.5359
Epoch 635/1000
73/73 - 1s - loss: 2.3072
Epoch 636/1000
73/73 - 1s - loss: 3.4459
Epoch 637/1000
73/73 - 1s - loss: 3.8144
Epoch 638/1000
73/73 - 1s - loss: 3.7469
Epoch 639/1000
73/73 - 1s - loss: 3.9103
Epoch 640/1000
73/73 - 0s - loss: 4.2959
Epoch 641/1000
73/73 - 1s - loss: 3.0751
Epoch 642/1000
73/73 - 0s - loss: 2.8803
Epoch 643/1000
73/73 - 1s - loss: 2.6350
Epoch 644/1000
73/73 - 1s - loss: 3.8112
Epoch 645/1000
73/73 - 1s - loss: 2.3324
Epoch 646/1000
73/73 - 1s - loss: 3.5829
Epoch 647/1000
73/73 - 1s - loss: 3.2644
Epoch 648/1000
73/73 - 1s - loss: 3.2955
Epoch 649/1000
73/73 - 1s - loss: 3.1138
Epoch 650/1000
73/73 - 1s - loss: 2.6880
Epoch 651/1000
73/73 - 1s - loss: 2.1914
Epoch 652/1000
73/73 - 1s - loss: 2.6946
Epoch 653/1000
73/73 - 1s - loss: 2.3212
Epoch 654/1000
73/73 - 1s - loss: 2.6693
Epoch 655/1000
73/73 - 1s - loss: 2.9540
Epoch 656/1000
73/73 - 1s - loss: 2.5963
Epoch 657/1000
73/73 - 1s - loss: 2.1219
Epoch 658/1000
73/73 - 1s - loss: 2.8445
Epoch 659/1000
73/73 - 1s - loss: 3.3075
Epoch 660/1000
73/73 - 1s - loss: 4.3980
Epoch 661/1000
73/73 - 1s - loss: 3.5406
Epoch 662/1000
73/73 - 1s - loss: 2.7513
Epoch 663/1000
73/73 - 1s - loss: 3.4944
Epoch 664/1000
73/73 - 1s - loss: 3.4761
Epoch 665/1000
73/73 - 1s - loss: 2.4438
Epoch 666/1000
73/73 - 1s - loss: 2.7296
Epoch 667/1000
73/73 - 1s - loss: 2.7658
Epoch 668/1000
73/73 - 1s - loss: 2.5275
Epoch 669/1000
73/73 - 1s - loss: 3.9271
Epoch 670/1000
73/73 - 0s - loss: 2.5102
Epoch 671/1000
73/73 - 1s - loss: 2.5355
Epoch 672/1000
73/73 - 1s - loss: 3.1792
Epoch 673/1000
73/73 - 1s - loss: 2.6304
Epoch 674/1000
73/73 - 1s - loss: 2.9811
Epoch 675/1000
73/73 - 1s - loss: 3.0271
Epoch 676/1000
73/73 - 1s - loss: 4.3054
Epoch 677/1000
73/73 - 1s - loss: 2.3927
Epoch 678/1000
73/73 - 1s - loss: 3.3824
Epoch 679/1000
73/73 - 1s - loss: 2.3256
Epoch 680/1000
73/73 - 1s - loss: 2.8066
Epoch 681/1000
73/73 - 1s - loss: 3.3431
Epoch 682/1000
73/73 - 1s - loss: 4.2665
Epoch 683/1000
73/73 - 1s - loss: 2.8960
Epoch 684/1000
73/73 - 1s - loss: 2.3577
Epoch 685/1000
73/73 - 1s - loss: 3.1402
Epoch 686/1000
73/73 - 1s - loss: 2.9256
Epoch 687/1000
73/73 - 1s - loss: 3.0700
Epoch 688/1000
73/73 - 1s - loss: 2.3992
Epoch 689/1000
73/73 - 0s - loss: 2.7113
Epoch 690/1000
73/73 - 1s - loss: 2.1934
Epoch 691/1000
73/73 - 1s - loss: 2.2039
Epoch 692/1000
73/73 - 1s - loss: 2.8026
Epoch 693/1000
73/73 - 1s - loss: 2.7258
Epoch 694/1000
73/73 - 1s - loss: 2.6740
Epoch 695/1000
73/73 - 1s - loss: 3.1762
Epoch 696/1000
73/73 - 1s - loss: 2.0854
Epoch 697/1000
73/73 - 1s - loss: 2.4188
Epoch 698/1000
73/73 - 1s - loss: 3.1131
Epoch 699/1000
73/73 - 1s - loss: 2.2333
Epoch 700/1000
73/73 - 1s - loss: 3.1472
Epoch 701/1000
73/73 - 1s - loss: 4.9217
Epoch 702/1000
73/73 - 1s - loss: 4.0319
Epoch 703/1000
73/73 - 1s - loss: 2.7146
Epoch 704/1000
73/73 - 1s - loss: 2.3416
Epoch 705/1000
73/73 - 1s - loss: 2.9802
Epoch 706/1000
73/73 - 1s - loss: 3.1605
Epoch 707/1000
73/73 - 1s - loss: 3.3558
Epoch 708/1000
73/73 - 1s - loss: 2.4905
Epoch 709/1000
73/73 - 1s - loss: 2.7367
Epoch 710/1000
73/73 - 1s - loss: 2.4507
Epoch 711/1000
73/73 - 1s - loss: 2.7932
Epoch 712/1000
73/73 - 1s - loss: 2.4525
Epoch 713/1000
73/73 - 1s - loss: 2.7917
Epoch 714/1000
73/73 - 0s - loss: 2.0645
Epoch 715/1000
73/73 - 1s - loss: 3.4180
Epoch 716/1000
73/73 - 0s - loss: 2.7579
Epoch 717/1000
73/73 - 1s - loss: 2.5792
Epoch 718/1000
73/73 - 1s - loss: 1.9868
Epoch 719/1000
73/73 - 1s - loss: 2.2961
Epoch 720/1000
73/73 - 1s - loss: 2.5136
Epoch 721/1000
73/73 - 1s - loss: 3.0166
Epoch 722/1000
73/73 - 1s - loss: 2.6385
Epoch 723/1000
73/73 - 1s - loss: 2.4847
Epoch 724/1000
73/73 - 1s - loss: 3.7474
Epoch 725/1000
73/73 - 1s - loss: 2.5590
Epoch 726/1000
73/73 - 1s - loss: 3.2908
Epoch 727/1000
73/73 - 1s - loss: 3.1334
Epoch 728/1000
73/73 - 1s - loss: 3.3441
Epoch 729/1000
73/73 - 1s - loss: 3.4615
Epoch 730/1000
73/73 - 1s - loss: 2.6411
Epoch 731/1000
73/73 - 1s - loss: 2.9103
Epoch 732/1000
73/73 - 0s - loss: 3.5208
Epoch 733/1000
73/73 - 1s - loss: 2.6444
Epoch 734/1000
73/73 - 1s - loss: 2.2890
Epoch 735/1000
73/73 - 1s - loss: 2.7585
Epoch 736/1000
73/73 - 1s - loss: 3.0107
Epoch 737/1000
73/73 - 1s - loss: 2.7938
Epoch 738/1000
73/73 - 1s - loss: 3.2482
Epoch 739/1000
73/73 - 1s - loss: 3.3854
Epoch 740/1000
73/73 - 1s - loss: 2.3465
Epoch 741/1000
73/73 - 1s - loss: 2.4091
Epoch 742/1000
73/73 - 1s - loss: 2.9119
Epoch 743/1000
73/73 - 1s - loss: 3.4386
Epoch 744/1000
73/73 - 1s - loss: 2.6315
Epoch 745/1000
73/73 - 1s - loss: 2.7695
Epoch 746/1000
73/73 - 1s - loss: 2.5571
Epoch 747/1000
73/73 - 1s - loss: 2.5843
Epoch 748/1000
73/73 - 1s - loss: 2.2618
Epoch 749/1000
73/73 - 1s - loss: 2.2945
Epoch 750/1000
73/73 - 1s - loss: 3.3795
Epoch 751/1000
73/73 - 0s - loss: 2.6197
Epoch 752/1000
73/73 - 1s - loss: 2.4955
Epoch 753/1000
73/73 - 0s - loss: 2.3874
Epoch 754/1000
73/73 - 1s - loss: 2.5865
Epoch 755/1000
73/73 - 1s - loss: 2.8592
Epoch 756/1000
73/73 - 1s - loss: 2.6612
Epoch 757/1000
73/73 - 0s - loss: 3.5177
Epoch 758/1000
73/73 - 1s - loss: 2.6437
Epoch 759/1000
73/73 - 1s - loss: 2.3564
Epoch 760/1000
73/73 - 1s - loss: 2.8651
Epoch 761/1000
73/73 - 0s - loss: 3.1354
Epoch 762/1000
73/73 - 1s - loss: 2.6029
Epoch 763/1000
73/73 - 1s - loss: 3.1500
Epoch 764/1000
73/73 - 1s - loss: 3.1508
Epoch 765/1000
73/73 - 1s - loss: 2.9050
Epoch 766/1000
73/73 - 1s - loss: 4.3656
Epoch 767/1000
73/73 - 1s - loss: 3.5594
Epoch 768/1000
73/73 - 1s - loss: 2.4061
Epoch 769/1000
73/73 - 1s - loss: 2.4267
Epoch 770/1000
73/73 - 1s - loss: 2.3682
Epoch 771/1000
73/73 - 1s - loss: 3.6039
Epoch 772/1000
73/73 - 1s - loss: 2.0128
Epoch 773/1000
73/73 - 1s - loss: 2.6023
Epoch 774/1000
73/73 - 1s - loss: 2.3978
Epoch 775/1000
73/73 - 1s - loss: 4.0724
Epoch 776/1000
73/73 - 1s - loss: 3.1733
Epoch 777/1000
73/73 - 1s - loss: 2.0979
Epoch 778/1000
73/73 - 1s - loss: 2.8345
Epoch 779/1000
73/73 - 1s - loss: 3.6032
Epoch 780/1000
73/73 - 1s - loss: 1.9480
Epoch 781/1000
73/73 - 1s - loss: 2.5919
Epoch 782/1000
73/73 - 1s - loss: 2.6325
Epoch 783/1000
73/73 - 1s - loss: 2.1538
Epoch 784/1000
73/73 - 1s - loss: 1.9338
Epoch 785/1000
73/73 - 1s - loss: 2.1872
Epoch 786/1000
73/73 - 1s - loss: 3.4335
Epoch 787/1000
73/73 - 1s - loss: 2.1083
Epoch 788/1000
73/73 - 1s - loss: 2.1747
Epoch 789/1000
73/73 - 1s - loss: 3.7502
Epoch 790/1000
73/73 - 1s - loss: 2.4610
Epoch 791/1000
73/73 - 1s - loss: 2.5328
Epoch 792/1000
73/73 - 1s - loss: 2.9779
Epoch 793/1000
73/73 - 1s - loss: 2.5857
Epoch 794/1000
73/73 - 1s - loss: 2.6346
Epoch 795/1000
73/73 - 1s - loss: 3.0492
Epoch 796/1000
73/73 - 1s - loss: 2.7741
Epoch 797/1000
73/73 - 1s - loss: 2.4863
Epoch 798/1000
73/73 - 1s - loss: 2.8223
Epoch 799/1000
73/73 - 1s - loss: 3.2734
Epoch 800/1000
73/73 - 1s - loss: 2.7673
Epoch 801/1000
73/73 - 1s - loss: 2.5876
Epoch 802/1000
73/73 - 1s - loss: 2.5244
Epoch 803/1000
73/73 - 1s - loss: 3.5250
Epoch 804/1000
73/73 - 1s - loss: 2.9013
Epoch 805/1000
73/73 - 1s - loss: 3.0157
Epoch 806/1000
73/73 - 1s - loss: 3.9052
Epoch 807/1000
73/73 - 1s - loss: 2.6972
Epoch 808/1000
73/73 - 1s - loss: 2.5804
Epoch 809/1000
73/73 - 1s - loss: 1.7329
Epoch 810/1000
73/73 - 1s - loss: 2.1005
Epoch 811/1000
73/73 - 1s - loss: 1.9828
Epoch 812/1000
73/73 - 1s - loss: 2.0007
Epoch 813/1000
73/73 - 1s - loss: 2.3553
Epoch 814/1000
73/73 - 1s - loss: 3.3209
Epoch 815/1000
73/73 - 1s - loss: 3.1671
Epoch 816/1000
73/73 - 1s - loss: 3.2173
Epoch 817/1000
73/73 - 1s - loss: 1.9514
Epoch 818/1000
73/73 - 1s - loss: 2.4954
Epoch 819/1000
73/73 - 1s - loss: 3.0194
Epoch 820/1000
73/73 - 1s - loss: 2.6629
Epoch 821/1000
73/73 - 1s - loss: 3.2730
Epoch 822/1000
73/73 - 1s - loss: 2.3270
Epoch 823/1000
73/73 - 1s - loss: 2.2649
Epoch 824/1000
73/73 - 1s - loss: 2.1405
Epoch 825/1000
73/73 - 1s - loss: 2.8794
Epoch 826/1000
73/73 - 1s - loss: 2.6852
Epoch 827/1000
73/73 - 1s - loss: 2.9198
Epoch 828/1000
73/73 - 1s - loss: 3.1875
Epoch 829/1000
[illegible]
Epoch 929/1000
73/73 - 1s - loss: 3.2662
Epoch 930/1000
73/73 - 1s - loss: 2.1107
Epoch 931/1000
73/73 - 1s - loss: 2.3353
Epoch 932/1000
73/73 - 1s - loss: 2.3564
Epoch 933/1000
73/73 - 1s - loss: 1.7035
Epoch 934/1000
73/73 - 1s - loss: 2.2296
Epoch 935/1000
73/73 - 1s - loss: 3.8195
Epoch 936/1000
73/73 - 1s - loss: 2.6939
Epoch 937/1000
73/73 - 1s - loss: 2.4978
Epoch 938/1000
73/73 - 1s - loss: 2.6177
Epoch 939/1000
73/73 - 1s - loss: 1.7893
Epoch 940/1000
73/73 - 1s - loss: 2.2292
Epoch 941/1000
73/73 - 1s - loss: 1.8206
Epoch 942/1000
73/73 - 1s - loss: 1.6177
Epoch 943/1000
73/73 - 1s - loss: 2.9670
Epoch 944/1000
73/73 - 1s - loss: 3.5130
Epoch 945/1000
73/73 - 1s - loss: 2.3486
Epoch 946/1000
73/73 - 1s - loss: 2.2993
Epoch 947/1000
73/73 - 1s - loss: 1.8339
Epoch 948/1000
73/73 - 1s - loss: 2.7140
Epoch 949/1000
73/73 - 1s - loss: 2.5864
Epoch 950/1000
73/73 - 1s - loss: 2.5130
Epoch 951/1000
73/73 - 1s - loss: 2.5791
Epoch 952/1000
73/73 - 1s - loss: 2.0386
Epoch 953/1000
73/73 - 1s - loss: 2.5991
Epoch 954/1000
73/73 - 1s - loss: 1.6715
Epoch 955/1000
73/73 - 1s - loss: 2.0830
Epoch 956/1000
73/73 - 1s - loss: 2.0397
Epoch 957/1000
73/73 - 1s - loss: 2.7350
Epoch 958/1000
73/73 - 1s - loss: 2.5411
Epoch 959/1000
73/73 - 1s - loss: 2.8118
Epoch 960/1000
73/73 - 1s - loss: 3.4516
Epoch 961/1000
73/73 - 1s - loss: 3.5254
Epoch 962/1000
73/73 - 1s - loss: 3.5589
Epoch 963/1000
73/73 - 0s - loss: 3.9492
Epoch 964/1000
73/73 - 1s - loss: 2.5398
Epoch 965/1000
73/73 - 1s - loss: 2.5212
Epoch 966/1000
73/73 - 1s - loss: 2.2489
Epoch 967/1000
73/73 - 1s - loss: 2.4233
Epoch 968/1000
73/73 - 1s - loss: 2.6027
Epoch 969/1000
73/73 - 1s - loss: 2.4429
Epoch 970/1000
73/73 - 1s - loss: 2.3340
Epoch 971/1000
73/73 - 1s - loss: 2.5880
Epoch 972/1000
73/73 - 1s - loss: 2.0035
Epoch 973/1000
73/73 - 1s - loss: 2.6420
Epoch 974/1000
73/73 - 1s - loss: 1.6180
Epoch 975/1000
73/73 - 1s - loss: 1.8125
Epoch 976/1000
73/73 - 1s - loss: 1.9812
Epoch 977/1000
73/73 - 1s - loss: 1.9791
Epoch 978/1000
73/73 - 1s - loss: 3.5970
Epoch 979/1000
73/73 - 1s - loss: 2.7513
Epoch 980/1000
73/73 - 1s - loss: 2.4639
Epoch 981/1000
73/73 - 1s - loss: 2.5312
Epoch 982/1000
73/73 - 1s - loss: 1.7943
Epoch 983/1000
73/73 - 1s - loss: 1.2997
Epoch 984/1000
73/73 - 1s - loss: 1.8225
Epoch 985/1000
73/73 - 0s - loss: 1.8916
Epoch 986/1000
73/73 - 1s - loss: 2.5276
Epoch 987/1000
73/73 - 1s - loss: 2.6642
Epoch 988/1000
73/73 - 1s - loss: 1.8982
Epoch 989/1000
73/73 - 1s - loss: 2.4872
Epoch 990/1000
73/73 - 1s - loss: 2.0362
Epoch 991/1000
73/73 - 1s - loss: 1.8625
Epoch 992/1000
73/73 - 1s - loss: 1.8769
Epoch 993/1000
73/73 - 1s - loss: 2.3081
Epoch 994/1000
73/73 - 1s - loss: 2.2607
Epoch 995/1000
73/73 - 1s - loss: 2.6769
Epoch 996/1000
73/73 - 1s - loss: 2.6503
Epoch 997/1000
73/73 - 1s - loss: 1.9227
Epoch 998/1000
73/73 - 1s - loss: 3.3118
Epoch 999/1000
73/73 - 1s - loss: 2.7140
Epoch 1000/1000
73/73 - 1s - loss: 3.7600
```

Out[59]: &lt;tensorflow.python.keras.callbacks.History at 0x7ff66bc7bcc0&gt;

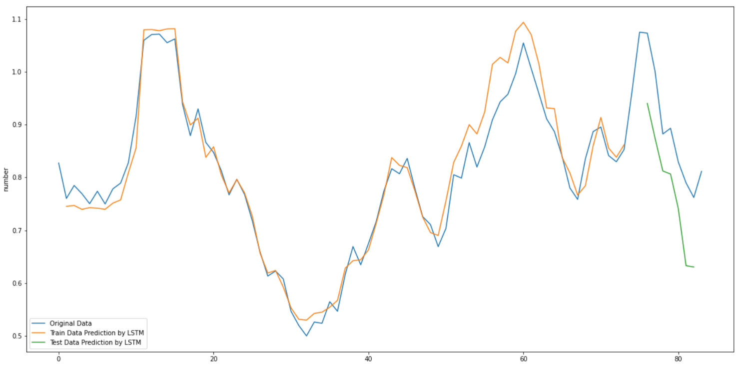
In [60]:

```
trainPredict = model.predict(X_tr_t, batch_size=batch_size)
model.reset_states()

testPredict = model.predict(X_tst_t, batch_size=batch_size)
```

In [61]:

```
look_back=1
trainPredictPlot = np.empty_like(dataset[:,0] )
trainPredictPlot[:] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back] = trainPredict.reshape(trainPredict.shape[0])
# shift test predictions for plotting
testPredictPlot = np.empty_like(dataset[:,0] )
```

```
testPredictPlot[:] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset )-1] = testPredict.reshape(testPredict.shape[0
# plot baseline and predictions
plt.figure(figsize=(20,10))
plt.plot(dataset[:,0], label='Original Data')
plt.plot(trainPredictPlot, label='Train Data Prediction by LSTM')
plt.plot(testPredictPlot,label='Test Data Prediction by LSTM')
plt.ylabel('number')
plt.legend()
plt.show()
```



In [62]:
```
print("MAPE Mean Absolute Percentage Error:", mean_absolute_percentage_error(y_test, testPredict.reshape(tes
print("Directional Symmetry Statistic:", directional_symmetry(y_test, testPredict.reshape(testPredict.shape[
```

```
MAPE Mean Absolute Percentage Error: 12.926832200564084
Directional Symmetry Statistic: 83.33333333333331
```

In [64]:
```
X
```

Out[64]:

| | ds | y | Steel_Inventory_M_USD | Steel_Orders_M_USD | IronOre_Global_USD | CoalAustralia_Global_USD | Nickel_Global_USD | Z |
|---|---|---|---|---|---|---|---|---|
| **0** | 2013-01-07 | 0.827352 | 20575 | 10568 | 127.191304 | 82.222360 | 13750.31522 | |
| **1** | 2013-01-08 | 0.760221 | 20609 | 10821 | 137.055000 | 82.221429 | 14308.26190 | |
| **2** | 2013-01-09 | 0.784715 | 20761 | 10774 | 134.185714 | 83.295918 | 13801.39286 | |
| **3** | 2013-01-10 | 0.769293 | 20735 | 11291 | 132.572727 | 85.443634 | 14117.65217 | |
| **4** | 2013-01-11 | 0.750242 | 20854 | 11559 | 136.323810 | 88.363265 | 13684.01190 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **79** | 2020-01-02 | 0.893123 | 19753 | 9914 | 87.625000 | 71.050179 | 12715.55000 | |
| **80** | 2020-01-03 | 0.829167 | 19945 | 9072 | 88.659091 | 70.204870 | 11846.22727 | |
| **81** | 2020-01-04 | 0.789704 | 19824 | 7800 | 83.750000 | 64.620000 | 11804.00950 | |
| **82** | 2020-01-05 | 0.762035 | 19441 | 8908 | 91.333333 | 54.148308 | 12179.60526 | |
| **83** | 2020-01-06 | 0.811477 | 18998 | 8979 | 103.340909 | 55.002760 | 12727.15227 | |

84 rows × 15 columns

# Machine Learning Models: Xgboost, Random Forest, Linear Regression LASSO, Ridge regression

In [66]:
```
# Split the data into X & y
df_tmp=data.copy()
X = df_tmp.drop(columns=['Date','dd','mm','yyyy','StainlessSteelPrice']+columns_need_to_remove,axis=1)
y = df_tmp["StainlessSteelPrice"]
#X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train,X_test,y_train,y_test=X[:78],X[78:],y[:78], y[78:]
print(X_train.shape)
X_test=np.array(X_test)
y_test=np.array(y_test)
col_index=X_test[:,-1].argsort()[::-1]
X_test=X_test[col_index]
y_test=y_test[col_index]
scaler = StandardScaler()
scaler.fit(X_train)
print('Feature,Mean, Variance:')
for i, j ,k in zip(every_column_except_y,scaler.mean_,scaler.var_):
  print(i,j,k)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
print(len(X_train))
print(len(X_test))

########################xgboost model with grid search to find hyperparameters####################
n_estimators = [10,20,30,40,50,60,70,80,100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster = ['gbtree', 'gblinear']
base_score = [0.25, 0.5, 0.75, 1]
learning_rate = [0.001,0.01,0.05, 0.1, 0.15, 0.20,1]
min_child_weight = [1, 2, 3, 4]
# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'learning_rate' : learning_rate,
    'min_child_weight' : min_child_weight,
    'booster' : booster,
    'base_score' : base_score
    }
mape=make_scorer(mean_absolute_percentage_error,greater_is_better=False)
regressor = xgboost.XGBRegressor()

random_cv = RandomizedSearchCV(estimator=regressor, param_distributions=hyperparameter_grid, cv=5,
                              n_iter=50, scoring = mape,n_jobs = 4,
                              verbose = 5, return_train_score = True, random_state=42)
random_cv.fit(X_train,y_train)
print(random_cv.best_params_)
regressor = xgboost.XGBRegressor(base_score= 0.5,
 booster= 'gbtree',
 learning_rate= 0.2,
 max_depth= 5,
 min_child_weight=1,
 n_estimators= 80)
y_preds = regressor.fit(X_train,y_train).predict(X_test)
mae_rf = mean_absolute_percentage_error(y_test,y_preds)
ds= directional_symmetry(y_test, y_preds)
print('MAPE',mae_rf, 'DS',ds)
#############################Random Forest Regressor###################
model = RandomForestRegressor(n_jobs=-1)
print(model)
model.fit(X_train,y_train)
# Evaluate model using mean absolute error
from sklearn.metrics import mean_absolute_error
y_preds = model.predict(X_test)
mae_rf = mean_absolute_percentage_error(y_test,y_preds)
ds= directional_symmetry(y_test, y_preds)
print('MAPE',mae_rf, 'DS',ds)

###########################linear Regression model#########################

model=LinearRegression().fit(X_train, y_train)
y_preds = model.predict(X_test)
mae_rf = mean_absolute_percentage_error(y_test,y_preds)
ds= directional_symmetry(y_test, y_preds)
print('MAPE',mae_rf, 'DS',ds)
```

```
(78, 13)
Feature,Mean, Variance:
Steel_Inventory_M_USD 20106.24358974359 3285945.158612755
Steel_Orders_M_USD 9885.141025641025 1552303.6852399735
IronOre_Global_USD 79.06360583807694 592.1812298678882
CoalAustralia_Global_USD 83.17662011166667 373.18185355984764
Nickel_Global_USD 12731.432463602567 7694755.987312424
Zinc_Global_USD 2382.852253205128 225549.51405673832
Freight_Transport_Index 0.20641025641025643 0.8931640368178833
Baltic_Dry_Index 1091.128205128205 185145.5476660092
BCI_China 98.82314756410256 0.35982123769278335
CLI_China 99.96203089743588 0.5995813608902194
CLI_US 100.07963346153846 0.4800161938662228
Copper_Global_USD  6078.979102564103 619322.7926056051
number_of_days_from_today 1780.576923076923 474628.5261341223
78
6
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  12 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 216 tasks      | elapsed:    4.2s
[Parallel(n_jobs=4)]: Done 250 out of 250 | elapsed:    6.0s finished
[02:43:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
{'n_estimators': 70, 'min_child_weight': 2, 'max_depth': 15, 'learning_rate': 0.15, 'booster': 'gbtree', 'ba
se_score': 0.25}
[02:43:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
MAPE 10.488548320169022 DS 60.0
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
MAPE 7.794590077106099 DS 60.0
MAPE 33.62484255590004 DS 100.0
```

In [67]:
```
!pip install autofeat
```

```
Requirement already satisfied: autofeat in /usr/local/lib/python3.6/dist-packages (2.0.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from autofeat) (1.19.5)
Requirement already satisfied: sympy>=1.7.1 in /usr/local/lib/python3.6/dist-packages (from autofeat) (1.7.
1)
Requirement already satisfied: pint in /usr/local/lib/python3.6/dist-packages (from autofeat) (0.16.1)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from autofeat) (0.16.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from autofeat) (1.0.0)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages (from autofeat) (1.
1.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from autofeat) (0.22.
2.post1)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.6/dist-packages (from sympy>=1.7.1->au
tofeat) (1.1.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-packages (from pint->autofeat) (2
0.9)
Requirement already satisfied: importlib-resources; python_version < "3.7" in /usr/local/lib/python3.6/dist-
packages (from pint->autofeat) (5.1.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-p
ackages (from pint->autofeat) (3.4.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-packages (from pandas
>=0.24.0->autofeat) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24.0->
autofeat) (2018.9)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->a
utofeat) (1.4.1)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from packaging->p
int->autofeat) (2.4.7)
Requirement already satisfied: zipp>=0.4; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages
(from importlib-resources; python_version < "3.7"->pint->autofeat) (3.4.0)
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /usr/local/lib/python3.6/
dist-packages (from importlib-metadata; python_version < "3.8"->pint->autofeat) (3.7.4.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.
7.3->pandas>=0.24.0->autofeat) (1.15.0)
```

# Machine Learning Models: LASSO, Ridge Regression, Random forest with additional created features with the help of AutoFeat Library

In [68]:
```
def test_autofeat( feateng_steps=2,units=None):
    #X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12)
    # run autofeat
```

```python
    afreg = AutoFeatRegressor(verbose=1, feateng_steps=feateng_steps, units=units)
    # fit autofeat on less data, otherwise ridge reg model with xval will overfit on new features
    X_train_tr = afreg.fit_transform(X_train, y_train)
    X_test_tr = afreg.transform(X_test)
    print("autofeat new features:", len(afreg.new_feat_cols_))
    print("MAPE on training data:", mean_absolute_percentage_error(y_train, afreg.predict(X_train_tr)))
    print("MAPE on test data:", mean_absolute_percentage_error(y_test, afreg.predict(X_test_tr)))
    print("Directional Symmetry Statistic on test data:", directional_symmetry(y_test, afreg.predict(X_test_

    # train rreg on transformed train split incl cross-validation for parameter selection
    print("# Ridge Regression")
    rreg = Ridge()
    param_grid = {"alpha": [0.00001, 0.0001, 0.001, 0.01, 0.1, 1., 2.5, 5., 10., 25., 50., 100., 250., 500.,
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        gsmodel = GridSearchCV(rreg, param_grid, scoring = mape, cv=5)
        gsmodel.fit(X_train_tr, y_train)
    print("best params:", gsmodel.best_params_)
    print("best score:", gsmodel.best_score_)
    print("MAPE on training data:", mean_absolute_percentage_error(y_train, gsmodel.predict(X_train_tr)))
    print("MAPE on test data:", mean_absolute_percentage_error(y_test, gsmodel.predict(X_test_tr)))
    print("Directional Symmetry Statistic on test data:", directional_symmetry(y_test,  gsmodel.predict(X_te
    print('Lasso')
    lasso=Lasso()
    param_grid = {"alpha": [0.00001, 0.0001, 0.001, 0.01, 0.1, 1., 2.5, 5., 10., 25., 50., 100., 250., 500.,
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        gsmodel = GridSearchCV(lasso, param_grid, scoring = mape, cv=5)
        gsmodel.fit(X_train_tr, y_train)
    print("best params:", gsmodel.best_params_)
    print("best score:", gsmodel.best_score_)
    print("MAPE on training data:", mean_absolute_percentage_error(y_train, gsmodel.predict(X_train_tr)))
    print("MAPE on test data:", mean_absolute_percentage_error(y_test, gsmodel.predict(X_test_tr)))
    print("Directional Symmetry Statistic on test data:", directional_symmetry(y_test,  gsmodel.predict(X_te

    print("# Random Forest")
    rforest = RandomForestRegressor(n_estimators=100, random_state=13)
    param_grid = {"min_samples_leaf": [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2]}
    gsmodel = GridSearchCV(rforest, param_grid, scoring = mape, cv=5)
    gsmodel.fit(X_train_tr, y_train)
    print("best params:", gsmodel.best_params_)
    print("best score:", gsmodel.best_score_)
    print("MAPE on training data:", mean_absolute_percentage_error(y_train, gsmodel.predict(X_train_tr)))
    print("MAPE on test data:", mean_absolute_percentage_error(y_test, gsmodel.predict(X_test_tr)))
    print("Directional Symmetry Statistic on test data:", directional_symmetry(y_test,  gsmodel.predict(X_te
for i in range(1):
  print('##################feateng_steps=',i+1,' ######################################')
  test_autofeat( feateng_steps=i+1)
```

```
##################feateng_steps= 1  ######################################
[AutoFeat] The 1 step feature engineering process could generate up to 91 features.
[AutoFeat] With 78 data points this new feature matrix would use about 0.00 gb of space.
[feateng] Step 1: transformation of original features
[feateng] Generated 65 transformed features from 13 original features - done.
[feateng] Generated altogether 65 new features in 1 steps
[feateng] Removing correlated features, as well as additions at the highest level
[feateng] Generated a total of 64 additional features
[featsel] Scaling data...done.
[featsel] Feature selection run 1/5
[featsel] Feature selection run 2/5
[featsel] Feature selection run 3/5
[featsel] Feature selection run 4/5
[featsel] Feature selection run 5/5
[featsel] 16 features after 5 feature selection runs
[featsel] 15 features after correlation filtering
[featsel] 7 features after noise filtering
[AutoFeat] Computing 3 new features.
[AutoFeat]     3/    3 new features ...done.
[AutoFeat] Final dataframe with 16 feature columns (3 new).
[AutoFeat] Training final regression model.
[AutoFeat] Trained model: largest coefficients:
0.872330035358846
-0.067691 * x009
-0.062035 * Abs(x005)
0.050315 * x008
0.049729 * x004
0.042602 * x010
-0.012619 * 1/x012
-0.005232 * x009**3
[AutoFeat] Final score: 0.9183
[AutoFeat] Computing 3 new features.
[AutoFeat]     3/    3 new features ...done.
autofeat new features: 3
MAPE on training data: 4.195956502166654
MAPE on test data: 799.357029084828
Directional Symmetry Statistic on test data: 100.0
# Ridge Regression
best params: {'alpha': 5.0}
best score: -7.187508365683845
MAPE on training data: 4.132097758345593
MAPE on test data: 1549.8060507084022
Directional Symmetry Statistic on test data: 80.0
Lasso
best params: {'alpha': 0.01}
best score: -8.483435133967436
MAPE on training data: 5.7269215338437744
MAPE on test data: 1680.152507326647
Directional Symmetry Statistic on test data: 80.0
# Random Forest
best params: {'min_samples_leaf': 0.0001}
best score: -9.921552340277907
MAPE on training data: 1.9242567368543437
MAPE on test data: 5.744874357641039
Directional Symmetry Statistic on test data: 60.0
##################feateng_steps= 2  ######################################
[AutoFeat] The 2 step feature engineering process could generate up to 4186 features.
[AutoFeat] With 78 data points this new feature matrix would use about 0.00 gb of space.
[feateng] Step 1: transformation of original features
[feateng] Generated 65 transformed features from 13 original features - done.
[feateng] Step 2: first combination of features
[feateng] Generated 2951 feature combinations from 3003 original feature tuples - done.
[feateng] Generated altogether 3017 new features in 2 steps
[feateng] Removing correlated features, as well as additions at the highest level
[feateng] Generated a total of 2724 additional features
[featsel] Scaling data...done.
[featsel] Feature selection run 1/5
[featsel] Feature selection run 2/5
[featsel] Feature selection run 3/5
[featsel] Feature selection run 4/5
[featsel] Feature selection run 5/5
[featsel] 32 features after 5 feature selection runs
[featsel] 31 features after correlation filtering
[featsel] 13 features after noise filtering
[AutoFeat] Computing 13 new features.
[AutoFeat]    13/   13 new features ...done.
[AutoFeat] Final dataframe with 26 feature columns (13 new).
[AutoFeat] Training final regression model.
[AutoFeat] Trained model: largest coefficients:
0.7888834716284335
0.026916 * x004*Abs(x003)
-0.022164 * x001*x002
0.018083 * x001*x010**2
0.017497 * x010*Abs(x003)
-0.016485 * x003**2*x009**3
0.010553 * exp(x000)*exp(x008)
0.008463 * x004*x010**2
0.007131 * x010**2*exp(x000)
-0.005927 * x009**3*exp(x005)
-0.005770 * exp(x002)/x012
0.005258 * x004**3*Abs(x008)
0.003167 * x010**2*exp(x001)
-0.001791 * x010**3/x005
[AutoFeat] Final score: 0.9546
[AutoFeat] Computing 13 new features.
[AutoFeat]    13/   13 new features ...done.
autofeat new features: 13
MAPE on training data: 3.142559846642727
MAPE on test data: 1486.988172679632
Directional Symmetry Statistic on test data: 80.0
# Ridge Regression
best params: {'alpha': 10.0}
best score: -4.186713664183551
MAPE on training data: 2.970506263959172
MAPE on test data: 1652.6049667489517
Directional Symmetry Statistic on test data: 100.0
Lasso
best params: {'alpha': 0.001}
best score: -4.847311714859951
MAPE on training data: 2.9876546255945593
MAPE on test data: 1707.7409348257806
Directional Symmetry Statistic on test data: 80.0
# Random Forest
best params: {'min_samples_leaf': 0.0001}
best score: -9.163474665173638
MAPE on training data: 1.7629865403399354
MAPE on test data: 6.446181646908834
Directional Symmetry Statistic on test data: 80.0
##################feateng_steps= 3  ######################################
[AutoFeat] The 3 step feature engineering process could generate up to 102466 features.
[AutoFeat] With 78 data points this new feature matrix would use about 0.03 gb of space.
[feateng] Step 1: transformation of original features
[feateng] Generated 65 transformed features from 13 original features - done.
[feateng] Step 2: first combination of features
[feateng] Generated 11960 feature combinations from 3003 original feature tuples - done.
[feateng] Step 3: transformation of new features
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
[feateng] Generated 51000 transformed features from 11960 original features - done.
[feateng] Generated altogether 63095 new features in 3 steps
[feateng] Removing correlated features, as well as additions at the highest level
[feateng] Generated a total of 48162 additional features
[featsel] Scaling data...done.
[featsel] Feature selection run 1/5
[featsel] Feature selection run 2/5
[featsel] Feature selection run 3/5
[featsel] Feature selection run 4/5
[featsel] Feature selection run 5/5
[featsel] 42 features after 5 feature selection runs
[featsel] 35 features after correlation filtering
[featsel] 15 features after noise filtering
[AutoFeat] Computing 15 new features.
[AutoFeat]    15/   15 new features ...done.
[AutoFeat] Final dataframe with 28 feature columns (15 new).
[AutoFeat] Training final regression model.
[AutoFeat] Trained model: largest coefficients:
0.740203172867836
0.021719 * x004*Abs(x003)
0.021290 * Abs(x012**2 - exp(x010))
0.015285 * x004*x010**2
-0.013924 * Abs(x005 - Abs(x006))
0.012575 * Abs(x009**2 - exp(x010))
0.012055 * exp(-x009 + x010)
0.011366 * x001*x010**2
0.007637 * exp(Abs(x002) - Abs(x012))
0.005511 * 1/(x005**3 + Abs(x002))
0.003304 * Abs(Abs(x012)/x005)
-0.003145 * Abs(x011**3 - exp(x006))
0.003051 * (x009 - exp(x001))**2
-0.002110 * Abs(x002**2 + 1/x012)
-0.001909 * exp(-x009)/x012
0.000263 * x001*Abs(x003)
[AutoFeat] Final score: 0.9366
[AutoFeat] Computing 15 new features.
[AutoFeat]    15/   15 new features ...done.
autofeat new features: 15
MAPE on training data: 3.7733739677289413
MAPE on test data: 12671758.856304035
Directional Symmetry Statistic on test data: 80.0
# Ridge Regression
best params: {'alpha': 25.0}
best score: -4.382095317109185
MAPE on training data: 3.5966112297460633
MAPE on test data: 11966432.016471004
Directional Symmetry Statistic on test data: 80.0
Lasso
best params: {'alpha': 0.01}
best score: -5.773574298441052
MAPE on training data: 4.170429007553731
MAPE on test data: 17050319.48959439
Directional Symmetry Statistic on test data: 80.0
# Random Forest
best params: {'min_samples_leaf': 0.0001}
best score: -6.256048120619475
MAPE on training data: 1.6670156451813518
MAPE on test data: 7.266776968436818
Directional Symmetry Statistic on test data: 60.0
##################feateng_steps= 4  ######################################
[AutoFeat] The 4 step feature engineering process could generate up to 10536526 features.
[AutoFeat] With 78 data points this new feature matrix would use about 3.29 gb of space.
[feateng] Step 1: transformation of original features
[feateng] Generated 65 transformed features from 13 original features - done.
[feateng] Step 2: first combination of features
[feateng] Generated 11960 feature combinations from 3003 original feature tuples - done.
[feateng] Step 3: transformation of new features
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
<string>:2: RuntimeWarning:

divide by zero encountered in log

<string>:2: RuntimeWarning:

divide by zero encountered in log
```

```
[feateng] Generated 51000 transformed features from 11960 original features - done.
[feateng] Step 4: combining old and new features
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-68-576d4d7feb70> in <module>()
     49 for i in range(4):
     50   print('##################feateng_steps=',i+1,' ######################################')
---> 51   test_autofeat( feateng_steps=i+1)

<ipython-input-68-576d4d7feb70> in test_autofeat(feateng_steps, units)
      4     afreg = AutoFeatRegressor(verbose=1, feateng_steps=feateng_steps, units=units)
      5     # fit autofeat on less data, otherwise ridge reg model with xval will overfit on new features
----> 6     X_train_tr = afreg.fit_transform(X_train, y_train)
      7     X_test_tr = afreg.transform(X_test)
      8     print("autofeat new features:", len(afreg.new_feat_cols_))

/usr/local/lib/python3.6/dist-packages/autofeat/autofeat.py in fit_transform(self, X, y)
    299         # generate features
    300         df_subs, self.feature_formulas_ = engineer_features(df_subs, self.feateng_cols_, _parse_unit
s(self.units, verbose=self.verbose),
--> 301                                                              self.feateng_steps, self.transformation
s, self.verbose)
    302         # select predictive features
    303         if self.featsel_runs <= 0:

/usr/local/lib/python3.6/dist-packages/autofeat/feateng.py in engineer_features(df_org, start_features, unit
s, max_steps, transformations, verbose)
    331             if verbose > 0:
    332                 print("[feateng] Step %i: combining old and new features" % steps)
--> 333             new_new_features, temp_uncorr = get_feature_combinations(list(product(original_features,
 new_features)))
    334             uncorr_features.update(temp_uncorr)
    335             steps += 1

/usr/local/lib/python3.6/dist-packages/autofeat/feateng.py in get_feature_combinations(feature_tuples)
    288                     new_feat = np.array(f(df[feat1].to_numpy(), df[feat2].to_numpy()), dtype=np.floa
t32)
    289                     # near 0 variance test - sometimes all that's left is "e"
```

```
--> 290                     if np.isfinite(new_feat).all() and np.var(new_feat) > 1e-10:
    291                         corr = max(abs(np.corrcoef(new_feat, df[feat1])[0, 1]), abs(np.corrcoef(new_
feat, df[feat2])[0, 1]))
    292                         if corr < 1.:

/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py in _all(a, axis, dtype, out, keepdims)
     55 
     56 def _all(a, axis=None, dtype=None, out=None, keepdims=False):
---> 57     return umr_all(a, axis, dtype, out, keepdims)
     58 
     59 def _count_reduce_items(arr, axis):

KeyboardInterrupt: 
```

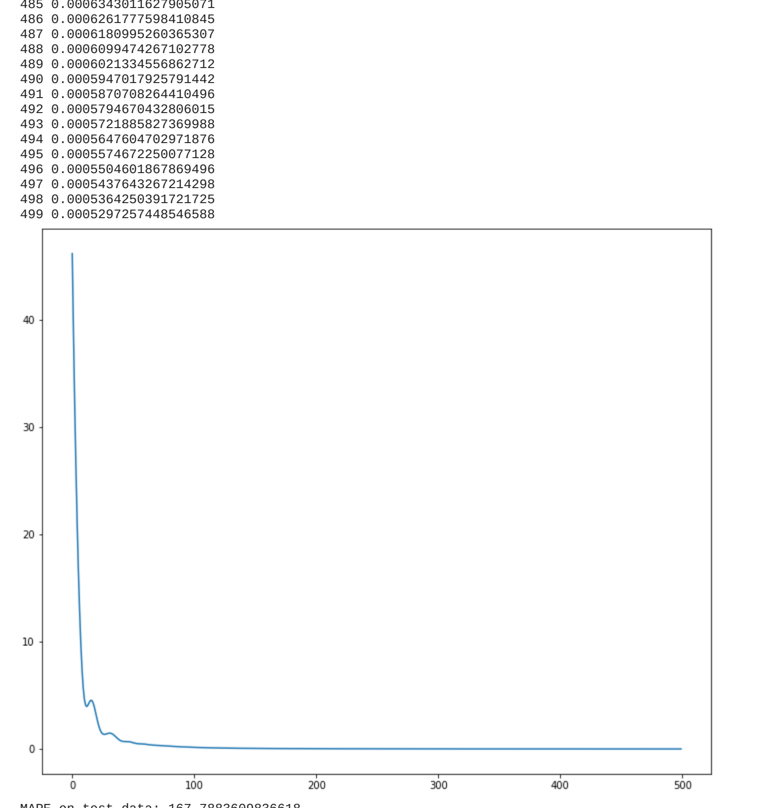
# Neural Network Model with Pytorch

In [69]:

```
X_train = torch.from_numpy(np.array(X_train)).float()
y_train = torch.from_numpy(np.array(y_train)).float()
X_test = torch.from_numpy(np.array(X_test)).float()
y_train=y_train.reshape(y_train.shape[0],1)
class Net(nn.Module):
    def __init__(self, D_in, H1, H2, H3, D_out):
        super(Net, self).__init__()

        self.linear1 = nn.Linear(D_in, H1)
        self.linear2 = nn.Linear(H1, H2)
        self.linear3 = nn.Linear(H2, H3)
        self.linear4 = nn.Linear(H3, D_out)

    def forward(self, x):
        y_pred = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(y_pred).clamp(min=0)
        y_pred = self.linear3(y_pred).clamp(min=0)
        y_pred = self.linear4(y_pred)
        return y_pred
H1, H2, H3 = 500, 1000, 200
D_in, D_out = X_train.shape[1], y_train.shape[1]
model1 = Net(D_in, H1, H2, H3, D_out)
criterion = nn.MSELoss(reduction='sum')
#optimizer = torch.optim.SGD(model1.parameters(), lr=1e-4 * 2)
optimizer = torch.optim.Adam(model1.parameters(), lr=1e-4 )
losses1 = []

for t in range(500):
    y_pred = model1(X_train)

    loss = criterion(y_train, y_pred)
    print(t, loss.item())
    losses1.append(loss.item())

    if torch.isnan(loss):
        break

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
plt.figure(figsize=(12, 10))
plt.plot(range(len(losses1)), losses1)
#plt.plot(range(len(losses3)), losses3)
plt.show()
y_pred = model1(X_test)
#print("MAPE on training data:", mean_absolute_percentage_error(y_train,
print("MAPE on test data:", mean_absolute_percentage_error(y_test, y_pred.detach().numpy().reshape(y_pred.de
print("Directional Symmetry Statistic on test data:", directional_symmetry(y_test, y_pred.detach().numpy().r
```

```
0 46.13474655151367
1 38.7170295715332
2 32.116214752197266
3 26.284027099609375
4 21.168235778808594
5 16.717897415161133
6 12.938552856445312
7 9.84252643585205
8 7.42533016204834
9 5.684387683868408
10 4.588488578796387
11 4.056254863739014
12 3.9528818130493164
13 4.109633922576904
14 4.351409912109375
15 4.523345470428467
16 4.523411273956299
17 4.325128078460693
18 3.957186222076416
19 3.478966236114502
20 2.958502769470215
21 2.4665677547454834
22 2.0495247840881348
23 1.7325681447982788
24 1.5196270942687988
25 1.4027447700500488
26 1.3607028722763062
27 1.369882345199585
28 1.4049620628356934
29 1.4444235563278198
30 1.4713078737258911
31 1.4744250774383545
32 1.4478845596313477
33 1.3912551403045654
34 1.3088860511779785
35 1.2080076932907104
36 1.0986831188201904
37 0.9906559586524963
38 0.8929235935211182
39 0.812466025352478
40 0.7538750767707825
41 0.7175580263137817
42 0.7002329230308533
43 0.6959221959114075
44 0.697254478931427
45 0.6969152092933655
46 0.6891797780990601
47 0.6717238426208496
48 0.6451902985572815
49 0.61272132396698
50 0.5785945057868958
51 0.5472161173820496
52 0.5214511156082153
53 0.5026105642318726
54 0.4903176724910736
55 0.4829690158367157
56 0.4781830310821533
57 0.473430871963501
58 0.4667607247829437
59 0.4570922255516052
60 0.4443293809890747
61 0.42912691831588745
62 0.41279172897338867
63 0.396751344203949
64 0.38225826621055603
65 0.3700830936431885
66 0.360459566116333
67 0.3530237078666687
68 0.3468590974807739
69 0.3409501612186432
70 0.33442583680152893
71 0.3268594741821289
72 0.31817275285720825
73 0.308747798204422
74 0.29918476939201355
75 0.28998327255249023
76 0.2815108895301819
77 0.2739503085613251
78 0.2671717405319214
79 0.2609010636806488
80 0.2548539936542511
81 0.24879159033298492
82 0.24259406328201294
83 0.23618115484714508
84 0.22978278994560242
85 0.22358711063861847
86 0.21770086884498596
87 0.2121489942073822
88 0.20693925023078918
89 0.2019374519586563
90 0.19698429107666016
91 0.19203610718250275
92 0.18704862892627716
93 0.18204158544540405
94 0.17708343267440796
95 0.172344848513603
96 0.16784833371639252
97 0.16357465088367462
98 0.1594621241092682
99 0.1554691344499588
100 0.15154176950454712
101 0.14765459299087524
102 0.14381316304206848
103 0.14003746211528778
104 0.13638737797737122
105 0.1328936219215393
106 0.1295451670885086
107 0.12632563710212708
108 0.12321862578392029
109 0.1201961487531662
110 0.11724226176738739
111 0.11437389254570007
112 0.11160626262426376
113 0.10894104838371277
114 0.10636011511087418
```

```
[illegible]
205 0.025528844445943832
206 0.025218812748789787
207 0.024909276515245438
208 0.02460477128624916
209 0.024307917803525925
210 0.024012181907892227
211 0.02371966652572155
212 0.02343478612601757
213 0.02315657213330269
214 0.022878974676132202
215 0.022605370730161667
216 0.02233516424894333
217 0.02206588350236416
218 0.021799422800540924
219 0.021538488566875458
220 0.021277684718370438
221 0.021021336317062378
222 0.020770933479070663
223 0.020516959950327873
224 0.020269213244318962
225 0.02002648264169693
226 0.019782444462180138
227 0.01954219676554203
228 0.019310105592012405
229 0.019082151353359222
230 0.01885337382555008
231 0.01862761378288269
232 0.01840571127831936
233 0.018185408785939217
234 0.01797408238053322
235 0.01775813288986683
236 0.017548657953739166
237 0.017340660095214844
238 0.017139092087745667
239 0.01692829839885235
240 0.016729062423110008
241 0.01653357967734337
242 0.01633457839488983
243 0.01613619178533554
244 0.015937406569719315
245 0.015748441219329834
246 0.015566342510282993
247 0.015389680862426758
248 0.015207558870315552
249 0.015024375170469284
250 0.014847565442323685
251 0.014669552445411682
252 0.014491760171949863
253 0.014324519783258438
254 0.014154338277876377
255 0.013985424302518368
256 0.013816927559673786
257 0.013654359616339207
258 0.013492782600224018
259 0.013327587395906448
260 0.013162524439394474
261 0.013005156069993973
262 0.01284809224307537
263 0.01269752997905016
264 0.012542685493826866
265 0.012392205186188221
266 0.01223919726908207
267 0.012087572365999222
268 0.011939652264118195
269 0.01179167814552784
270 0.011642593890428543
271 0.011496154591441154
272 0.011353721842169762
273 0.01120942272245884
274 0.01107148639857769
275 0.010931392200291157
276 0.010789641179144382
277 0.010655361227691174
278 0.010524162091314793
279 0.01039797067642212
280 0.010267267003655434
281 0.010138168931007385
282 0.010012289509177208
283 0.009883180260658264
284 0.009758176282048225
285 0.009633093141019344
286 0.009509583003818989
287 0.009386060759425163
288 0.00926338043063879
289 0.009145726449787617
290 0.009030036628246307
291 0.008914472535252571
292 0.008799567818641663
293 0.008685890585184097
294 0.008577559143304825
295 0.008466370403766632
296 0.008357960730791092
297 0.008251870051026344
298 0.008143438026309013
299 0.008035840466618538
300 0.007934022694826126
301 0.007831327617168427
302 0.007728002965450287
303 0.0076255095191299915
304 0.007527539040893316
305 0.007429372053593397
306 0.0073309000581502914
307 0.007234819233417511
308 0.00714386161416769
309 0.007050088606774807
310 0.006958177778869867
311 0.0068709938786923885
312 0.006781395990401506
313 0.0066899508237838745
314 0.006604218389838934
315 0.006521415431052446
316 0.00643565971404314
317 0.006349063478410244
318 0.0062637655064463615
319 0.006185567006468773
320 0.006104792468249798
321 0.006025707349181175
322 0.005942904390394688
323 0.00586280319839716
324 0.005788854788988829
325 0.005713220685720444
326 0.0056359246373176575
327 0.005560468882322311
328 0.005490126553922892
329 0.005419058725237846
330 0.005345559678971767
331 0.005276657175272703
332 0.005209573544561863
333 0.005138453096151352
334 0.005071983207017183
335 0.005007809493690729
336 0.004941171500831842
337 0.004873717203736305
338 0.00480653578415513
339 0.00474680308252573
340 0.004684197250753641
341 0.004617904778569937
342 0.004555315710604191
343 0.004495255183428526
344 0.004433720372617245
345 0.004375315736979246
346 0.004318200051784515
347 0.0042592911049723625
348 0.00420176750048995
349 0.004146356135606766
350 0.004089174792170525
351 0.00403622118756175
352 0.003984702751040459
353 0.003930762410163879
354 0.003880048170685768
355 0.003830503672361374
356 0.0037791302893310785
357 0.0037303296849131584
358 0.0036814608611166477
359 0.0036334346514195204
360 0.0035844682715833187
361 0.003535717725753784
362 0.00348723866045475
363 0.0034442951437085867
364 0.003399488516151905
365 0.0033539575524628162
366 0.0033103092573583126
367 0.0032660416327416897
368 0.0032222203444689512
369 0.0031784791499376297
370 0.0031356383115053177
371 0.003092745551839471
372 0.003051002509891987
373 0.003008914412930608
374 0.002966737374663353
375 0.002924772445112467
376 0.0028840999584645033
377 0.0028429192025214434
378 0.0028033757116645575
379 0.002761838026344776
380 0.002721304539591074
381 0.0026807833928614855
382 0.0026430990546941757
383 0.0026054196059703827
384 0.0025687329471111298
385 0.0025317214895039797
386 0.00249496940523386
387 0.0024589081294834614
388 0.002423797035589814
389 0.0023872917518019676
390 0.002353176474571228
391 0.002319265389814973
392 0.0022848478984087706
393 0.002251939382404089
394 0.002218669280409813
395 0.0021855321247130632
396 0.0021535800769925117
397 0.0021211812272667885
398 0.0020901085808873177
399 0.0020587118342518806
400 0.002027775626629591
401 0.00199905619956553
402 0.001969918143004179
403 0.001939747133292578
404 0.0019122555386275053
405 0.0018853810615837574
406 0.001857978175394237
407 0.0018323491094633937
408 0.0018069808138534427
[illegible]
```



[illegible]