Kayla Brown
Master of Science in Data Analytics, Western Governors University

D214: Graduate Capstone Project

Dr. Smith and Dr. Sewell

August 28, 2023

Research Question

Section 1.a

Yahoo Finance provides past and current information of precious metals and their related trading prices (YahooFinance, 2023). This project focuses on each of the documented commodities (Gold, Silver, Platinum, and Palladium) in combination with time series analysis to predict the closing market price for the next year into 2024. Predicting closing market value with each of the precious metals can indicate which metals are less of a financial risk for traders.

My research question is as stated: To what extent can Neural Prophet accurately predict variable "cost" for precious metals "Gold", "Silver", "Platinum", and "Palladium" over a forecast period of 365 days?

For gold trades alone, approximately 27 million ounces are traded daily. Precious metals and their trades impact the financial and investment world due to each precious metal having their own usefulness (i.e. jewelry, industrial parts, chemical applications), various intrinsic inflation protection, and history as an international currency – especially in disrupted economies and war-time periods (Schwab,2023). Analyzing the trends, seasonality, and patterns of closing market value for metals can greatly prepare traders to have a well-planned investment portfolio and educated decisions.

The basis of my project is model accuracy in relation to precious metals' cost. For my alternative hypothesis, I state that Neural Prophet can predict the market closing price of precious metals over a forecast period of 365 days with at least 75% accuracy. For my null hypothesis, I state Neural Prophet cannot predict the market closing price of precious metals over a forecast period of 365 days with at least 75% accuracy.

While time series forecasting is a highly utilized machine learning technique for businesses and their profits, accuracy cannot be 100% due to unplanned irregularities and seasonality (Talaviya, 2022). However, I will be utilizing Neural Prophet for my time series analysis. Neural Prophet is a mathematical and machine learning focused model that provides high accuracy in comparison to other models. This is due to Neural Prophet accounting for and being flexible with seasonal patterns, decomposition method like exponential smoothing for varied weighted averages based on recency, and curve-fitting approach (LaBarr, 2022). Though it is impossible to achieve a flawless model, my hypothesis for a minimum of 75% accuracy should be achievable with Neural Prophet as my time series analysis model.

Data Collection
Section 2.a

The data utilized for this time series project is provided by Guillem Servera on Kaggle.com and his data extraction from Yahoo Finance (Servera,2023). The license for this acquired dataset is Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) in which users are free to share and adapt the dataset if appropriate attribution is performed and the dataset's use is not used for financial/commercial purposes. The attributions associated with dataset-allowance is that the user gives appropriate credit, provides a link to the license, and indicate changes made to the dataset. The link to Guillem Servera's extracted "Gold, Silver & Precious Metals Futures Daily Data" dataset from Yahoo Finance:

https://www.kaggle.com/datasets/guillemservera/precious-metals-data. The related license link:

https://creativecommons.org/licenses/by-nc/4.0/. To ensure appropriate credit and documentation, these sources will be provided in the "Sources" section as well (Kaggle, 2023).

The original dataset has 27,994 rows and 8 columns. These 8 columns, or features, are 'ticker', 'commodity', 'date', 'open', 'high', 'low', 'close', and 'volume'. I transformed the original dataset by creating four new, smaller datasets based off the unique values in column 'commodity': 'Gold', 'Silver', 'Platinum', and 'Palladium'. I did not include the unique value 'Copper' due to copper being a base metal and not a precious metal, which is what I am analyzing. With the new datasets 'golddf', 'silverdf', 'platdf', 'palldf' for 'Gold', 'Silver', 'Platinum', and 'Palladium' respectively, there are two columns 'date' and 'close'. The 'date' column is transformed into a datetime format with the pandas python package with the 'close' column containing the closing market price for that respective date listed. For modeling purposes, these later become 'ds' for 'date' and 'y' for 'close'.

Section 2.b

Dataset licensing was a challenge during my dataset collecting due to data property ownership. After thorough research of license CC BY-NC 4.0, I have understood and undertaken necessary steps to ensure legal and ethical use of this dataset.

An advantage of using the Kaggle dataset is that the data is already structured with definitive columns and rows in a CSV file. The data collecting phase of the data analysis life cycle was completed with a simple download which minimizes the time I expended on the project. However, a disadvantage of using an already structured CSV file was the necessary data transformation to meet my project needs such as date-time conversion, creation of 4 smaller datasets originating from the one original dataset and renaming of features to meet Neural Prophet model requirements.

Data Extraction and Preparation
Section 3.a

To prepare my Jupyter notebook environment for Python and time series analysis, I first import all necessary Python packages. Pandas and Numpy are the primary data manipulation packages for data analysis. For visualizations, Matplotlib and and Seaborn are used for user-friendly graphs and colors. Time series analysis packages are Neural Prophet for modeling, DateTime for date conversion, and Sklearn for data splitting for a 90% train dataset and 10% validation test set for model evaluation. Lastly, I imported Warnings so as to minimize text disruption in my functional code.

```
In [1]: #importing necessary packages

#general data analysis packages
import pandas as pd
import numpy as np

#visualization / plotting packages
import matplotlib.pyplot as plt
import seaborn as sns

#these packages pertain to time series
from neuralprophet import NeuralProphet
import datetime
from sklearn.model_selection import TimeSeriesSplit

#ignore code warnings
import warnings
warnings.filterwarnings('ignore')
```

After preparing the environment, I upload the CSV “PreciousMetals” as “df” which contains the downloaded dataset from Kaggle (Servera, 2023) by using Pandas (pd). With my dataset uploaded to my Jupyter notebook, I began data exploration. First, I assessed the column names with “df.columns” which resulted in “ticker”, “commodity”, “date”, “open”, ”high”, “low”, “close”, and “volume”. Next, I assessed the number of rows and columns with “df.shape” with the output of 27,994 rows and 8 columns.

```
In [2]: #importing csv using file path
        df = pd.read_csv(r"C:\Users\kmbro\OneDrive\Documents\MSDA_Courses\D214\PreciousMetals.csv")

In [3]: #EDA - column names
        df.columns

Out[3]: Index(['ticker', 'commodity', 'date', 'open', 'high', 'low', 'close',
               'volume'],
              dtype='object')

In [4]: #EDA - number of rows, columns
        df.shape

Out[4]: (27994, 8)
```

Before proceeding to further exploration, I assess the need for data cleaning. First, I checked for null values in which missing entries occur in the dataset. Second, I checked for duplicated values for accidental repeat entries. Both assessments were negative for a need to perform data cleaning.

```
In [9]: #data cleaning - assessing need for data prep
        df.isnull().sum()

Out[9]: ticker       0
        commodity    0
        date         0
        open         0
        high         0
        low          0
        close        0
        volume       0
        dtype: int64
```

```
In [10]: #data cleaning - assessing for duplicate values
         df.duplicated()

Out[10]: 0        False
         1        False
         2        False
         3        False
         4        False
                  ...
         27989    False
         27990    False
         27991    False
         27992    False
         27993    False
         Length: 27994, dtype: bool
```

With a better understanding of the size of my dataset, I look closer at the columns to determine their data type (i.e. string, float, integer). Out of the three project-significant columns (‘commodity’, ‘date’, ’close’), only ‘date’ needs to have a transformed data type.

```
In [5]: #EDA - assessing data types of columns
        df.dtypes

Out[5]: ticker        object
        commodity     object
        date          object
        open         float64
        high         float64
        low          float64
        close        float64
        volume         int64
        dtype: object
```

Time series analysis requires a feature containing dates that are formatted as date-time rather than an object data type as seen in the code above. To correct this, I utilize Pandas (pd) to convert my ‘date’ feature into a date-time format. I then assess my newly transformed column to ensure the conversion worked.

(Please see visualization below)

```
In [11]: #convert "date" entry to datetime
         df['date'] = pd.to_datetime(df['date'])

In [12]: #ensuring datetime function was successful
         df.date

Out[12]: 0       2000-08-30
         1       2000-08-31
         2       2000-09-01
         3       2000-09-05
         4       2000-09-06
                    ...
         27989   2023-08-14
         27990   2023-08-15
         27991   2023-08-16
         27992   2023-08-17
         27993   2023-08-18
         Name: date, Length: 27994, dtype: datetime64[ns]
```

With general data assessments performed, one of the last data preparation steps is creating exclusive, commodity-specific datasets. Because Neural Prophet time series analysis is a univeriate model, I can only have one variable to trend over time: closing market price, known as 'close'. Thus, I separate the unique values in the 'commodity' column to create four smaller datasets containing only the specific dates for that commodity ('date') and the closing market price ('close') for that specific commodity. Though 'copper' is a unique value in the 'commodity' column, copper is not a precious metal and will not be included in this project

```
In [13]: #creating smaller datasets for unique values 'gold','silver','platinum', and 'palladium' from 'commodity' column
         golddf = df[df['commodity'] == 'Gold']
         silverdf = df[df['commodity'] == 'Silver']
         platdf  = df[df['commodity'] == 'Platinum']
         palldf = df[df['commodity'] == 'Palladium']

In [14]: #selecting only necessary columns for new datasets
         golddf = golddf[['date','close']]
         silverdf = silverdf[['date','close']]
         platdf = platdf[['date','close']]
         palldf = palldf[['date','close']]
```

pertaining to only precious metals and their forecasted market prices.

Finally, I explore my new datasets and assess their rows and columns. With each containing two columns as expected ('date' and 'close') and at least 5,200 entries in each, these datasets are ready for analysis.

```
In [15]: #EDA - number of rows, columns
         golddf.shape
Out[15]: (5762, 2)

In [16]: #EDA - number of rows, columns
         silverdf.shape
Out[16]: (5764, 2)

In [17]: #EDA - number of rows, columns
         platdf.shape
Out[17]: (5231, 2)

In [18]: #EDA - number of rows, columns
         palldf.shape
Out[18]: (5471, 2)
```

Section 3.b

Frequency of the market closing prices in the Kaggle Precious Metals dataset had been a challenge. The prices are updated on a daily frequency whereas for the purposes of this project, I will be calculating and predicting using a monthly frequency. Due to Neural Prophet, however, this potential challenge was corrected during model fitting.

Both an advantage and disadvantage found during data extraction and preparation is the use of Python for this project. R is a programming language meant specifically for data analysis and created with intentions of data visualization and statistical use. In comparison, Python is a general-purpose programming language in which various imported packages allow for a wide, diverse range of data manipulation (Luna, 2022). Though R may accelerate the time series analysis process with less chance of human error, imported Python packages are up to date with the best machine learning libraries and techniques for data analysis (Valeriy Manokhin, 2023).

Analysis
Section 4.a

To predict and forecast the upcoming year's market closing prices (variable 'close') for precious metals Gold, Silver, Platinum, and Palladium, I will be utilizing time series analysis. Time series analysis consists of dates collected from consistent intervals in relation to one or more observed variables. For my univariate time series analysis, the observed variable is 'close'. Neural Prophet is a neural-network based time series model utilized for accurate, univariate analysis (Samal, 2022).

The first step of my analysis is to convert the datetime 'date' column and float 'close' column to 'ds' and 'y' respectively due to Neural Prophet only responding to these two column names for model fitting. Secondly, I create the variable 'm' to represent the Neural Prophet model for easier coding. Third, I split my datasets into train and test datasets (typically 90% train, 10% test). This dataset split allows for evaluation of my model. Fourth, I fit my datasets with the Neural Prophet model for visualization and prediction.

(Please see next page)

Section 4.b
Gold

Gold Model & Validation

In [23]:
```
#NeuralProphet expects two columns: 'ds' for dates and 'y' for observed value
golddf.rename(columns = {'date':'ds', 'close' : 'y'}, inplace=True)
```

In [24]:
```
#gold model and visualizations

#fitting model to a split dataset (90/10 : train/test)
m = NeuralProphet()
golddf_train, golddf_val = m.split_df(golddf, freq='M', valid_p = 0.1)
goldmetrics = m.fit(golddf_train, freq='M', validation_df=golddf_val)
```

```
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
WARNING - (NP.forecaster.fit) - When Global modeling with local normalization, metrics are displayed in normalized scale.
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 96.201% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to ove
rride this.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 32
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 122
WARNING - (NP.config.set_lr_finder_args) - Learning rate finder: The number of batches (163) is too small than the required
number for the learning rate finder (243). The results might not be optimal.
```

Finding best initial lr: 100% 243/243 [00:02<00:00, 232.42it/s]

Epoch 122: 100%

122/122 [00:00<00:00, 178.17it/s, loss=0.00122, v_num=58, MAE_val=75.00, RMSE_val=102.0, Loss_val=0.00231, RegLoss_val=0.000, MAE=58.20, RMSE=82.80, Loss=0.00142, RegLoss=0.000]

With my dataset now fitted to the model, I can use Neural Prophet parameters to adjust the forecast as needed. For my intentions, I set the “periods” parameter to 12 for a total of twelve months due to the model frequency set to “M” for monthly intervals. After creating the forecast predictions, I assessed the first and last 5 entries with the first prediction being August 31,2023 and the last prediction being July 31, 2024. This verifies a correct prediction length. With a verified prediction model, I continue with my visualizations.

(Please see visualization below)

In [26]:
```
#creating predictions
goldfuture = m.make_future_dataframe(golddf, periods=12)
goldforecast = m.predict(goldfuture)

#view the first 5 entries of forecasted prices
goldforecast.head()
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 96.217% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
```

Predicting DataLoader 0: 100% 1/1 [00:00<?, ?

```
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

Out[26]:

	ds	y	yhat1	trend	season_yearly	season_weekly
0	2023-08-31	None	2020.005493	2003.580200	21.984966	-5.559675
1	2023-09-30	None	1054.450806	2013.074219	9.150304	-967.773743
2	2023-10-31	None	2020.333008	2022.884766	2.959490	-5.511282
3	2023-11-30	None	2012.475342	2032.378784	-14.343842	-5.559675
4	2023-12-31	None	3024.880859	2042.189209	-13.728808	996.420166

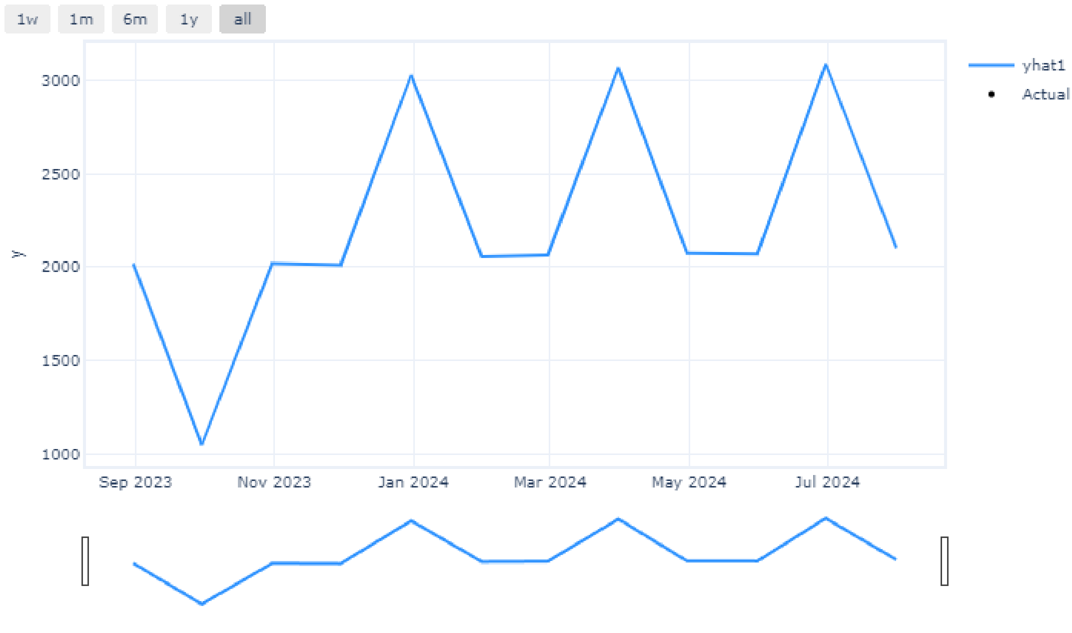
In [27]:
```
#view the last 5 entries of forecasted prices
goldforecast.tail(5)
```

Out[27]:

	ds	y	yhat1	trend	season_yearly	season_weekly
7	2024-03-31	None	3065.817627	2070.987549	-1.590038	996.420166
8	2024-04-30	None	2076.164307	2080.481689	1.194038	-5.511282
9	2024-05-31	None	2071.779785	2090.292236	-12.271945	-6.240614
10	2024-06-30	None	3084.480225	2099.786133	-11.726340	996.420166
11	2024-07-31	None	2103.266602	2109.596436	-0.864809	-5.464949

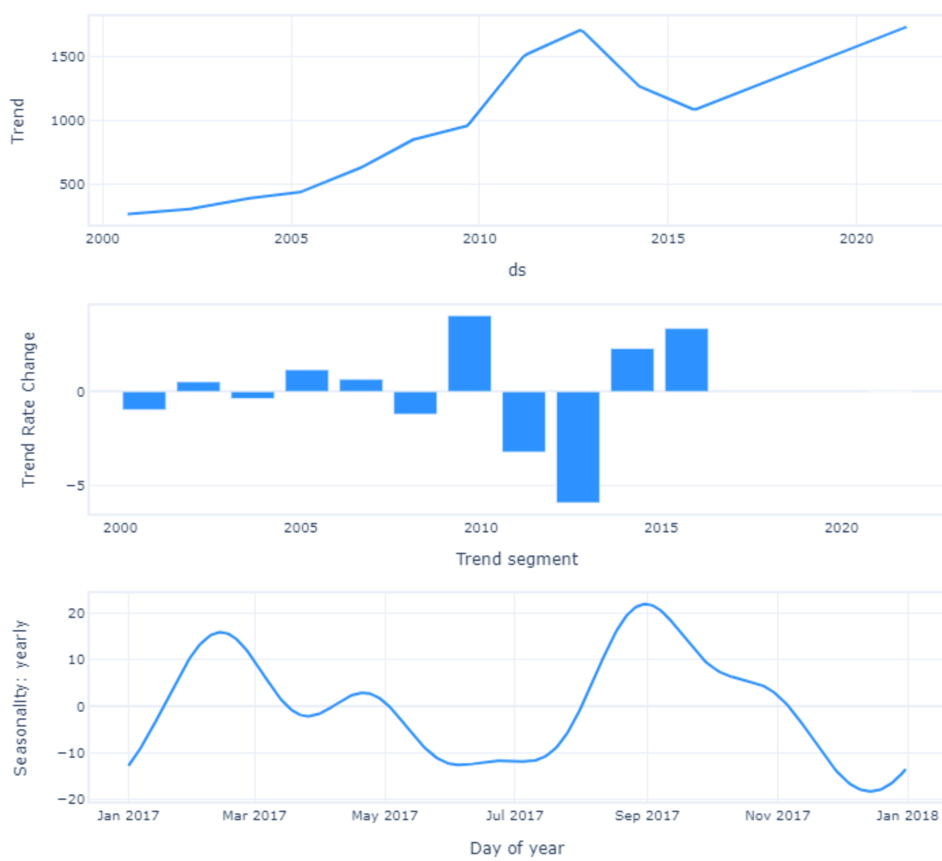
In [28]:
```
#visualization of prediction
m.plot(goldforecast)
```

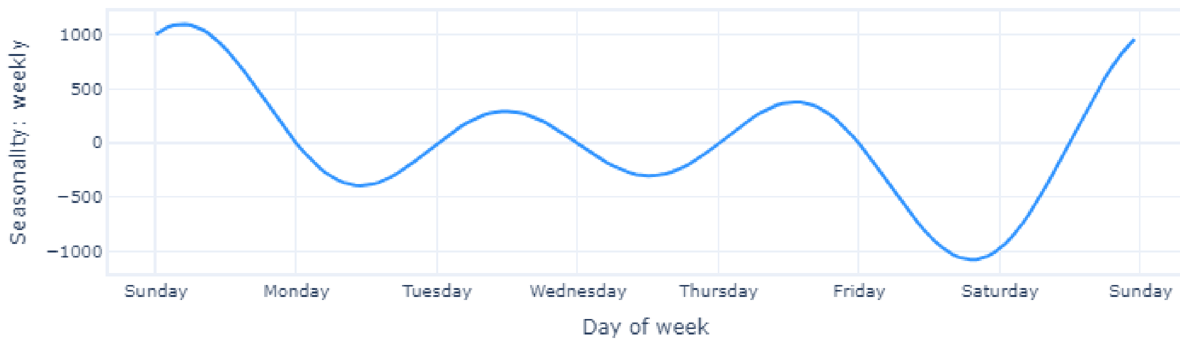
Out[28]:



In [29]:
```
#visualizing components
m.plot_parameters()
```

Out[29]:





To assess the accuracy of my model and predictions, I utilize the metrics calculated with Neural Prophet in relation to my train and validation datasets. The first evaluation metric used is MAE, Mean Absolute Error. MAE evaluates regression models and their accuracy by calculating the amount of deviation from the predictions and the actual values (Acharya, 2021). The resulting MAE for this project indicates how many dollars ($) the prediction deviated by.

In [25]:
```
#evaluation metrics
goldmetrics
```

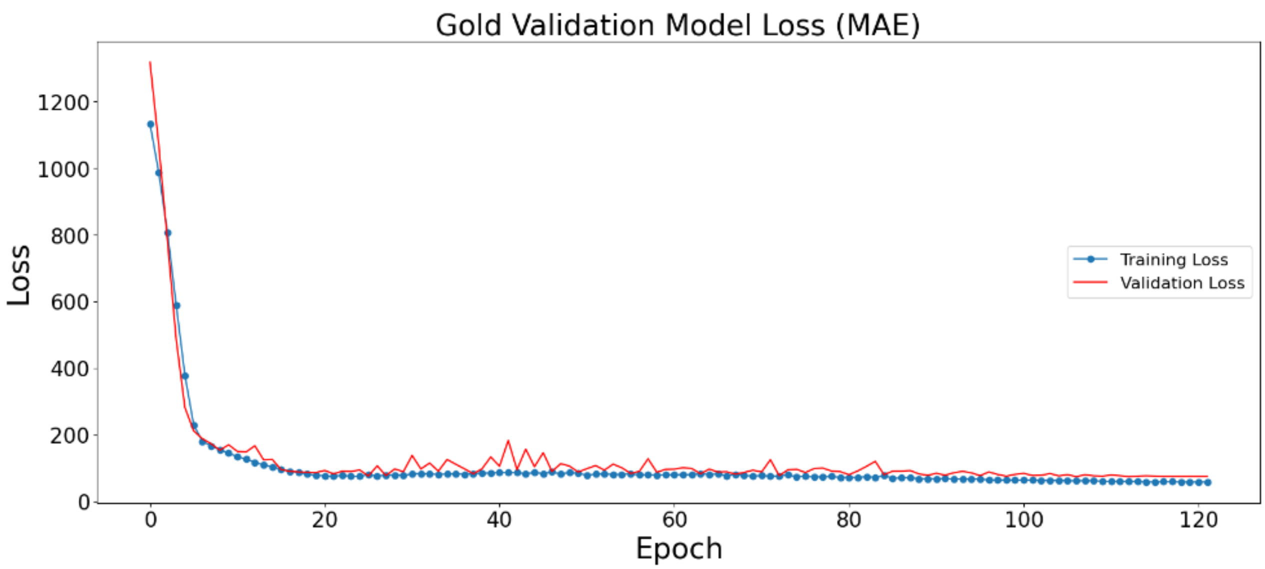
Out[25]:

	MAE_val	RMSE_val	Loss_val	RegLoss_val	epoch	MAE	RMSE	Loss	RegLoss
0	1317.828369	1664.233887	0.512099	0.0	0	1133.322388	1423.124023	0.308874	0.0
1	1070.880249	1374.999268	0.374737	0.0	1	987.325989	1234.439941	0.243054	0.0
2	779.996826	1011.912659	0.219418	0.0	2	807.251038	998.207947	0.167252	0.0
3	485.073364	627.900574	0.088127	0.0	3	588.894714	710.542542	0.087028	0.0
4	280.898560	357.912994	0.028649	0.0	4	377.589783	447.912323	0.034983	0.0
...	...	...	...	...	...	...	...	...	...
117	74.965897	101.372650	0.002298	0.0	117	58.620220	82.803101	0.001428	0.0
118	74.884750	101.192665	0.002290	0.0	118	58.274960	82.551033	0.001420	0.0
119	75.057655	101.442810	0.002301	0.0	119	58.267822	82.654999	0.001419	0.0
120	74.930946	101.484894	0.002303	0.0	120	58.030766	82.255806	0.001415	0.0
121	74.970680	101.524925	0.002305	0.0	121	58.229755	82.779060	0.001418	0.0

122 rows × 9 columns

In [30]:
```
#visualizing MAE, evaluation metric
fig, ax = plt.subplots(figsize=(20, 8))
ax.plot(goldmetrics["MAE"], '-o', label="Training Loss")
ax.plot(goldmetrics["MAE_val"], '-r', label="Validation Loss")
ax.legend(loc='center right', fontsize=16)
ax.tick_params(axis='both', which='major', labelsize=20)
ax.set_xlabel("Epoch", fontsize=28)
ax.set_ylabel("Loss", fontsize=28)
ax.set_title("Gold Validation Model Loss (MAE)", fontsize=28)
```

Out[30]:
```
Text(0.5, 1.0, 'Gold Validation Model Loss (MAE)')
```



For another evaluation metric, I calculated the MAPE (Mean Absolute Percentage Error). This percentage represents the average of absolute percentage errors to indicate forecast accuracy (Indeed Editorial Team, 2023).

In [31]:
```
#calculating MAPE for model accuracy
goldmean = golddf_val['y'].mean()
```

In [32]:
```
goldresult = 75.167763/goldmean *100
print("The MAPE of the gold model is: ", goldresult)

if goldresult >=26:
    print("Null Hypothesis: The accuracy of this model does not meet 75% accuracy")
else:
    print("Alternate Hypothesis: The accuracy of this model does meet 75% accuracy")
```

```
The MAPE of the gold model is:  4.088707833058266
Alternate Hypothesis: The accuracy of this model does meet 75% accuracy
```

The same steps were applied to the silver (**Section 4.c)**, platinum (**Section 4.d**), and palladium (**Section 4.e**) datasets.

Section 4.c
Silver

Silver Model & Validation

In [33]:
```
#NeuralProphet expects two columns: 'ds' for dates and 'y' for observed value
silverdf.rename(columns = {'date':'ds', 'close' : 'y'}, inplace=True)
```

In [34]:
```
#silver model and visualizations

#fitting model to a split dataset (90/10 : train/test)
m = NeuralProphet()
silverdf_train, silverdf_val = m.split_df(silverdf, freq='M', valid_p = 0.1)
silvermetrics = m.fit(silverdf_train, freq='M', validation_df=silverdf_val)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 96.253% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
WARNING - (NP.forecaster.fit) - When Global modeling with local normalization, metrics are displayed in normalized scale.
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 96.241% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to overri
de this.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 32
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 122
WARNING - (NP.config.set_lr_finder_args) - Learning rate finder: The number of batches (163) is too small than the required num
ber for the learning rate finder (243). The results might not be optimal.
```

Finding best initial lr: 100% 243/243 [00:02<00:00, 240.54it/s]

Epoch 122: 100%

122/122 [00:00<00:00, 170.80it/s, loss=0.0025, v_num=59, MAE_val=2.280, RMSE_val=2.890, Loss_val=0.0051, RegLoss_val=0.000, MAE=1.710, RMSE=2.410, Loss=0.00306, RegLoss=0.000]

(Please see visualization below)

In [37]:
```
#creating predictions
silverfuture = m.make_future_dataframe(silverdf, periods=12)
silverforecast = m.predict(silverfuture)

#view the first 5 entries of forecasted prices
silverforecast.head()
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 96.253% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
```

Predicting DataLoader 0: 100% 1/1 [00:00<00:00, 124.98it/s]

```
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

Out[37]:

	ds	y	yhat1	trend	season_yearly	season_weekly
0	2023-08-31	None	24.109432	28.813553	0.619215	-5.323334
1	2023-09-30	None	45.903011	28.905380	0.082477	16.915157
2	2023-10-31	None	23.583633	29.000267	-0.086218	-5.330416
3	2023-11-30	None	23.516205	29.092094	-0.252554	-5.323334
4	2023-12-31	None	38.569206	29.186981	-0.358715	9.740939

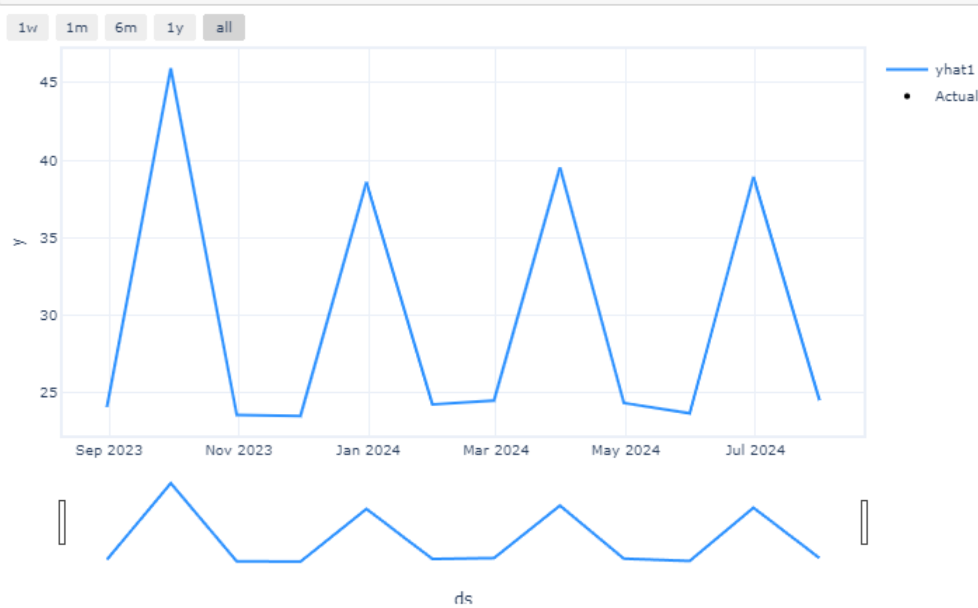
In [38]:
```
#view the last 5 entries of forecasted prices
silverforecast.tail()
```

Out[38]:

	ds	y	yhat1	trend	season_yearly	season_weekly
7	2024-03-31	None	39.519638	29.465519	0.313177	9.740939
8	2024-04-30	None	24.363960	29.557346	0.137032	-5.330416
9	2024-05-31	None	23.692505	29.652233	-0.592013	-5.367714
10	2024-06-30	None	38.906734	29.744061	-0.578266	9.740939
11	2024-07-31	None	24.546604	29.838943	0.012803	-5.305142

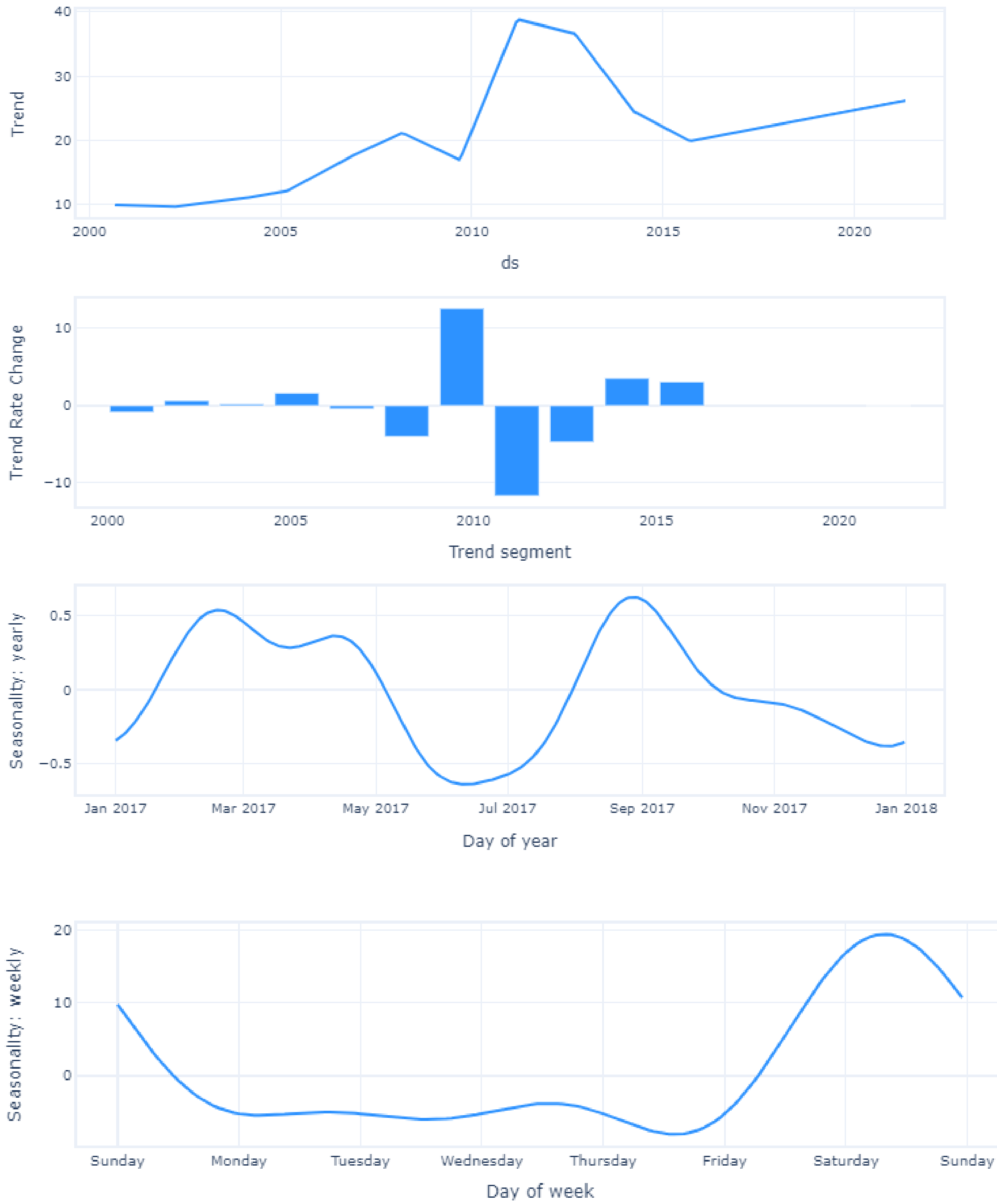
In [39]:
```
#visualization of prediction
m.plot(silverforecast)
```

Out[39]:



In [40]:
```
#visualized components
m.plot_parameters()
```

Out[40]:



In [36]:
```
#evaluation metrics
silvermetrics
```

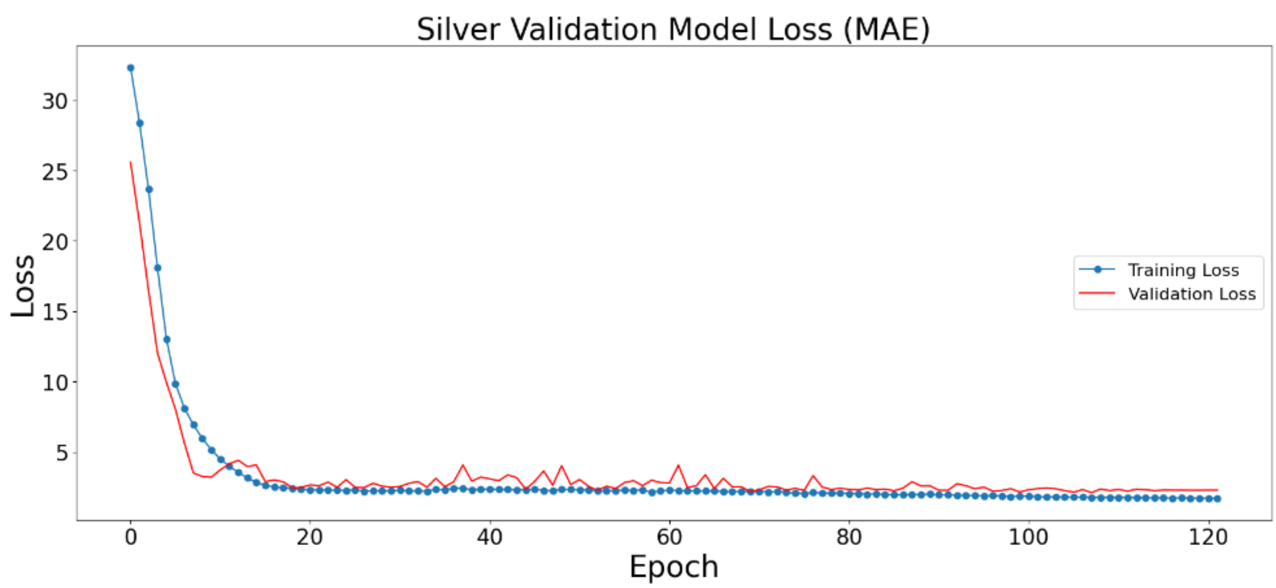
Out[36]:

	MAE_val	RMSE_val	Loss_val	RegLoss_val	epoch	MAE	RMSE	Loss	RegLoss
0	25.580793	31.756506	0.514968	0.0	0	32.345600	39.439766	0.518590	0.0
1	21.284040	26.235191	0.378855	0.0	1	28.409147	34.766003	0.425412	0.0
2	16.507843	19.992775	0.236811	0.0	2	23.688686	29.022619	0.315130	0.0
3	12.040068	14.784774	0.133163	0.0	3	18.105019	22.324593	0.194069	0.0
4	9.934315	11.218673	0.076965	0.0	4	13.031738	16.074493	0.100712	0.0
...	...	...	...	...	...	...	...	...	...
117	2.280636	2.889101	0.005104	0.0	117	1.723534	2.422128	0.003101	0.0
118	2.273276	2.885720	0.005092	0.0	118	1.715999	2.411256	0.003077	0.0
119	2.274243	2.886542	0.005095	0.0	119	1.708238	2.412714	0.003069	0.0
120	2.280257	2.888735	0.005103	0.0	120	1.712816	2.406723	0.003077	0.0
121	2.279687	2.888404	0.005102	0.0	121	1.706480	2.409130	0.003061	0.0

122 rows × 9 columns

In [41]:
```
#visualizing MAE, evaluation metric
fig, ax = plt.subplots(figsize=(20, 8))
ax.plot(silvermetrics["MAE"], '-o', label="Training Loss")
ax.plot(silvermetrics["MAE_val"], '-r', label="Validation Loss")
ax.legend(loc='center right', fontsize=16)
ax.tick_params(axis='both', which='major', labelsize=20)
ax.set_xlabel("Epoch", fontsize=28)
ax.set_ylabel("Loss", fontsize=28)
ax.set_title("Silver Validation Model Loss (MAE)", fontsize=28)
```

Out[41]: `Text(0.5, 1.0, 'Silver Validation Model Loss (MAE)')`



In [43]:
```
#calculating MAPE for model accuracy
silvermean = silverdf_val['y'].mean()
```

In [44]:
```
silverresult =2.279687/silvermean *100
print("The MAPE of the silver model is: ", silverresult)


if silverresult >=26:
    print("Null Hypothesis: The accuracy of this model does not meet 75% accuracy")
else:
    print("Alternate Hypothesis: The accuracy of this model does meet 75% accuracy"
```

```
The MAPE of the silver model is:  9.887798303308317
Alternate Hypothesis: The accuracy of this model does meet 75% accuracy
```

Section 4.d
Platinum

Platinum Model & Validation

In [45]:
```
#NeuralProphet expects two columns: 'ds' for dates and 'y' for observed value
platdf.rename(columns = {'date':'ds', 'close' : 'y'}, inplace=True)
```

In [46]:
```
#platinum model and visualizations

#fitting model to a split dataset (90/10 : train/test)
m = NeuralProphet()
platdf_train, platdf_val = m.split_df(platdf, freq='M', valid_p = 0.1)
platmetrics = m.fit(platdf_train, freq='M', validation_df=platdf_val)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.775% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
WARNING - (NP.forecaster.fit) - When Global modeling with local normalization, metrics are displayed in normalized scale.
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.709% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to overri
de this.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 32
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 125
WARNING - (NP.config.set_lr_finder_args) - Learning rate finder: The number of batches (148) is too small than the required num
ber for the learning rate finder (242). The results might not be optimal.
```

Finding best initial lr: 100% 242/242 [00:03<00:00, 195.30it/s]

Epoch 125: 100%

125/125 [00:00<00:00, 189.03it/s, loss=0.00249, v_num=60, MAE_val=70.10, RMSE_val=85.50, Loss_val=0.00206, RegLoss_val=0.000, MAE=80.30, RMSE=112.0, Loss=0.00284, RegLoss=0.000]

In [48]:
```
#creating predictions
platfuture = m.make_future_dataframe(platdf, periods=12)
platforecast = m.predict(platfuture)

#view the first 5 entries of forecasted prices
platforecast.head()
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.775% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
```

Predicting DataLoader 0: 100% 1/1 [00:00<00:00, 151.41it/s]

```
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

Out[48]:

	ds	y	yhat1	trend	season_yearly	season_weekly
0	2023-08-31	None	926.221680	796.043091	1.999288	128.179321
1	2023-09-30	None	151.427948	795.766113	-39.880630	-604.457520
2	2023-10-31	None	881.895447	795.479858	-42.872696	129.288239
3	2023-11-30	None	893.176392	795.202942	-30.205893	128.179321
4	2023-12-31	None	723.180054	794.916748	-34.333561	-37.403202

In [49]:
```
#view the last 5 entries of forecasted prices
platforecast.tail()
```

Out[49]:

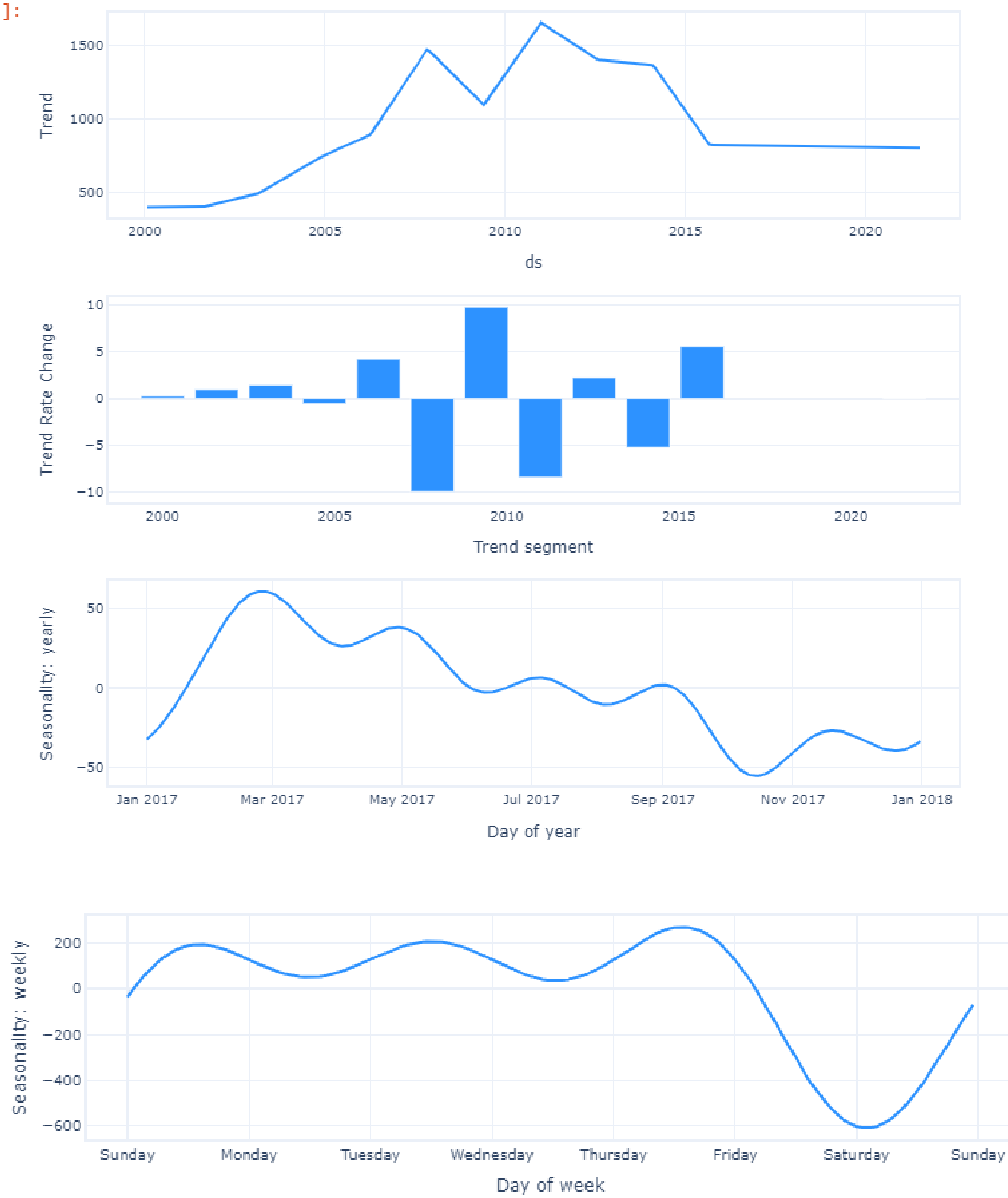
	ds	y	yhat1	trend	season_yearly	season_weekly
7	2024-03-31	None	783.663452	794.076660	26.990042	-37.403202
8	2024-04-30	None	961.301697	793.799683	38.213783	129.288239
9	2024-05-31	None	922.541870	793.513489	1.823134	127.205269
10	2024-06-30	None	761.633911	793.236572	5.800559	-37.403202
11	2024-07-31	None	912.168945	792.950378	-9.462717	128.681290

In [50]:
```
#visualization of prediction
m.plot(platforecast)
```

Out[50]:

In [51]:
```
#visualized components
m.plot_parameters()
```

Out[51]:



In [47]:
```
#evaluation metrics
platmetrics
```

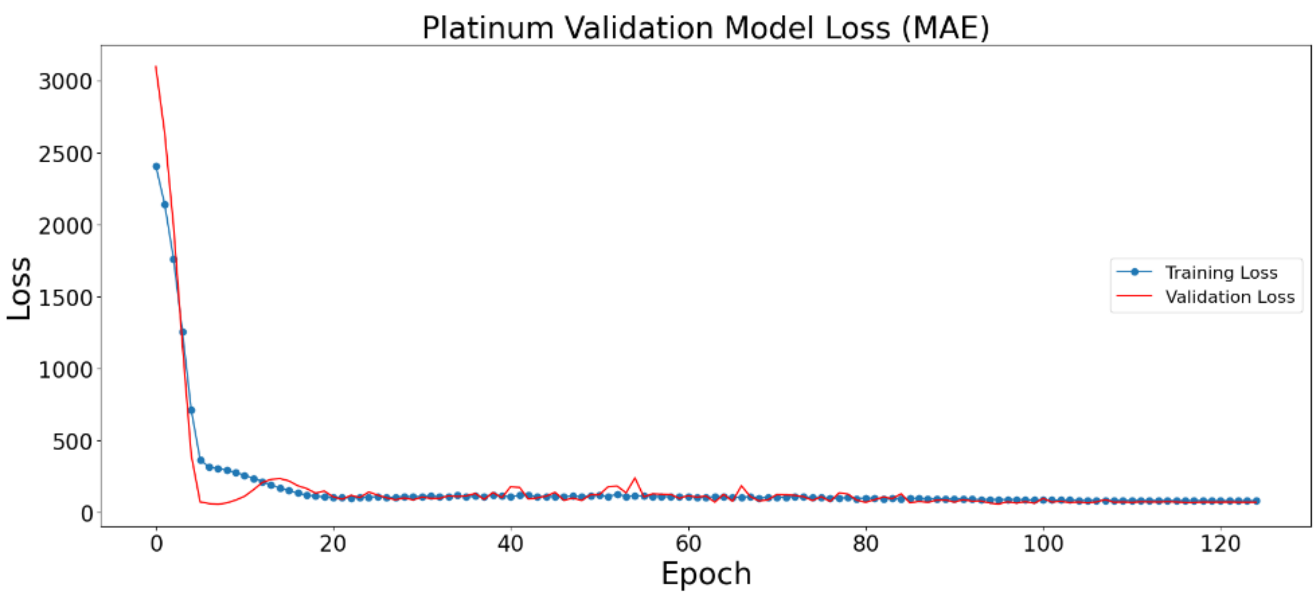
Out[47]:

	MAE_val	RMSE_val	Loss_val	RegLoss_val	epoch	MAE	RMSE	Loss	RegLoss
0	3098.565918	3346.454834	1.829101	0.0	0	2408.846191	2763.897217	1.106084	0.0
1	2637.992920	2904.004639	1.489025	0.0	1	2141.836182	2489.600830	0.947931	0.0
2	1991.578247	2273.272705	1.024357	0.0	2	1763.016602	2101.295410	0.727284	0.0
3	1166.517700	1429.988281	0.478832	0.0	3	1255.644653	1553.931396	0.448721	0.0
4	396.617218	552.873657	0.085557	0.0	4	709.695007	908.333374	0.179408	0.0
...	...	...	...	...	...	...	...	...	...
120	71.027992	86.721016	0.002120	0.0	120	80.263885	112.129036	0.002857	0.0
121	71.847237	87.442673	0.002155	0.0	121	80.564949	112.230888	0.002855	0.0
122	70.167709	85.609344	0.002066	0.0	122	80.087097	112.050529	0.002842	0.0
123	70.120934	85.491692	0.002060	0.0	123	80.344025	112.662415	0.002844	0.0
124	70.095779	85.483612	0.002060	0.0	124	80.268753	111.792969	0.002840	0.0

125 rows × 9 columns

In [52]:
```
#visualizing MAE, evaluation metric
fig, ax = plt.subplots(figsize=(20, 8))
ax.plot(platmetrics["MAE"], '-o', label="Training Loss")
ax.plot(platmetrics["MAE_val"], '-r', label="Validation Loss")
ax.legend(loc='center right', fontsize=16)
ax.tick_params(axis='both', which='major', labelsize=20)
ax.set_xlabel("Epoch", fontsize=28)
ax.set_ylabel("Loss", fontsize=28)
ax.set_title("Platinum Validation Model Loss (MAE)", fontsize=28)
```

Out[52]: `Text(0.5, 1.0, 'Platinum Validation Model Loss (MAE)')`



In [53]:
```
#calculating MAPE for model accuracy
platmean = platdf_val['y'].mean()
platresult =70.095779/platmean *100
print("The MAPE of the platinum model is: ", platresult)


if platresult >=26:
    print("Null Hypothesis: The accuracy of this model does not meet 75% accuracy")
else:
    print("Alternate Hypothesis: The accuracy of this model does meet 75% accuracy")
```

```
The MAPE of the platinum model is:  7.15312596065906
Alternate Hypothesis: The accuracy of this model does meet 75% accuracy
```

Section 4.e
Palladium

Palladium Model & Validation

In [50]:
```
#NeuralProphet expects two columns: 'ds' for dates and 'y' for observed value
palldf.rename(columns = {'date':'ds', 'close' : 'y'}, inplace=True)
```

In [51]:
```
#palladium model and visualizations

#fitting model to a split dataset (80/20 : train/test)
m = NeuralProphet()
palldf_train, palldf_val = m.split_df(palldf, freq='M', valid_p = 0.2)
pallmetrics = m.fit(palldf_train, freq='M', validation_df=palldf_val)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.869% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
WARNING - (NP.forecaster.fit) - When Global modeling with local normalization, metrics are displayed in normalized scale.
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.796% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to overri
de this.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 32
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 127
WARNING - (NP.config.set_lr_finder_args) - Learning rate finder: The number of batches (137) is too small than the required num
ber for the learning rate finder (241). The results might not be optimal.
```

Finding best initial lr: 100% 241/241 [00:01<00:00, 250.00it/s]

Epoch 127: 100%

127/127 [00:00<00:00, 229.26it/s, loss=0.00662, v_num=48, MAE_val=466.0, RMSE_val=518.0, Loss_val=0.271, RegLoss_val=0.000, MAE=79.20, RMSE=109.0, Loss=0.00698, RegLoss=0.000]

In [53]:
```
#creating predictions
pallfuture = m.make_future_dataframe(palldf, periods=12)
pallforecast = m.predict(pallfuture)

#view the first 5 entries of forecasted prices
pallforecast.head()
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency B corresponds to 95.869% of the data.
WARNING - (NP.df_utils._infer_frequency) - Defined frequency M is different than major frequency B
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
INFO - (NP.df_utils._infer_frequency) - Major frequency M corresponds to [91.667]% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - M
```

Predicting DataLoader 0: 100% 1/1 [00:00<00:00, 481.61it/s]

```
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

Out[53]:

	ds	y	yhat1	trend	season_yearly	season_weekly
0	2023-08-31	None	1365.468262	1016.805298	-14.106594	362.769653
1	2023-09-30	None	40.221832	1023.120789	-12.846444	-970.052490
2	2023-10-31	None	1377.234497	1029.646851	-14.662849	362.250427
3	2023-11-30	None	1395.311768	1035.962402	-3.420250	362.769653
4	2023-12-31	None	217.306427	1042.488525	19.365164	-844.547363

In [54]:
```
#view the last 5 entries of forecasted prices
pallforecast.tail()
```

Out[54]:

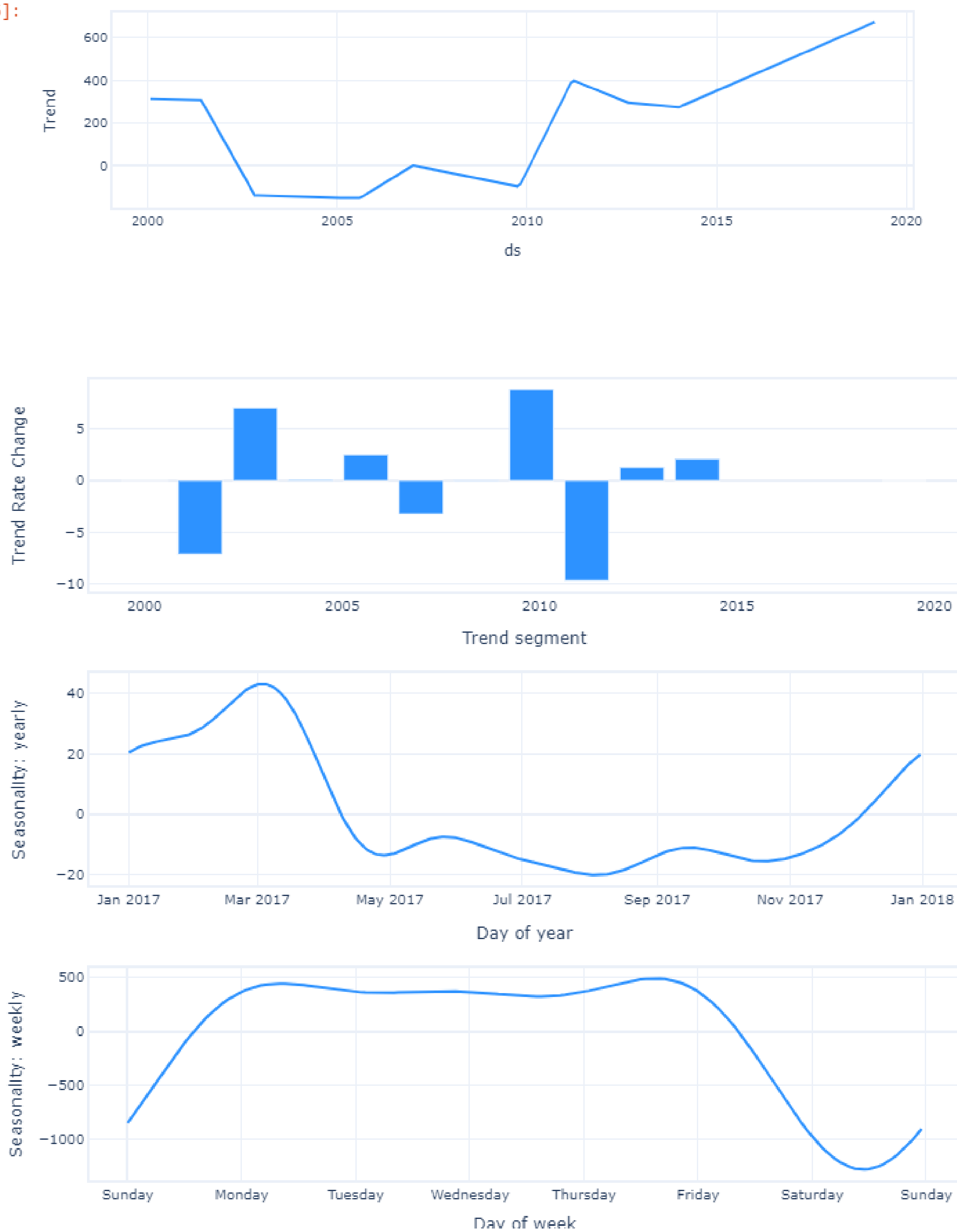
	ds	y	yhat1	trend	season_yearly	season_weekly
7	2024-03-31	None	230.191162	1061.645752	13.092800	-844.547363
8	2024-04-30	None	1416.676025	1067.961182	-13.535585	362.250427
9	2024-05-31	None	1431.277344	1074.487305	-7.873512	364.663605
10	2024-06-30	None	221.272461	1080.802979	-14.983175	-844.547363
11	2024-07-31	None	1428.792358	1087.328857	-20.020420	361.483978

In [55]:
```
#visualization of prediction
m.plot(pallforecast)
```

Out[55]:

In [56]:
```
#visualized components
m.plot_parameters()
```

Out[56]:



```
In [52]: #evaluation metrics
         pallmetrics
```

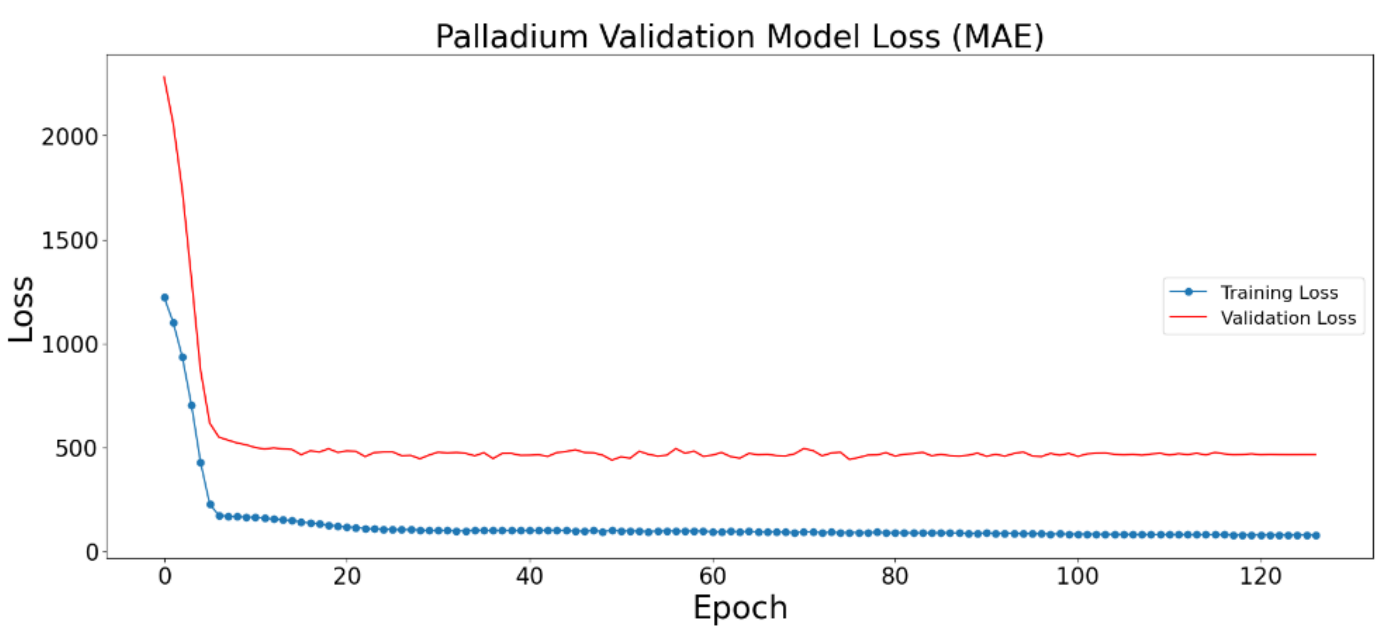
Out[52]:

	MAE_val	RMSE_val	Loss_val	RegLoss_val	epoch	MAE	RMSE	Loss	RegLoss
0	2282.214844	2431.922363	2.164549	0.0	0	1223.385376	1439.418823	0.778515	0.0
1	2058.609863	2198.680664	1.904866	0.0	1	1102.158569	1302.871216	0.672859	0.0
2	1736.322754	1861.595337	1.532286	0.0	2	935.169006	1113.254028	0.530456	0.0
3	1311.546753	1410.752075	1.048065	0.0	3	704.562927	846.906677	0.341790	0.0
4	869.846375	938.571167	0.579217	0.0	4	427.622345	522.124084	0.143976	0.0
...	...	...	...	...	...	...	...	...	...
122	465.798737	518.112061	0.270909	0.0	122	79.310799	109.026314	0.007004	0.0
123	465.428131	517.735291	0.270413	0.0	123	79.264587	109.385574	0.006997	0.0
124	465.615417	517.899780	0.270633	0.0	124	79.255653	108.857712	0.006984	0.0
125	465.832520	518.115723	0.270904	0.0	125	79.216652	109.122101	0.006980	0.0
126	465.773529	518.057373	0.270838	0.0	126	79.210518	108.712318	0.006978	0.0

127 rows × 9 columns

```
In [57]: #visualizing MAE, evaluation metric
         fig, ax = plt.subplots(figsize=(20, 8))
         ax.plot(pallmetrics["MAE"], '-o', label="Training Loss")
         ax.plot(pallmetrics["MAE_val"], '-r', label="Validation Loss")
         ax.legend(loc='center right', fontsize=16)
         ax.tick_params(axis='both', which='major', labelsize=20)
         ax.set_xlabel("Epoch", fontsize=28)
         ax.set_ylabel("Loss", fontsize=28)
         ax.set_title("Palladium Validation Model Loss (MAE)", fontsize=28)
```

```
Out[57]: Text(0.5, 1.0, 'Palladium Validation Model Loss (MAE)')
```



```
In [96]: #calculating MAPE for model accuracy
         pallmean = palldf_val['y'].mean()
```

```
In [106]: pallresult = 465.615784/pallmean *100
          print("The MAPE of the palladium model is: ", pallresult)
          if pallresult >= 26:
              print("Null Hypothesis: The accuracy of this model does not meet 75% accuracy")
          else:
              print("Alternate Hypothesis: The accuracy of this model does meet 75% accuracy")

          The MAPE of the palladium model is:  23.404944522451814
          Alternate Hypothesis: The accuracy of this model does meet 75% accuracy
```

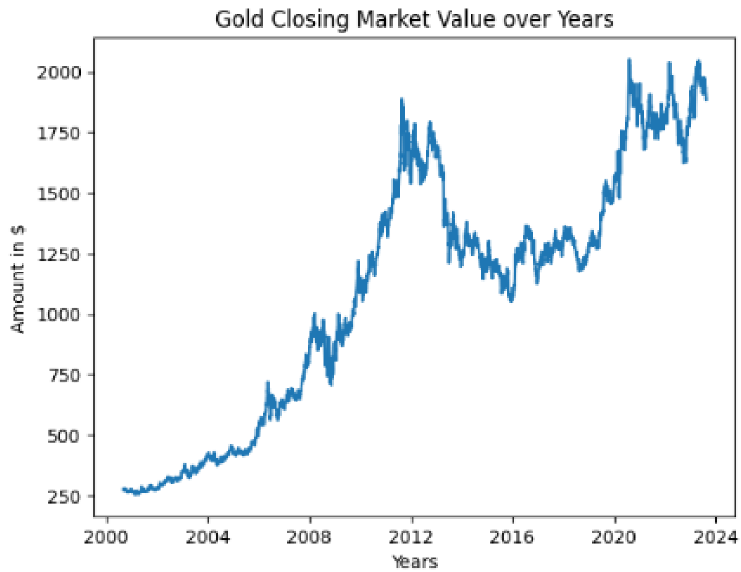
Section 4.f

To accurately perform time series analysis, I sought one of the newest and researched methods: Neural Prophet. . Neural Prophet is a library created with Facebook's user-friendly and powerful Prophet library and Pytorch as a foundation with modern neural networking. By taking a strong time series tool (i.e. Prophet) and advancing it further with automated parameters and increased dataset adaptability, Neural Prophet is a highly effective and accurate forecasting tool.

Using Neural Prophet has multiple advantages. A significant advantage is that feedforward neural networks allow deeper, more detailed model training due feedback processes improving predictions over time. This process can have the potential to be a disadvantage, however, if there is not enough data to be pushed through the model. Feedforward neural networks can only process data in one direction which means that if the model runs out of data to train on, the model will not be optimal and accurate for intended purposes (Amazon, 2023). For this reason, I ensured during the data collecting process that my individual, smaller datasets would have enough data for an adequate model.

Data Summary and Implications
Section 5.a
Gold

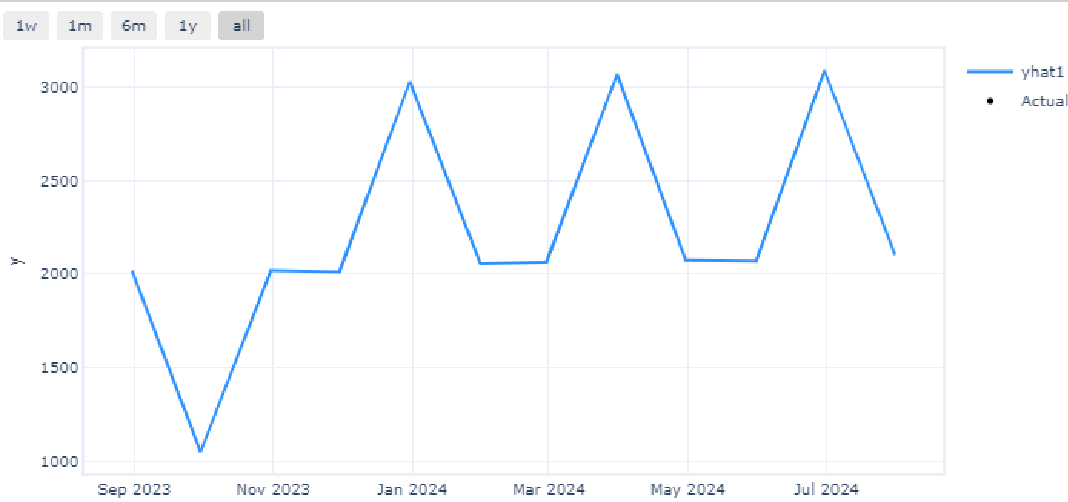
Visualization of past and current closing market prices:



Visualization of forecasted closing market prices:

```
m.plot(goldforecast)
```

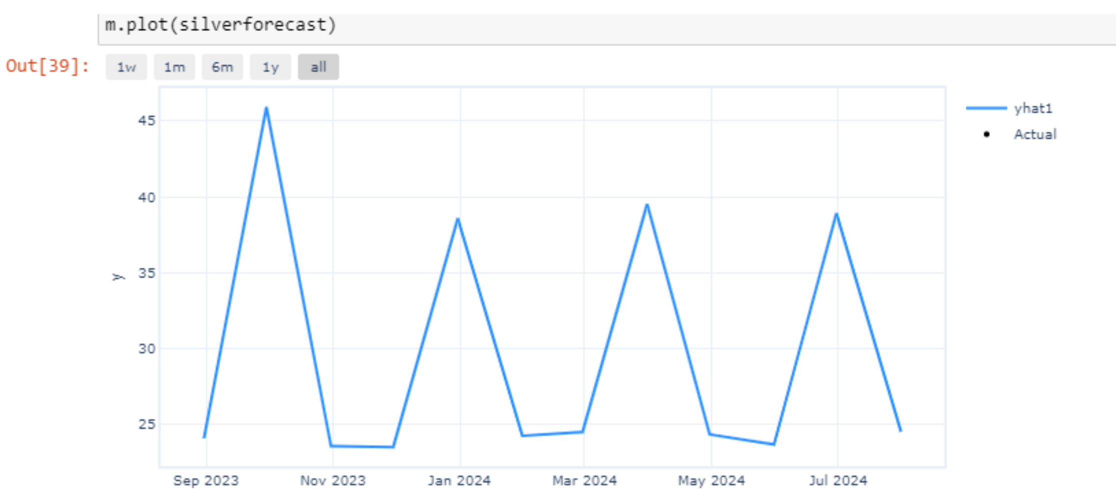
Out[28]:



Section 5.b
Silver

Visualization of past and current closing market prices:

Visualization of forecasted closing market prices:



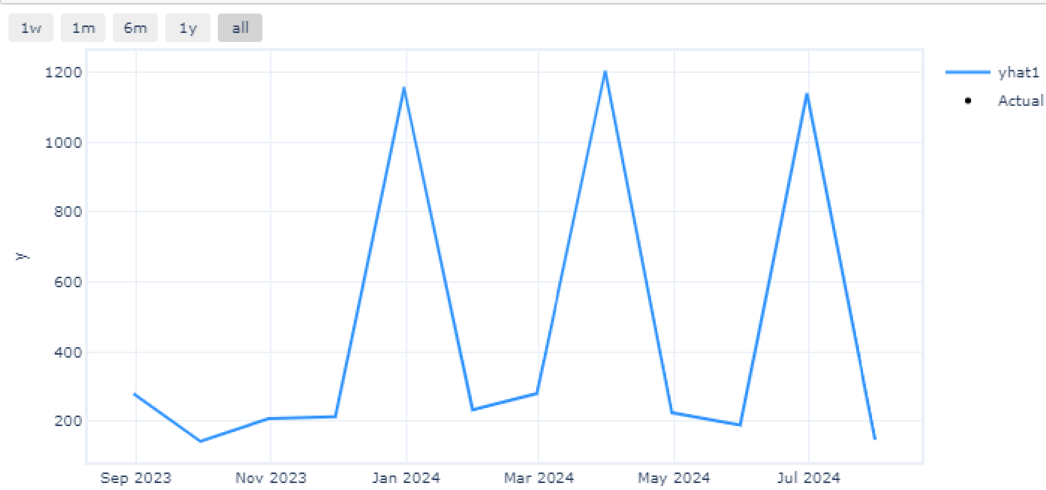
Section 5.c
Platinum

Visualization of past and current closing market prices:

Visualization of forecasted closing market prices:

```
In [46]: #visualization of prediction
         m.plot(platforecast)
```

Out[46]:



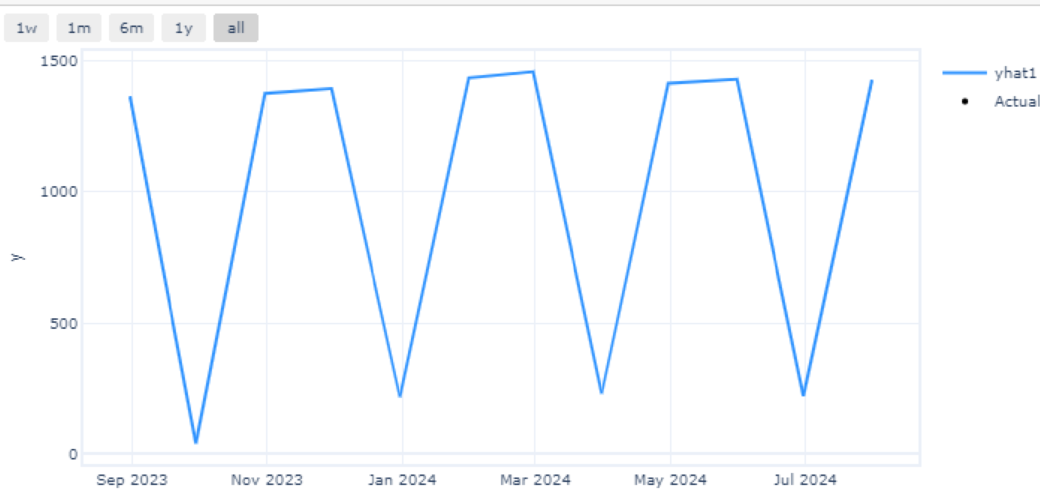
Section 5.d
Palladium

Visualization of past and current closing market prices:

Visualization of forecasted closing market prices:

```
In [55]: #visualization of prediction
         m.plot(pallforecast)
```

Out[55]:



Section 5.e
Implications

Every time the model is fitted and trained, a new model will be outputted. As a result, the implications stated in this report are solely formed from the captured screenshots of a chosen model.

Gold market closing price forecasts indicate that new maximum-priced predictions for approximately $3000. The peaks and valleys also indicate that active trading can be beneficial for the average interval of $1000 between forecasted peaks and valleys. This model has an accuracy of 95.9% based off the validation dataset and evaluation. As a result, this model achieved the accuracy goal of 75% and above.

For silver, the price is the lowest of all the precious medals with an approximate forecasted price of $35-$40. Silver has not had a significant increase in price since the early 2010s and forecasts do not suggest high likelihood of profit with silver investments. This model has an accuracy of 90% based off the validation dataset and evaluation. As a result, this model achieved the accuracy goal of 75% and above.

Thirdly, platinum forecasts indicate that while platinum is no longer as expensive and profitable from previous years, the fluctuations and price intervals are significant with a range from approximately $200 to $1200. Understanding the peaks and valleys can be profitable for active, short-term trading. This model has an accuracy of 92.8% based off the validation dataset and evaluation. As a result, this model achieved the accuracy goal of 75% and above.

Lastly, palladium forecasted closing market prices are the most similar to the platinum forecasts. Palladium also has a visualized maximum price decrease but has strong fluctuations that make this precious metal profitable for active trading as well. This model has an accuracy of 76.5% based off the validation dataset and evaluation. As a result, this model achieved the accuracy goal of 75% and above.

A limitation of my analysis is Neural Prophet's feed-forward neural network in which machine learning loops are not utilized. Large amounts of data are required for a functional, accurate model. Additionally, model overfitting can occur more easily with this model than comparable forecasting models. To combat this limitation, the forecast dataset was evaluated for model loss between the training dataset and the testing dataset. My model was overfitted with an 80% / 20% dataset split for most of the models and was corrected when switched to a 90% / 10% dataset split.

Section 5.f
Model Accuracy

My research question "To what extent can Neural Prophet accurately predict variable "cost" for precious metals "Gold", "Silver", "Platinum", and "Palladium" over a forecast period of 365 days?" and accompanying hypothesis of Neural Prophet having a minimum of 75% accuracy rely on an evaluation metric. I utilized two forecast error metrics: MAE and MAPE. MAE, Mean Absolute Error, is a scale-dependent metric in which the difference between actual values and predicted values calculates model-specific accuracy. This metric is simple to compute and highly accurate, however, the scale-dependency does not allow this metric to compare model accuracy with another model. As a result, I utilized MAPE, Mean Absolute Percentage Error, to evaluate model accuracy. This metric is scale-independent and can compare my forecast prediction accuracy with my four models. MAPE is calculated by taking the MAE and dividing by the mean of the train dataset i.e., the mean of the predicted closing market prices (Indeed Editorial Team, 2023). All four of the models produced a MAPE lower than 25% which dictate that less than 25% difference is found between the predictions and forecasts.

Within the context of your research question, recommend a course of action based on your results. Then propose **two** directions or approaches for future study of the data set.

Section 5.g
Recommendations and Future Approaches

Neural Prophet forecasting model proved my hypothesis that Neural Prophet can accurately forecast precious metals over a 365-day period with a minimum of 75% accuracy. In order from the most accurate models to lowest are the following: gold forecast, platinum forecast, silver forecast, and palladium forecast. In combination of potential for profit, the gold forecasts are accurate and profitable for interested parties i.e. stock traders and investors. To better increase model use and actionability, two future directions are suggested. The first direction is to perform model updates with each interval period of new data collected. As an example, re-run the model every month for more accurate predictions as new data is collected. A second approach is to decrease the forecast period from 365 days to smaller forecast periods such as quarterly, 30 days, and daily for an increased accuracy. Accuracy becomes worse as forecast period lengths are extended so users and analysts can benefit from forecasting with shorter periods. Overall, Neural Prophet met the hypothesis with approximately 5000 entries for each precious metal to determine a minimum of 75% accuracy for all models.

Sources

I. YahooFinance. (2023). *Commodities futures prices & day charts*. Yahoo! Finance. https://finance.yahoo.com/commodities/

   a. (YahooFinance, 2023) in Section 1.a

II. Schwab. (2023). *Gold futures*. Schwab Brokerage. https://www.schwab.com/futures/gold

   a. (Schwab, 2023) in Section 1.a

III. Talaviya, A. (2022, November 14). *Time-series analysis and stock price forecast modeling: All you need to know*. Medium. https://medium.com/@avikumart_/time-series-analysis-and-stock-price-forecast-modeling-all-you-need-to-know-fe66c06e50ae#:~:text=time%2Dseries%20data%20can%20be,learned%20in%20the%20previous%20section.&text=Time%2Dseries%20analysis%20is%20the%20study%20of%20data%20sets%20over%20time.

   a. (Talaviya, 2022) in Section 1.a

IV. YouTube. (2022). *What is the Prophet Model*. *YouTube*. Retrieved August 28, 2023, from https://www.youtube.com/watch?v=2XFro0nIHQM.

   a. (LaBarr, 2022) in Section 1.a

V. Servera, G. (2023, August 23). *Gold, silver & precious metals futures daily data*. Kaggle. https://www.kaggle.com/datasets/guillemservera/precious-metals-data

   a. (Servera, 2023) in Section 2.a and Section 3.a

VI. Luna, J. C. (2022, December 28). *Python vs R for data science: Which should you learn?*. DataCamp. https://www.datacamp.com/blog/python-vs-r-for-data-science-whats-the-difference

a. (Luna, 2022) in Section 3.b

VII. Valeriy Manokhin, P. (2023, January 26). *Python vs R for time-series forecasting*. Medium. https://valeman.medium.com/python-vs-r-for-time-series-forecasting-395390432598

a. (Valeriy Manokhin, 2023) in Section 3.b

VIII. Samal, C. (2022a, February 25). *Time series tutorial using NeuralProphet*. Medium. https://medium.com/analytics-vidhya/time-series-tutorial-using-neuralprophet-e918a1b437ed#:~:text=NeuralProphet%20is%20a%20neural%2Dnetwork,passengers%20from%201949%20to%201960.

a. (Samal, 2022) in Section 4.a

IX. Acharya, S. (2021, June 15). *What are RMSE and Mae?*. Medium. https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383

a. (Acharya, 2021) in Section 4.b

X. Indeed Editorial Team. (2023, February 3). *What is Mape? A guide to mean absolute percentage error*. Indeed.com. https://www.indeed.com/career-advice/career-development/what-is-mape

a. (Indeed Editorial Team, 2023) in Section 4.b and Section 5.e

XI. Amazon. (2023). *What is* ... https://aws.amazon.com/what-is/neural-network/#:~:text=Neural%20network%20training%20is%20the,process%20unknown%20inputs%20more%20accurately.

a. (Amazon, 2023) in Section 4.f