On-Chain Forecasting

This notebook is intended to be an extension of the on-chain-regression models previously used. Specifically, I will go into the different methods of forecasting and a small amount of their statistics. Similarly to the regression notebook, I am looking to aggregate syntax and usage for forecasting models. I do not predict that seasonality will work very well; however, having this notebook will easily allow me to go back and quickly understand forecasting on other datasets.

Too Long Didn't Read

- Went through a complete exploratory data analysis -- cleaned, dropped, and fixed data in order to best answer the questions at hand.
- Set up forecasting methods (Holt, Exponential smoothing, simple exponential smoothing) to attempt to forecast price over a few timeframes.
- Forecasting included simulations, using measures of center of these simulations painted a more clear picture of how multiplicative or additive settings in the model may change outputs.

Issues

- Mathematically forecasting doesn't take into account real world events like the eth beacon chain merge.

Future applications of stats models and on-chain data

- Using a clustering model, or outlier detection, to find abnormal weeks or months with on chain data.
- This new model could use PCA, or generally, more methods of data tranformation in order to more accurately

In [143]:
```
#Imports and data
import numpy as np
import pandas as pd
import missingno as msno
import statistics
import json
import requests
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, make_scorer, r2_score
from sklearn.ensemble import RandomForestRegressor
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, H
from datetime import date
import warnings
warnings.filterwarnings('ignore')

# Glassnode API to create a dataframe.

API_KEY = '2AWlM3FJda1LoK0Eq4pDADN96d6'

urls = ['https://api.glassnode.com/v1/metrics/addresses/count',
       'https://api.glassnode.com/v1/metrics/addresses/active_count',
       'https://api.glassnode.com/v1/metrics/addresses/new_non_zero_count',
       'https://api.glassnode.com/v1/metrics/mining/difficulty_latest',
       'https://api.glassnode.com/v1/metrics/mining/hash_rate_mean',
       'https://api.glassnode.com/v1/metrics/addresses/min_1_count',
       'https://api.glassnode.com/v1/metrics/addresses/min_10_count',
       'https://api.glassnode.com/v1/metrics/addresses/min_1k_count',
       'https://api.glassnode.com/v1/metrics/market/price_usd_close']

data = []
for url in urls:
    label = url.split('/')[-1]
    res = requests.get(url, params = {'a':'ETH','api_key':API_KEY})
    df = pd.read_json(res.text, convert_dates=['t'])
    df.set_index('t', inplace = True)
    df.rename(columns = {'v':label}, inplace = True)
    data.append(df)
eth_df = pd.concat(data, axis=1)
```

In [144]:
```
eth_df.describe()
```

Out[144]:

	count	active_count	new_non_zero_count	difficulty_latest	hash_rate_mean	min_1_(
count	2.547000e+03	2547.000000	2547.000000	2.547000e+03	2.547000e+03	2.18200(
mean	5.321660e+07	267442.731841	61592.233608	3.465860e+15	2.470402e+14	6.58459:
std	4.979418e+07	193215.173150	50174.689875	3.825861e+15	2.796223e+14	4.54547(
min	9.203000e+03	0.000000	41.000000	3.382792e+11	1.910758e+10	9.20200(
25%	1.317367e+06	42611.000000	13056.000000	3.196885e+14	2.041062e+13	7.03247{
50%	4.421414e+07	263038.000000	58085.000000	2.375547e+15	1.679992e+14	9.25784{
75%	8.802063e+07	426776.000000	90845.000000	3.517995e+15	2.514803e+14	1.01258(
max	1.568754e+08	794922.000000	348434.000000	1.545466e+16	1.064275e+15	1.24559(

Issues with previous Regression notebook:

There were tons of nan values that essentially chopped the dataset in half. The worst part was that many of these nan values were at the tail end of the dataset. This left us with price data that was from nearly a year ago. This renders the entire notebook pretty useless: who cares about price predictions a year ago? It was fine to initially work with just to validate the different algorithms used for continious value regression: which random forest won. Also did not tune hyperparameters for any of the models. I bet we could get some pretty sweet results with tuning.

Response to issues in past notebook:

Since I am forecasting, I would like to have as up to date as up to date as possible. This may mean dropping columns with too many nans or nans that directly interfere with the up-to-dateness of the data. Would like to incorporate more exploratory data analysis in order to validate the up-to-dateness but also the statistical assumptions of linear regression. Using a random forest regressor as the only model will keep this notebook short and sweet. Also tuning some hyperparameters of the model. GET STOKED!

Exploratory Data Analysis:

Structure Investigation:

Exploring the shape and dtypes of the dataset. dtypes will all be floats for this particular dataset. If they were strings or booleans you could convert them into floats in order to be used as features for models. One-hot-encoder comes to mind.

In [145]:
```
#General shape of the DataFrame Matrix:
print(f'Rows-->{eth_df.shape}<--Columns')
print(f'Count of Column data types:{pd.value_counts(eth_df.dtypes)}')
```

```
Rows-->(2548, 9)<--Columns
Count of Column data types:float64    9
dtype: int64
```

In [146]:
```
#Another way to easily present this information
#Shows shape, dtype, datetime index, frequency, and non-null value count
eth_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2548 entries, 2015-07-30 to 2022-07-20
Freq: D
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   count              2547 non-null   float64
 1   active_count       2547 non-null   float64
 2   new_non_zero_count 2547 non-null   float64
 3   difficulty_latest  2547 non-null   float64
 4   hash_rate_mean     2547 non-null   float64
 5   min_1_count        2182 non-null   float64
 6   min_10_count       2182 non-null   float64
 7   min_1k_count       2182 non-null   float64
 8   price_usd_close    2539 non-null   float64
dtypes: float64(9)
memory usage: 199.1 KB
```

This is done to confirm the suspected data type. There could have been a previous issue in loading the DataFrame that would lead to some types being not of the correct data type.

Quality Investigation:

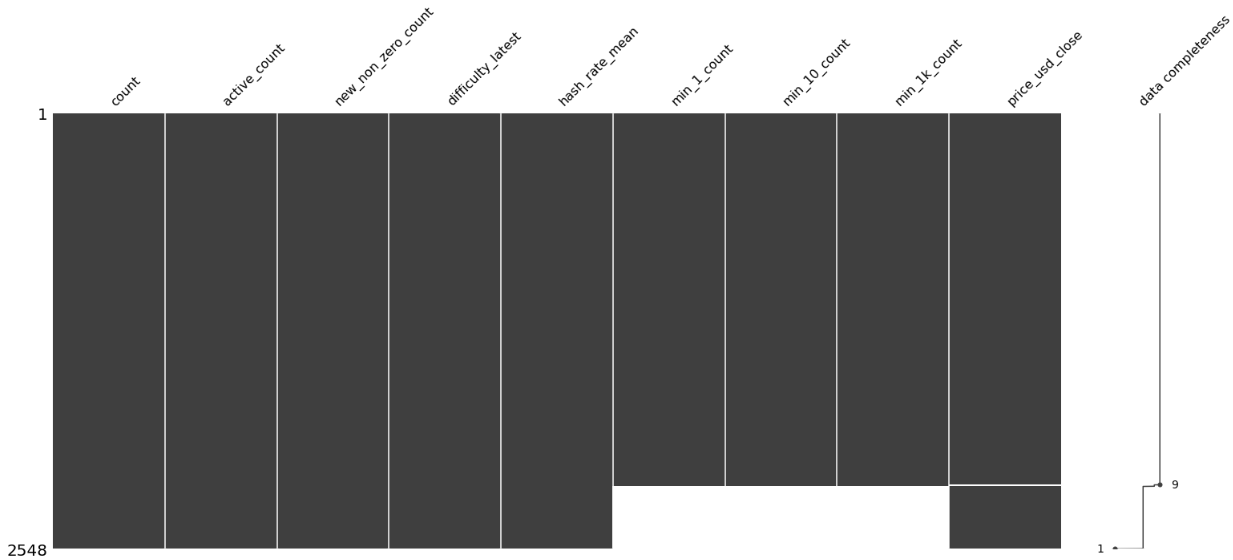
Confirming the quality of the dataset. Important to understand duplicates, missing values, unwanted entries, or recording erros.

In a case with categorical, binary, ordinal, categorical, or continuous variables slightly more will need to be done in order to understand the quality of data.

This includes checking duplicates agaisnt a primary key; however, we have no such primary key in this DataFrame.

In [147]:
```
msno.matrix(eth_df, labels = True, sort = 'descending')
```

Out[147]: <AxesSubplot:>



We can see that there are are nan values from min_1_count, min_10_count, and min_1k count from index 2000 to essentially the end of the DataFrame. This will severly hurt our ability to forecast a more recent price. It is likely in our best interest to remove these from the dataset. in addition to any nan values that are few and far between after dropping wallet value counts.

In [148]:
```
eth_df = eth_df.drop(columns = ['min_1_count', 'min_10_count', 'min_1k_coun
```

In [149]:
```
dates = [pd.to_datetime(d, format = '%Y%m%d') for d in eth_df.index]
eth_df['date'] = dates
eth_df = eth_df.set_index(eth_df['date'])
eth_df
```

Out[149]:

	count	active_count	new_non_zero_count	difficulty_latest	hash_rate_mean	price_usd_c
date						
2015-08-08	10641.0	800.0	353.0	1.606016e+12	9.644985e+10	0.76
2015-08-09	10894.0	731.0	253.0	1.741399e+12	1.013969e+11	0.71
2015-08-10	11543.0	997.0	649.0	1.948102e+12	1.116431e+11	0.70
2015-08-11	13432.0	2339.0	1889.0	2.171897e+12	1.240757e+11	1.08
2015-08-12	13744.0	904.0	312.0	2.248238e+12	1.308930e+11	1.21
...	...	...	...	...	...	
2022-07-15	156620572.0	544139.0	66121.0	1.165769e+16	8.685408e+14	1233.47
2022-07-16	156680292.0	581997.0	59720.0	1.184357e+16	8.787750e+14	1354.16
2022-07-17	156736689.0	549322.0	56397.0	1.140954e+16	8.677008e+14	1347.98
2022-07-18	156805778.0	505030.0	69089.0	1.162624e+16	8.749357e+14	1567.85
2022-07-19	156875419.0	486985.0	69641.0	1.190754e+16	8.921721e+14	1541.39

2538 rows × 7 columns

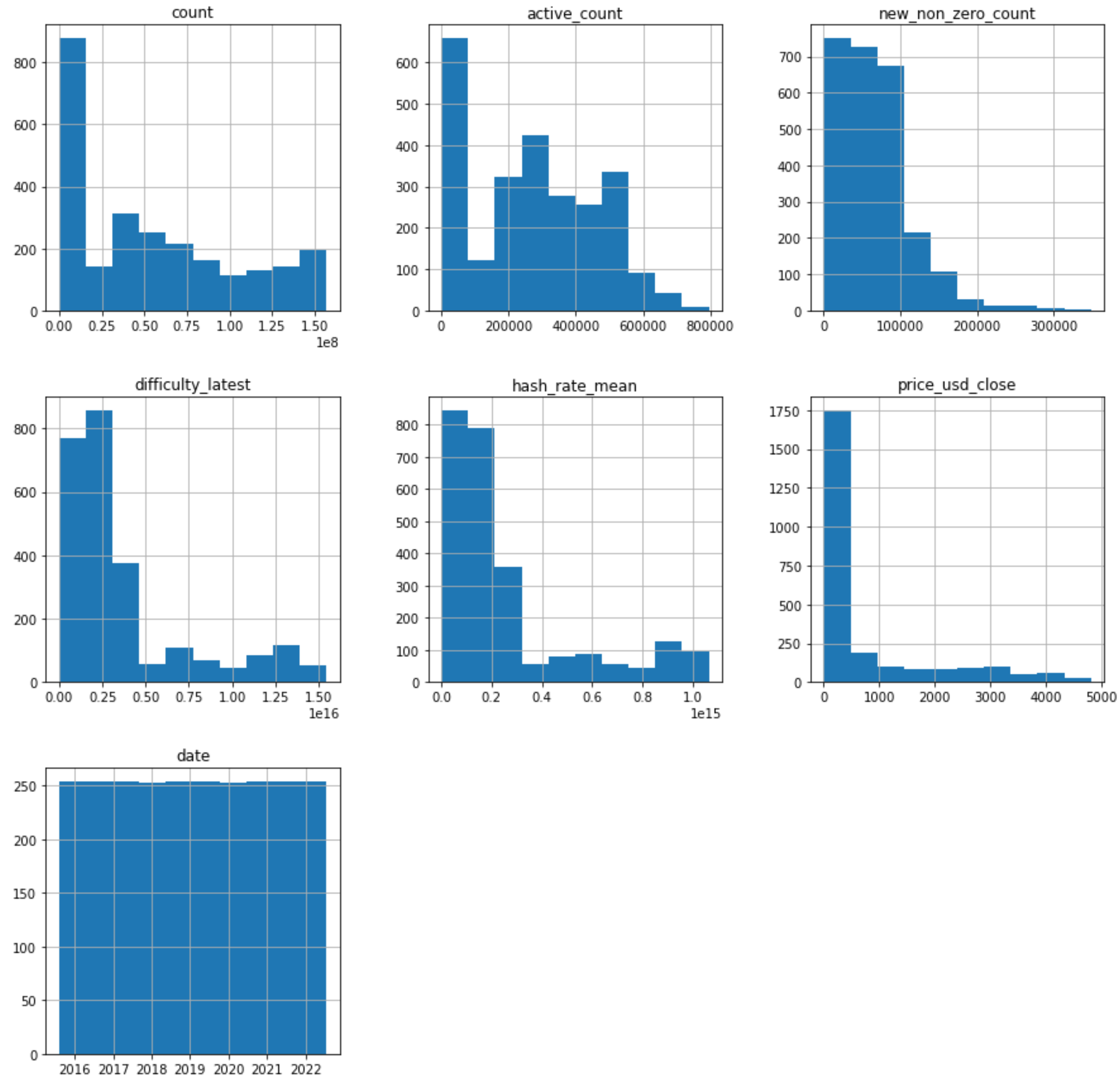
From this, we can notice that each of the columns of the dataframe, generally increase as time goes on. This makes sense due to the increasing demand on the eth network. These subplots allow us to further examine the assumptions of linear regression.

Content Investigation:

This section is to understand the content of the features and how they may be related.

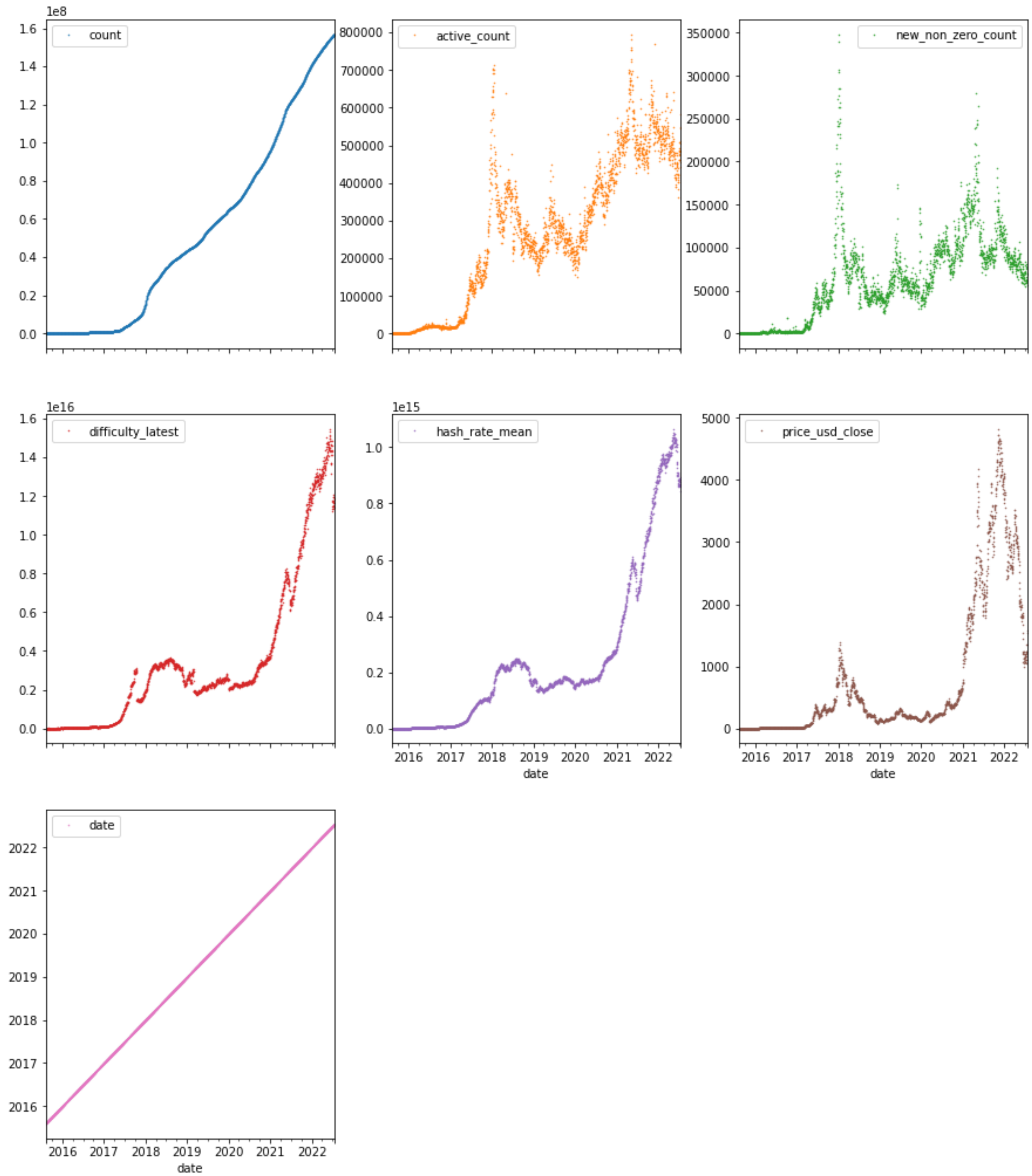
To do this, I will use histogram and line chart subplots.

In [150]:
```
eth_df.hist(bins = 10, figsize = (15,15));
```



In [151]:

```
eth_df.plot(lw = 0,
            marker = '.',
            subplots = True,
            layout = (5,3),
            figsize = (15,30),
            markersize = 1);
```



It seems that many of these features have a more or less linear relationship. Examining the correlation through a heat map is a good way to quanitfy the relationship. For this analysis go down to section with the linear regression assumptions below the fitting of the model.

Graphical and goodness of fit functions

In [152]:
```
def model_fit(y_test, preds):
    ''''
    Function to print Mean Squared Error,
    Root Mean Squared Error, and r^2.
    Need y_test and predictions from model and train test split
    '''''
    mse = mean_squared_error(y_test, preds)
    print(f'Mean Squared Error: {mse}')
    rmse = np.sqrt(mse)
    print(f'Root Mean Squared Error: {rmse}.\nSimply put, this means that e
    r2 = r2_score(y_test, preds)
    print("r^2:", r2)
```

In [153]:
```
def preds_act_plot(y_test, preds):
    ''''
    Function to linearly plot
    predicted price vs actual price in the model.
    Need y_test and predictions from model and train test split
    '''''
    plt.figure(figsize=(8, 6), dpi=80)
    plt.scatter(y_test, preds)
    plt.xlabel('Actual Labels')
    plt.ylabel('Predicted Labels')
    plt.title('Predicted vs Actual eth Price')
    z = np.polyfit(y_test, preds, 1)
    p = np.poly1d(z)
    plt.plot(y_test,p(y_test), color='magenta')
    plt.show()
```

In [154]:
```
def price_date_plot(preds):
    ''''
    Function to time-series plot predicted vs actual prices
    need predictions from model.predict(X_test)
    '''''
    plt.figure(figsize=(8, 6), dpi=80)
    preds = model.predict(X)
    plt.plot(eth_df.index, preds, label = 'predicted', color = 'red')
    plt.plot(eth_df.index,eth_price, label = 'actual', color = 'blue')
    plt.legend()
    plt.xlabel('Date')
    plt.ylabel('eth price')
    plt.title('eth price vs date')
```

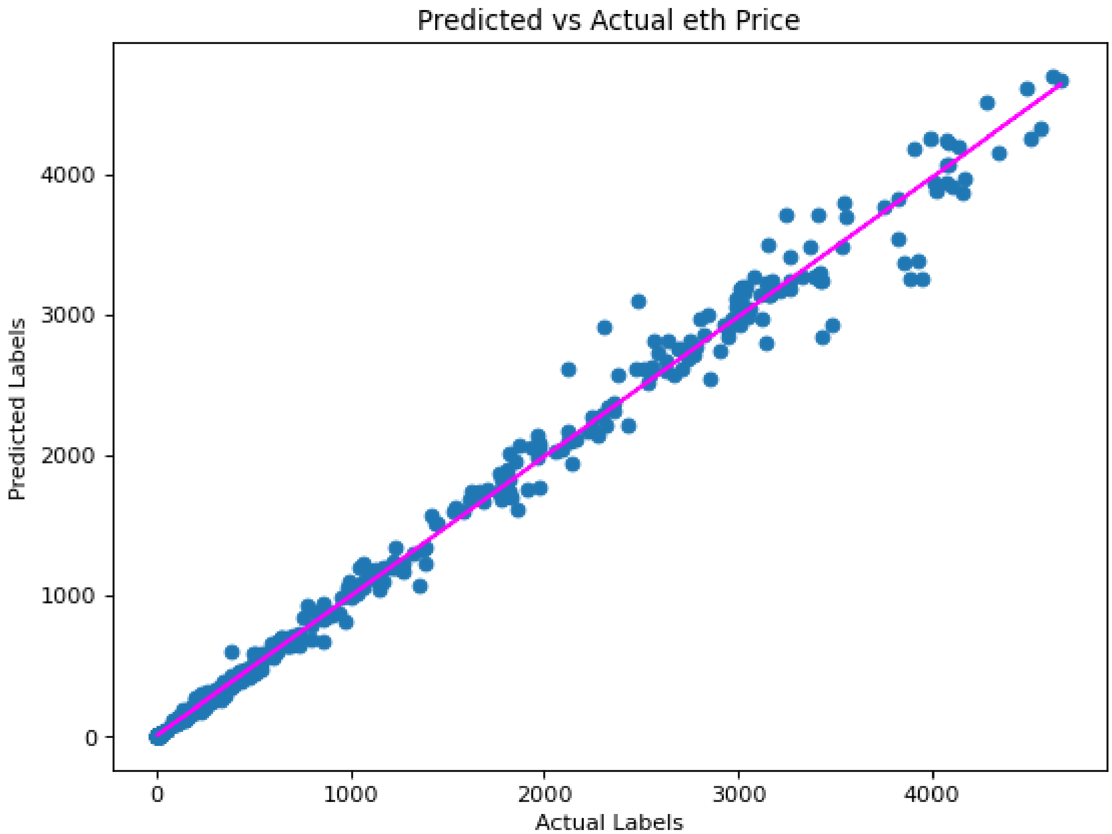
Random Forest Model

In [155]:
```
#Train Test Split for all future regression models
eth_df = eth_df.dropna()
X = eth_df[['count','active_count','new_non_zero_count','difficulty_latest'
eth_price = eth_df['price_usd_close']
X, y = X, eth_price
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.30,
print (f'Training Set: {X_train.shape[0]} rows \nTest Set: {X_test.shape[0]
```

```
Training Set: 1776 rows
Test Set: 762 rows
```

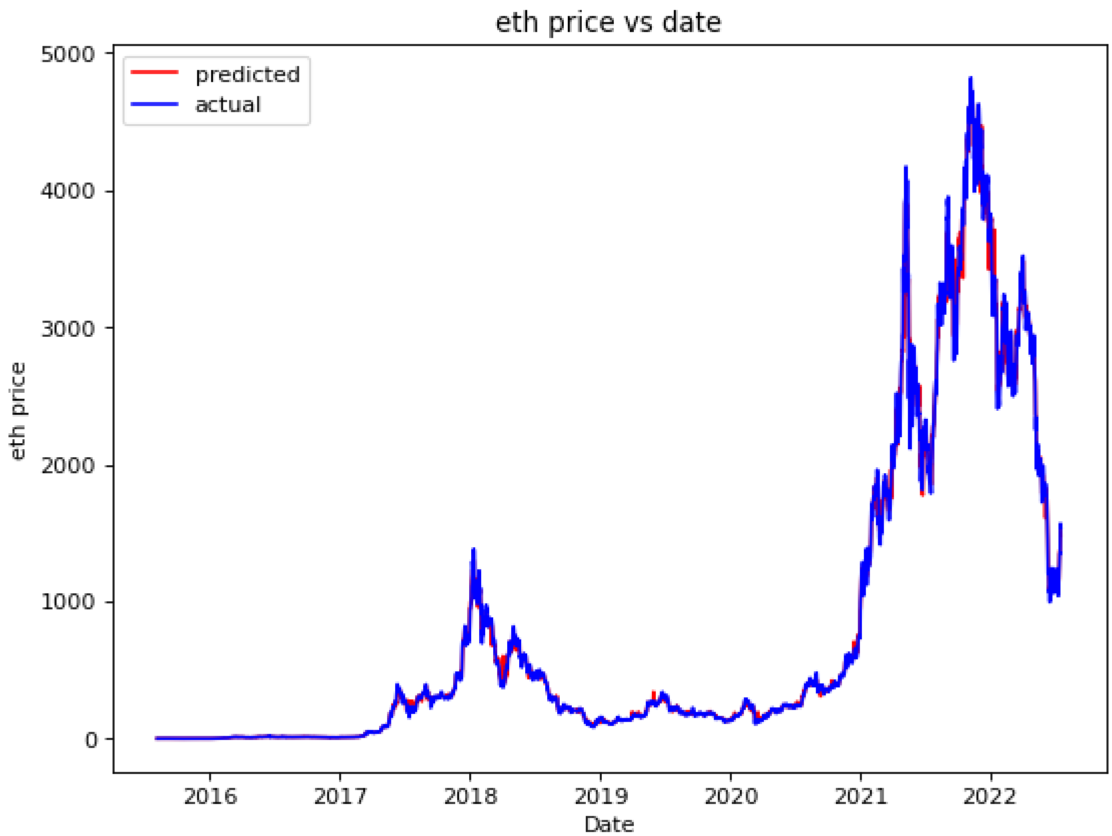
In [156]:
```
model = RandomForestRegressor().fit(X_train, y_train)
preds = model.predict(X_test)
```

In [157]:
```
preds_act_plot(y_test, preds)
model_fit(y_test, preds)
price_date_plot(preds)
```




```
Mean Squared Error: 8401.205809443785
Root Mean Squared Error: 91.65809189288082.
Simply put, this means that each prediction was, on average, $91.66 diffe
```

```
rent from actual.
r^2: 0.9933077221429054
```



Linear Regression Assumptions:

1st Assumption -- Linear Relationship:

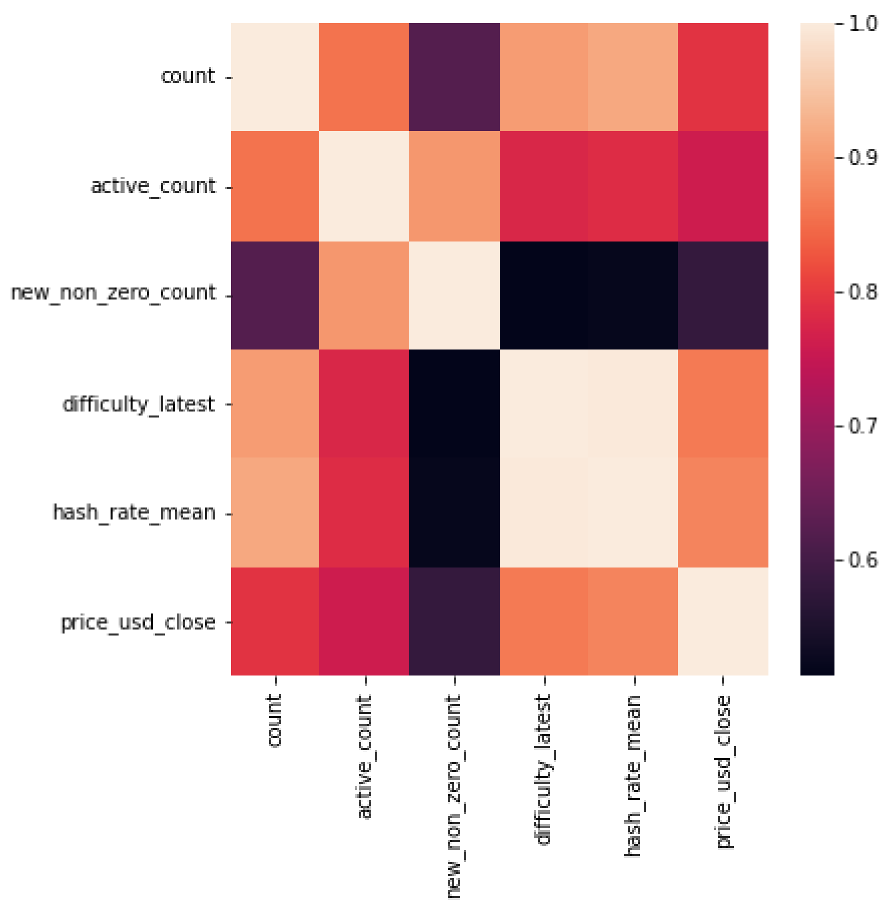
- There is a linear relationship between the independent variable x (feature, hash_rate_mean, difficulty_latest, new_non_zero_count, etc.) and dependent variable (label, eth_usd_close)
- There does seem to be a more or less linear relationship between the features and the label. It is obviously not completely linear; however, visually there seems to be a linear relationship and quantitatively there seems to be a linear relationship via the correlation DataFrame below.
- This is also shown well above with predictive labels vs actual labels.

In [158]:

```
corr_df = eth_df.corr(method = 'pearson')
plt.figure(figsize = (6,6))
sns.heatmap(corr_df)
corr_df
```

Out[158]:

	count	active_count	new_non_zero_count	difficulty_latest	hash_rate_mean
count	1.000000	0.857252	0.621569	0.902888	0.914626
active_count	0.857252	1.000000	0.895655	0.776950	0.784418
new_non_zero_count	0.621569	0.895655	1.000000	0.513253	0.519030
difficulty_latest	0.902888	0.776950	0.513253	1.000000	0.996773
hash_rate_mean	0.914626	0.784418	0.519030	0.996773	1.000000
price_usd_close	0.792044	0.759223	0.579870	0.864914	0.876262



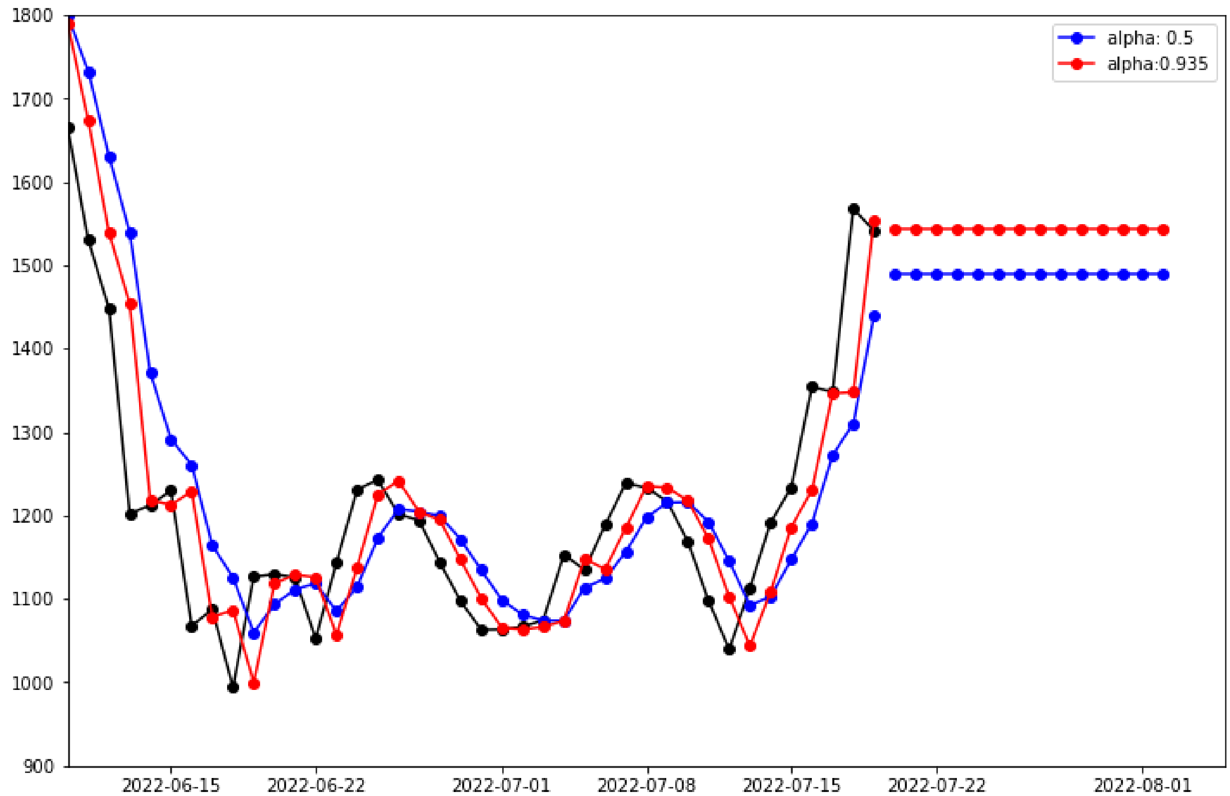
Clearly, we see a relatively high correlation between pretty much all of the features. There is a minimum of 0.5 or so for each of the features.

Forecasting with Different Methods: ExponentialSmoothing, SimpleExpSmoothing, Holt

[Docs (https://www.statsmodels.org/devel/examples/notebooks/generated/exponential_smoothing.html)](https://www.statsmodels.org/devel/examples/notebooks/generated/exponential_smoothing.html) to reference: trend, seasonality, smoothing level. Simply put, SimpleExponentialSmoothing does not have any trend. Holt has no seasonality but does have a trend , and ExponentialSmoothing has both a trend and seasonality.

In [159]:
```
fit1 = SimpleExpSmoothing(eth_df['price_usd_close'], initialization_method=
smoothing_level=0.5, optimized=False)
fcast1 = fit1.forecast(14).rename('alpha: 0.5')

fit2 = SimpleExpSmoothing(eth_df['price_usd_close'], initialization_method=
fcast2 = fit2.forecast(14).rename(f'alpha:{fit2.model.params["smoothing_lev

plt.figure(figsize=(12, 8))
plt.plot(eth_df['price_usd_close'], marker="o", color="black")
plt.plot(fit1.fittedvalues, marker="o", color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")
plt.plot(fit2.fittedvalues, marker="o", color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.legend([line1, line2], [fcast1.name, fcast2.name])
plt.xlim(date(2022,6,10),date(2022,8,5))
plt.ylim(900,1800)
```

Out[159]:
```
(900.0, 1800.0)
```



Simple Exponential Smoothing Analysis:

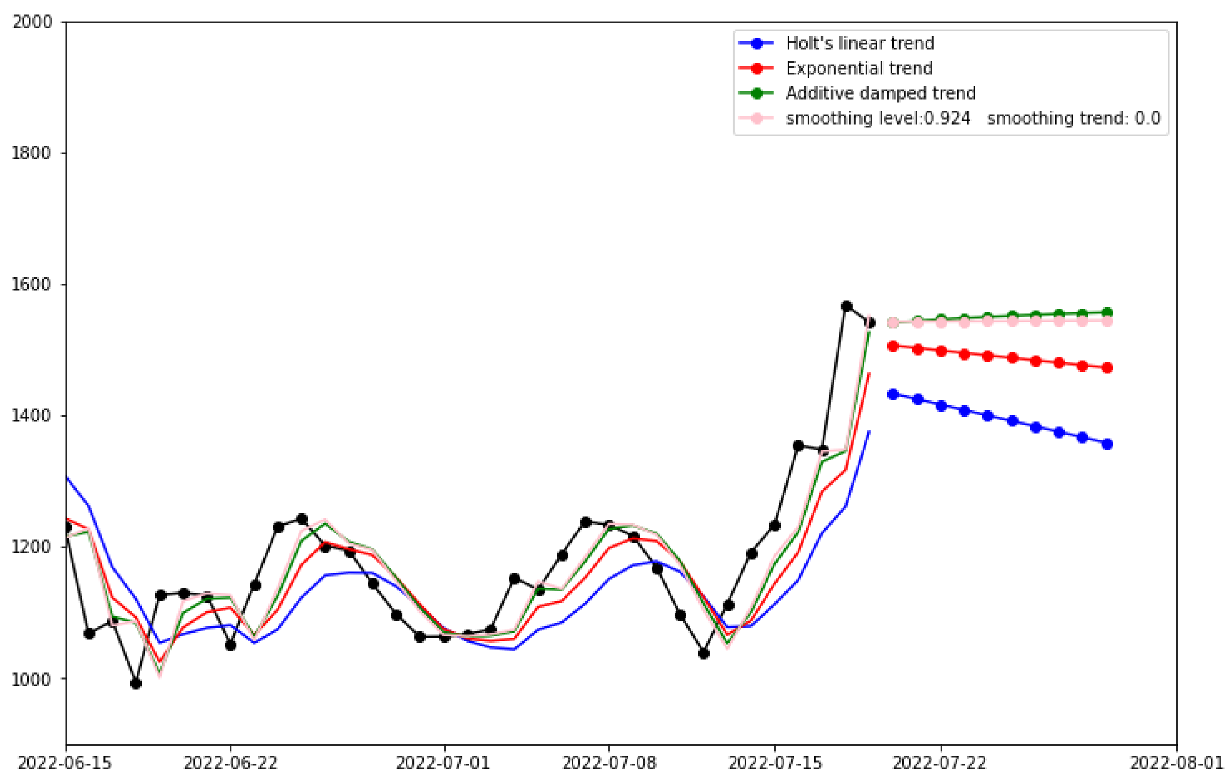
Clearly there is little value in exponential smoothing for price data like this. In this example the naive method was used which all future forecasts equal the last observation of the series. Using an average method would return a simple average of all of the data. Clearly, these have no use case for us in this application.

- Using other forecasting methods may be more useful for attempting to predict the price of ether. Exponential smoothing with trend and seasonal components. Holt is Exponential smoothing with a trend component.

Holt Forecasting:

In [161]:
```
fit1 = Holt(eth_df['price_usd_close'], initialization_method="estimated").f
    smoothing_level=.4, smoothing_trend=0.01, optimized=False
)
fcast1 = fit1.forecast(10).rename("Holt's linear trend")
fit2 = Holt(eth_df['price_usd_close'], exponential=True, initialization_met
    smoothing_level=.6, smoothing_trend=0.01, optimized=False
)
fcast2 = fit2.forecast(10).rename("Exponential trend")
fit3 = Holt(eth_df['price_usd_close'], damped_trend=True, initialization_me
    smoothing_level=.8, smoothing_trend=0.01
)
fcast3 = fit3.forecast(10).rename("Additive damped trend")

fit4 = Holt(eth_df['price_usd_close'], initialization_method="estimated").f
s_level = fit4.model.params['smoothing_level']
s_trend = fit4.model.params['smoothing_trend']
fcast4 = fit4.forecast(10).rename(f"smoothing level:{s_level.round(3)}"+ f"

plt.figure(figsize=(12, 8))
plt.plot(eth_df['price_usd_close'], marker="o", color="black")
plt.plot(fit1.fittedvalues, color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")
plt.plot(fit2.fittedvalues, color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.plot(fit3.fittedvalues, color="green")
(line3,) = plt.plot(fcast3, marker="o", color="green")
plt.plot(fit4.fittedvalues, color="pink")
(line4,) = plt.plot(fcast4, marker="o", color="pink")
plt.legend([line1, line2, line3, line4], [fcast1.name, fcast2.name, fcast3.
plt.xlim(date(2022,6,15),date(2022,8,1))
plt.ylim(900,2000)
```

Out[161]: (900.0, 2000.0)



Holt Forecasting Analysis:

- Holt forecasting is based on recent trends, thus the uptrend from july 17th would indicate a positive trend which was made obvious from the varying methods of trends. The influence of a trend may be changed by the smoothing trend. A higher smoothing trend puts more influence on the more recent price fluctuations.

1. Holt's Linear trend: Will increase at the same slope indefinitely increasing or decreasing. This forecast equation take in the output of a level equation and trend equation.

$$\begin{aligned} \text{Forecast equation} \quad & \hat{y}_{t+h|t} = \ell_t + hb_t \\ \text{Level equation} \quad & \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ \text{Trend equation} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}, \end{aligned}$$

- The Level equation denotes an estimate of the level of the series at time t. The Trend equations denotes an estimate of the slope of the series at time t.
- α is the smoothing level: 0<α<1
- β* is the smoothing trend: 0<β*<1

- Taking this all into consideration, we see that the observations are merely a h-step ahead forecast which is equal to the last estimated level plus h times the last estiamted trend value. --> Forecasts are a linear function of h.

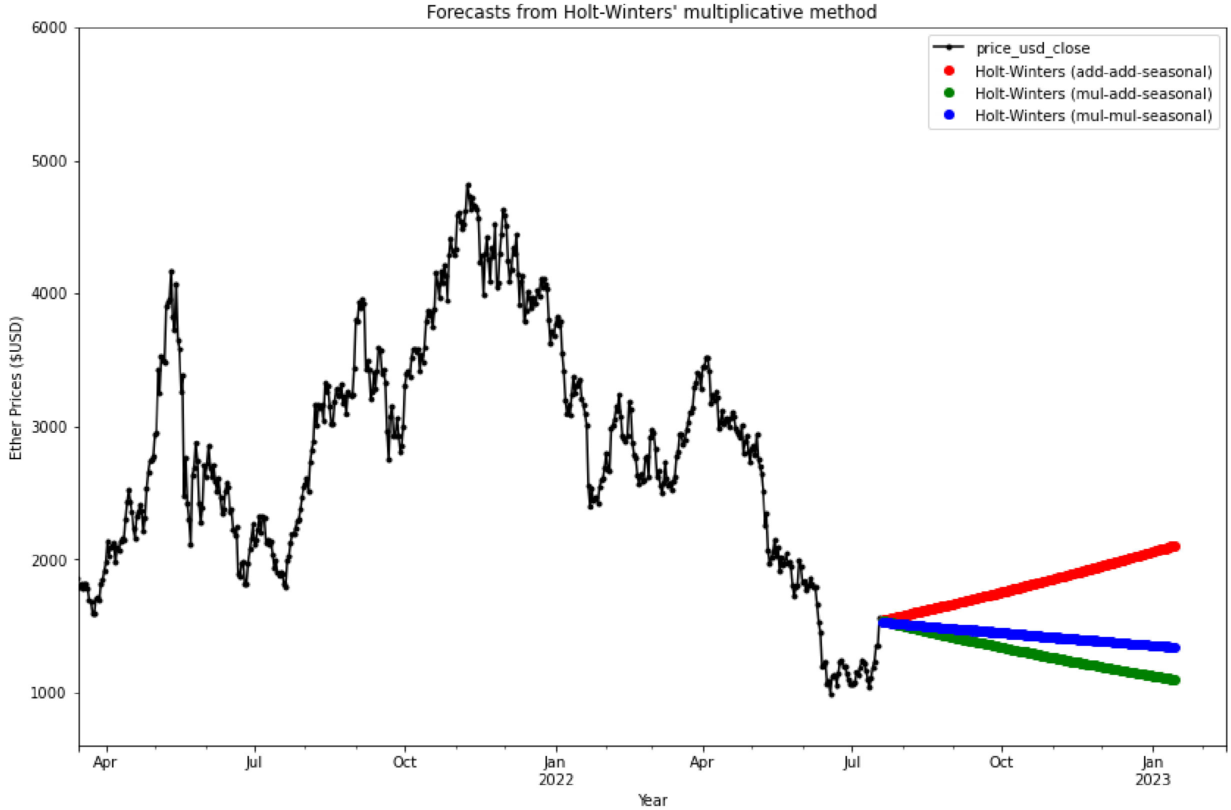
2. Additive Damped trend: Taking into account the Holt's linear function, the damped trend adds a dampening to a linear function 0<ϕ<1. This done to curb the effects of over forecasting, especially across long time frames. If ϕ = 1 there is effectively no applied dampening to holt's linear forecasting.

$$\begin{aligned} \hat{y}_{t+h|t} &= \ell_t + (\phi + \phi^2 + \cdots + \phi^h)b_t \\ \ell_t &= \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}) \\ b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}. \end{aligned}$$

3. The optimized method with a listed smoothing level and trend is merely a linear trend with optimized smoothing trend and level based on the previous data points. It should be very interesting to apply seasonal meathods to the entire data set using holt-winters method.

Exponential Smoothing:

In [390]:
```
fit1 = ExponentialSmoothing(eth_df['price_usd_close'],seasonal_periods=2,tr
initialization_method="estimated").fit()

fit2 = ExponentialSmoothing(eth_df['price_usd_close'],seasonal_periods=2,tr
initialization_method="estimated").fit()

fit3 = ExponentialSmoothing(eth_df['price_usd_close'],seasonal_periods=2,tr
initialization_method="estimated").fit()

ax = eth_df['price_usd_close'].plot(figsize=(14, 9), marker = ".",color="bl
title="Forecasts from Holt-Winters' multiplicative method",)

ax.set_ylabel("Ether Prices ($USD)")
ax.set_xlabel("Year")

#fit1.fittedvalues.plot(ax=ax, style="--", color="red")
#fit2.fittedvalues.plot(ax=ax, style="--", color="green")
#fit3.fittedvalues.plot(ax=ax, style = "--", color = "blue")

fit1.forecast(180).rename("Holt-Winters (add-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="red",lw = 0.01, legend=True
)
fit2.forecast(180).rename("Holt-Winters (mul-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="green",lw = 0.01, legend=True
)

fit3.forecast(180).rename("Holt-Winters (mul-mul-seasonal)").plot(
    ax=ax, style="--", marker="o", color="blue",lw = 0.01, legend=True
)

plt.legend()
plt.ylim(600,6000)
plt.xlim(date(2021,3,15),date(2023,2,15))
plt.show()
print("Forecasting Ether Prices using Holt-Winters method with both additiv
```




```
Forecasting Ether Prices using Holt-Winters method with both additive and
multiplicative seasonality.
```

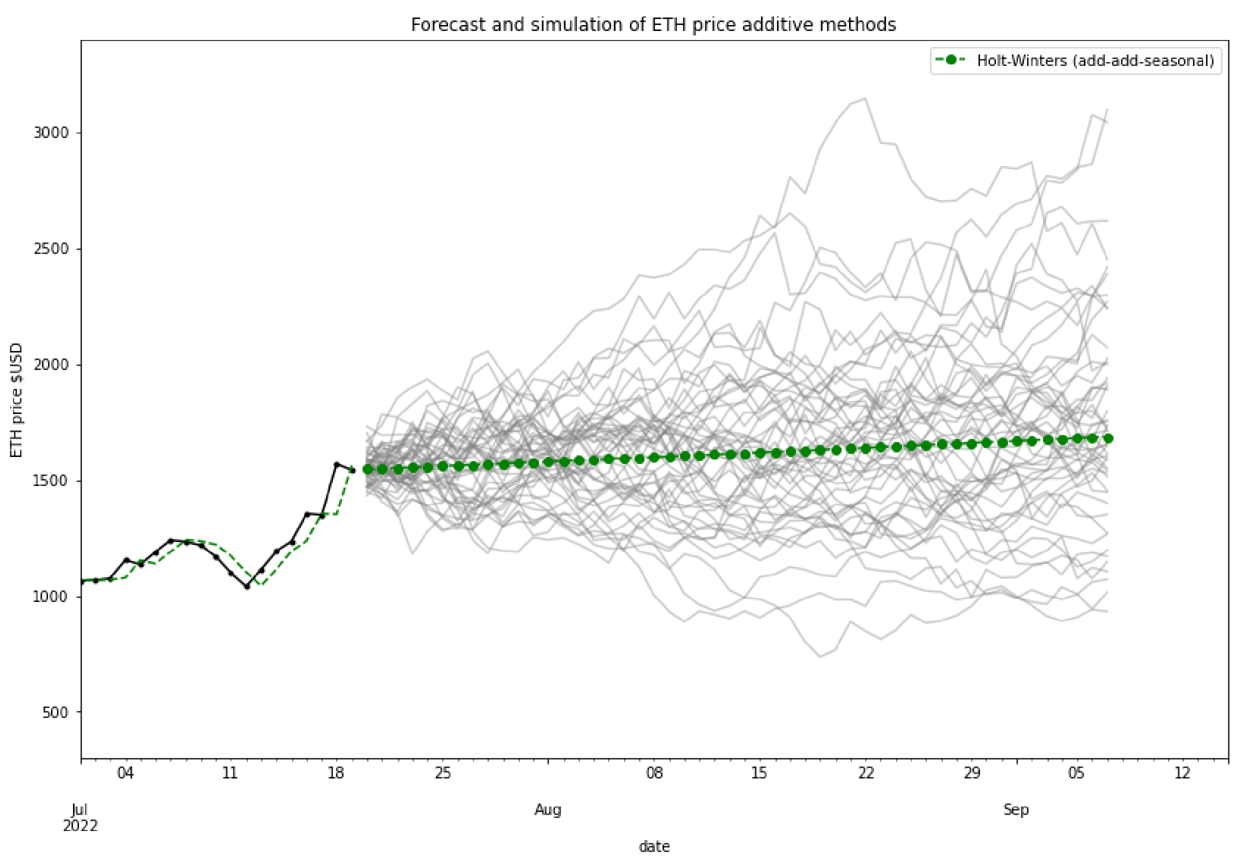
Exponential Smoothing Analysis:

- Mathematically, what is a multiplicative trend

Simulations: Additive and Multiplicative

In [391]:

```
#Additive trend and seasonal Model Fitting
fit = ExponentialSmoothing(eth_df['price_usd_close'], seasonal_periods = 2,
                           initialization_method = "estimated", use_boxcox
#Simulation Fitting
add_add_simulations = fit.simulate(50, repetitions = 100, error = "add")

#Historical Price Plotting
ax= eth_df['price_usd_close'].plot(figsize = (14,9), marker = ".",
                                   color = "black", title = "Forecast and
ax.set_ylabel("ETH price $USD")
ax.set_xlabel("Date")

#Model and Simulation Plotting
fit.fittedvalues.plot(ax=ax, style="--", color="green")
simulations.plot(ax=ax, style="-", alpha=0.4, color="grey", legend=False)
fit.forecast(50).rename("Holt-Winters (add-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)

#Graph limits for better visual
plt.ylim(300,3400)
plt.xlim(date(2022,7,1),date(2022,9,15))
plt.show()
print("Forecasting Ether Prices using Holt-Winters additive methods. Parame
print(fit.params)
```



```
Forecasting Ether Prices using Holt-Winters additive methods. Parameters
below:
{'smoothing_level': 0.980169226538637, 'smoothing_trend': 0.0, 'smoothing
_seasonal': 0.0, 'damping_trend': nan, 'initial_level': -0.21848550186925
184, 'initial_trend': 0.005412908529524337, 'initial_seasons': array([-0.
0445814 , -0.04601229]), 'use_boxcox': True, 'lamda': 0.1515586338288304
2, 'remove_bias': False}
```

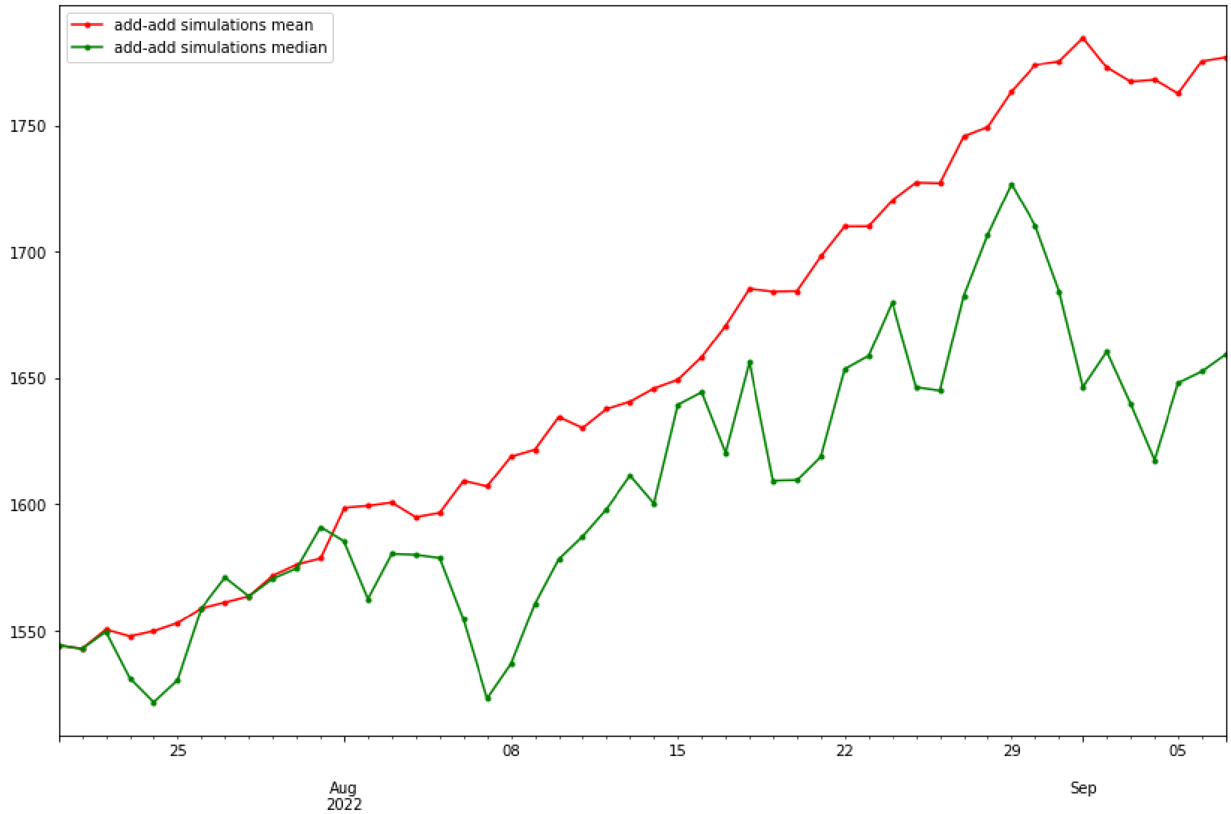
Analysis:

We can clearly see a mildly bullish forecast and simulations. To test the center of the simulations we can compute the mean and median of each day and simulation. This can be done pretty simply in pandas:

In [392]:
```
add_add_simulations['mean'] = add_add_simulations.mean(axis=1)
add_add_simulations['median'] = add_add_simulations.median(axis = 1)
```

In [393]:
```
add_add_simulations['mean'].rename("add-add simulations mean").plot(figsize
add_add_simulations['median'].rename("add-add simulations median").plot(fig
plt.legend()
```

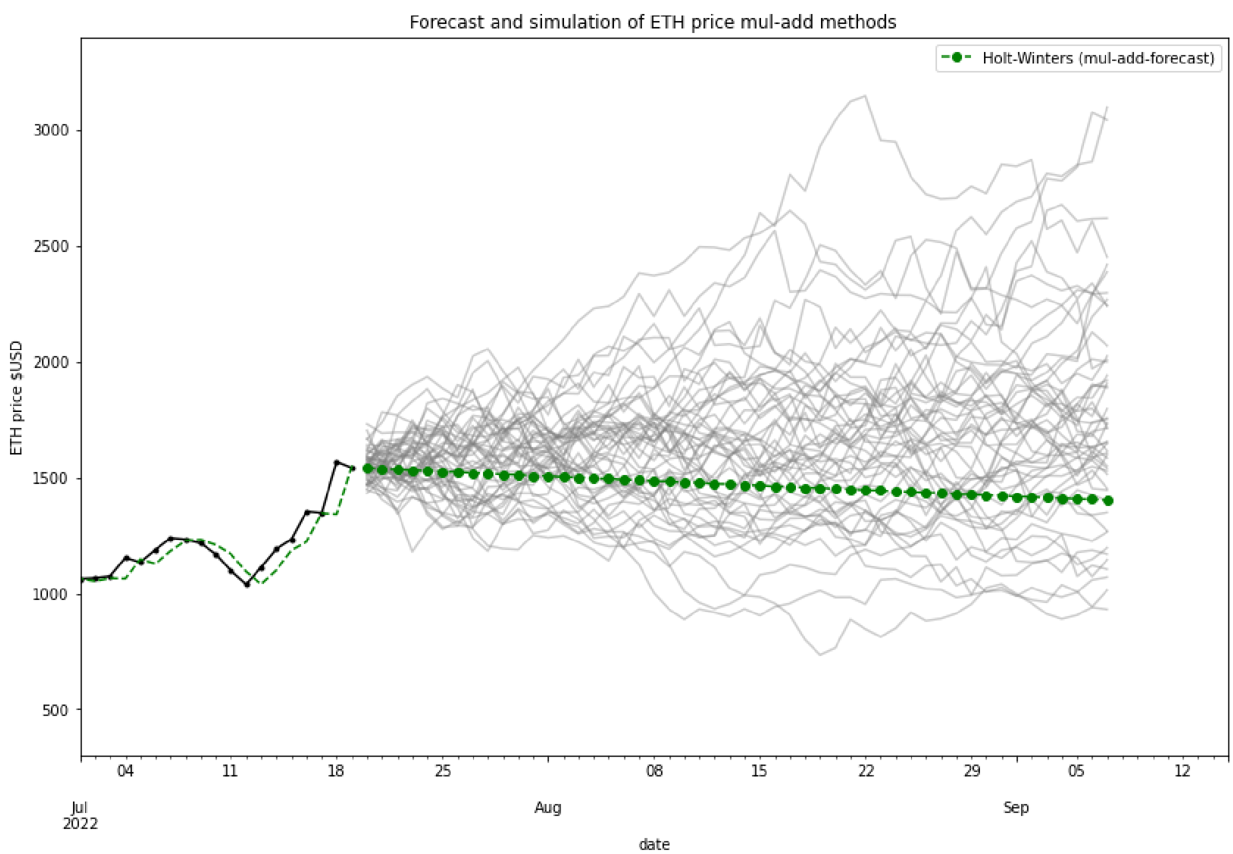
Out[393]: <matplotlib.legend.Legend at 0x7fea7a0f89a0>



Center of Simulation Analysis:

- It is interesting to see that the mean and median of the simulations are both bullish. Is this because of a optimized smoothing level and trend that puts too much weight on the recent uptrend and extrapolates that to aggresively? Looking at fit.params above the mean and median graph.
- Smoothing Trend: 0.98, Smoothing Level: 0.00. A higher smoothing trend directly leads to higher influence on more recent points. Manually tuning that lower may give a less biased simulaiton.
- Additionally, validating forecasting methods would be wise. Forecasting from a point in the past comparing that to actual values would be a good way to see which different smoothing level, trends, and seasonality settings have the best output when it comes to forecasting crypto prices.

In [394]:
```
#Additive seasonal and multiplicative trend model Fitting
fit = ExponentialSmoothing(eth_df['price_usd_close'], seasonal_periods = 2,
                            initialization_method = "estimated", use_boxcox
#Simulation Fitting
mul_add_simulations = fit.simulate(50, repetitions = 100, error = "add")

#Historical Price Plotting
ax= eth_df['price_usd_close'].plot(figsize = (14,9), marker = ".",
                                   color = "black", title = "Forecast and
ax.set_ylabel("ETH price $USD")
ax.set_xlabel("Date")

#Model and Simulation Plotting
fit.fittedvalues.plot(ax=ax, style="--", color="green")
simulations.plot(ax=ax, style="-", alpha=.4, color="grey", legend=False)
fit.forecast(50).rename("Holt-Winters (mul-add-forecast)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)

#Graph limits for better visual
plt.ylim(300,3400)
plt.xlim(date(2022,7,1),date(2022,9,15))
plt.show()
print("Forecasting Ether Prices using Holt-Winters mul-add methods")
```




```
Forecasting Ether Prices using Holt-Winters mul-add methods
```

In [395]:
```
mul_add_simulations['mean'] = mul_add_simulations.mean(axis=1)
mul_add_simulations['median'] = mul_add_simulations.median(axis = 1)
mul_add_simulations.tail(1)
```

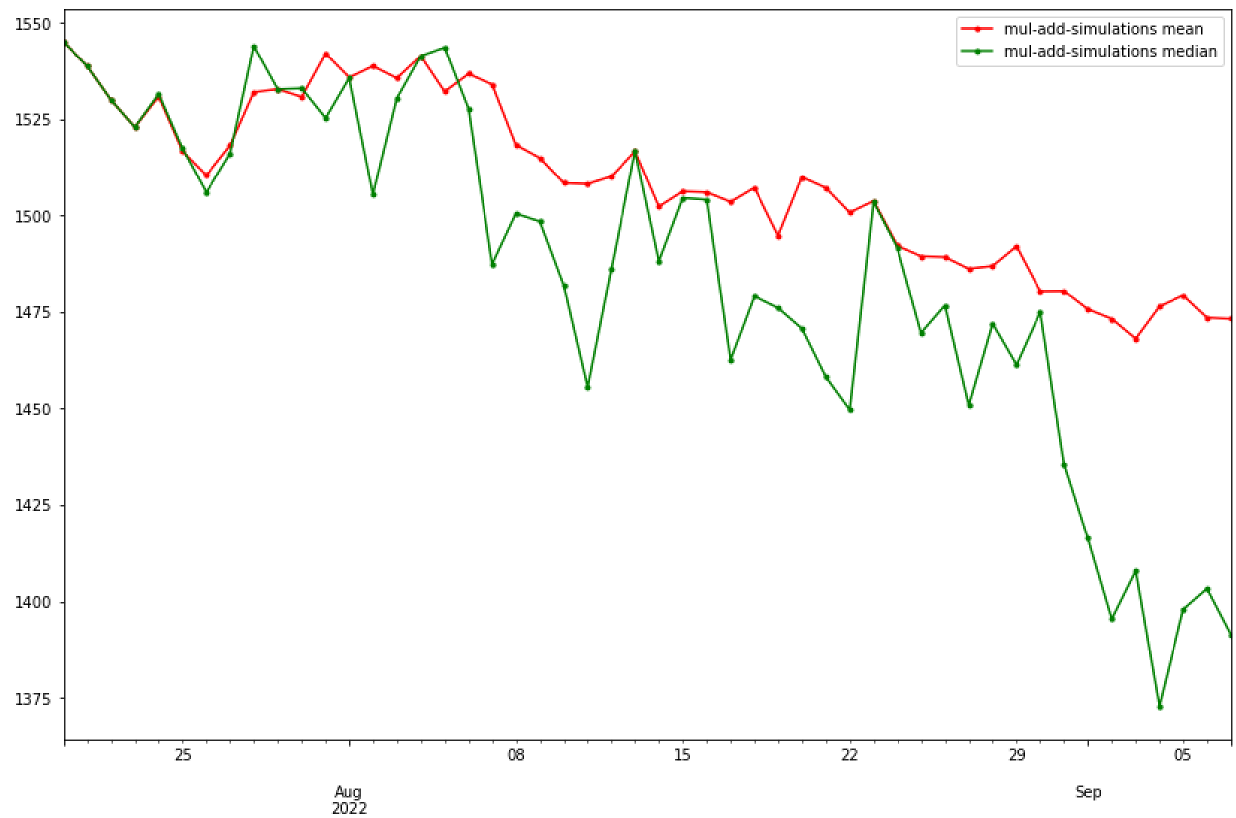
Out[395]:

	0	1	2	3	4	5	6	
2022-09-07	1971.743234	1127.785482	869.089821	1567.47158	1628.120018	818.812865	1777.764912	15

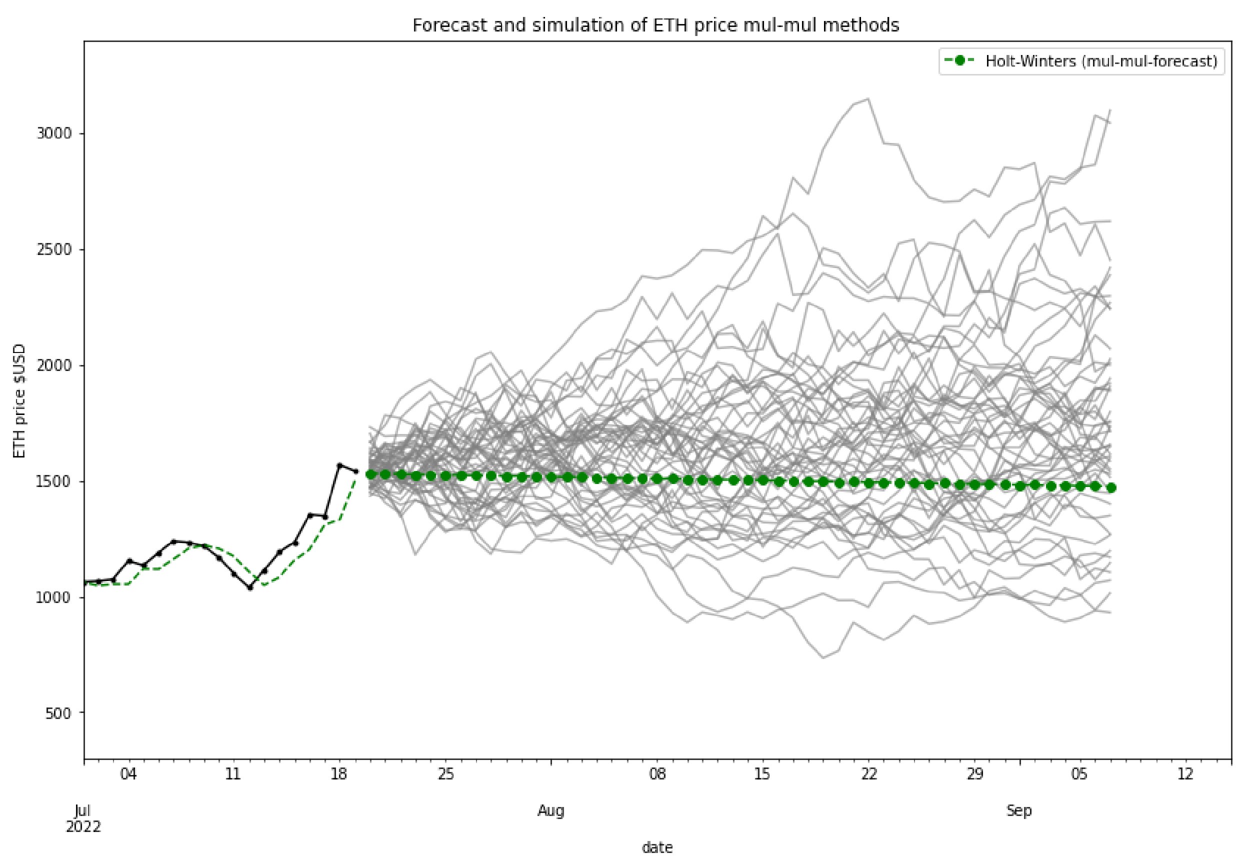
1 rows × 102 columns

In [396]:
```
mul_add_simulations['mean'].rename("mul-add-simulations mean").plot(figsize
mul_add_simulations['median'].rename("mul-add-simulations median").plot(fig
plt.legend()
```

Out[396]: <matplotlib.legend.Legend at 0x7fea5a5cea90>



In [397]:
```
#Additive seasonal and multiplicative trend model Fitting
fit = ExponentialSmoothing(eth_df['price_usd_close'], seasonal_periods = 2,
                            initialization_method = "estimated", use_boxcox
#Simulation Fitting
mul_mul_simulations = fit.simulate(50, repetitions = 100, error = "add")

#Historical Price Plotting
ax= eth_df['price_usd_close'].plot(figsize = (14,9), marker = ".",
                                   color = "black", title = "Forecast and
ax.set_ylabel("ETH price $USD")
ax.set_xlabel("Date")

#Model and Simulation Plotting
fit.fittedvalues.plot(ax=ax, style="--", color="green")
simulations.plot(ax=ax, style="-", alpha=.6, color="grey", legend=False)
fit.forecast(50).rename("Holt-Winters (mul-mul-forecast)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)

#Graph limits for better visual
plt.ylim(300,3400)
plt.xlim(date(2022,7,1),date(2022,9,15))
plt.show()
print("Forecasting Ether Prices using Holt-Winters mul-mul methods")
```



Forecasting Ether Prices using Holt-Winters mul-mul methods

In [398]:
```
mul_mul_simulations['mean'] = mul_mul_simulations.mean(axis=1)
mul_mul_simulations['median'] = mul_mul_simulations.median(axis = 1)
mul_mul_simulations.tail(1)
```

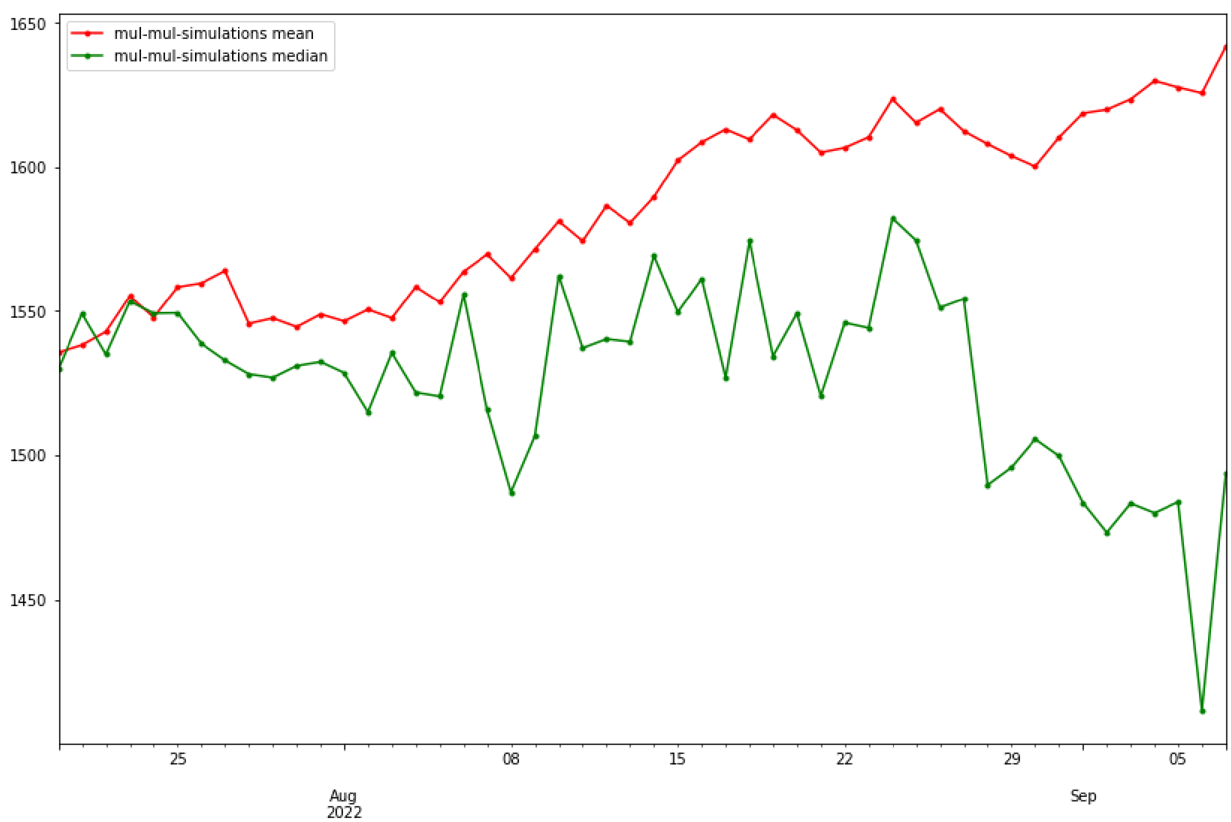
Out[398]:

	0	1	2	3	4	5	6
2022-09-07	1630.395519	2237.870226	1031.771041	626.060674	1936.292535	1924.712835	3280.248309

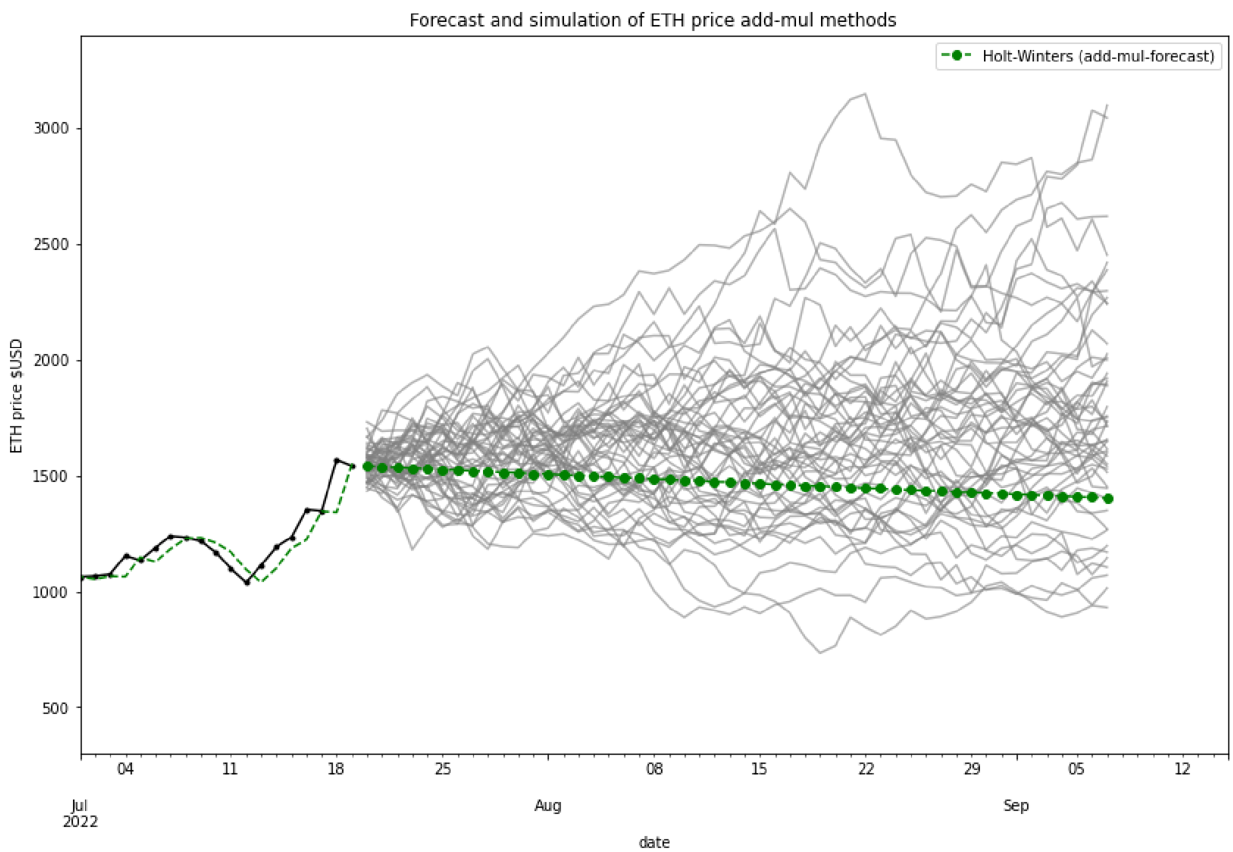
1 rows × 102 columns

In [399]:
```
mul_mul_simulations['mean'].rename("mul-mul-simulations mean").plot(figsize
mul_mul_simulations['median'].rename("mul-mul-simulations median").plot(fig
plt.legend()
```

Out[399]: <matplotlib.legend.Legend at 0x7fea8d707ac0>



In [400]:
```
#Additive seasonal and multiplicative trend model Fitting
fit = ExponentialSmoothing(eth_df['price_usd_close'], seasonal_periods = 2,
                            initialization_method = "estimated", use_boxcox
#Simulation Fitting
mul_add_simulations = fit.simulate(50, repetitions = 100, error = "add")

#Historical Price Plotting
ax= eth_df['price_usd_close'].plot(figsize = (14,9), marker = ".",
                                    color = "black", title = "Forecast and
ax.set_ylabel("ETH price $USD")
ax.set_xlabel("Date")

#Model and Simulation Plotting
fit.fittedvalues.plot(ax=ax, style="--", color="green")
simulations.plot(ax=ax, style="-", alpha=.6, color="grey", legend=False)
fit.forecast(50).rename("Holt-Winters (add-mul-forecast)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)

#Graph limits for better visual
plt.ylim(300,3400)
plt.xlim(date(2022,7,1),date(2022,9,15))
plt.show()
print("Forecasting Ether Prices using Holt-Winters add-mul methods")
```

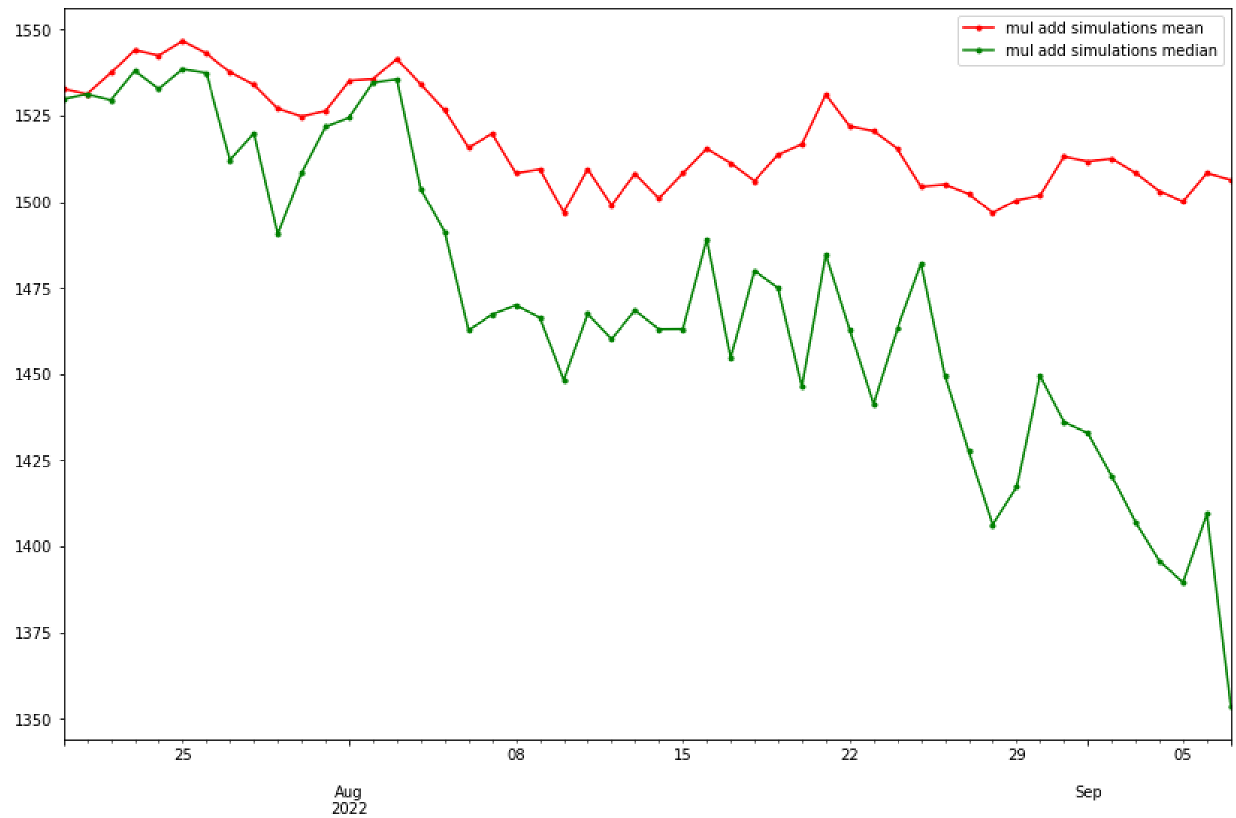



```
Forecasting Ether Prices using Holt-Winters add-mul methods
```

In [401]:
```
mul_add_simulations['mean'] = mul_add_simulations.mean(axis=1)
mul_add_simulations['median'] = mul_add_simulations.median(axis = 1)
```

In [402]:
```
mul_add_simulations['mean'].rename("mul add simulations mean").plot(figsize
mul_add_simulations['median'].rename("mul add simulations median").plot(fig
plt.legend()
```

Out[402]:
```
<matplotlib.legend.Legend at 0x7fea8d55a7f0>
```



Issues with Mathematically forecasting ETH price:

- The equations may be able to model future prices regardless of on-chain activity or outside factors; however, it doesn't give a wholistic view on other factors or on-chain demand that may determine price fluctuations. These types of forecasting may be best suited for more seasonal data. For example sales data by month or more broadly speaking demand data for a retial company.
- More braod applications toward a bull cycle or bear cycle may be of use. For example, creating different data frames starting with the bitcoin halving date and ending with the 'blow off top' that is general seen in these cycles.
- More explicitly, these price fluctuations may be a result of the upcoming ethereum merge to the beacon chain.

In [ ]: