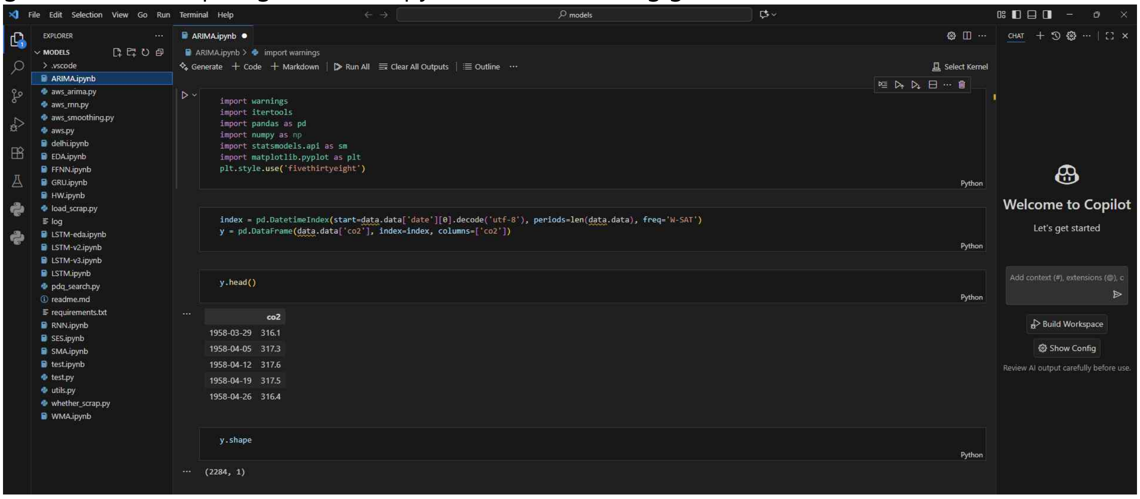
에너지시스템과AI

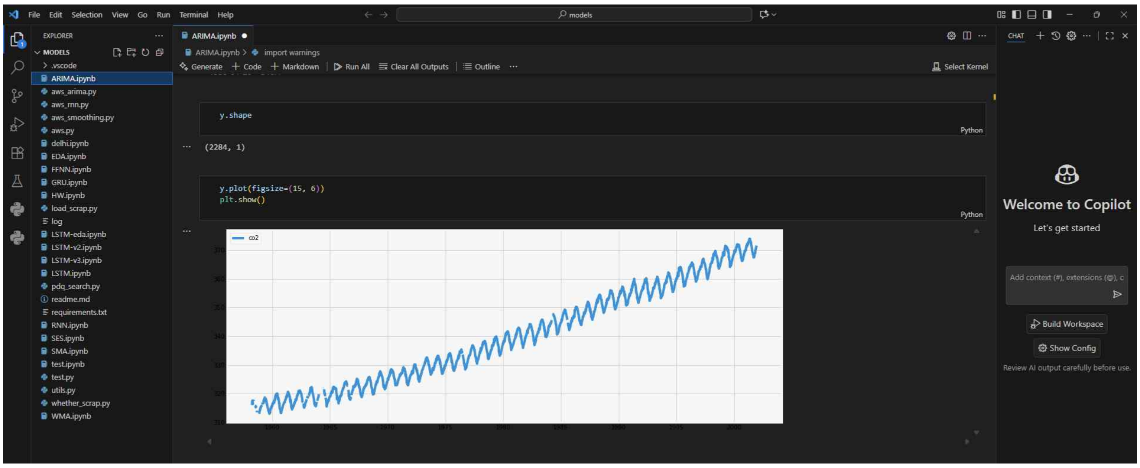
2주차 과제

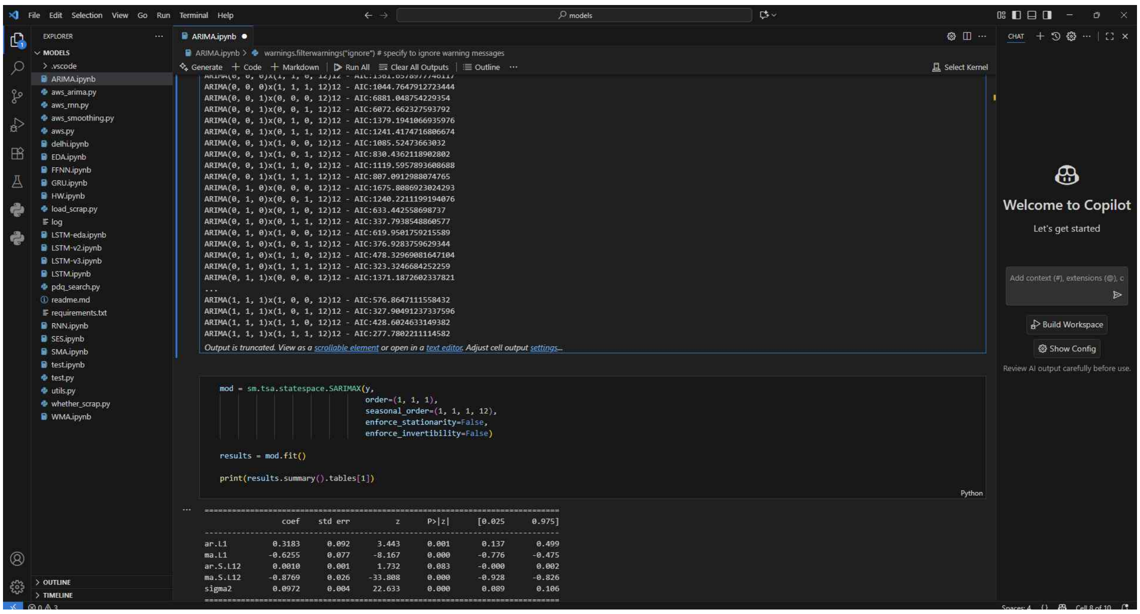
<목차>

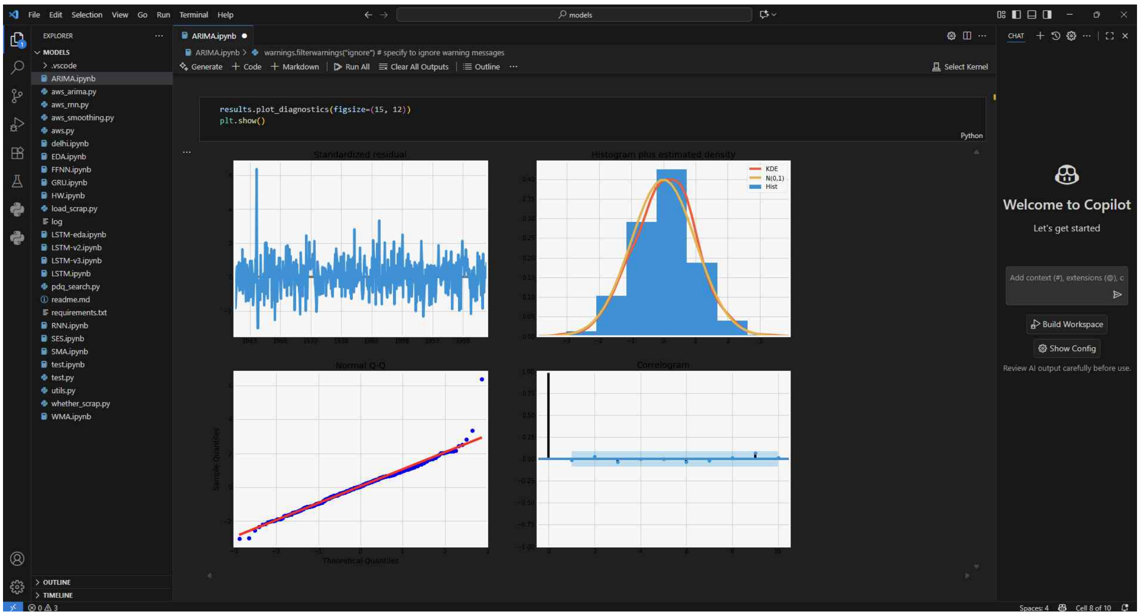
1-1. ARIMA.ipynb 실행 결과

github주소 : https://github.com/pyaf/load_forecasting.git

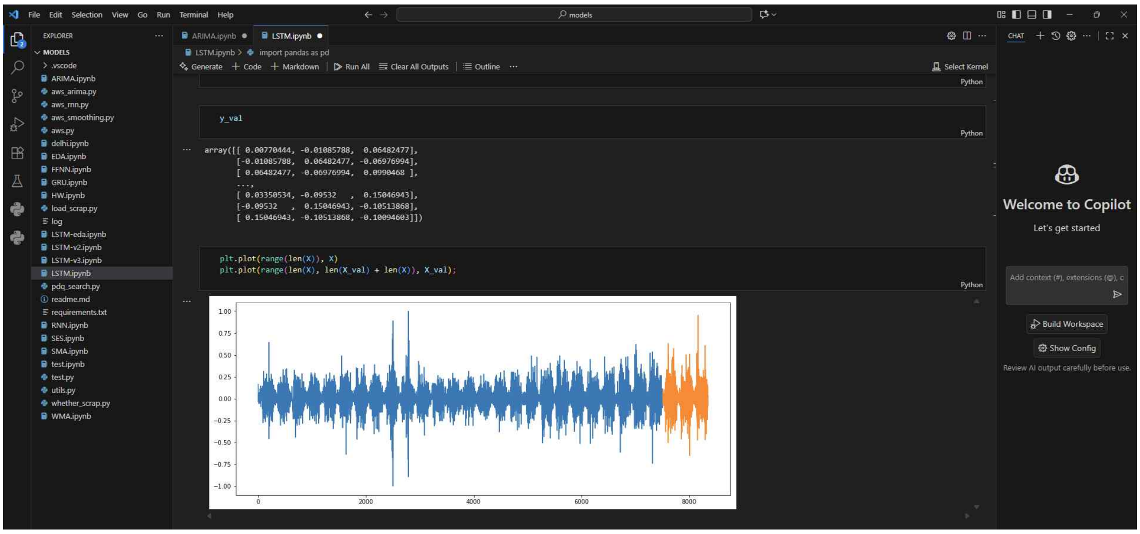


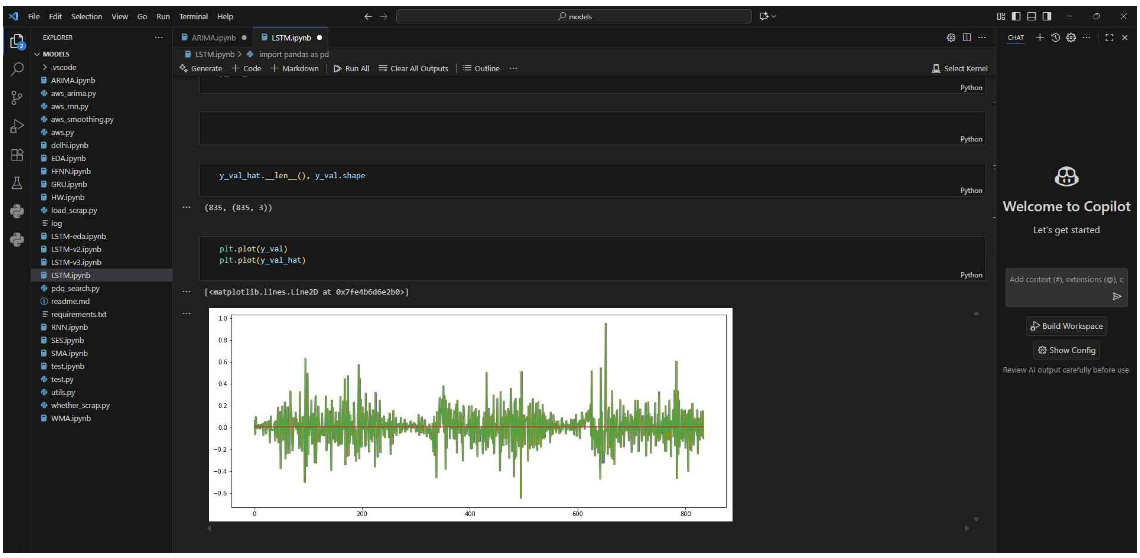


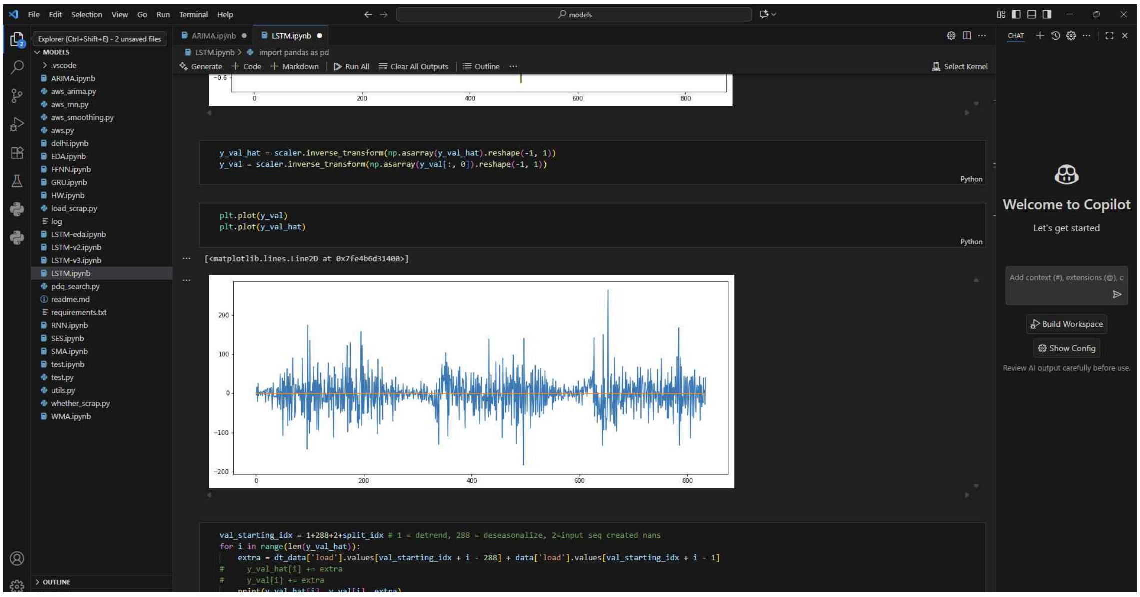




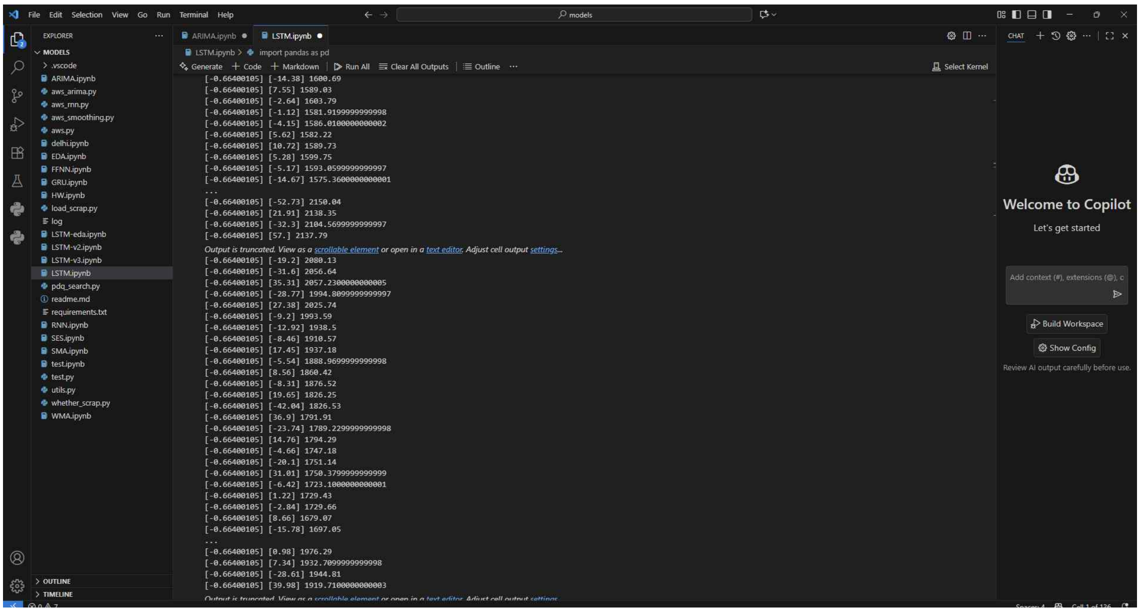
1-2. LSTM.ipynb 실행 결과



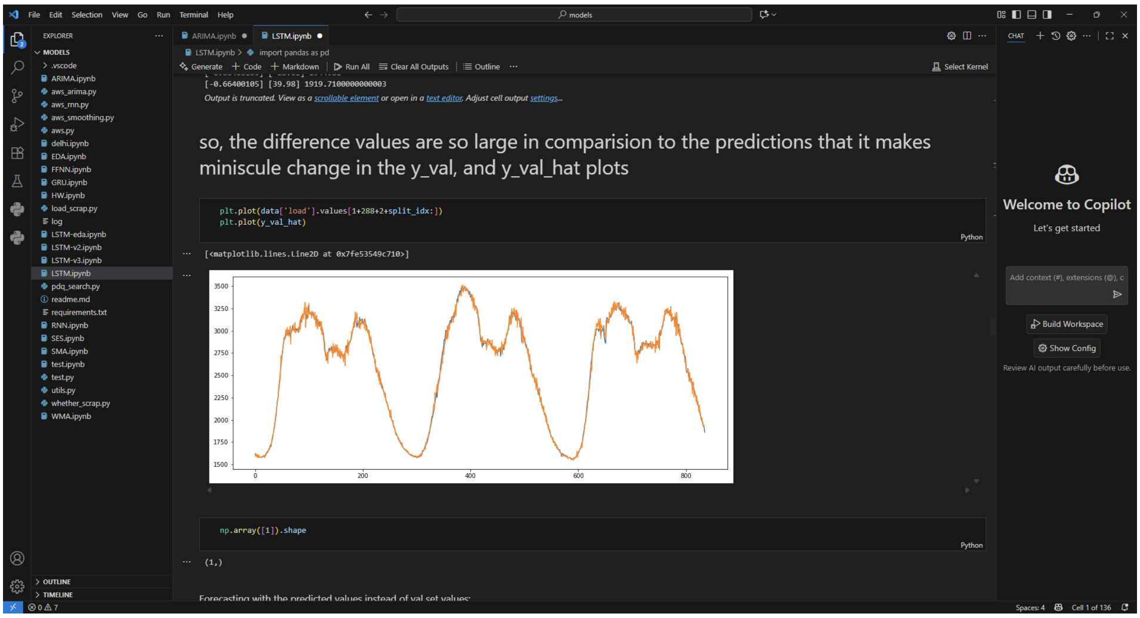




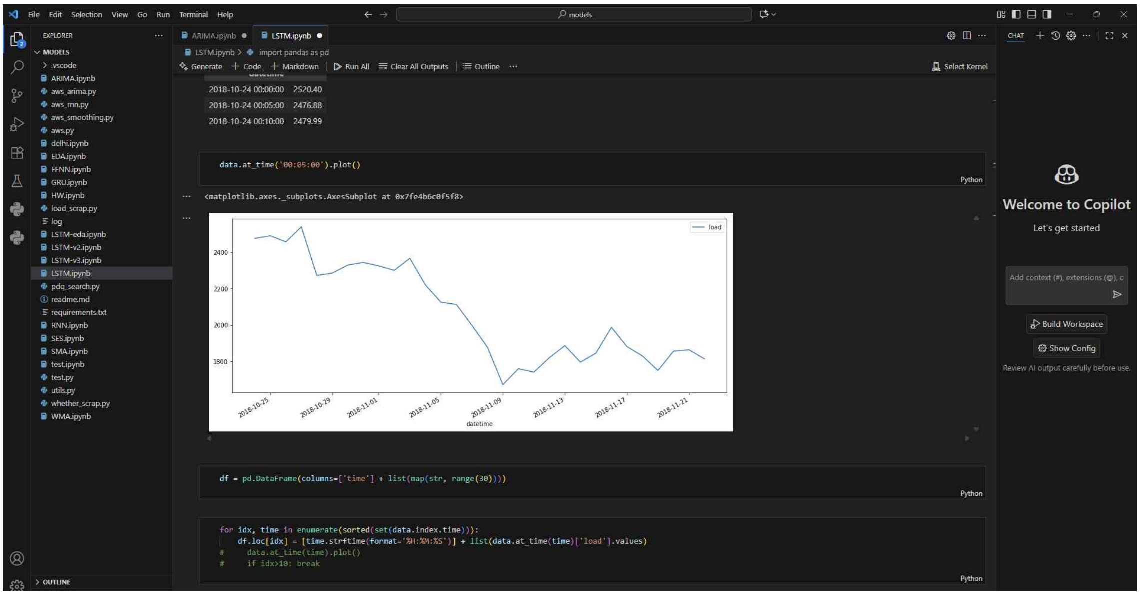
y_val_hat = scaler.inverse_transform(np.asarray(y_val_hat).reshape(-1, 1))
y_val = scaler.inverse_transform(np.asarray(y_val[:, 0]).reshape(-1, 1))
plt.plot(y_val)
plt.plot(y_val_hat)
[<matplotlib.lines.Line2D at 0x7fe4b6d31400>]
val_starting_idx = 1+288+2+split_idx # 1 = detrend, 288 = deseasonalize, 2=input seq created nans
for i in range(len(y_val_hat)):
extra = dt_data['load'].values[val_starting_idx + i - 288] + data['load'].values[val_starting_idx + i - 1]
# y_val_hat[i] += extra
# y_val[i] += extra



[-0.66400105] [-14.38] 1600.69
[-0.66400105] [7.55] 1589.03
[-0.66400105] [-2.64] 1603.79
[-0.66400105] [-1.12] 1581.9199999999998
[-0.66400105] [-4.15] 1586.0100000000002
[-0.66400105] [5.62] 1582.22
[-0.66400105] [10.72] 1589.73
[-0.66400105] [5.28] 1599.75
[-0.66400105] [-5.17] 1593.0599999999997
[-0.66400105] [-14.67] 1575.3600000000001
...
[-0.66400105] [-52.73] 2150.04
[-0.66400105] [21.91] 2138.35
[-0.66400105] [-32.3] 2104.5699999999997
[-0.66400105] [57.] 2137.79
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
[-0.66400105] [-19.2] 2080.13
[-0.66400105] [-31.6] 2056.64
[-0.66400105] [35.31] 2057.2300000000005
[-0.66400105] [-28.77] 1994.8099999999997
[-0.66400105] [27.38] 2025.74
[-0.66400105] [-9.2] 1993.59
[-0.66400105] [-12.92] 1938.5
[-0.66400105] [-8.46] 1910.57
[-0.66400105] [17.45] 1937.18
[-0.66400105] [-5.54] 1888.9699999999998
[-0.66400105] [8.56] 1860.42
[-0.66400105] [-8.31] 1876.52
[-0.66400105] [19.65] 1826.25
[-0.66400105] [-42.04] 1826.53
[-0.66400105] [36.9] 1791.91
[-0.66400105] [-23.74] 1789.2299999999998
[-0.66400105] [14.76] 1794.29
[-0.66400105] [-4.66] 1747.18
[-0.66400105] [-20.1] 1751.14
[-0.66400105] [31.01] 1750.3799999999999
[-0.66400105] [-6.42] 1723.1000000000001
[-0.66400105] [1.22] 1729.43
[-0.66400105] [-2.84] 1729.66
[-0.66400105] [8.66] 1679.07
[-0.66400105] [-15.78] 1697.05
...
[-0.66400105] [0.98] 1976.29
[-0.66400105] [7.34] 1932.7099999999998
[-0.66400105] [-28.61] 1944.81
[-0.66400105] [39.98] 1919.7100000000003



[-0.66400105] [39.98] 1919.7100000000003
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
so, the difference values are so large in comparision to the predictions that it makes miniscule change in the y_val, and y_val_hat plots
plt.plot(data['load'].values[1+288+2+split_idx:])
plt.plot(y_val_hat)
[<matplotlib.lines.Line2D at 0x7fe53549c710>]
np.array([1]).shape
(1,)



datetime	
2018-10-24 00:00:00	2520.40
2018-10-24 00:05:00	2476.88
2018-10-24 00:10:00	2479.99

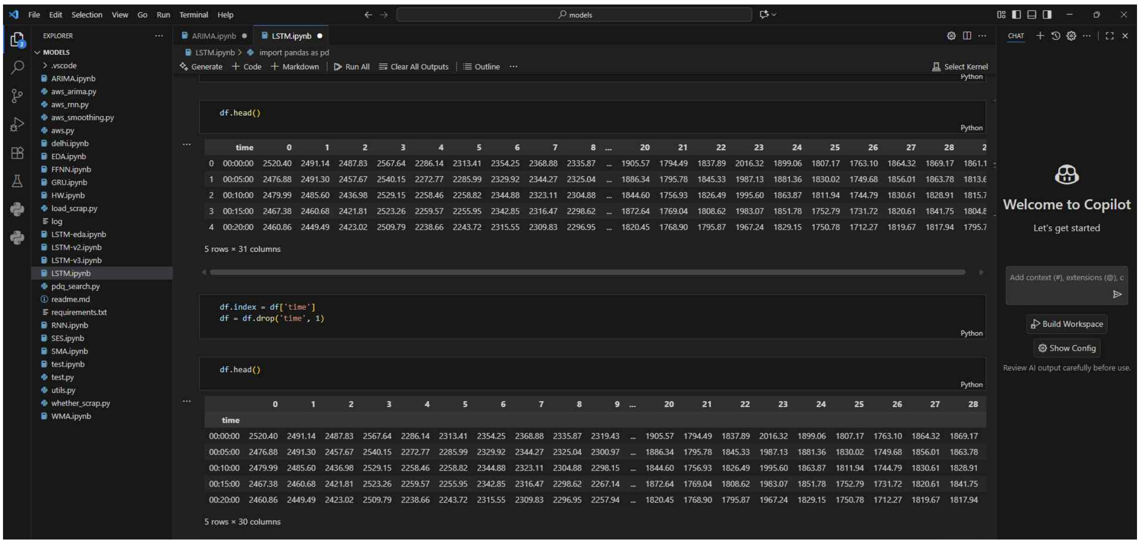
```python
data.at_time('00:05:00').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4b6c0f5f8>
```



```python
df = pd.DataFrame(columns=['time'] + list(map(str, range(30))))
```

```python
for idx, time in enumerate(sorted(set(data.index.time))):
    df.loc[idx] = [time.strftime(format='%H:%M:%S')] + list(data.at_time(time)['load'].values)
#     data.at_time(time).plot()
#     if idx>10: break
```



```python
df.head()
```

	time	0	1	2	3	4	5	6	7	8	...	20	21	22	23	24	25	26	27	28	2
0	00:00:00	2520.40	2491.14	2487.83	2567.64	2286.14	2313.41	2354.25	2368.88	2335.87	...	1905.57	1794.49	1837.89	2016.32	1899.06	1807.17	1763.10	1864.32	1869.17	1861.1
1	00:05:00	2476.88	2491.30	2457.67	2540.15	2272.77	2285.99	2329.92	2344.27	2325.04	...	1886.34	1795.78	1845.33	1987.13	1881.36	1830.02	1749.68	1856.01	1863.78	1813.6
2	00:10:00	2479.99	2485.60	2436.98	2529.15	2258.46	2258.82	2344.88	2323.11	2304.88	...	1844.60	1756.93	1826.49	1995.60	1863.87	1811.94	1744.79	1830.61	1828.91	1815.7
3	00:15:00	2467.38	2460.68	2421.81	2523.26	2259.57	2255.95	2342.85	2316.47	2298.62	...	1872.64	1769.04	1808.62	1983.07	1851.78	1752.79	1731.72	1820.61	1841.75	1804.8
4	00:20:00	2460.86	2449.49	2423.02	2509.79	2238.66	2243.72	2315.55	2309.83	2296.95	...	1820.45	1768.90	1795.87	1967.24	1829.15	1750.78	1712.27	1819.67	1817.94	1795.7

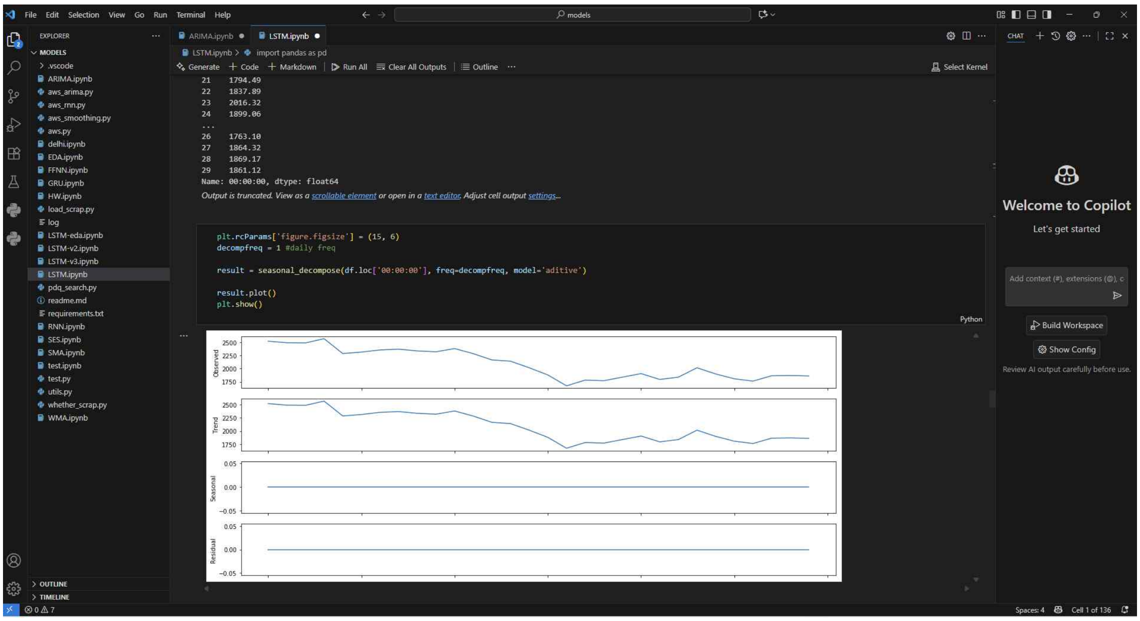
5 rows × 31 columns

```python
df.index = df['time']
df = df.drop('time', 1)
```

```python
df.head()
```

time	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28
00:00:00	2520.40	2491.14	2487.83	2567.64	2286.14	2313.41	2354.25	2368.88	2335.87	2319.43	...	1905.57	1794.49	1837.89	2016.32	1899.06	1807.17	1763.10	1864.32	1869.17
00:05:00	2476.88	2491.30	2457.67	2540.15	2272.77	2285.99	2329.92	2344.27	2325.04	2300.97	...	1886.34	1795.78	1845.33	1987.13	1881.36	1830.02	1749.68	1856.01	1863.78
00:10:00	2479.99	2485.60	2436.98	2529.15	2258.46	2258.82	2344.88	2323.11	2304.88	2298.15	...	1844.60	1756.93	1826.49	1995.60	1863.87	1811.94	1744.79	1830.61	1828.91
00:15:00	2467.38	2460.68	2421.81	2523.26	2259.57	2255.95	2342.85	2316.47	2298.62	2267.14	...	1872.64	1769.04	1808.62	1983.07	1851.78	1752.79	1731.72	1820.61	1841.75
00:20:00	2460.86	2449.49	2423.02	2509.79	2238.66	2243.72	2315.55	2309.83	2296.95	2257.94	...	1820.45	1768.90	1795.87	1967.24	1829.15	1750.78	1712.27	1819.67	1817.94

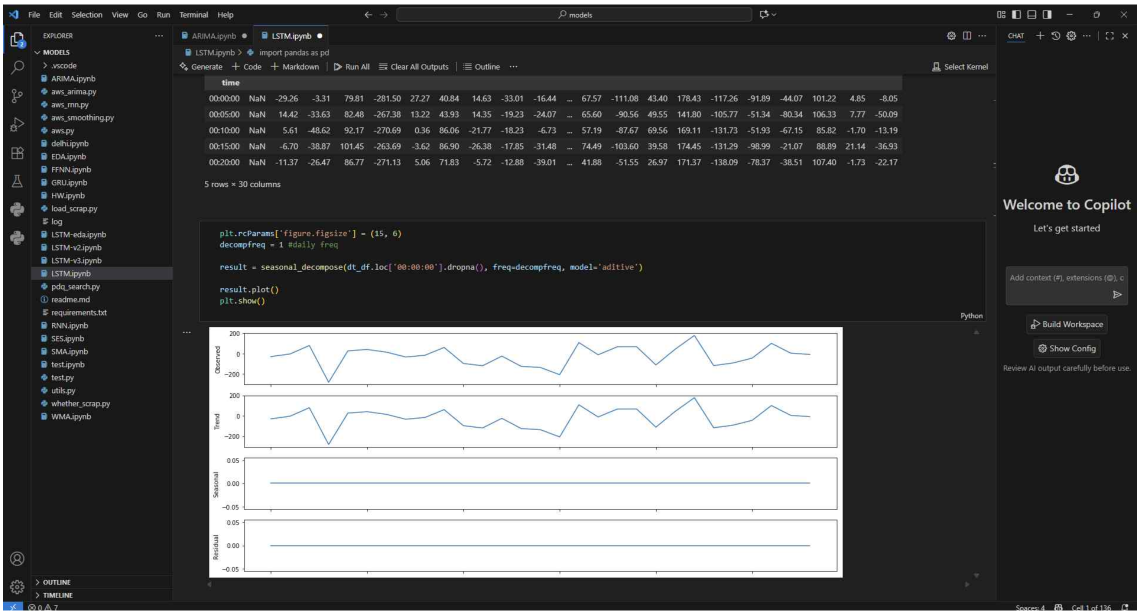
5 rows × 30 columns



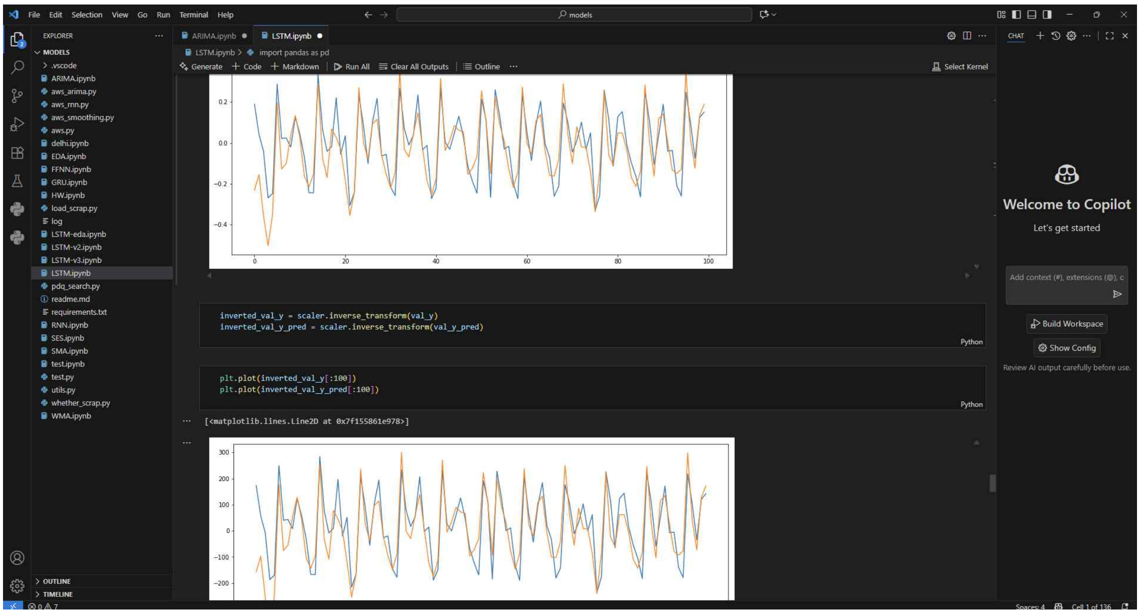
```
21    1794.49
22    1837.89
23    2016.32
24    1899.06
...
26    1763.10
27    1864.32
28    1869.17
29    1861.12
Name: 00:00:00, dtype: float64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

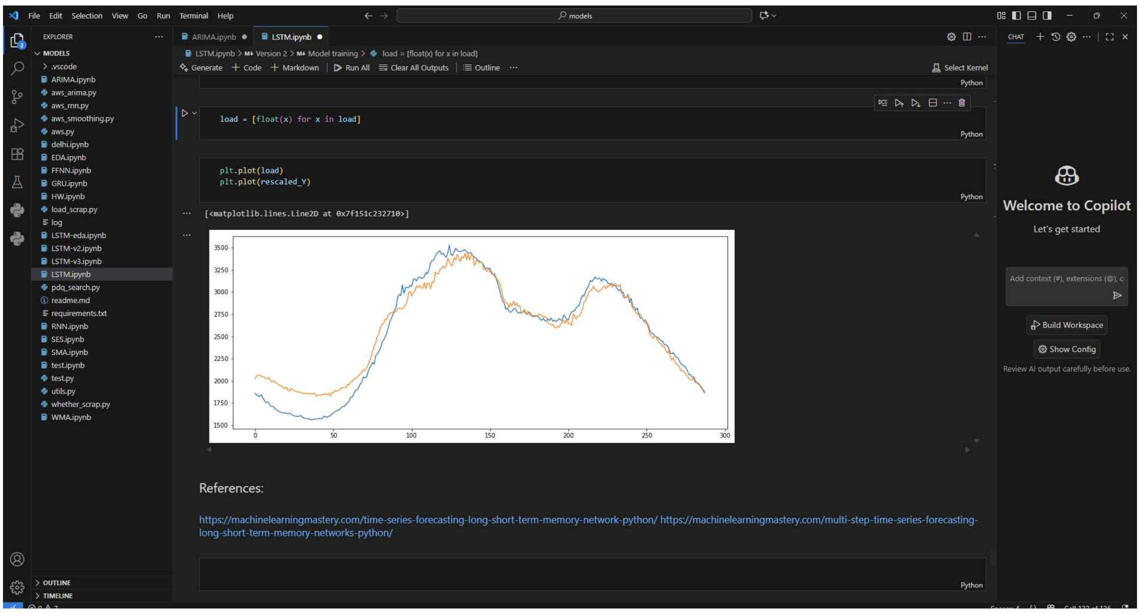
```python
plt.rcParams['figure.figsize'] = (15, 6)
decompfreq = 1 #daily freq

result = seasonal_decompose(df.loc['00:00:00'], freq=decompfreq, model='aditive')

result.plot()
plt.show()
```



LSTM.ipynb > import pandas as pd
time
00:00:00 NaN -29.26 -3.31 79.81 -281.50 27.27 40.84 14.63 -33.01 -16.44 ... 67.57 -111.08 43.40 178.43 -117.26 -91.89 -44.07 101.22 4.85 -8.05
00:05:00 NaN 14.42 -33.63 82.48 -267.38 13.22 43.93 14.35 -19.23 -24.07 ... 65.60 -90.56 49.55 141.80 -105.77 -51.34 -80.34 106.33 7.77 -50.09
00:10:00 NaN 5.61 -48.62 92.17 -270.69 0.36 86.06 -21.77 -18.23 -6.73 ... 57.19 -87.67 69.56 169.11 -131.73 -51.93 -67.15 85.82 -1.70 -13.19
00:15:00 NaN -6.70 -38.87 101.45 -263.69 -3.62 86.90 -26.38 -17.85 -31.48 ... 74.49 -103.60 39.58 174.45 -131.29 -98.99 -21.07 88.89 21.14 -36.93
00:20:00 NaN -11.37 -26.47 86.77 -271.13 5.06 71.83 -5.72 -12.88 -39.01 ... 41.88 -51.55 26.97 171.37 -138.09 -78.37 -38.51 107.40 -1.73 -22.17
5 rows × 30 columns
plt.rcParams['figure.figsize'] = (15, 6)
decompfreq = 1 #daily freq
result = seasonal_decompose(dt_df.loc['00:00:00'].dropna(), freq=decompfreq, model='aditive')
result.plot()
plt.show()
Observed
Trend
Seasonal
Residual
Welcome to Copilot
Let's get started



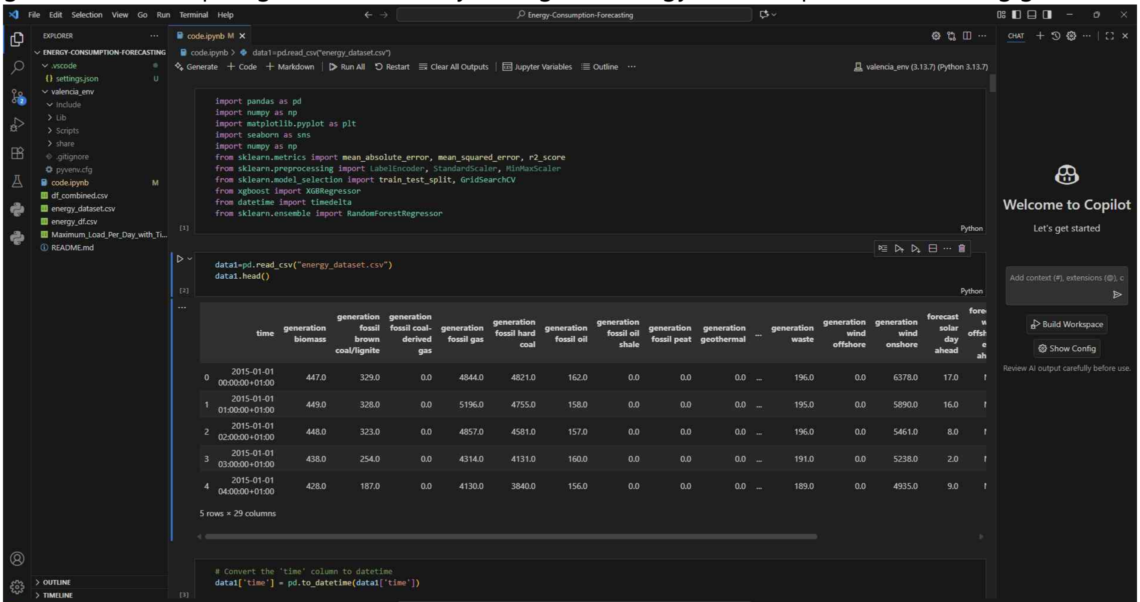
inverted_val_y = scaler.inverse_transform(val_y)
inverted_val_y_pred = scaler.inverse_transform(val_y_pred)
plt.plot(inverted_val_y[:100])
plt.plot(inverted_val_y_pred[:100])
[<matplotlib.lines.Line2D at 0x7f155861e978>]
Welcome to Copilot
Let's get started

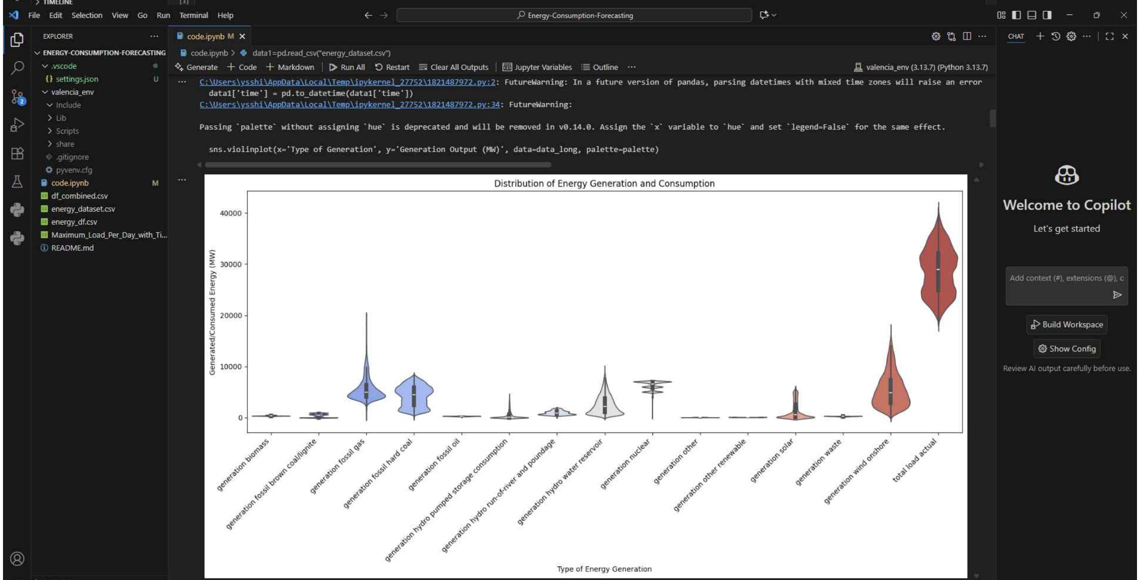


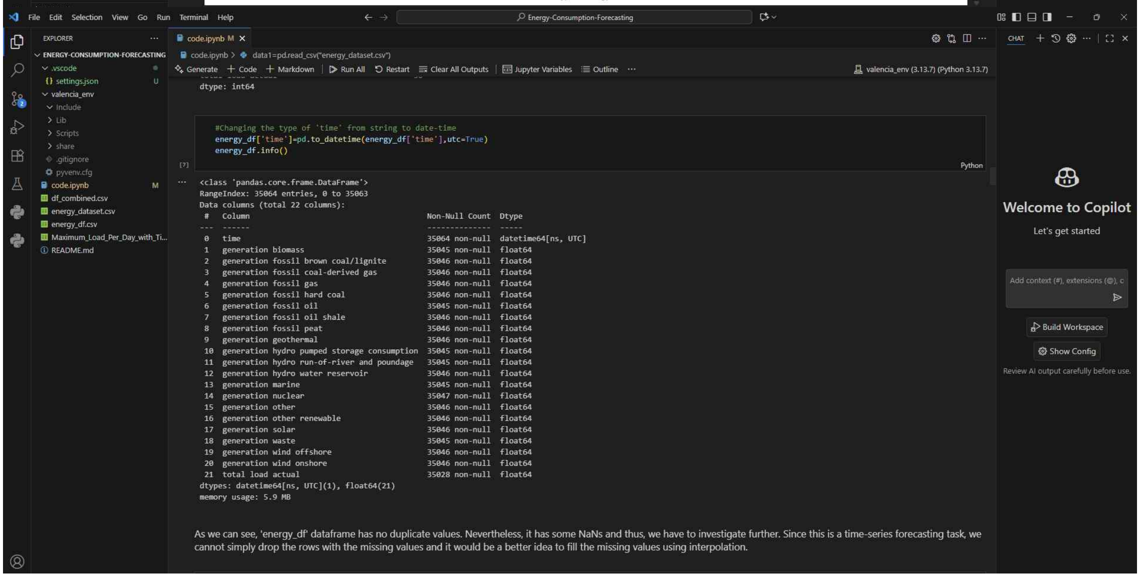
LSTM.ipynb > Version 2 > Model training > load = [float(x) for x in load]
load = [float(x) for x in load]
plt.plot(load)
plt.plot(rescaled_Y)
[<matplotlib.lines.Line2D at 0x7f151c232710>]
References:
https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/ https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/
Welcome to Copilot
Let's get started

2-1. code.ipynb 실행 결과

github주소 : https://github.com/ShreyasLengade/Energy-Consumption-Forecasting.git

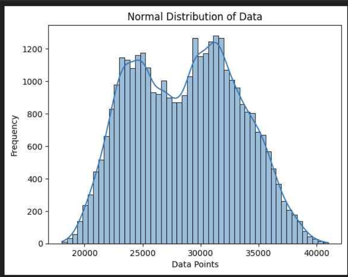






Most null values can be found in the 'total load actual' column which represents the energy consumption. Therefore, it is a good idea to visualize it and see what we can do. The similar numbers in null values in the columns which have to do with the type of energy generation probably indicate that they will also appear in the same rows. Let us first define a normal distribution to see the irregualrities.

```python
# Plotting the distribution
sns.histplot(energy_df['total load actual'], kde=True)  # 'kde=True' adds the Kernel Density Estimate to smooth the histogram
plt.title('Normal Distribution of Data')
plt.xlabel('Data Points')
plt.ylabel('Frequency')
plt.show()
```



Now lets see using a line plot.

```python
# Zoom into the plot of the hourly (actual) total load

ax = plot_series(df=energy_df, column='total load actual', ylabel='Total Load (MWh)',
                 title='Actual Total Load (First 2 weeks - Original)', end=24*7*2)
plt.show()
```

After zooming into the first 2 weeks of the 'total load actual' column, we can already see that there are null values for a few hours. However, the number of the missing values and the behavior of the series indicate that an interpolation would fill the NaNs quite well. Let us further investigate if the null values coincide across the different columns. Let us display the last five rows.

After zooming into the first 2 weeks of the 'total load actual' column, we can already see that there are null values for a few hours. However, the number of the missing values and the behavior of the series indicate that an interpolation would fill the NaNs quite well. Let us further investigate if the null values coincide across the different columns. Let us display the last five rows.

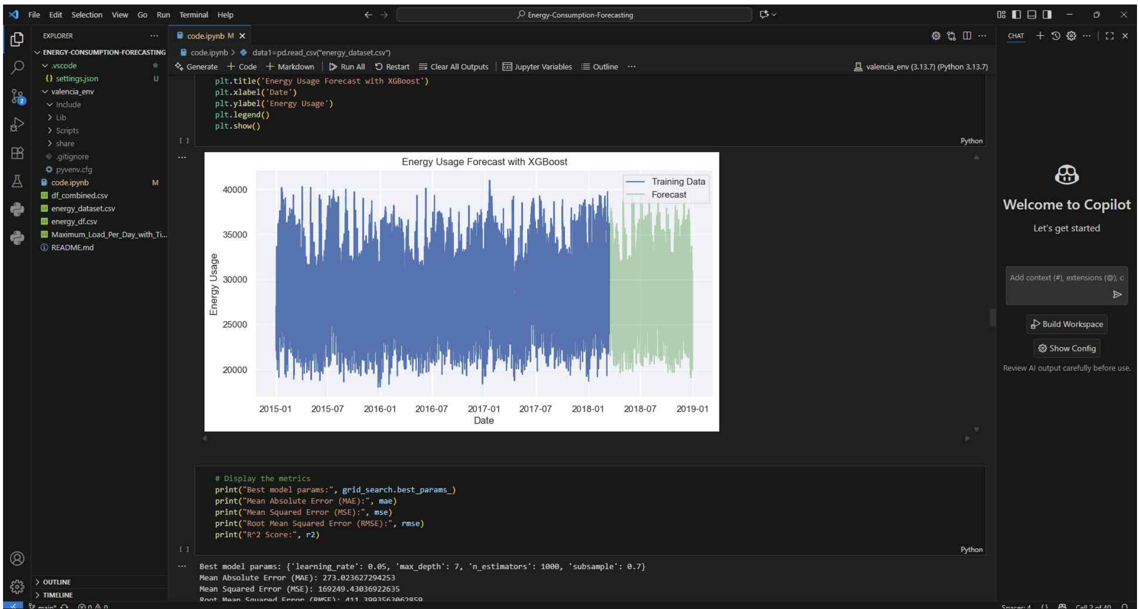
```python
# Display the rows with null values
energy_df[energy_df.isnull().any(axis=1)].tail()
```

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...	generation hydro water reservoir	generation marine	generation nuclear	generation other
16612	2016-11-23 03:00:00+00:00	NaN	900.0	0.0	4838.0	4547.0	269.0	0.0	0.0	0.0	...	435.0	0.0	5040.0	60.0
25164	2017-11-14 11:00:00+00:00	0.0	0.0	0.0	10064.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
25171	2017-11-14 18:00:00+00:00	0.0	0.0	0.0	12336.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
30185	2018-06-11 16:00:00+00:00	331.0	506.0	0.0	7538.0	5360.0	300.0	0.0	0.0	0.0	...	4258.0	0.0	5856.0	52.0
30896	2018-07-11 07:00:00+00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

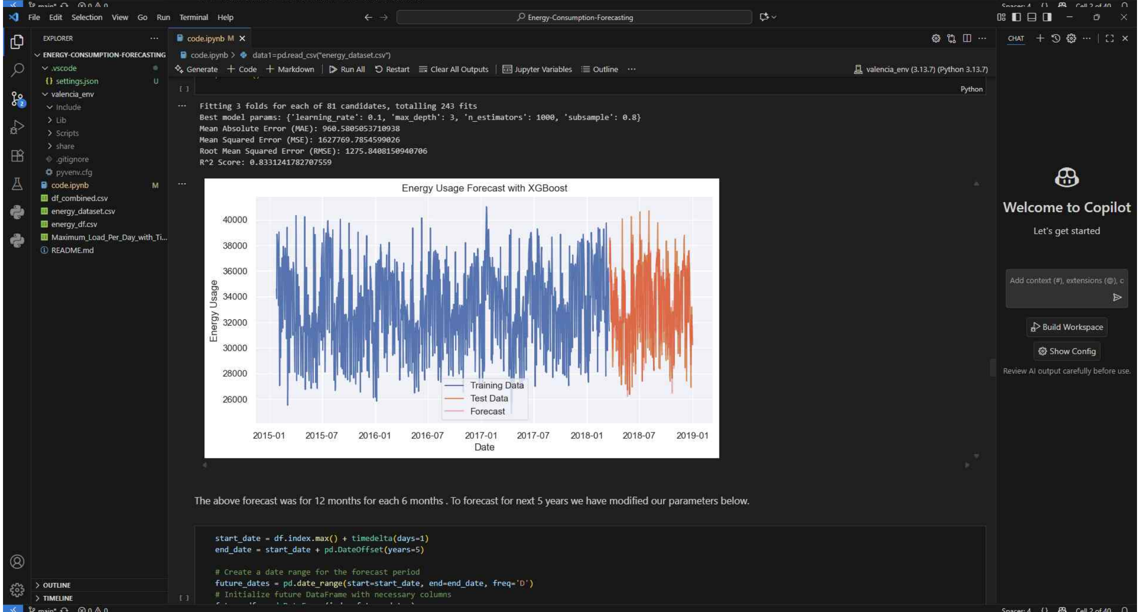
5 rows × 22 columns

If we manually searched through all of them, we would confirm that the null values in the columns which have to do with the type of energy generation mostly coincide. The null values in 'actual total load' also coincide with the aforementioned columns, but also appear in other rows as well. In order to handle the null values in df_energy, we will use a linear interpolation with a forward direction. Perhaps other kinds of interpolation would be better; nevertheless, we prefer to use the simplest model possible. Only a small part of our input data will be noisy and it will not affect performance noticeably.

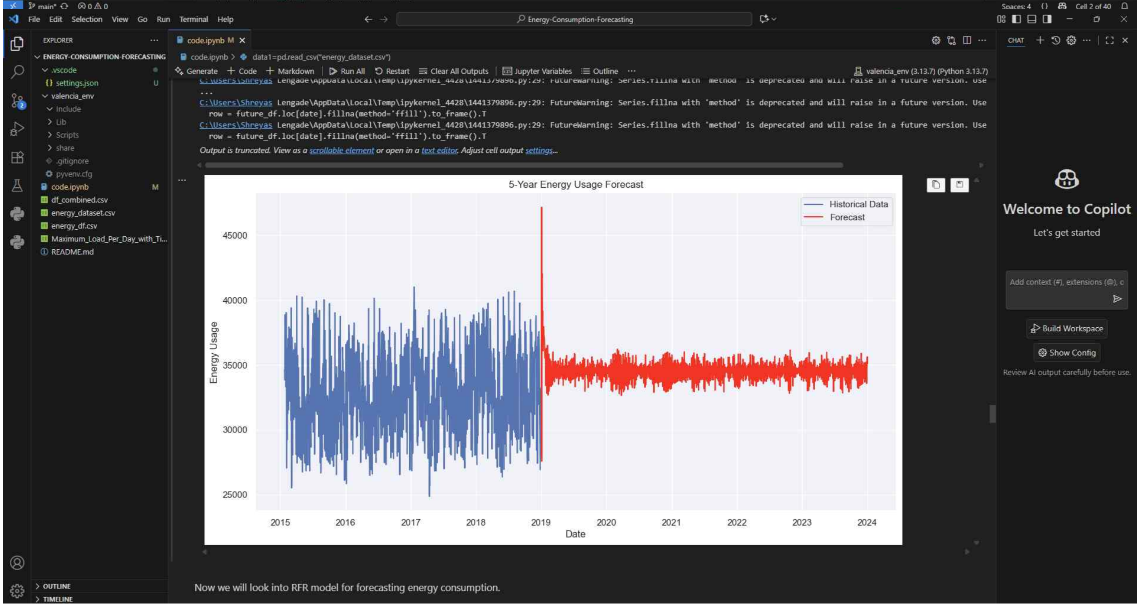
```python
energy_df.replace(0, np.nan, inplace=True)
# Fill null values using interpolation
energy_df.interpolate(method='linear', limit_direction='forward', inplace=True, axis=0)
# Display the number of non-zero values in each column

print('Non-zero values in each column:\n', energy_df.astype(bool).sum(axis=0), sep='\n')
```



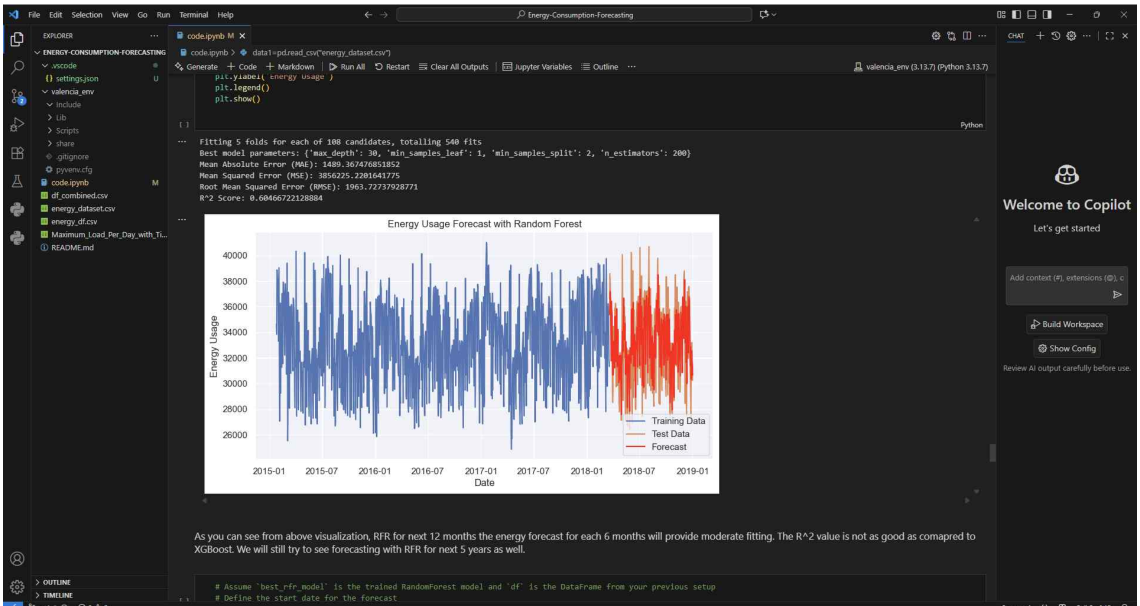
Energy-Consumption-Forecasting
code.ipynb M
data1=pd.read_csv("energy_dataset.csv")
plt.title('Energy Usage Forecast with XGBoost')
plt.xlabel('Date')
plt.ylabel('Energy Usage')
plt.legend()
plt.show()
Energy Usage Forecast with XGBoost
Training Data
Forecast
Energy Usage
Date
# Display the metrics
print("Best model params:", grid_search.best_params_)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R^2 Score:", r2)
Best model params: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 1000, 'subsample': 0.7}
Mean Absolute Error (MAE): 273.023627294253
Mean Squared Error (MSE): 169249.43036922635
Welcome to Copilot
Let's get started



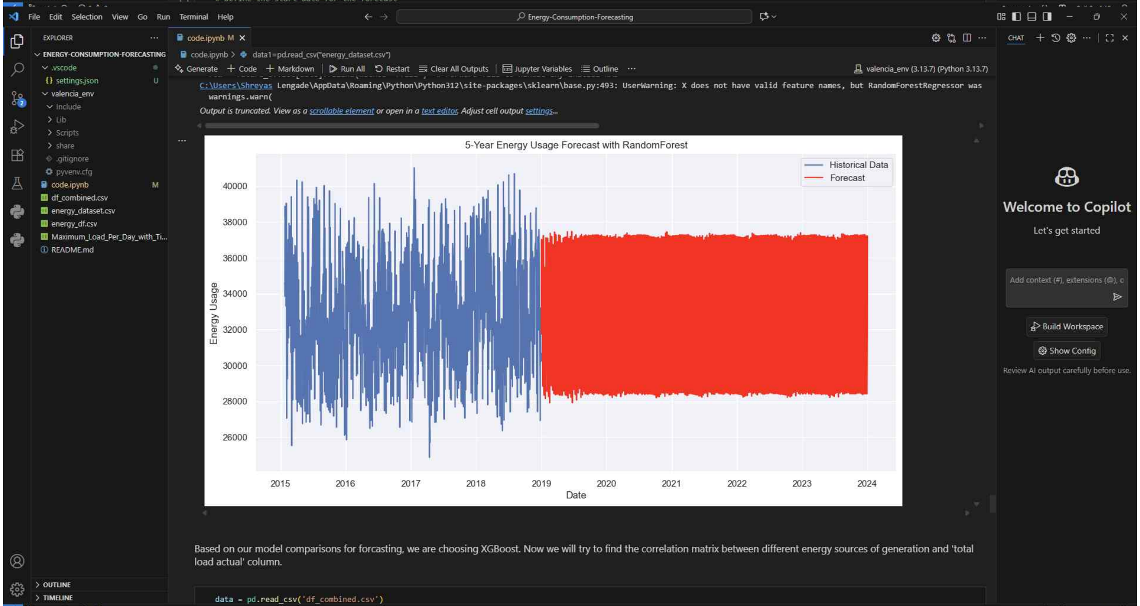
Energy-Consumption-Forecasting
code.ipynb M
Fitting 3 folds for each of 81 candidates, totalling 243 fits
Best model params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 1000, 'subsample': 0.8}
Mean Absolute Error (MAE): 960.5805053710938
Mean Squared Error (MSE): 1627769.7854599026
Root Mean Squared Error (RMSE): 1275.8408150940706
R^2 Score: 0.8331241782707559
Energy Usage Forecast with XGBoost
Training Data
Test Data
Forecast
Energy Usage
Date
The above forecast was for 12 months for each 6 months . To forecast for next 5 years we have modified our parameters below.
start_date = df.index.max() + timedelta(days=1)
end_date = start_date + pd.DateOffset(years=5)
# Create a date range for the forecast period
future_dates = pd.date_range(start=start_date, end=end_date, freq='D')
# Initialize future DataFrame with necessary columns
Welcome to Copilot
Let's get started



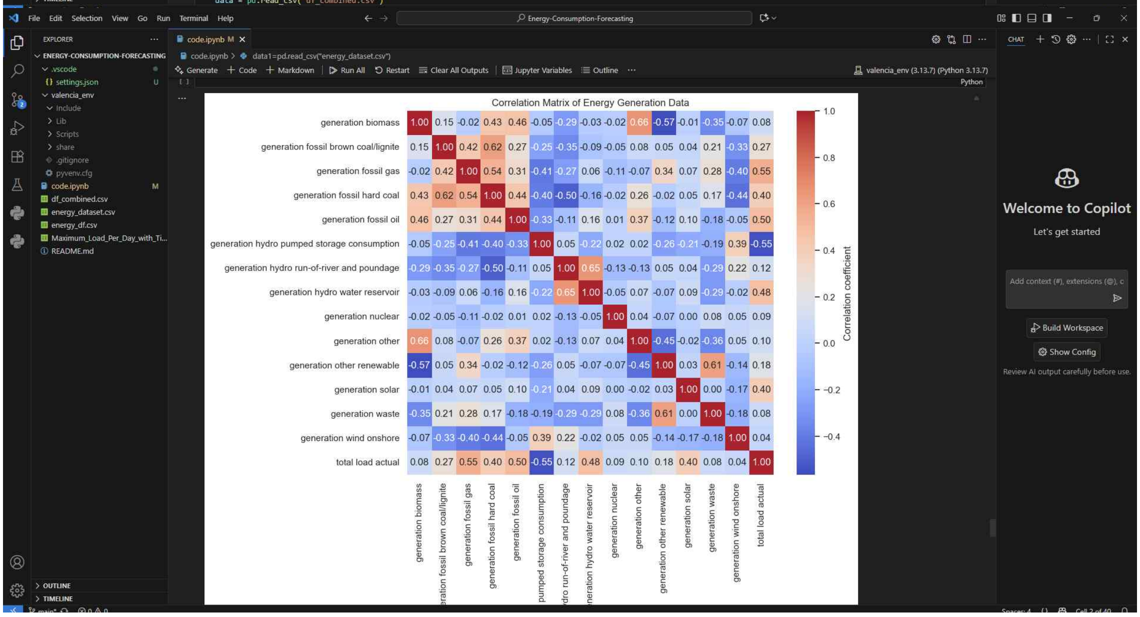
Energy-Consumption-Forecasting
code.ipynb M
C:\Users\Shreyas Lengade\AppData\Local\Temp\ipykernel_4428\1441379896.py:29: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use
row = future_df.loc[date].fillna(method='ffill').to_frame().T
C:\Users\Shreyas Lengade\AppData\Local\Temp\ipykernel_4428\1441379896.py:29: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use
row = future_df.loc[date].fillna(method='ffill').to_frame().T
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
5-Year Energy Usage Forecast
Historical Data
Forecast
Energy Usage
Date
Now we will look into RFR model for forecasting energy consumption.
Welcome to Copilot
Let's get started



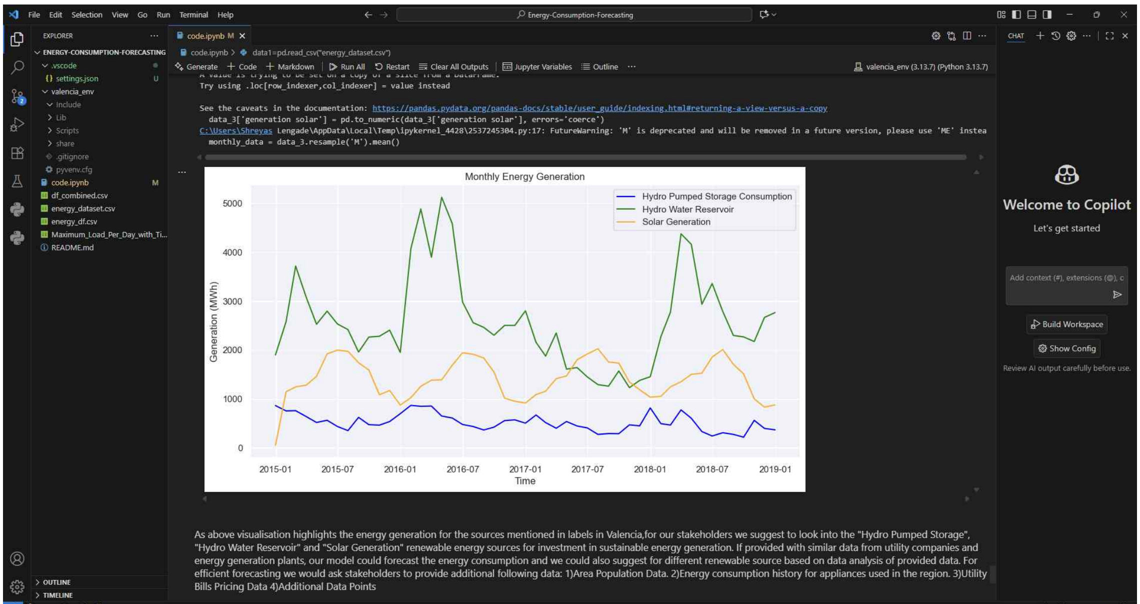
Energy-Consumption-Forecasting
code.ipynb M
data1=pd.read_csv("energy_dataset.csv")
valencia_env (3.13.7) (Python 3.13.7)
plt.legend()
plt.show()
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best model parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Mean Absolute Error (MAE): 1489.367476851852
Mean Squared Error (MSE): 3856225.2201641775
Root Mean Squared Error (RMSE): 1963.72737928771
R^2 Score: 0.60466722128884
Energy Usage Forecast with Random Forest
Energy Usage
Date
Training Data
Test Data
Forecast
As you can see from above visualization, RFR for next 12 months the energy forecast for each 6 months will provide moderate fitting. The R^2 value is not as good as comapred to XGBoost. We will still try to see forecasting with RFR for next 5 years as well.
Assume `best_rfr_model` is the trained RandomForest model and `df` is the DataFrame from your previous setup
Define the start date for the forecast
Welcome to Copilot
Let's get started



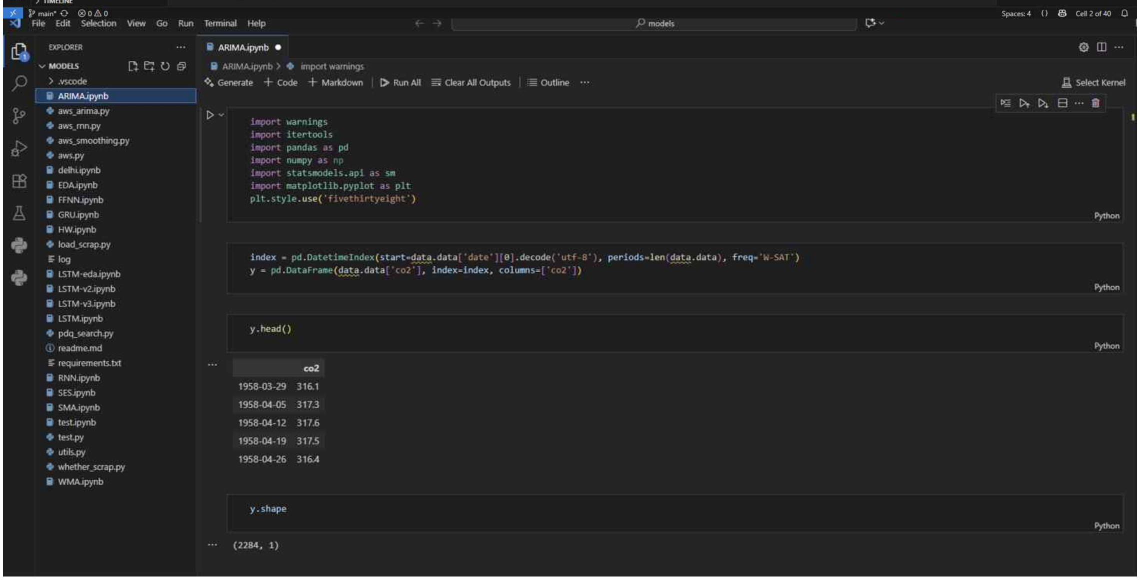
Energy-Consumption-Forecasting
code.ipynb M
C:\Users\Shreyas Lengade\AppData\Roaming\Python\Python312\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but RandomForestRegressor was
warnings.warn(
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
5-Year Energy Usage Forecast with RandomForest
Historical Data
Forecast
Energy Usage
Date
Based on our model comparisons for forcasting, we are choosing XGBoost. Now we will try to find the correlation matrix between different energy sources of generation and 'total load actual' column.
data = pd.read_csv('df_combined.csv')
Welcome to Copilot
Let's get started



Energy-Consumption-Forecasting
code.ipynb M
Correlation Matrix of Energy Generation Data
generation biomass 1.00 0.15 -0.02 0.43 0.46 -0.05 -0.29 -0.03 -0.02 0.66 -0.57 -0.01 -0.35 -0.07 0.08
generation fossil brown coal/lignite 0.15 1.00 0.42 0.62 0.27 -0.25 -0.35 -0.09 -0.05 0.08 0.05 0.04 0.21 -0.33 0.27
generation fossil gas -0.02 0.42 1.00 0.54 0.31 -0.41 -0.27 0.06 -0.11 -0.07 0.34 0.07 0.28 -0.40 0.55
generation fossil hard coal 0.43 0.62 0.54 1.00 0.44 -0.40 -0.50 -0.16 -0.02 0.26 -0.02 0.05 0.17 -0.44 0.40
generation fossil oil 0.46 0.27 0.31 0.44 1.00 -0.33 -0.11 0.16 0.01 0.37 -0.12 0.10 -0.18 -0.05 0.50
generation hydro pumped storage consumption -0.05 -0.25 -0.41 -0.40 -0.33 1.00 0.05 -0.22 0.02 0.02 -0.26 -0.21 -0.19 0.39 -0.55
generation hydro run-of-river and poundage -0.29 -0.35 -0.27 -0.50 -0.11 0.05 1.00 0.65 -0.13 -0.13 0.05 0.04 -0.29 0.22 0.12
generation hydro water reservoir -0.03 -0.09 0.06 -0.16 0.16 -0.22 0.65 1.00 -0.05 0.07 -0.07 0.09 -0.29 -0.02 0.48
generation nuclear -0.02 -0.05 -0.11 -0.02 0.01 0.02 -0.13 -0.05 1.00 0.04 -0.07 0.00 0.08 0.05 0.09
generation other 0.66 0.08 -0.07 0.26 0.37 0.02 -0.13 0.07 0.04 1.00 -0.45 -0.02 -0.36 0.05 0.10
generation other renewable -0.57 0.05 0.34 -0.02 -0.12 -0.26 0.05 -0.07 -0.07 -0.45 1.00 0.03 0.61 -0.14 0.18
generation solar -0.01 0.04 0.07 0.05 0.10 -0.21 0.04 0.09 0.00 -0.02 0.03 1.00 0.00 -0.17 0.40
generation waste -0.35 0.21 0.28 0.17 -0.18 -0.19 -0.29 -0.29 0.08 -0.36 0.61 0.00 1.00 -0.18 0.08
generation wind onshore -0.07 -0.33 -0.40 -0.44 -0.05 0.39 0.22 -0.02 0.05 0.05 -0.14 -0.17 -0.18 1.00 0.04
total load actual 0.08 0.27 0.55 0.40 0.50 -0.55 0.12 0.48 0.09 0.10 0.18 0.40 0.08 0.04 1.00
Correlation coefficient
Welcome to Copilot
Let's get started



File Edit Selection View Go Run Terminal Help
Energy-Consumption-Forecasting
EXPLORER
ENERGY-CONSUMPTION-FORECASTING
.vscode
settings.json
valencia_env
Include
Lib
Scripts
share
.gitignore
pyvenv.cfg
code.ipynb
df_combined.csv
energy_dataset.csv
energy_df.csv
Maximum_Load_Per_Day_with_Ti...
README.md
OUTLINE
TIMELINE
code.ipynb M
code.ipynb > data1=pd.read_csv("energy_dataset.csv")
Generate + Code + Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
valencia_env (3.13.7) (Python 3.13.7)
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
data_3['generation solar'] = pd.to_numeric(data_3['generation solar'], errors='coerce')
C:\Users\Shreyas Lengade\AppData\Local\Temp\ipykernel_4428\2537245304.py:17: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instea
monthly_data = data_3.resample('M').mean()
Monthly Energy Generation
Hydro Pumped Storage Consumption
Hydro Water Reservoir
Solar Generation
Generation (MWh)
0
1000
2000
3000
4000
5000
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
Time
As above visualisation highlights the energy generation for the sources mentioned in labels in Valencia,for our stakeholders we suggest to look into the "Hydro Pumped Storage", "Hydro Water Reservoir" and "Solar Generation" renewable energy sources for investment in sustainable energy generation. If provided with similar data from utility companies and energy generation plants, our model could forecast the energy consumption and we could also suggest for different renewable source based on data analysis of provided data. For efficient forecasting we would ask stakeholders to provide additional following data: 1)Area Population Data. 2)Energy consumption history for appliances used in the region. 3)Utility Bills Pricing Data 4)Additional Data Points
CHAT
Welcome to Copilot
Let's get started
Build Workspace
Show Config
Review AI output carefully before use.
main*
Spaces: 4
Cell 2 of 40



File Edit Selection View Go Run Terminal Help
models
EXPLORER
MODELS
.vscode
ARIMA.ipynb
aws_arima.py
aws_rnn.py
aws_smoothing.py
aws.py
delhi.ipynb
EDA.ipynb
FFNN.ipynb
GRU.ipynb
HW.ipynb
load_scrap.py
log
LSTM-eda.ipynb
LSTM-v2.ipynb
LSTM-v3.ipynb
LSTM.ipynb
pdq_search.py
readme.md
requirements.txt
RNN.ipynb
SES.ipynb
SMA.ipynb
test.ipynb
test.py
utils.py
whether_scrap.py
WMA.ipynb
ARIMA.ipynb
ARIMA.ipynb > import warnings
Generate + Code + Markdown Run All Clear All Outputs Outline
Select Kernel
import warnings
import itertools
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
Python
index = pd.DatetimeIndex(start=data.data['date'][0].decode('utf-8'), periods=len(data.data), freq='W-SAT')
y = pd.DataFrame(data.data['co2'], index=index, columns=['co2'])
Python
y.head()
Python
co2
1958-03-29 316.1
1958-04-05 317.3
1958-04-12 317.6
1958-04-19 317.5
1958-04-26 316.4
y.shape
Python
(2284, 1)