Appliance Energy Use Prediction


```
Author: Alexis Kaldany
Spring 2022
DATS_6313_10
Overseen by Professor Reza Jafari
```

Abstract

Introduction

Description of Data

- The data set is at 10 min for about 4.5 months. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network. Each wireless node transmitted the temperature and humidity conditions around 3.3 min. Then, the wireless data was averaged for 10 minutes periods. The energy data was logged every 10 minutes with m-bus energy meters.
- Weather from the nearest airport weather station (Chievres Airport, Belgium) was downloaded from a public data set from Reliable Prognosis (rp5.ru), and merged together with the experimental data sets using the date and time column.
- Two random variables have been included in the data set for testing the regression models and to filter out non predictive attributes (parameters).

Visualization of Variable Locations

Table of Variables

- All temperatures in Celsius
- All humidity is in % terms
- All energy use in watts per hour

Variable	Description	Variable	Description
date	10 minute intervals	Appliances	energy use of appliances
lights	energy use of light fixtures	Press_mm_hg	Pressure
RH_1	Humidity in kitchen	T1	Temp in kitchen
RH_2	Humidity in living room	T2	Temp in living room
RH_3	Humidity in laundry area	T3	Temp in laundry area
RH_4	Humidity in office room	T4	Temp in office room
RH_5	Humidity in bathroom	T5	Temp in bathroom
RH_6	Humidity in north area	T6	Temp in north area
RH_7	Humidity in ironing room	T7	Temp in ironing room
RH_8	Humidity in teenager room	T8	Temp in teenager room
RH_9	Humidity in parents room	T9	Temp in parents room
RH_out	Humidity outside	T_out	Temp outside
Windspeed	in m/s	Visibility	kilometers
rv1	random variable 1	rv2	andom variable 2
Tdewpoint	A*C		

1a

- Below is the plot of the dependent variable "Appliances" against time. It should be noted that much of the time the value of the dependent variable hovers around 0.

- Lag for 90 was chosen. Significant autocorrelation across time.

- There is a classic checkbox pattern in this heatplot. This indicates a variety of relationships among the independent variables.

- The PACF is interesting as there is very little partial correlation after the first few time-steps.

Stationarity

- I attempted to find a combination of differences and diff-logarithmic transformations that would eliminate the trend seen in the mean in the below rolling mean/variance of the original data set.

- The original dataset had KPSS and ADF values that indicated stationarity, but I sought improved stationarity to ensure the remaining tests/methods were total stabile. So I took the difference and difference-logarithm up to an interval of 300, and graphed the change in ADF and KPSS values, looking for a point of maximum stability.

- The adf values didn't really have any big movement towards stationarym, just trends away fron stationarity.

- The KPSS statistics showed decreasing stationarity as lags increased, after an initial improvement

- Below we see the rolling mean/variance of the 150 interval lag. The mean clearly is extremely stable, however the variance is slowly decline, at a rate worse than the original set.

- Taking the log of the difference doesn't help the trend in the variance either, and the mean is about as placid as in the plain 150 difference set and the original set.

- I chose at this point to keep the original data undifferenced and otherwise as is rather than mess about with more attempt at making it stationary.

Decomposition

- I did this section on both data differenced at the 144 interval (daily differencing) and on the original data
- Below we see the PACF at the 144 interval.

- Below we see the symetrical ACF of the 144 differenced data.

- Below is the decomposition of the original dependent variable.

- Below is the decomposition of the 144 period differenced dependent variable.

- Below is the decomposition of the differenced dependent variable.

- Below is the adjusted seasonal data compared to the un-adjusted 144 differenced data.

- Below is the detrended decomposition compared to the original set. It is impossible for the values to be negative, so the detrended data isn't all that useful.

Base Models

Holt-Winters

- Below is the rolling mean/variance of the prediction of Holt-Winters. Clearly the prediction stabilizes at a value.

- Below is the plot of the train, test, and forecast of the Holt-Winters model.

- Below is a graph of just the test and forecast data. Holt-Winters results in a cyclical predictions apparently.

- Below is the ACF of the Holt-Winters prediction array.

- Below is the ACF-PACF of the forecast values of the Holt-Winters model.

Average Method

- Below is the train-test-predict for the Average Method.

- Below is the test and forecast of the average method.

- Below is the ACF of the error of the forecast values.

Naive Method

- Below is the plot train-test-forecast of the Naive method.

- Below is the plot of the test and forecast for the Naive Method.

- Below is the plot of the ACF of the errors

Drift Method

- Below is the plot of the train-test-forecast for the Drift method.

- Below is the plot of the test data and the forecast data.

- Below is the ACF of the errors of the forecast-test arrays. The ACF indicates significant ACF values.

SES Method

- Below is the train-test-forecast of the SES method.

- Below is the test-forecast of the SES method.

- Below is the ACF of the forecast error for the SES method.

Backwards Selection and Linear Model

Before

- Below are the summary statistics of the linear regression before any variables are removed.

OLS Regression Results			
Dep. Variable:	Appliances	R-squared (uncentered):	0.557
Model:	OLS	Adj. R-squared (uncentered):	0.557
Method:	Least Squares	F-statistic:	793.7
Date:	Fri, 22 Apr 2022	Prob (F-statistic):	0.00
Time:	14:45:06	Log-Likelihood:	-94410.
No. Observations:	15788	AIC:	1.889e+05
Df Residuals:	15763	BIC:	1.891e+05
Df Model:	25		
Covariance Type:	nonrobust		

- Below is a table of the p-values of all 25 independent variables. 5 Variables have p-values greater than 0.05, indicating they are not statistically significant.

	coef	std err	t	P>|t|	[0.025	0.975]
lights	2.1405	0.105	20.446	0.000	1.935	2.346
T1	-6.3577	2.239	-2.840	0.005	-10.746	-1.969
RH_1	12.8278	0.757	16.946	0.000	11.344	14.312
T2	-13.2161	2.097	-6.304	0.000	-17.326	-9.107
RH_2	-12.4343	0.912	-13.636	0.000	-14.222	-10.647
T3	27.6756	1.242	22.278	0.000	25.241	30.111
RH_3	8.6016	0.839	10.250	0.000	6.957	10.246
T4	-3.7388	1.090	-3.429	0.001	-5.876	-1.601
RH_4	-2.4344	0.735	-3.313	0.001	-3.874	-0.994
T5	-1.6770	1.509	-1.111	0.266	-4.635	1.281
RH_5	0.1043	0.102	1.026	0.305	-0.095	0.303
T6	7.5826	0.829	9.143	0.000	5.957	9.208
RH_6	0.3347	0.080	4.178	0.000	0.178	0.492
T7	1.2152	1.527	0.796	0.426	-1.777	4.208
RH_7	-0.5249	0.493	-1.066	0.287	-1.490	0.441
T8	8.9150	1.086	8.208	0.000	6.786	11.044
RH_8	-5.9552	0.429	-13.875	0.000	-6.797	-5.114
T9	-14.6278	2.028	-7.212	0.000	-18.604	-10.652
RH_9	-0.9481	0.468	-2.024	0.043	-1.866	-0.030
T_out	-9.3318	2.563	-3.641	0.000	-14.356	-4.308
Press_mm_hg	0.1630	0.059	2.763	0.006	0.047	0.279
RH_out	-0.6707	0.460	-1.459	0.145	-1.572	0.230
Windspeed	1.1960	0.374	3.195	0.001	0.462	1.930
Visibility	0.1909	0.063	3.032	0.002	0.068	0.314
Tdewpoint	4.5302	2.399	1.889	0.059	-0.171	9.232

Omnibus:	11048.846	Durbin-Watson:	0.622
Prob(Omnibus):	0.000	Jarque-Bera (JB):	160765.586
Skew:	3.283	Prob(JB):	0.00

After

- After removing the statistically insignificant variables ['T5','RH_5','T7','RH_7','RH_out'], we get a summary statistics shown below.

OLS Regression Results			
Dep. Variable:	Appliances	R-squared (uncentered):	0.557
Model:	OLS	Adj. R-squared (uncentered):	0.557
Method:	Least Squares	F-statistic:	991.8
Date:	Fri, 22 Apr 2022	Prob (F-statistic):	0.00
Time:	14:55:19	Log-Likelihood:	-94413.
No. Observations:	15788	AIC:	1.889e+05
Df Residuals:	15768	BIC:	1.890e+05
Df Model:	20		
Covariance Type:	nonrobust		

- There is very little change in the adjusted R^2 values after removing the insignificant independent variables.

	coef	std err	t	P>|t|	[0.025	0.975]
lights	2.1491	0.104	20.614	0.000	1.945	2.353
T1	-6.7459	2.221	-3.038	0.002	-11.099	-2.393
RH_1	12.9590	0.751	17.263	0.000	11.488	14.430
T2	-13.3953	2.070	-6.470	0.000	-17.453	-9.337
RH_2	-12.5307	0.895	-14.001	0.000	-14.285	-10.776
T3	27.4552	1.215	22.602	0.000	25.074	29.836
RH_3	8.4737	0.827	10.249	0.000	6.853	10.094
T4	-3.8113	1.070	-3.563	0.000	-5.908	-1.714
RH_4	-2.6735	0.702	-3.810	0.000	-4.049	-1.298
T6	7.3525	0.801	9.177	0.000	5.782	8.923
RH_6	0.3185	0.078	4.084	0.000	0.166	0.471
T8	9.4176	0.980	9.608	0.000	7.496	11.339
RH_8	-6.3048	0.362	-17.402	0.000	-7.015	-5.595
T9	-15.1285	1.697	-8.916	0.000	-18.454	-11.803
RH_9	-0.8216	0.443	-1.855	0.064	-1.690	0.047
T_out	-5.7160	0.927	-6.165	0.000	-7.533	-3.899
Press_mm_hg	0.1025	0.031	3.260	0.001	0.041	0.164
Windspeed	1.2841	0.372	3.456	0.001	0.556	2.012
Visibility	0.1816	0.063	2.904	0.004	0.059	0.304
Tdewpoint	0.9710	0.680	1.428	0.153	-0.362	2.304

Omnibus:	11050.350	Durbin-Watson:	0.622
Prob(Omnibus):	0.000	Jarque-Bera (JB):	160868.059
Skew:	3.283	Prob(JB):	0.00
Kurtosis:	17.192	Cond. No.	3.06e+03

- The prediction and forecast values more or less resemble the train-test values, but are significantly more compressed (have lower variance)

OLS model Prediction Plot
Training set
Prediction values
Test set
Forecasted Values
Electricity (kWh)
1000
800
600
400
200
0
2016-01-11
2016-02-11
2016-03-14
2016-04-14
2016-05-15
Date

GPAC

GPAC Table
j values
k values
0 0.7 -0.097 0.11 0.059 0.038 0.011 0.0057 0.078 0.099 0.056
1 0.62 0.69 0.16 -0.013 0.021 0.0093 0.14 0.071 0.056 0.013
2 0.74 -0.49 0.14 0.31 0.019 -0.62 -0.26 0.081 0.063 -0.2
3 0.86 -0.14 0.31 0.1 1.2 -0.093 -0.1 0.051 0.028 0.087
4 0.89 1.2 0.25 -2.3 0.7 -1.2 -0.13 0.071 -0.063 0.073
5 0.85 -0.55 2.4 -0.52 -0.95 -0.3 0.062 0.12 0.21 0.055
6 0.81 4.8 3.4 -3.9 -0.56 -0.45 0.9 0.18 0.077 0.073
7 1 -0.84 1.1 -0.72 0.73 0.086 -0.35 0.41 -0.032 0.08
8 1.3 0.73 0.69 0.18 0.92 3.6 -0.21 0.44 1.4 0.09
9 1.1 -1.5 0.63 -4.4 1.1 1.4 2.9 0.39 0.15 -0.43
1 2 3 4 5 6 7 8 9 10
4
2
0
−2
−4

ARMA (3,0)

ACF plot of Stem-ACF-3-0-Errors
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

ACF plot of Stem-ACF-3-0-Residuals
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

ARMA(3,0)
Train Data
Test Data
Prediction Data
Forecasting Data
Electricity (Wh)
0
200
400
600
800
1000
2016-01-11
2016-02-11
2016-03-14
2016-04-14
2016-05-15
Time

Forecasting on ARMA(3,0) for Electricity (Wh)
Test Data
Forecasting Data
Electricity (Wh)
0
200
400
600
800
2016-04-30
2016-05-05
2016-05-10
2016-05-15
2016-05-21
2016-05-26
Time

ARMA(3,3)

ACF plot of Stem-ACF-3-3-Errors
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

ACF plot of Stem-ACF-3-3-Residuals
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

ARIMA(3,0,0) x (0,3,0,12)

ACF plot of Stem-ACF-SARIMA-Errors
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

ACF plot of Stem-ACF-SARIMA-Residuals
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−75
−50
−25
0
25
50
75
lags

ARIMA(3,0,0)x(0,3,0,12) on Electricity (Wh)
Prediction Data
Train Data
Test Data
Electricity (Wh)
2000
1500
1000
500
0
−500
−1000
−1500
2016-01-11
2016-02-11
2016-03-14
2016-04-14
2016-05-15
Date

ARIMA(3,0,0)x(0,3,0,12) on Electricity (Wh)
Prediction Data
Train Data
Test Data
Forecasting Data
Electricity (Wh)
2000
1500
1000
500
0
−500
−1000
−1500
2016-01-11
2016-02-11
2016-03-14
2016-04-14
2016-05-15
Date

LSTM

ACF plot of Stem-ACF-LSTM-Errors
Autocorrelation value
1.0
0.8
0.6
0.4
0.2
0.0
−75
−50
−25
0
25
50
75
lags

Model Summary

models	mse	ljb
Holt-Winters	80324.1355	17055.560919
Average-Method	11074.3459	21037.197590
Naive-Method	5381.1617	1129.402962
Drift-Method	5385.1842	1128.966050
SES-Method	5717.6955	2705.998085
Linear-Regression	9156.2115	12769.092638
ARMA-3-0	4614.1882	40.116973
ARMA-3-3	4527.7083	1.145882
ARIMA(3,0,0)x(0,3,0,12)	57564.6157	NaN
LSTM	11369.3577	16878.570533

Conclusion

Citations

Appendix + Code

```
# %%
from matplotlib.cbook import flatten
from toolbox import *
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller, kpss
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from statsmodels.tsa.api import SARIMAX, AutoReg
from keras.layers import Dense
from keras.layers import LSTM
from keras import Sequential
from statsmodels.tsa.seasonal import STL
from sklearn.model_selection import train_test_split
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.preprocessing.timeseries import timeseries_dataset_from_array
import statsmodels.tsa.holtwinters as ets
from sklearn.metrics import mean_squared_error
from numpy import linalg as LA
warnings.filterwarnings('ignore')
image_folder = 'final-images/'
# %%


def visualize_loss(history, title, epochs):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title(title)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.savefig(image_folder+'LSTM-Training-Val-Loss-{epochs}.png',
dpi=1000)
    plt.show()

# Lists
models = []
model_mse = []
model_ljb = []
model_error_var = []
model_notes = []
# %%
# Reading csv

df_models = pd.DataFrame(columns=['Model', 'MSE','Ljung-Box','Error-
Var','Notes'])
df_raw = pd.read_csv('final-data.csv', parse_dates=["date"])
df = df_raw.copy()
df['Date'] = pd.to_datetime(df_raw['date'])
del df['date']
del df['rv1']
del df['rv2']
#df['Date'] = pd.to_datetime(df_raw['date'])
df.set_index('Date', inplace=True)
df_undiff = df.copy()
#%%
# Dataset Cleaning
df_na = df[df.isnull().any(axis=1)]
print(f"There are {len(df_na)} many rows with NaN values")
# Checking column types
df_types = df.dtypes
print(df_types)
# All columns are either int, float, or datetime, looks good
```

```
# Description Section
# 1a. Plot of the dependent variable versus time.
plt.figure(figsize=(12, 8), layout='constrained')
plt.plot(df.index, df['Appliances'])
plt.title('Appliance Energy Use')
plt.xlabel('10 Minute Intervals')
plt.ylabel('Energy Use in Wh')
# plt.legend(bbox_to_anchor=(1, 1), loc="upper left")
plt.grid()
plt.savefig(image_folder+'1a-dependent.png', dpi=1000)
plt.show()

# 1c. ACF/PACF of the dependent variable
lags = 90
acf = sm.tsa.stattools.acf(df.Appliances, nlags=lags)
pacf = sm.tsa.stattools.pacf(df.Appliances, nlags=lags)
fig = plt.figure()
plt.subplot(211)
plt.title('ACF/PACF of the raw data')
plot_acf(df.Appliances, ax=plt.gca(), lags=lags)
plt.subplot(212)
plot_pacf(df.Appliances, ax=plt.gca(), lags=lags)
fig.tight_layout(pad=3)
plt.savefig(image_folder+'1c-ACF-PACF-Original.png', dpi=1000)
plt.show()

stem_acf('Original-Appliances', acf_df(df.Appliances, 90), 19735)
####################
# 1d. Correlation Matrix + Heatmap
corr = df.corr()
ax = plt.axes()
ax = sns.heatmap(data=corr)
ax.set_title("Heatmap of Pearson's Correlation Matrix")
plt.savefig(image_folder+'1d-heatmap-corr.png', dpi=1000)
plt.show()
##################
# 1e. Train/Test (80/20)
index_80 = int(len(df)*0.8)
index_20 = int(len(df)-index_80)
df_train = df[:index_80]
df_test = df[index_80:]

##########################
# Ok, sticking with original dependent variable
adf_original, kpss_original = adf_kpss_statistic(df.Appliances)
print(f" The ADF p-value for the original data is: {adf_original}")
print(f" The KPSS p-value for the original data is: {kpss_original}")
original_mean_var = cal_rolling_mean_var(df.Appliances)
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Original Data')
ax1.plot(original_mean_var.index, original_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(original_mean_var.index, original_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
```

```
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_original.png', dpi=1000)
plt.show()

# ############ Difference Dataframes
log_df = log_transform(df.Appliances,df.index)
diff_df = pd.DataFrame()
diff_df['Original'] = df.Appliances
diff_df_list = [diff(df.Appliances,x,df.index) for x in range(1,301)]
diff_combined_df = pd.concat(diff_df_list,axis=1)
diff_combined_df.to_csv('diff_combined_df.csv')
##
log_diff_list = [diff(log_df['log_transform'],x,log_df.index) for x in
range(1,301)]
log_diff_df = pd.concat(log_diff_list,axis=1)
log_diff_df.to_csv('log_diff_df.csv')
log_diff_df = pd.read_csv('log_diff_df.csv')
diff_combined_df= pd.read_csv('diff_combined_df.csv')
## Log Transform Dataframes

## Cannot take log after difference because of negatives
log_diff_list = [log_transform(diff_combined_df[f"
{x}_diff"],diff_combined_df.index) for x in range(1,301)]
diff_log_df = pd.concat(log_diff_list,axis=1)
# Testing log transform on original Appliances
original_transform = adf_kpss_statistic(log_df.log_transform)
print(original_transform)
#################
# Testing for Stationarity
adf_list = []
kpss_list = []
for x in range(1,301):
    output = adf_kpss_statistic(diff_combined_df[f"{x}_diff"])
    adf_list.append(output[0])
    kpss_list.append(output[1])
diff_stats = pd.DataFrame(index=range(1,301))
diff_stats['ADF'] = adf_list
diff_stats['KPSS'] = kpss_list
diff_stats.to_csv('diff_stats.csv')
## ADF Test statistic high == low p value == alternative hypothesis ==
stationarity
## KPSS Test statistic high == low p value == alternative hypothesis ==
non-stationarity
# ADF
plt.figure(figsize=(12, 8), layout='constrained')
plt.plot(diff_stats.index,diff_stats['ADF'])
plt.title('ADF Statistics vs Diff Intervals')
plt.xlabel('Diff Intervals')
plt.ylabel('ADF Statistics')
plt.grid()
plt.savefig(image_folder+'1a-ADF-Stats.png',dpi=1000)
plt.show()
## KPSS
plt.figure(figsize=(12, 8), layout='constrained')
```

```
plt.plot(diff_stats.index,diff_stats['KPSS'])
plt.title('KPSS Statistics vs Diff Intervals')
plt.xlabel('Diff Intervals')
plt.ylabel('KPSS Statistics')
plt.grid()
plt.savefig(image_folder+'1a-KPSS-Stats.png',dpi=1000)
plt.show()

#########
# Rolling mean/var
log_mean_var = cal_rolling_mean_var(log_df.log_transform)
diff_10_mean_var = cal_rolling_mean_var(diff_combined_df['10_diff'])
diff_25_mean_var = cal_rolling_mean_var(diff_combined_df['25_diff'])
diff_50_mean_var = cal_rolling_mean_var(diff_combined_df['50_diff'])
diff_150_mean_var = cal_rolling_mean_var(diff_combined_df['150_diff'])
diff_300_mean_var = cal_rolling_mean_var(diff_combined_df['300_diff'])
log_diff_1_mean_var = cal_rolling_mean_var(log_diff_df['1_diff'])
log_diff_2_mean_var = cal_rolling_mean_var(log_diff_df['2_diff'])
log_diff_3_mean_var = cal_rolling_mean_var(log_diff_df['3_diff'])
log_diff_4_mean_var = cal_rolling_mean_var(log_diff_df['4_diff'])
log_diff_5_mean_var = cal_rolling_mean_var(log_diff_df['5_diff'])
log_diff_10_mean_var = cal_rolling_mean_var(log_diff_df['10_diff'])
log_diff_25_mean_var = cal_rolling_mean_var(log_diff_df['25_diff'])
log_diff_50_mean_var = cal_rolling_mean_var(log_diff_df['50_diff'])
log_diff_150_mean_var = cal_rolling_mean_var(log_diff_df['150_diff'])
log_diff_300_mean_var = cal_rolling_mean_var(log_diff_df['300_diff'])
## Plotting
# Log rolling
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Log Data')
ax1.plot(log_mean_var.index, log_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_mean_var.index, log_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log.png', dpi=1000)
plt.show()
# 10 diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('10 Diff')
ax1.plot(diff_10_mean_var.index, diff_10_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(diff_10_mean_var.index, diff_10_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_diff_10.png', dpi=1000)
plt.show()
# 25 diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('25 Diff')
ax1.plot(diff_25_mean_var.index, diff_25_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(diff_25_mean_var.index, diff_25_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
```

```
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_diff_25.png', dpi=1000)
plt.show()
# 50 dif
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('50 Diff ')
ax1.plot(diff_50_mean_var.index, diff_50_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(diff_50_mean_var.index, diff_50_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_diff_50.png', dpi=1000)
plt.show()
#150 dif
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('150 Diff ')
ax1.plot(diff_150_mean_var.index, diff_150_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(diff_150_mean_var.index, diff_150_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_diff_150.png', dpi=1000)
plt.show()
#300 dif
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('300 Diff ')
ax1.plot(diff_300_mean_var.index, diff_300_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(diff_300_mean_var.index, diff_300_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_diff_300.png', dpi=1000)
plt.show()
# 1 log diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('1 Log Diff ')
ax1.plot(log_diff_1_mean_var.index, log_diff_1_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_1_mean_var.index, log_diff_1_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_1.png', dpi=1000)
plt.show()
# 2 Log Diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('2 Log Diff ')
ax1.plot(log_diff_2_mean_var.index, log_diff_2_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_2_mean_var.index, log_diff_2_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_2.png', dpi=1000)
```

```
plt.show()
# 3 Log Diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('3 Log Diff ')
ax1.plot(log_diff_3_mean_var.index, log_diff_3_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_3_mean_var.index, log_diff_3_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_3.png', dpi=1000)
plt.show()
# 4 Log Diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('4 Log Diff ')
ax1.plot(log_diff_4_mean_var.index, log_diff_4_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_4_mean_var.index, log_diff_4_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_4.png', dpi=1000)
plt.show()
# 5 Log Diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('5 Log Diff ')
ax1.plot(log_diff_5_mean_var.index, log_diff_5_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_5_mean_var.index, log_diff_5_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_5.png', dpi=1000)
plt.show()
# 10 Log Diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('10 Log Diff ')
ax1.plot(log_diff_10_mean_var.index, log_diff_10_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_10_mean_var.index, log_diff_10_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_10.png', dpi=1000)
plt.show()
# 25 log diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('25 Log Diff ')
ax1.plot(log_diff_25_mean_var.index, log_diff_25_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_25_mean_var.index, log_diff_25_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
```

```
plt.savefig(image_folder+'rolling_log_diff_25.png', dpi=1000)
plt.show()
# 50 log diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('50 Log Diff ')
ax1.plot(log_diff_50_mean_var.index, log_diff_50_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_50_mean_var.index, log_diff_50_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_50.png', dpi=1000)
plt.show()
#150 log diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('150 Log Diff ')
ax1.plot(log_diff_150_mean_var.index, log_diff_150_mean_var['Rolling
Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_150_mean_var.index, log_diff_150_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_150.png', dpi=1000)
plt.show()
#300 log diff
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('300 Log Diff ')
ax1.plot(log_diff_300_mean_var.index, log_diff_300_mean_var['Rolling
Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(log_diff_300_mean_var.index, log_diff_300_mean_var['Rolling
Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling_log_diff_300.png', dpi=1000)
plt.show()
# %%
##############################
# Using differenced data in y from now on
#diff_df = diff(df.Appliances, 144, df.index).copy()
#df['Appliances'] = diff_df['144_diff'].copy()

# Using differenced data exclusively now
index_80 = int(len(df)*0.8)
index_20 = int(len(df)-index_80)
df_train = df[:index_80]
df_test = df[index_80:]
log_diff_df = pd.read_csv('log_diff_df.csv')
diff_combined_df = pd.read_csv('diff_combined_df.csv')


# Time Series Decomposition
acf = sm.tsa.stattools.acf(diff_combined_df['144_diff'], nlags=lags)
```

```
pacf = sm.tsa.stattools.pacf(diff_combined_df['144_diff'], nlags=lags)
fig = plt.figure()
plt.subplot(211)
plt.title('ACF/PACF of the raw data')
plot_acf(diff_combined_df['144_diff'], ax=plt.gca(), lags=lags)
plt.subplot(212)
plot_pacf(diff_combined_df['144_diff'], ax=plt.gca(), lags=lags)
fig.tight_layout(pad=3)
plt.savefig(image_folder+'Decomposition-144-ACF-PACF-Original.png',
dpi=1000)
plt.show()
stem_acf('Appliances-144-diff-ACF',
         acf_df(diff_combined_df['144_diff'], 90), 19735)
###################
res = STL(df.Appliances, period=10).fit()
fig = res.plot()
plt.title('Original')
plt.ylabel('Residual')
plt.xlabel('Iterations')
plt.tight_layout()
plt.savefig(image_folder+'Original-Decomposition.png', dpi=1000)
plt.show()

res = STL(diff_combined_df['144_diff'], period=10).fit()
fig = res.plot()
plt.title('150 Diff')
plt.ylabel('Residual')
plt.xlabel('Iterations')
plt.tight_layout()
plt.savefig(image_folder+'144-Diff-Decomposition.png', dpi=1000)
plt.show()

origin_stl = STL(df.Appliances, period=10)
res = origin_stl.fit()


origin_SOT = strength_of_trend(res.resid, res.trend)
origin_season = strength_of_seasonal(res.resid, res.seasonal)

diff_stl = STL(diff_combined_df['144_diff'], period=10)
res = diff_stl.fit()


diff_SOT = strength_of_trend(res.resid, res.trend)
diff_season = strength_of_seasonal(res.resid, res.seasonal)

plt.figure()
plt.plot(df.index, res.trend, label='Trend')
plt.plot(df.index, res.resid, label='Residual')
plt.plot(df.index, res.seasonal, label='Seasonal')
plt.title('Trend, Residual, and Seasonal Plot')
plt.xticks(df.index[::4500], fontsize=10)
plt.ylabel('Electricity (Wh)')
plt.xlabel('Time')
```

```
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Cleaner-144-Decomposition.png', dpi=1000)
plt.show()
#
adjusted_seasonal = np.subtract(
    np.array(df.Appliances), np.array(res.seasonal))
detrended = np.subtract(np.array(df.Appliances), np.array(res.trend))
residual = np.array(res.resid)
adjust_seas = np.array(adjusted_seasonal)

plt.figure()
plt.plot(df.index, df.Appliances, label='Original Data', color='black')
plt.plot(df.index, adjusted_seasonal,
         label='Adjusted Seasonal', color='yellow')
plt.xticks(df.index[::4500], fontsize=10)
plt.title('Seasonaly Adjusted Data vs. Differenced')
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Seasonal-Adjusted-Decomposition.png', dpi=1000)
plt.show()

plt.figure()
plt.plot(df.index, df.Appliances, label='Original Data')
plt.plot(df.index, detrended, label='Detrended')
plt.xticks(df.index[::4500], fontsize=10)
plt.title('Detrended vs. Original')
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Detrended-Decomposition.png', dpi=1000)
plt.show()
#

# ####Holt-Winters
model_hw = ets.ExponentialSmoothing(
    df_undiff.Appliances, seasonal_periods=144, trend=None,
seasonal='add').fit()

# prediction on train set
hw_train = model_hw.forecast(steps=df_train.shape[0])
hw_train_mean_var = cal_rolling_mean_var(hw_train)
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Rolling mean/var of H-W prediction Data')
ax1.plot(hw_train_mean_var.index, hw_train_mean_var['Rolling Mean'])
ax1.set_ylabel('Rolling Mean')
ax2.plot(hw_train_mean_var.index, hw_train_mean_var['Rolling Variance'])
ax2.set_xlabel('Date')
ax2.set_ylabel('Rolling Variance')
plt.savefig(image_folder+'rolling-HW-prediction.png', dpi=1000)
plt.show()
```

```
#
train_hw = pd.DataFrame(
    hw_train, columns=['Appliances']).set_index(df_train.index)

hw_test = model_hw.forecast(steps=df_test.shape[0])
test_hw = pd.DataFrame(
    hw_test, columns=['Appliances']).set_index(df_test.index)


# model assessment
hw_train_error = np.array(df_train['Appliances'] - train_hw['Appliances'])
models.append('Holt-Winters')
model_mse.append(mse(hw_train_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(hw_train_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(hw_train_error))
model_notes.append('Flat prediction')
# print(sm.stats.acorr_ljungbox(hw_train_error,
#       lags=[5], boxpierce=True, return_df=True))

# print('HW train error mean is', np.mean(hw_train_error))
# print('the variance of the Holt-winter model prediction error is',
#       np.var(hw_train_error))

# # test data
hw_test_error = np.array(df_test['Appliances'] - test_hw['Appliances'])
# print(f"Test MSE is: {mse(hw_test_error)}")
# print(sm.stats.acorr_ljungbox(hw_test_error,
#       lags=[5], boxpierce=True, return_df=True))
# print('HW test error mean is', np.mean(hw_test_error))
# print('the variance of the Holt-winter model error is',
np.var(hw_test_error))


# plot Holt-Winter model

# plot of full model

plt.figure()
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Holt-Winter Method on Data')
plt.plot(df_train.index, df_train['Appliances'],
         label="Train Data", color='green')
plt.plot(df_test.index, df_test['Appliances'], label="Test Data",
color='blue')
plt.plot(test_hw.index, test_hw.Appliances,
         label='Forecasting Data', color='yellow')
plt.xticks(df.index[::4500], fontsize=10)
plt.legend()
plt.savefig(image_folder+'HW-Train-Test-Predict.png', dpi=1000)
plt.show()
```

```
# plot of test data
plt.figure()
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='green')
plt.plot(test_hw.index, test_hw.Appliances,
         label='Forecasting Data', color='blue')
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title(
    f'Holt-Winter Method on Data with MSE =
{mse(hw_test_error).round(4)}')
plt.xticks(test_hw.index[::725], fontsize=10)
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'HW-Test-Predict.png', dpi=1000)
plt.show()

# holt-winter train data
acf = sm.tsa.stattools.acf(hw_train, nlags=lags)
pacf = sm.tsa.stattools.pacf(hw_train, nlags=lags)
fig = plt.figure()
plt.subplot(211)
plt.title('ACF/PACF of the H-W Train data')
plot_acf(hw_train, ax=plt.gca(), lags=lags)
plt.subplot(212)
plot_pacf(hw_train, ax=plt.gca(), lags=lags)
fig.tight_layout(pad=3)
plt.savefig(image_folder+'HW-Train-PACF.png', dpi=1000)
plt.show()

stem_acf('H-W-Train', acf_df(hw_train, 90), 19735)

# holt winter test data
acf = sm.tsa.stattools.acf(test_hw.Appliances, nlags=lags)
pacf = sm.tsa.stattools.pacf(test_hw.Appliances, nlags=lags)
fig = plt.figure()
plt.subplot(211)
plt.title('ACF/PACF of the HW Test data')
plot_acf(test_hw.Appliances, ax=plt.gca(), lags=lags)
plt.subplot(212)
plot_pacf(test_hw.Appliances, ax=plt.gca(), lags=lags)
fig.tight_layout(pad=3)
plt.savefig(image_folder+'HW-Test_PACF.png', dpi=1000)
plt.show()

########
# Base Models
# Starting with average
avg_predict = h_step_average_method(
    df_train['Appliances'], df_test['Appliances'])

plt.figure()
plt.plot(df_train.index, df_train.Appliances,
         label="Train Data", color='green')
```

```
plt.xticks(df.index[::4500], rotation=90, fontsize=10)
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='blue')
plt.plot(df_test.index, avg_predict, label='Average Method', color='pink')
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Average Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Average-train-test-predict.png', dpi=1000)
plt.show()


plt.figure()
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, avg_predict, label='Average Method',
color='yellow')
plt.xticks(df_test.index[::725], fontsize=10)
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Average Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Average-test-predict.png', dpi=1000)
plt.show()


predict, forcast = average_prediction(df_train['Appliances'], len(df))

# train
one_step_predict = np.array(predict)
yarray = np.array(df_train.Appliances[1:])
avg_yt_error = np.subtract(one_step_predict[2:], yarray)
print("Mean square error for train:", mse(avg_yt_error).round(4))

avg_yf_error = np.array(df_test.Appliances) - np.array(avg_predict)
print("Mean square error for test:", mse(avg_yf_error).round(4))

# Average method statistics
print('variance of the error:', np.var(avg_yt_error))
print('the RMSE of the Average forecasting model error is, ',
mean_squared_error(
    df_test['Appliances'], np.array(avg_predict), squared=False))
print('the mean of the Average forecasting  model error is',
np.mean(avg_yf_error))
print(sm.stats.acorr_ljungbox(avg_yf_error,
      lags=[5], boxpierce=True, return_df=True))


stem_acf('Average-Error-ACF', acf_df(avg_yf_error, 90), len(df_train))

models.append('Average-Method')
model_mse.append(mse(avg_yt_error).round(4))
```

```
model_ljb.append(sm.stats.acorr_ljungbox(avg_yt_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(avg_yt_error))
model_notes.append('Slightly better')
# naive method
naive_predict = h_step_naive_method(df_train.Appliances,
df_test.Appliances)


plt.figure()
plt.plot(df_train.index, df_train.Appliances, label="Train Data",
color='blue')
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, naive_predict, label='Naive Method',
color='yellow')
plt.xticks(df.index[::4500], rotation=90, fontsize=10)
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Naive Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Naive-train-test-predict.png', dpi=1000)
plt.show()

plt.figure()
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, naive_predict, label='Naive Method',
color='yellow')
plt.xticks(df_test.index[::725], fontsize=10)
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Naive Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Naive-test-predict.png', dpi=1000)
plt.show()


# train
naive_error = np.array(
    df_train.Appliances[1:]) -
np.array(one_step_naive_method(df_train.Appliances))
print("Mean square error for the Naive method training set is ",
mse(naive_error))
# forecast
N_yf_error = np.array(df_test.Appliances) - np.array(naive_predict)
print("Mean square error for the Naive method testing set is ",
mse(naive_error))

# Q6d: Naive method statistics
print('the Variance of the error of the Naive method training set is ',
      np.var(naive_error))
```

```
print('the Variance of the error of the Naive method testing set is ',
      np.var(naive_error))
print('the RMSE of the Naive model forecasting error is, ',
mean_squared_error(
    df_test['Appliances'], np.array(naive_predict), squared=False))
print('the mean of the Naive model forecasting error is',
np.mean(naive_error))
print(sm.stats.acorr_ljungbox(N_yf_error, lags=[
      5], boxpierce=True, return_df=True))
print('The Q value was found to be 5014.178759  with a p-value of 0.0')
print('the variance for the prediction error appeared less than the
variance of the forecasting error')


stem_acf('Stem-ACF-Naive-Err', acf_df(naive_error, 90), len(df_train))

models.append('Naive-Method')
model_mse.append(mse(naive_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(naive_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(naive_error))
model_notes.append('Slightly better')
# drift method
one_step_predict = one_step_drift_method(df_train.Appliances)
h_step_predict = h_step_drift_method(df_train.Appliances,
df_test.Appliances)

plt.figure()
plt.plot(df_train.index, df_train.Appliances, label="Train Data",
color='blue')
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, one_step_predict[len(
    df_train)-len(df_test)-1:], label='Drift Method', color='yellow')
plt.xticks(df.index[::4500], rotation=90, fontsize=10)
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Drift Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'Drift-Train-Test-Predict.png', dpi=1000)
plt.show()

plt.figure()
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, one_step_predict[len(
    df_train)-len(df_test)-1:], label='Drift Method', color='yellow')
plt.xticks(df_test.index[::725], fontsize=10)
plt.xlabel('Time')
plt.ylabel('Electricity (Wh)')
plt.title('Drift Method on Electricity (Wh)')
plt.legend()
plt.tight_layout()
```

```
plt.savefig(image_folder+'Drift-Test-Predict.png', dpi=1000)
plt.show()


# train
drift_yt_error = np.subtract(np.array(df_train.Appliances[1:]), np.array(
    one_step_drift_method(df_train.Appliances)))
print("Mean square error for the drift method training set is ",
mse(drift_yt_error))
# forecast
drift_yf_error = np.subtract(np.array(df_test.Appliances)[1:], np.array(
    one_step_drift_method(df_test.Appliances)))
print("Mean square error for the drift method testing set is ",
mse(drift_yf_error))

print('the Variance of the error of the Drift method training set is ',
      np.var(drift_yt_error))
print('the Variance of the error of the Drift method testing set is ',
      np.var(drift_yf_error))
print('the RMSE of the Drift model forecasting error is, ',
mean_squared_error(
    df_test['Appliances'], np.array(one_step_predict)[len(df_train)-
len(df_test)-1:], squared=False))
print('the mean of the Drift model forecasting error is',
np.mean(drift_yf_error))
print(sm.stats.acorr_ljungbox(drift_yf_error,
      lags=[5], boxpierce=True, return_df=True))
print('The Q value was found to be 5129.25267 with a p-value of 0.0')
print('the variance for the prediction error appeared less than the
variance of the forecasting error')


stem_acf('drift-Stem-ACF-Drift-Err', acf_df(drift_yf_error, 90),
len(df_train))
models.append('Drift-Method')
model_mse.append(mse(drift_yt_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(drift_yt_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(drift_yt_error))
model_notes.append('Slightly better')
# Seasonal exponential smoothing


holtt = ets.ExponentialSmoothing(
    df_train.Appliances, trend=None, damped_trend=False,
seasonal=None).fit(smoothing_level=0.5)
holtf = holtt.forecast(steps=len(df_test))
holtf = pd.DataFrame(holtf)


# plot train, test, and SES

plt.figure()
plt.plot(df_train.index, df_train.Appliances, label="Train Data",
```

```
color='blue')
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, np.array(holtf), label='SES Method',
color='yellow')
plt.xticks(df.index[::4500], rotation=90, fontsize=10)
plt.xlabel('Date')
plt.ylabel('Appliances (Wh)')
plt.title('SES Method on Appliances(Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'SES-Train-Test-Predict.png', dpi=1000)
plt.show()

plt.figure()
plt.plot(df_test.index, df_test.Appliances, label="Test Data",
color='red')
plt.plot(df_test.index, np.array(holtf), label='SES Method',
color='yellow')
plt.xticks(df_test.index[::725], fontsize=10)
plt.xlabel('Date')
plt.ylabel('Appliances (Wh)')
plt.title('SES Method on Appliances(Wh)')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'SES-Test-Predict.png', dpi=1000)
plt.show()


# train

SES_yt_error = np.subtract(
    np.array(df_train.Appliances), SES_train(df_train.Appliances, .5))
print("Mean square error for the SES method training set is ",
mse(SES_yt_error))
# forecast
holtf = holtt.forecast(steps=len(df_test.Appliances))
holtf = pd.DataFrame(holtf)
SES_yf_error = np.subtract(np.array(df_test.Appliances), holtf[0])
print("the mean square error for the SES method testing set is ",
mse(SES_yf_error))
print('the Variance of the error of the SES method training set is ',
      np.var(SES_yt_error))
print('the Variance of the error of the SES method testing set is ',
      np.var(SES_yf_error))
print('the RMSE of the SES model forecasting error is, ',
      mean_squared_error(df_test['Appliances'], holtf[0], squared=False))
print('the mean of the SES model forecasting error is',
np.mean(SES_yf_error))
print(sm.stats.acorr_ljungbox(SES_yf_error,
      lags=[5], boxpierce=True, return_df=True))
print('The Q value was found to be 5014.178759 with a p-value of 0.0')
print('The variance of the prediction error appeared less than the
variance of the forecasting error')
```

```
stem_acf('Stem-ACF-SES-Err', acf_df(SES_yf_error, 90), len(df_train))
models.append('SES-Method')
model_mse.append(mse(SES_yt_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(SES_yt_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(SES_yt_error))
model_notes.append('Slightly better')
###############
# Doing Backwards regression and multiple linear regression together
# - Develop the multiple linear regression model that represent the
dataset. Check the accuracy of
# the developed model.
#   - You need to include the complete regression analysis into your
report. - Perform one-step ahead prediction and compare the performance
versus the test set.
#   - Hypothesis tests analysis: F-test, t-test.
#   - AIC, BIC, RMSE, R-squared and Adjusted R-squared
#   - ACF of residuals.
#   - Q-value
#   - Variance and mean of the residuals.
# Backwords Selection
features = []
x_train = df_train.drop(columns=['Appliances'])
y_train = df_train.Appliances
#x_train_ols = sm.add_constant(x_train)
OLS_model = sm.OLS(y_train, x_train)
OLS_fit = OLS_model.fit()
print(OLS_fit.summary())
OLS_coefficients = OLS_fit.params
initial_aic_bic_rsquared = aic_bic_rsquared_df(OLS_fit)


def loop_backwards(x, y, df):
    fit = sm.OLS(y, x).fit()
    remove_this_feature = worst_feature(fit.pvalues)
    print(remove_this_feature)
    features.append(remove_this_feature)
    new_x = new_x_train(remove_this_feature, x)
    new_x_df = aic_bic_rsquared_df(sm.OLS(y_train, new_x).fit())
    new_df = pd.concat([df, new_x_df])
    return new_df, new_x


newer_df, newer_x = loop_backwards(x_train, y_train,
initial_aic_bic_rsquared)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
newer_df, newer_x = loop_backwards(newer_x, y_train, newer_df)
features.append(newer_x.columns[0])
```

```
newer_df['feature_to_drop'] = features


x_trainer = sm.add_constant(x_train)
H = np.matmul(x_trainer.T, x_trainer)
print('This is H dim', H.shape)
s, d, v = np.linalg.svd(H)
print('SingularValues = ', d)
# Condition number
print(" the condition number for X is = ", LA.cond(x_trainer))
print(features)
# %%
# modeling after selection
df_after_selection = df.drop(columns=['T5','RH_5','T7','RH_7','RH_out'])
df_after_selection_train = df_after_selection[:index_80]
df_after_selection_test = df_after_selection[index_80:]
x_train = df_after_selection.drop(columns=['Appliances'])[:index_80]
x_test = df_after_selection.drop(columns=['Appliances'])[index_80:]
y_train = df_after_selection.Appliances[:index_80]
y_test = df_after_selection.Appliances[index_80:]


# One Step
OLS_model_t = sm.OLS(y_train, x_train)
OLS_results_t = OLS_model_t.fit()
y_pred = OLS_results_t.predict(x_train)
#%%

# Forecast
OLS_model_r = sm.OLS(y_train, x_train)
OLS_results_r = OLS_model_r.fit()
y_forecast = OLS_results_r.predict(x_test)
residuals_multi_model_r = np.subtract(
    np.array(df_after_selection_test.Appliances), np.array(y_forecast))
forecast = OLS_results_r.summary()

plt.figure()
plt.plot(df_after_selection_train.index,
         df_after_selection_train.Appliances, label='Training set')
plt.xticks(df.index[::4500], fontsize=10)
plt.plot(df_after_selection_train.index, y_pred, label='Prediction
values')
plt.plot(df_after_selection_test.index,
         df_after_selection_test.Appliances, label='Test set')
plt.plot(df_after_selection_test.index, y_forecast, label='Forecasted
Values')
plt.legend()
plt.tight_layout()
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.title('OLS model Prediction Plot')
plt.savefig(image_folder+'OLS-Train-Test-Predict.png', dpi=1000)
plt.show()
```

```
plt.figure()
plt.plot(df_after_selection_test.index,
         df_after_selection_test.Appliances, label='Test set')
plt.xticks(df.index[::4500], fontsize=10)
plt.plot(df_after_selection_test.index, y_forecast, label='Forecasted
Values')
plt.legend()
plt.tight_layout()
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.title('OLS model Prediction-Test Plot')
plt.savefig(image_folder+'OLS-Test-Predict.png', dpi=1000)
plt.show()


stem_acf('Stem-ACF-Multi-Var-Model-Prediction',
         acf_df(y_pred, 90), len(df_train))
stem_acf('Stem-ACF-Multi-Var-Model-Forecast',
         acf_df(y_forecast, 90), len(df_train))
stem_acf('Stem-ACF-Multi-Var-Model-Residuals',
         acf_df(residuals_multi_model_r, 90), len(df_train))

# train data
print(f"MSE : {mse(y_pred).round(4)}")
train_ljung = sm.stats.acorr_ljungbox(
    y_pred, lags=[5], boxpierce=True, return_df=True)
print(
    f'The Q value was found to be {train_ljung.iloc[:,[0]]} with a p-value
of {train_ljung.iloc[:,[1]]}')
print('mean of the regression model prediction error:', np.mean(y_pred))
print(' variance of the regression model prediction error:',
np.var(y_pred))
print(' RMSE of the regression model prediction error:, ',
mean_squared_error(
    df_after_selection_train['Appliances'], y_pred, squared=False))

# test data
print("Mean square error for the regression method forecasting on
Electricity (Wh) is ",
      mse(y_forecast).round(4))
test_ljung = sm.stats.acorr_ljungbox(
    y_forecast, lags=[5], boxpierce=True, return_df=True)
print(
    f'The Q value was found to be {test_ljung.iloc[:,[0]]} with a p-value
of {train_ljung.iloc[:,[1]]}')
print('the mean of the regression model forecasting error is',
np.mean(y_forecast))
print('the variance of the regression model forecasting error is',
np.var(y_forecast))
print('the RMSE of the regression model forecasting error is, ',
mean_squared_error(
    df_after_selection_test['Appliances'], y_forecast.values,
squared=False))
```

```
print('the variance of the prediction error appeared larger than the
variance of the testing error')
linear_regression_error = np.subtract(np.array(df_train.Appliances),
np.array(y_pred))
models.append('Linear-Regression')
model_mse.append(mse(linear_regression_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(linear_regression_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(linear_regression_error))
model_notes.append('Likely best')
# %%
# ARIMA Modeling
lags = 20
acf_train_y = sm.tsa.stattools.acf(diff_combined_df['144_diff'],
nlags=lags)


# GPAC plot

da_pac = gpac_matrix(acf_train_y, 10, 10)
plt.figure()
sns.heatmap(da_pac,  annot=True)
plt.title('GPAC Table')
plt.xlabel('k values')
plt.ylabel('j values')
plt.savefig(image_folder+'GPAC-Plot.png', dpi=1000)
plt.show()

#%%
#### ARMA (3,0)
na = 3
nb = 0

model_3_0 = sm.tsa.ARIMA(endog=df_train['Appliances'], order=(na, 0,
nb)).fit()
print(model_3_0.summary())
# MSE calculation
# train data
arma_3_0_pred = model_3_0.predict(start=0, end=15787)
arma_3_0_error = df_train.Appliances - arma_3_0_pred.values

# test data
arma_3_0_forecast = model_3_0.predict(start=15788, end=19734)
arma_3_0_residuals = df_test.Appliances - arma_3_0_forecast.values

stem_acf('Stem-ACF-3-0-Errors', acf_df(arma_3_0_error, 90), len(df_train))

stem_acf('Stem-ACF-3-0-Residuals',
         acf_df(arma_3_0_residuals, 90), len(df_train))

# train data
print("Mean square error for the ARMA(3,0) method forecasting on
Electricity (Wh) is\n ",
      mse(arma_3_0_error).round(4))
```

```
print(sm.stats.acorr_ljungbox(arma_3_0_error,
      lags=[5], boxpierce=True, return_df=True))
print(' Q value :')
print(' mean for the ARMA(3,0) model error is\n', np.mean(arma_3_0_error))
print(' variance for the ARMA(3,0) model error is\n',
np.var(arma_3_0_error))
print("covariance Matrix is\n", model_3_0.cov_params())
print("standard error coefficients are \n", model_3_0.bse)
print(' confidence intervals are\n', model_3_0.conf_int())
print(' RMSE for the ARMA(3,0) model error is\n ', mean_squared_error(
    df_train['Appliances'], arma_3_0_pred.values, squared=False))


# forecasting data
print('The MSE for the forecasting data was found to be',
mse(arma_3_0_residuals))
print(sm.stats.acorr_ljungbox(arma_3_0_residuals,
      lags=[5], boxpierce=True, return_df=True))
print('The Q value was found to be 4824.96647  with a p-value of 0.0 ')
print('the mean for the ARMA(3,0) model forecasting error is\n',
      np.mean(arma_3_0_residuals))
print('the variance for the ARMA(3,0) model forecasting error is\n',
      np.var(arma_3_0_residuals))
print('the RMSE for the ARMA(3,0) model error is\n ', mean_squared_error(
    df_test['Appliances'], arma_3_0_forecast.values, squared=False))
print('The variance of the prediction error was less than the variance of
the forecasting error')

plt.figure()
plt.title('ARMA(3,0) ')
plt.plot(df_train.index, df_train.Appliances, color='red', label='Train
Data')
plt.plot(df_test.index, df_test.Appliances, color='green', label='Test
Data')
plt.plot(df_train.index, arma_3_0_pred.values,
         color='yellow', label='Prediction Data')
plt.plot(df_test.index, arma_3_0_forecast.values,
         color='blue', label='Forecasting Data')
plt.xticks(df.index[::4500], fontsize=10)
plt.ylabel('Electricity (Wh)')
plt.xlabel('Time')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'ARMA-3-0-Train-Test-Predict.png', dpi=1000)
plt.show()


plt.figure()
plt.title('Forecasting on ARMA(3,0) for Electricity (Wh)')
plt.plot(df_test.index, df_test.Appliances, color='red', label='Test
Data')
plt.plot(df_test.index, arma_3_0_forecast.values,
         color='yellow', label='Forecasting Data')
plt.xticks(df_test.index[::750], fontsize=10)
```

```
plt.ylabel('Electricity (Wh)')
plt.xlabel('Time')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'ARMA-3-0-Test-Predict.png', dpi=1000)
plt.show()

models.append('ARMA-3-0')
model_mse.append(mse(arma_3_0_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(arma_3_0_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(arma_3_0_error))
model_notes.append('slightly worse')
###########
#%%
# ARMA (3,3)
na = 3
nb = 3

model_3_3 = sm.tsa.ARIMA(endog=df_train['Appliances'], order=(na, 0,
nb)).fit()
print(model_3_3.summary())
# MSE calculation
# train data
arma_3_3_pred = model_3_3.predict(start=0, end=15787)
arma_3_3_error = df_train.Appliances - arma_3_3_pred.values

# test data
arma_3_3_forecast = model_3_3.forecast(steps=len(df_test))
arma_3_3_residuals = df_test.Appliances - arma_3_3_forecast.values

stem_acf('Stem-ACF-3-3-Errors', acf_df(arma_3_3_error, 90), len(df_train))

stem_acf('Stem-ACF-3-3-Residuals',
         acf_df(arma_3_3_residuals, 90), len(df_train))

# train data
print("Mean square error for the ARMA(3,3) method forecasting on
Electricity (Wh) is\n ",
      mse(arma_3_3_error).round(4))
print(sm.stats.acorr_ljungbox(arma_3_3_error,
      lags=[5], boxpierce=True, return_df=True))
print(' Q value :')
print(' mean for the ARMA(3,3) model error is\n', np.mean(arma_3_3_error))
print(' variance for the ARMA(3,3) model error is\n',
np.var(arma_3_3_error))
print("covariance Matrix is\n", model_3_3.cov_params())
print("standard error coefficients are \n", model_3_3.bse)
print(' confidence intervals are\n', model_3_3.conf_int())
print(' RMSE for the ARMA(3,3) model error is\n ', mean_squared_error(
    df_train['Appliances'], arma_3_3_pred.values, squared=False))

#%%
# forecasting data
```

```
print('The MSE for the forecasting data was found to be',
mse(arma_3_3_residuals))
print(sm.stats.acorr_ljungbox(arma_3_3_residuals,
      lags=[5], boxpierce=True, return_df=True))
print('The Q value was found to be 4824.96647  with a p-value of 0.0 ')
print('the mean for the ARMA(3,0) model forecasting error is\n',
      np.mean(arma_3_3_residuals))
print('the variance for the ARMA(3,3) model forecasting error is\n',
      np.var(arma_3_3_residuals))
print('the RMSE for the ARMA(3,3) model error is\n ', mean_squared_error(
    df_test['Appliances'], arma_3_3_forecast.values, squared=False))
print('The variance of the prediction error was less than the variance of
the forecasting error')

plt.figure()
plt.title('ARMA(3,3) ')
plt.plot(df_train.index, df_train.Appliances, color='red', label='Train
Data')
plt.plot(df_test.index, df_test.Appliances, color='green', label='Test
Data')
plt.plot(df_train.index, arma_3_3_pred.values,
         color='yellow', label='Prediction Data')
plt.plot(df_test.index, arma_3_3_forecast.values,
         color='blue', label='Forecasting Data')
plt.xticks(df.index[::4500], fontsize=10)
plt.ylabel('Electricity (Wh)')
plt.xlabel('Time')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'ARMA-3-3-Train-Test-Predict.png', dpi=1000)
plt.show()


plt.figure()
plt.title('Forecasting on ARMA(3,3) for Electricity (Wh)')
plt.plot(df_test.index, df_test.Appliances, color='red', label='Test
Data')
plt.plot(df_test.index, arma_3_3_forecast.values,
         color='yellow', label='Forecasting Data')
plt.xticks(df_test.index[::750], fontsize=10)
plt.ylabel('Electricity (Wh)')
plt.xlabel('Time')
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'ARMA-3-3-Test-Predict.png', dpi=1000)
plt.show()

models.append('ARMA-3-3')
model_mse.append(mse(arma_3_3_error).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(arma_3_3_error,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(arma_3_3_error))
model_notes.append('Likely best')
#%%
```

```
######
# SARIMA
SARIMA = sm.tsa.statespace.SARIMAX(df_train.Appliances, order=(3,0,0),
seasonal_order=(0,3,0,12),
                                   enforce_stationarity=False,
                                   enforce_invertibility=False,)
result = SARIMA.fit()
print(result.summary())

# Train
sarima_prediction = result.predict(start=0, end=len(df_train),
dynamic=False)
sarima_prediction_mean = np.mean(np.array(sarima_prediction))
sarima_errors = df_train.Appliances[:-1] - sarima_prediction[:-1]

# test data

sarima_forecast = result.predict(start=0, end=
(len(df_test['Appliances'])))
# ST_pred_f = ST_pred_f.predicted_mean
# pred_f_plot= ST_pred_f
sarima_residuals = df_test.Appliances - sarima_forecast.values[1:]

# train
stem_acf('Stem-ACF-SARIMA-Errors', acf_df(sarima_errors, 90),
len(df_train))

# test
stem_acf('Stem-ACF-SARIMA-Residuals', acf_df(sarima_residuals, 90),
len(df_train))



# training statistics
print("Mean square error for the  ARIMA(3,0,0)x(0,0,0,12) method
forecasting on Electricity (Wh) is\n ", mse(sarima_errors).round(4))
print(sm.stats.acorr_ljungbox(sarima_errors, lags=[5], boxpierce=True,
return_df=True))
print('The Q value was found to be 78.976724  with a p-value of 1.373662e-
15')
print('the mean for the  ARIMA(3,0,0)x(0,3,0,12) model error is\n',
np.mean(sarima_errors))
print('the variance for the  ARIMA(3,0,0)x(0,3,0,12) model error is\n',
np.var(sarima_errors))
print("the covariance Matrix for the data is\n", result.cov_params())
print("the Standard Error for the coefficients are is\n", result.bse)
print('The confidence intervals are\n', result.conf_int())
print('the RMSE for the ARIMA(3,0,0)x(0,3,0,12) model error is\n ',
mean_squared_error(df_train.Appliances, sarima_prediction.values[1:],
squared=False))

# testing statistics
print('The MSE for the forecasting data was found to
be',mse(sarima_residuals))
```

```
print(sm.stats.acorr_ljungbox(sarima_residuals, lags=[5], boxpierce=True,
return_df=True))
print('The Q value was found to be 923.111417 with a p-value of 2.648581e-
197')
print('the mean for the ARIMA(3,0,0)x(0,3,0,12) model forecasting error
is\n', np.mean(sarima_residuals))
print('the variance for the ARIMA(3,0,0)x(0,3,0,12) model forecasting
error is\n', np.var(sarima_residuals))
print('the RMSE for the ARIMA(3,0,0)x(0,3,0,12) model error is\n ',
mean_squared_error(df_test['Appliances'], sarima_forecast[1:].values,
squared=False))
print('the variance of the prediction error appeared less than the
variance of the forecasting error')
# without forecasting
plt.figure()
plt.title('ARIMA(3,0,0)x(0,3,0,12) on Electricity (Wh)')
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.plot(df_train.index, sarima_prediction[1:], color ='yellow',
label='Prediction Data')
plt.plot(df_train.index, df_train.Appliances, color='red', label='Train
Data')
plt.plot(df_test.index, df_test.Appliances, color='green', label='Test
Data')
plt.xticks(df.index[::4500], rotation= 90, fontsize= 10)
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'SARIMA-No-Forecast-Test-Predict.png', dpi=1000)
plt.show()


# with forecasting
plt.figure()
plt.title('ARIMA(3,0,0)x(0,3,0,12) on Electricity (Wh)')
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.plot(df_train.index, sarima_prediction[1:], color ='yellow',
label='Prediction Data')
plt.plot(df_train.index, df_train.Appliances, color='red', label='Train
Data')
plt.plot(df_test.index, df_test.Appliances, color='green', label='Test
Data')
plt.plot(df_test.index, sarima_forecast[1:], color ='blue',
label='Forecasting Data')
plt.xticks(df.index[::4500], rotation= 90, fontsize= 10)
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'SARIMA-WITH-Forecast-Test-Predict.png',
dpi=1000)
plt.show()

models.append('ARIMA(3,0,0)x(0,3,0,12)')
model_mse.append(mse(sarima_errors).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(sarima_errors,
```

```
        lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(sarima_errors))
model_notes.append('Likely best')

# %%
# LSTM
# Keras Notes
# Sequence_length has no relationship with input shape
# n_features and unit much be the same
# arrays from dataframes need to be reshaped to be three dimensional
# Any inp
# ModelCheckpoint callback
# ReduceLROnPlateau callback
reduce_lr_on_plateau_cb = keras.callbacks.ReduceLROnPlateau(factor=0.1,
                                                            patience=1)
epochs = 50
sequence_length = 1
learning_rate = 10 ** -4
n_features = 25

df_lstm = df_undiff.copy()
#df_lstm = df.copy()
try:
    df_lstm.set_index('Date', inplace=True)
    del df_lstm['rv1']
    del df_lstm['rv2']
except:
    print('whatever')


timestep = [x for x in range(len(df_lstm))]
index_train = int(len(df)*0.8)
index_test = int(len(df)-index_80)
index_val = int(index_train*0.8)
length_val = index_train-index_val
df_train = df_lstm.iloc[:index_val, :]
df_val = df_lstm.iloc[index_val:index_train, :]
df_test = df_lstm.iloc[index_train:, :]

tr_x = np.array(df_train.drop(columns=['Appliances'])).reshape(
    len(df_train), n_features, 1)

tr_y = np.array(df_train[['Appliances']]).reshape(len(df_train), 1, 1)

val_x = np.array(df_val.drop(columns=['Appliances'])).reshape(
    len(df_val), n_features, 1)

val_y = np.array(df_val[['Appliances']]).reshape(len(df_val), 1, 1)

test_x = np.array(df_test.drop(columns=['Appliances'])).reshape(
    len(df_test), n_features, 1)

test_y = np.array(df_test[['Appliances']]).reshape(df_test.shape[0], 1, 1)
```

```
inputs = timeseries_dataset_from_array(
    data=tr_x, targets=None, sequence_length=sequence_length)
targets = timeseries_dataset_from_array(
    data=tr_y, targets=None, sequence_length=sequence_length)

dataset = tf.data.Dataset.zip((inputs, targets))

inputs_val = timeseries_dataset_from_array(
    data=val_x, targets=None, sequence_length=sequence_length)
targets_val = timeseries_dataset_from_array(
    data=val_y, targets=None, sequence_length=sequence_length)

inputs_test = timeseries_dataset_from_array(
    data=test_x, targets=None, sequence_length=sequence_length)
targets_test = timeseries_dataset_from_array(
    data=test_y, targets=None, sequence_length=sequence_length)


dataset = tf.data.Dataset.zip((inputs, targets))
dataset_val = tf.data.Dataset.zip((inputs_val, targets_val))

model_lstm = Sequential()
model_lstm.add(LSTM(25, activation='relu',
          return_sequences=True, input_shape=(None, 25)))
model_lstm.add(LSTM(25, activation='relu',
          return_sequences=True, input_shape=(None, 25)))
model_lstm.add(Dense(1))
model_lstm.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              metrics=['mean_squared_error'])

history = model_lstm.fit(
    dataset, validation_data=dataset_val, epochs=epochs, verbose=1,
    callbacks=[reduce_lr_on_plateau_cb])


visualize_loss(history, "Training and Validation Loss", epochs)
lstm_pred = np.array(model_lstm.predict(inputs)).flatten()
lstm_forecast = np.array(model_lstm.predict(inputs_test)).flatten()
lstm_errors = np.subtract(np.array(df_train.Appliances),lstm_pred)

stem_acf('Stem-ACF-LSTM-Errors',
         acf_df(lstm_errors, 90), len(df_train))

plt.figure()
plt.title('LSTM on Electricity (Wh)')
plt.xlabel('Date')
plt.ylabel('Electricity (Wh)')
plt.plot(df_train.index, lstm_pred, color ='yellow', label='Prediction
Data')
plt.plot(df_train.index, df_train.Appliances, color='red', label='Train
Data')
plt.plot(df_test.index, df_test.Appliances, color='green', label='Test
Data')
```

```
plt.plot(df_test.index, lstm_forecast, color ='blue', label='Forecasting
Data')
plt.xticks(df.index[::4500], rotation= 90, fontsize= 10)
plt.legend()
plt.tight_layout()
plt.savefig(image_folder+'LSTM-WITH-Forecast-Test-Predict.png', dpi=1000)
plt.show()
models.append('LSTM')
model_mse.append(mse(lstm_errors).round(4))
model_ljb.append(sm.stats.acorr_ljungbox(lstm_errors,
      lags=[5], boxpierce=True).iat[0,0])
model_error_var.append(np.var(lstm_errors))
model_notes.append('150 epochs')

# Creating model dataframe
df_models = pd.DataFrame()
df_models['models'] = models
df_models['mse'] = model_mse
df_models['ljb'] = model_ljb
#df_models['error_var'] = model_error_var
#df_models['notes'] = model_notes
df_models.to_csv('models_results.csv')
#%%
```