# Topic: Forecasting – Time Series

**Instructions**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Forecasting – Time Series**

**Guidelines:**

**1. An assignment submission is considered complete only when correct and executable code(s) are submitted along with the documentation explaining the method and results. Failing to submit either of those will be considered an invalid submission and will not be considered as correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to keys provided. (will be available only post the submission).**

**Hints:**

1. Business Problem
   1. What is the business objective?
   2. Are there any constraints?
2. Work on each feature of the dataset to create a data dictionary as displayed in the below image:

Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature

1. Data Pre-processing

3.1 Data Cleaning, Feature Engineering, etc.

3.2 Outlier Treatment

1. Exploratory Data Analysis (EDA):
   1. Summary
   2. Identify the trend
   3. Identify seasonality
2. Model Building:
   1. Perform Forecasting on the given datasets (both data-driven and moving averages)
   2. Apply techniques like exponential smoothing, model-based approach, and ARIMA
   3. Briefly explain the output in the documentation for each step (as explained in the class)
3. Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided

**Problem Statement: -**

1. Solar power consumption has been recorded by city councils at regular intervals. The reason behind doing so is to understand how businesses are using solar power so that they can cut down on nonrenewable sources of energy and shift towards renewable energy. Based on the data, build a forecasting model, and provide insights on it.

Solarpower.csv

A picture containing table

Description automatically generated



**Code:**

'''# CRISP-ML(Q)

Business Problem: 1. Solar power consumption has been recorded by city councils at regular intervals. The reason behind doing so is to understand how businesses are using solar power so that they can cut down on nonrenewable sources of energy and shift towards renewable energy. Based on the data, build a forecasting model, and provide insights on it.

Business Objective: Maximize Solar Power Consumption

Business Constraints: Minimize the production cost of renewable sources

Success Criteria:

Business: Increase the production of renewable energy at least 20%

ML: Achieve an accuracy of at least 85%

Economic: Achieve an increase in revenue by at least $200K

Data Understanding:

Feature Description Type Relevance

date Date of power consumption Quantitative Relevant

cum\_power Cumulative power consumption Quantitative Relevant

'''

import pandas as pd

import numpy as np

# import pickle

from sqlalchemy import create\_engine, text

user = 'root' # user name

pw = '1234' # password

db = 'solar\_db' # database

# creating engine to connect database

engine = create\_engine(f"mysql+pymysql://{user}:{pw}@localhost/{db}")

df = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Forecasting/Assignment/solarpower\_cumuldaybyday2.csv")

# dumping data into database

df.to\_sql('solar', con = engine, if\_exists = 'replace', chunksize = 1000, index = False)

# loading data from database

sql = 'select \* from solar'

solar = pd.read\_sql\_query(text(sql), con = engine.connect() )

print(solar)

solar.shape

# Data Pre-processing

solar["t"] = np.arange(1, 2559) # Linear Trend is captured

solar["t\_square"] = solar["t"] \* solar["t"] # Quadratic trend or polynomial with '2' degrees trend is captured

solar["log\_cum\_power"] = np.log(solar["cum\_power"]) # Exponential trend is captured

solar.columns

solar.info()

p = solar["date"][0]

# Convert 'date' column to datetime format

solar['date'] = pd.to\_datetime(solar['date'], format='%d/%m/%Y')

# Extract month from 'date' column

solar['month'] = solar['date'].dt.strftime('%b')

date\_dummies = pd.DataFrame(pd.get\_dummies(solar['month']))

solar1 = pd.concat([solar, date\_dummies], axis = 1)

solar1 = solar1.drop(columns = "month")

# Visualization - Time plot

solar1.cum\_power.plot()

# Data Partition

Train = solar1.head(2193)

Test = solar1.tail(365)

# to change the index value in pandas data frame

# Test.set\_index(np.arange(1, 13))

####################### Linear ##########################

import statsmodels.formula.api as smf

linear\_model = smf.ols('cum\_power ~ t', data = Train).fit()

linear\_model.summary()

pred\_linear = pd.Series(linear\_model.predict(pd.DataFrame(Test['t'])))

rmse\_linear = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(pred\_linear))\*\*2))

rmse\_linear

##################### Exponential ##############################

Exp = smf.ols('log\_cum\_power ~ t', data = Train).fit()

pred\_Exp = pd.Series(Exp.predict(pd.DataFrame(Test['t'])))

rmse\_Exp = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(np.exp(pred\_Exp)))\*\*2))

rmse\_Exp

#################### Quadratic ###############################

Quad = smf.ols('cum\_power ~ t + t\_square', data = Train).fit()

pred\_Quad = pd.Series(Quad.predict(Test[["t", "t\_square"]]))

rmse\_Quad = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(pred\_Quad))\*\*2))

rmse\_Quad

################### Additive Seasonality ########################

add\_sea = smf.ols('cum\_power ~ Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov', data = Train).fit()

add\_sea.summary()

pred\_add\_sea = pd.Series(add\_sea.predict(Test[['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']]))

rmse\_add\_sea = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(pred\_add\_sea))\*\*2))

rmse\_add\_sea

################## Multiplicative Seasonality ##################

Mul\_sea = smf.ols('log\_cum\_power ~ Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov', data = Train).fit()

pred\_Mult\_sea = pd.Series(Mul\_sea.predict(Test[['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']]))

rmse\_Mult\_sea = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(np.exp(pred\_Mult\_sea)))\*\*2))

rmse\_Mult\_sea

################## Additive Seasonality Quadratic Trend ############################

add\_sea\_Quad = smf.ols('cum\_power ~ t + t\_square + Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov', data = Train).fit()

pred\_add\_sea\_quad = pd.Series(add\_sea\_Quad.predict(Test[['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 't', 't\_square']]))

rmse\_add\_sea\_quad = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(pred\_add\_sea\_quad))\*\*2))

rmse\_add\_sea\_quad

################## Multiplicative Seasonality Linear Trend ###########

Mul\_sea\_linear = smf.ols('log\_cum\_power ~ t + Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov', data = Train).fit()

pred\_Mult\_sea\_linear = pd.Series(Mul\_sea\_linear.predict(Test[['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 't']]))

rmse\_Mult\_sea\_linear = np.sqrt(np.mean((np.array(Test['cum\_power']) - np.array(np.exp(pred\_Mult\_sea\_linear)))\*\*2))

rmse\_Mult\_sea\_linear

################## Testing #######################################

data = {"MODEL":pd.Series(["rmse\_linear", "rmse\_Exp", "rmse\_Quad", "rmse\_add\_sea", "rmse\_Mult\_sea", "rmse\_add\_sea\_quad", "rmse\_Mult\_sea\_linear"]), "RMSE\_Values":pd.Series([rmse\_linear, rmse\_Exp, rmse\_Quad, rmse\_add\_sea, rmse\_Mult\_sea, rmse\_add\_sea\_quad, rmse\_Mult\_sea\_linear])}

table\_rmse = pd.DataFrame(data)

table\_rmse

# 'rmse\_add\_sea\_quad' has the least RMSE value among the models prepared so far. Use these features and build forecasting model using entire data

model\_full = smf.ols('cum\_power ~ t + t\_square + Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov', data = solar1).fit()

predict\_data = pd.read\_excel(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Forecasting/Assignment/Predict\_new.xlsx")

pred\_new = pd.Series(model\_full.predict(predict\_data))

pred\_new

predict\_data["forecasted\_cum\_power"] = pd.Series(pred\_new)

# The models and results have save and load method, so you don't need to use the pickle module directly.

# to save model

model\_full.save("Reg\_model.pickle")

import os

os.getcwd()

# to load model

from statsmodels.regression.linear\_model import OLSResults

model = OLSResults.load("Reg\_model.pickle")

# RESIDUALS MIGHT HAVE ADDITIONAL INFORMATION!

# Autoregression Model (AR)

# Calculating Residuals from best model applied on full data

# AV - FV

full\_res = solar1.cum\_power - model.predict(solar1)

# ACF plot on residuals

import statsmodels.graphics.tsaplots as tsa\_plots

tsa\_plots.plot\_acf(full\_res, lags = 12)

# ACF is an (complete) auto-correlation function gives values

# of auto-correlation of any time series with its lagged values.

# PACF is a partial auto-correlation function.

# It finds correlations of Y with lags of the residuals of the time series

tsa\_plots.plot\_pacf(full\_res, lags = 12)

# Alternative approach for ACF plot is explained in next 2 lines

# from pandas.plotting import autocorrelation\_plot

# autocorrelation\_ppyplot.show()

# AR Autoregressive model

from statsmodels.tsa.ar\_model import AutoReg

model\_ar = AutoReg(full\_res, lags = [1])

model\_fit = model\_ar.fit()

print('Coefficients: %s' % model\_fit.params)

pred\_res = model\_fit.predict(start = len(full\_res), end = len(full\_res) + len(predict\_data) - 1, dynamic = False)

pred\_res.reset\_index(drop = True, inplace = True)

# The Final Predictions using ASQT and AR(1) Model

final\_pred = pred\_new + pred\_res

final\_pred

**Output:**

solar.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2558 entries, 0 to 2557

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 date 2558 non-null datetime64[ns]

1 cum\_power 2558 non-null float64

2 t 2558 non-null int32

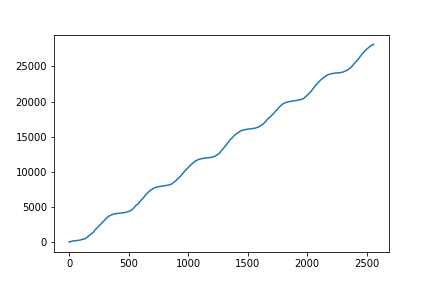
3 t\_square 2558 non-null int32

4 log\_cum\_power 2558 non-null float64

5 month 2558 non-null int32

dtypes: datetime64[ns](1), float64(2), int32(3)

memory usage: 90.1 KB



Out[144]:

MODEL RMSE\_Values

0 rmse\_linear 441.212895

1 rmse\_Exp 37012.422190

2 rmse\_Quad 465.492517

3 rmse\_add\_sea 14054.389247

4 rmse\_Mult\_sea 17562.360601

5 rmse\_add\_sea\_quad 145.495669

6 rmse\_Mult\_sea\_linear 38362.256774

pred\_new = pd.Series(model\_full.predict(predict\_data))

pred\_new

Out[148]:

0 28077.040032

1 28088.225824

2 28099.411791

3 28110.597935

4 28121.784254

5 27935.892027

6 27947.078698

7 27958.265545

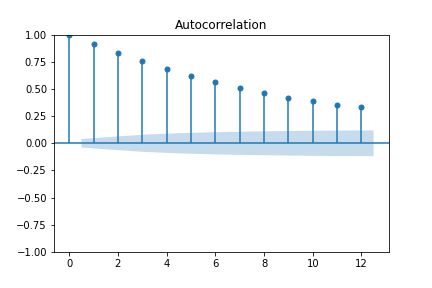
8 27969.452568

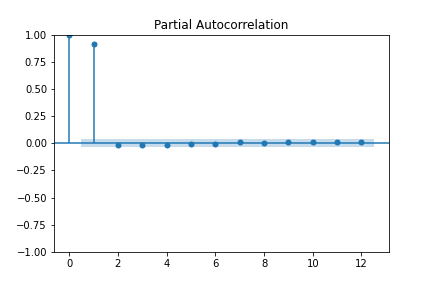
9 27980.639766

10 27991.827140

11 28003.014690

dtype: float64





print('Coefficients: %s' % model\_fit.params)

Coefficients: const 0.034716

y.L1 0.915314

dtype: float64

pred\_res = model\_fit.predict(start = len(full\_res), end = len(full\_res) + len(predict\_data) - 1, dynamic = False)

pred\_res.reset\_index(drop = True, inplace = True)

# The Final Predictions using ASQT and AR(1) Model

final\_pred = pred\_new + pred\_res

final\_pred

Out[167]:

0 28126.634950

1 28133.655455

2 28141.028877

3 28148.725345

4 28156.717516

5 27967.901642

6 27976.412257

7 27985.149673

8 27994.094698

9 28003.229765

10 28012.538795

11 28022.007070

dtype: float64