```python
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 11 14:49:28 2021

@author: Willem van der Schans

Related Github Directory:
    https://github.com/Kydoimos97/CapstoneMSBA2020
"""

#!/usr/bin/env python
# coding: utf-8


# In[1]:

# Math
from math import ceil, floor, sqrt

# Plotting
import matplotlib

matplotlib.use("Agg")
import matplotlib.pyplot as plt

# Numpy & Pandas
import numpy as np
import pandas as pd

# Stats models
import statsmodels.api as sm
from matplotlib.pyplot import figure

# Linear Imputation
from scipy.interpolate import interp1d

# Machine Learning
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA

# Augmented Dickey Fuller Test
from statsmodels.tsa.stattools import adfuller

# Options
pd.options.display.max_rows = 2500

# Low Verbose Warning Cooercion
import warnings
warnings.filterwarnings("ignore",
                        "statsmodels.tsa.arima_model.ARMA", FutureWarning)
warnings.filterwarnings("ignore",
                        "statsmodels.tsa.arima_model.ARIMA", FutureWarning)
warnings.filterwarnings("ignore")

# Set working Directory In code
import os

# MultiThreading
import subprocess as sub

# Dependency Import
import sys

# Time Tracking
import time

```

```
# First GUI element Imports
import tkinter as tk
import tkinter.font as tkFont
import tkinter.messagebox

# Download Files from the Web
import urllib

#Multi Threading
from threading import Thread

# Second GUI element Imports
from tkinter import *
from tkinter import filedialog, ttk
from tkinter.ttk import Separator, Style


# In[2]:
# Create all supporting Functions to be called in the GUI

def check_standard_dev():
    value = __std_dev_inp.get()
    if value.isdigit():
        __std_dev_inp_text_box.delete(1.0, "end-1c")
        __std_dev_inp_text_box.insert("end-1c", u"\u2714")
    else:
        __std_dev_inp_text_box.delete(1.0, "end-1c")
        __std_dev_inp_text_box.insert("end-1c", u"\u2717")


def check_item_id():
    value = __item_id_inp.get()
    value_str = str(value)
    if ("," in str(value)) and (value_str.startswith("-")):
        __item_id_inp_text_box.delete(1.0, "end-1c")
        __item_id_inp_text_box.insert("end-1c", "Item_ID List \u2714")
    elif value.isdigit():
        __item_id_inp_text_box.delete(1.0, "end-1c")
        __item_id_inp_text_box.insert("end-1c", "Top_N \u2714")
    elif value_str.startswith("-"):
        __item_id_inp_text_box.delete(1.0, "end-1c")
        __item_id_inp_text_box.insert("end-1c", "Item_ID \u2714")
    else:
        __item_id_inp_text_box.delete(1.0, "end-1c")
        __item_id_inp_text_box.insert("end-1c", u"\u2717")


def check_site_id():
    value = __site_id_inp.get()
    if value == "":
        __site_id_text_box.delete(1.0, "end-1c")
        __site_id_text_box.insert("end-1c", "All_Sites \u2714")
    elif value.isdigit():
        if len(value) == 3:
            __site_id_text_box.delete(1.0, "end-1c")
            __site_id_text_box.insert("end-1c", "Site_ID \u2714")
        else:
            pass
    elif "," in str(value):
        __site_id_text_box.delete(1.0, "end-1c")
        __site_id_text_box.insert("end-1c", "Site_ID List \u2714")
    else:
        __site_id_text_box.delete(1.0, "end-1c")
        __site_id_text_box.insert("end-1c", u"\u2717")


```

```
def check_time():
    value = __time_inp.get()
    if value.isdigit():
        __time_inp_text_box.delete(1.0, "end-1c")
        __time_inp_text_box.insert("end-1c", u"\u2714")
    else:
        __time_inp_text_box.delete(1.0, "end-1c")
        __time_inp_text_box.insert("end-1c", u"\u2717")


def browse_button():
    # Allow user to select a directory and store it in global var
    # called folder_path
    global __df_folder_path
    __filename = filedialog.askopenfilename()
    __df_folder_path.set(str(__filename))
    __browse_button_text.set("/".join(str(__filename).split("/", -1)[-3:]))


def browse_button2():
    # Allow user to select a directory and store it in global var
    # called folder_path
    global __df_folder_path2
    __filename2 = filedialog.askopenfilename()
    __df_folder_path2.set(str(__filename2))
    __browse_button_text2.set("/".join(str(__filename2).split("/", -1)[-3:]))


def browse_button3():
    # Allow user to select a directory and store it in global var
    # called folder_path
    global __df_folder_path2
    __filename3 = filedialog.askdirectory()
    __df_folder_path3.set(str(__filename3))
    __browse_button_text3.set("/".join(str(__filename3).split("/", -1)[-3:]))


def duplicate_remover(x):
    return list(dict.fromkeys(x))

# Main Starting Button
def button_click():
    global executing  # create global
    executing = True

    # Create new thread
    t = Thread(target=getInput)
    # Start new thread
    t.start()

# Stop Button
def stop():
    global executing  # create global
    executing = False
    # Disable Button to prevent backlog of clicks
    __stop_btn["state"] = "disabled"
    __timing_text_box.delete("1.0", "end")
    __timing_text_box.insert("end-1c", " ")
    __progress_text_box.delete("1.0", "end")
    __progress_text_box.insert("end-1c", " ")


running = True  # Global flag


# In[3]:
```

```
199
200 # Main Algorithm Loop
201 def getInput():
202
203     # Change GUI
204     __stop_button_text.set("Stop Execution")
205     __sub_button_text.set("Running")
206     # Disable Stop Button until startup sequence is complete.
207     __submit_btn["state"] = "disabled"
208
209     # Try First Inputs | Working Folder, DataFrame1, Dataframe 2
210     # Except show input specific errors
211     try:
212         os.chdir(__df_folder_path3.get())
213     except:
214         __stop_button_text.set("Exit Program")
215         __sub_button_text.set("Submit and Run")
216         __submit_btn["state"] = "normal"
217         __stop_btn["state"] = "disabled"
218         tk.messagebox.showerror(title="Program Stopped",
219                                 message="No output directory")
220         return
221
222     try:
223         df = pd.read_csv(str(__df_folder_path.get()), index_col=0)
224     except:
225         __stop_button_text.set("Exit Program")
226         __sub_button_text.set("Submit and Run")
227         __submit_btn["state"] = "normal"
228         __stop_btn["state"] = "disabled"
229         tk.messagebox.showerror(
230             title="Program Stopped", message="Wrong Database .CSV file"
231         )
232         return
233
234     try:
235         productsdf = pd.read_csv(str(__df_folder_path2.get()), index_col=0)
236     except:
237         __stop_button_text.set("Exit Program")
238         __sub_button_text.set("Submit and Run")
239         __submit_btn["state"] = "normal"
240         __stop_btn["state"] = "disabled"
241         tk.messagebox.showerror(
242             title="Program Stopped", message="Wrong ProductDF .CSV file"
243         )
244         return
245
246     # Try to Clean and prep the data and variables needed
247     # except show error = wrong csv file
248     try:
249         # All column names to lowercase
250         df.columns = map(str.lower, df.columns)
251
252         # Impute item names
253         productdict = dict(zip(productsdf.Item_ID, productsdf.Item_Desc))
254         df["product_name"] = df.item_id
255         df.product_name = df.product_name.map(productdict)
256         df["date"] = pd.to_datetime(df["date"])
257
258         # Project Column
259         projectdf = df.loc[(df["project"] != "NONE")]
260         df = df.loc[(df["project"] == "NONE")]
261
262         # Temprature Imputations
263         df["mintemp"].interpolate(method="linear", inplace=True)
264         df["maxtemp"].interpolate(method="linear", inplace=True)
```

```
265
266         # Sales Outlier Removal: Depreciated | Limits functionality
267         df["price"] = df["sales"] / df["quantity_sold"]
268         #df.loc[df.product_name == "TTS TOOTSIE ROLL $.10", "price"] = 0.10
269         df.sales = df.quantity_sold * df.price
270
271         #Remove Quantity Outliers
272         ## Create Aggregated Data Frames and dictionary
273         x = (
274             df.groupby(["date", "site_id"])
275             .agg(
276                 daily_sales=pd.NamedAgg(column="sales", aggfunc=sum),
277                 daily_quantity=pd.NamedAgg(column="quantity_sold", aggfunc=sum),
278             )
279             .reset_index()
280         )
281         x["date"] = pd.to_datetime(x["date"])
282
283         x["price"] = (x["daily_sales"] / x["daily_quantity"]).round(2)
284
285         z = (
286             df.groupby(["date", "site_id"])
287             .agg(total_sales=pd.NamedAgg(column="sales", aggfunc=sum))
288             .reset_index()
289         )
290
291         z = (
292             z.groupby(["site_id"])
293             .agg(average_sales=pd.NamedAgg(column="total_sales", aggfunc="mean"))
294             .reset_index()
295         )
296
297         salesdict = dict(zip(z.site_id, z.average_sales))
298
299         ## Find Average Sales and deviations and remove everything above the treshold
300         x["average_sales"] = x.site_id
301         x.average_sales = x.site_id.map(salesdict)
302         x["Sales_Difference"] = np.absolute(
303             ((x["daily_sales"] - x["average_sales"]) / x["average_sales"]) * 100
304         ).round(2)
305         x = x.sort_values("Sales_Difference", ascending=False)
306
307         th_std = np.std(x.Sales_Difference) * int(__std_dev_inp.get()) + np.mean(
308             x.Sales_Difference
309         )
310         temp = x
311
312         temp2 = temp.loc[(temp["Sales_Difference"] >= th_std)].sort_values(
313             "Sales_Difference", ascending=False
314         )
315
316         ## Recreate Data frame based on removed outliers.
317         site_vector = list(temp2["site_id"])
318         date_vector = list(temp2["date"])
319         index_list = []
320
321         df.reset_index(inplace=True)
322
323         for i in range(len(site_vector)):
324             temp = (
325                 df.loc[
326                     (df["site_id"] == site_vector[i]) & (df["date"] == date_vector[i])
327                 ]
328                 .sort_values("sales", ascending=False)
329                 .head(1)
330             )
```

```
            temp = temp.drop(
                columns=[
                    "location_id",
                    "open_date",
                    "sq_footage",
                    "locale",
                    "maxtemp",
                    "mintemp",
                    "fiscal_period",
                    "periodic_gbv",
                    "current_gbv",
                    "mpds",
                ]
            )
            index_list.append(temp.iloc[0, 0])

        ## Reduce Dimensions and Ram Usage
        df.rename(columns={df.columns[0]: "index_set"}, inplace=True)

        df = df[~df.index_set.isin(index_list)]

        df.drop(
            [
                "index_set",
                "location_id",
                "current_gbv",
                "product_name",
                "open_date",
                "fiscal_period",
                "project",
                "price",
            ],
            axis=1,
            inplace=True,
        )

        ### Recode Locale Variable
        locale_string = list(df.locale.unique())
        locale_replace = list(range(1, len(locale_string) + 1))
        df.locale.replace(locale_string, locale_replace, inplace=True)

        df = df[
            [
                "site_id",
                "item_id",
                "date",
                "sq_footage",
                "mpds",
                "locale",
                "periodic_gbv",
                "maxtemp",
                "mintemp",
                "quantity_sold",
                "sales",
            ]
        ]

        ## Create data set for item_id top N Input
        __df_agg = (
            df.groupby(["item_id"])
            .agg(
                total_sales=pd.NamedAgg(column="sales", aggfunc=sum),
                days_sold=pd.NamedAgg(column="sales", aggfunc="count"),
            )
            .reset_index()
            .sort_values("total_sales", ascending=False)
```

```
        )

        ## Get Item and Site Inputs and create lists
        item_list = []
        site_list = []

        value = __item_id_inp.get()
        value_str = str(value)
        if ("," in str(value)) and (value_str.startswith("-")):
            value_str = value_str.replace(" ", "")
            item_list = value_str.split(",")
            for __i in range(0, len(item_list)):
                item_list[__i] = int(item_list[__i])
        elif value.isdigit():
            item_list = list(__df_agg.item_id.head(int(value)))

        elif value_str.startswith("-"):
            item_list = [int(__item_id_inp.get())]
        else:
            item_list = list(__df_agg.item_id.unique())

        value = __site_id_inp.get()
        if value == "":
            site_list = list(df.site_id.unique())
        elif value.isdigit():
            if len(value) == 3:
                site_list = [int(__site_id_inp.get())]
            else:
                pass
        elif "," in str(value):
            site_list = str(value).replace(" ", "").split(",")
            for __i in range(0, len(site_list)):
                site_list[__i] = int(site_list[__i])

        ## Create further lists and dictionaries
        date_list = list(df.sort_values("date").date.unique())
        __app_list = []
        prediction_dict = {}
        holdout_prediction_dict = {}
        test_dict = {}
        model_sum_dict = {}

        ## Create a dictionary for none changing values and fill it
        df_dict = {}

        for __i in site_list:
            df_dict[int(__i)] = []

        for __i in range(0, len(site_list)):

            __site_id_value = site_list[__i]

            __df_temp = (
                df.loc[(df["site_id"] == __site_id_value)]
                .reset_index(drop=True)
                .sort_values("date")
            )

            df_dict[__site_id_value].append(int(__df_temp.sq_footage.mode()))
            df_dict[__site_id_value].append(int(__df_temp.mpds.mode()))
            df_dict[__site_id_value].append(int(__df_temp.locale.mode()))
            df_dict[__site_id_value].append(int(__df_temp.periodic_gbv.mode()))

    except:
        __stop_button_text.set("Exit Program")
        __sub_button_text.set("Submit and Run")
```

```
            __submit_btn["state"] = "normal"
            __stop_btn["state"] = "disabled"
            tk.messagebox.showerror(
                title="Program Stopped",
                message="Supplied inputs or files are not compatible",
            )
            return

        # Start Timing
        __timeC = time.time()

        # Prep the Parameter Inputs
        __p_values = []
        __d_values = []
        __q_values = []

        # Read the Input
        value = __p_gs_list_inp.get()
        value_str = str(value)
        if "," in str(value):
            value_str = value_str.replace(" ", "")
            __p_gs_list = value_str.split(",")
            for __i in range(0, len(__p_gs_list)):
                __p_gs_list[__i] = int(__p_gs_list[__i])
        else:
            __p_gs_list = [0, 1, 2]

        value = __d_gs_list_inp.get()
        value_str = str(value)
        if "," in str(value):
            value_str = value_str.replace(" ", "")
            __d_gs_list = value_str.split(",")
            for __i in range(0, len(__d_gs_list)):
                __d_gs_list[__i] = int(__d_gs_list[__i])
        else:
            __d_gs_list = [0, 1, 2]

        value = __q_gs_list_inp.get()
        value_str = str(value)
        if "," in str(value):
            value_str = value_str.replace(" ", "")
            __q_gs_list = value_str.split(",")
            for __i in range(0, len(__q_gs_list)):
                __q_gs_list[__i] = int(__q_gs_list[__i])
        else:
            __q_gs_list = [0, 1, 2]

        # Create Grid Searching Lists
        if __gridsearch_inp.get() == "Yes":
            if int(__p_inp.get()) == 0:
                if int(__q_inp.get()) == 0:
                    for __p in __p_gs_list:
                        __p_values.append(int(int(__p_inp.get()) + int(__p)))
                    for __d in __d_gs_list:
                        __d_values.append(int(int(__d_inp.get()) + int(__d)))
                    for __q in __q_gs_list:
                        __q_values.append(int(int(__q_inp.get()) + int(__q)))
                else:
                    for __p in __p_gs_list:
                        __p_values.append(int(int(__p_inp.get()) + int(__p)))
                    for __d in __d_gs_list:
                        __d_values.append(int(int(__d_inp.get()) + int(__d)))
                    for __q in __q_gs_list:
                        __q_values.append(int(int(__q_inp.get()) * int(__q)))
            elif int(__q_inp.get()) == 0:
                for __p in __p_gs_list:
```

```
                    __p_values.append(int(int(__p_inp.get()) * int(__p)))
                for __d in __d_gs_list:
                    __d_values.append(int(int(__d_inp.get()) + int(__d)))
                for __q in __q_gs_list:
                    __q_values.append(int(int(__q_inp.get()) + int(__q)))
            else:
                for __p in __p_gs_list:
                    __p_values.append(int(int(__p_inp.get()) * int(__p)))
                for __d in __d_gs_list:
                    __d_values.append(int(int(__d_inp.get()) + int(__d)))
                for __q in __q_gs_list:
                    __q_values.append(int(int(__q_inp.get()) * int(__q)))
        else:
            __p_values = [int(__p_inp.get())]
            __q_values = [int(__q_inp.get())]
            if __gridsearch_d_inp.get() == "Yes":
                for __d in __d_gs_list:
                    __d_values.append(int(int(__d_inp.get()) + int(__d)))
            else:
                __d_values = [int(__d_inp.get())]

        # Remove Potential Duplicate values to limit runtime.
        __p_values = duplicate_remover(__p_values)
        __d_values = duplicate_remover(__d_values)
        __q_values = duplicate_remover(__q_values)

        # Create Machine Learning Variables and dictionaries
        __best_score = 1000000000
        __best_cfg = 0
        counter = 0
        __timeA = time.time()
        __timeB = time.time()
        __site_id_value = "None"

        # Create Machine Inputs and Dictionaries
        __prediction_time_frame = int(__time_inp.get())
        score_dict = {}
        error_dict = {}

        # Set Binary Indicators for Errors
        trend_error = 0
        array_error = 0
        combination_error = 0



        # Start Main Algorithm Loop
        __stop_btn["state"] = "normal"
        for __i in range(0, len(item_list)):
            __item_id_value = item_list[__i]
            # Create a point to stop the loop
            if executing == False:
                __stop_button_text.set("Exit Program")
                __sub_button_text.set("Submit and Run")
                __submit_btn["state"] = "normal"
                __stop_btn["state"] = "disabled"
                __timing_text_box.delete("1.0", "end")
                __timing_text_box.insert("end-1c", " ")
                __progress_text_box.delete("1.0", "end")
                __progress_text_box.insert("end-1c", " ")
                tk.messagebox.showwarning(
                    title="Program Stopped", message=str(
                        "Program Stopped on Item = "
                        + str(__item_id_value)
                        + " and Site = "
                        + str(__site_id_value))
```

```
            )

            return
        elif __i > 0:
            # Update timing text box
            __timeB = time.time()
            __timediff = __timeB - __timeA
            __string_timing_text = ""
            __string_timing_text = str(
                "Item "
                + str(__i)
                + "/"
                + str(len(item_list))
                + " | Runtime = "
                + str(int(__timediff))
                + " Sec | Runtime Left = "
                + str(int((__timediff / __i) * (len(item_list) - __i)))
                + " Sec"
            )
            __timing_text_box.delete("1.0", "end")
            __timing_text_box.insert("end-1c", __string_timing_text)
        else:
            __timing_text_box.delete("1.0", "end")
            __timing_text_box.insert("end-1c", "Initializing")
        for __x in range(0, len(site_list)):
            # Nested Loop site_ID under Item_ID
            __site_id_value = site_list[__x]

            __df_temp = (
                df.loc[
                    (df["site_id"] == __site_id_value)
                    & (df["item_id"] == __item_id_value)
                ]
                .reset_index(drop=True)
                .sort_values("date")
            )

            # Check if item and site combination exist
            try:
                __index_pos = date_list.index(__df_temp.date.unique()[0])
            except:
                combination_error = 1
                error_dict[
                    __item_id_value, __site_id_value
                ] = "Combination Does not Exist"
                continue

            # Create a date list of differences to impute zero sales
            __trunc_date_list = list(date_list[__index_pos : len(date_list)])

            __diff_list = list(
                set(__trunc_date_list) - set(list(__df_temp.date.unique()))
            )

            if len(__diff_list) > 0:

                for __y in range(0, len(__diff_list)):
                    __app_list = []
                    __app_list.extend(
                        (
                            __site_id_value,
                            __item_id_value,
                            __diff_list[__y],
                            df_dict[__site_id_value][0],
                            df_dict[__site_id_value][1],
                            df_dict[__site_id_value][2],
```

```
                        df_dict[__site_id_value][3],
                        np.nan,
                        np.nan,
                        0,
                        0,
                    )
                )
                __app_series = pd.Series(__app_list, index=__df_temp.columns)
                __df_temp = __df_temp.append(__app_series, ignore_index=True)

            else:
                pass

            # sort newly created df by sate
            __df_temp = __df_temp.sort_values("date")
            # impute missing temprature values with linear regression
            __df_temp["mintemp"].interpolate(method="linear", inplace=True)
            __df_temp["maxtemp"].interpolate(method="linear", inplace=True)
            # Overwrite index to date
            __df_temp["dateind"] = __df_temp["date"]
            __df_temp.set_index("dateind", inplace=True)
            # Set Frequency for Arima Model
            __df_temp.index = pd.DatetimeIndex(
                __df_temp.index.values, freq=__df_temp.index.inferred_freq
            )

            # Hold-Out Evaluation
            ## Create Test Dataframe
            __test_temp = __df_temp.tail(__prediction_time_frame)
            __df_temp_bu = __df_temp
            ## Create Train Dataframe
            __df_temp = __df_temp.drop(__df_temp.tail(__prediction_time_frame).index)

            # Item and Site Combination First Sale Check
            if int(len(__df_temp)) < int(__time_inp.get()):
                if __warning_inp.get() == "Yes":
                    message_list = (
                        "Not enough data For Item_ID = "
                        + str(__item_id_value)
                        + " & Site_ID = "
                        + str(__site_id_value)
                        + "\n"
                        + "Total Data Points = "
                        + str(len(__df_temp_bu))
                        + "\n"
                        + "Not enough historical data present for this combination"
                        + "\n"
                        + "No Prediction can be made at this time"
                    )

                    tk.messagebox.showerror(title="Warning", message=message_list)

                    array_error = 1
                    error_dict[
                        __item_id_value, __site_id_value
                    ] = "No Historical Data | Item too new for the store"
                elif __warning_inp.get() != "Yes":
                    array_error = 1
                    error_dict[
                        __item_id_value, __site_id_value
                    ] = "No Historical Data | Item too new for the store"

                continue
            else:
                pass

```

```
                # Gridsearching Loop
                __best_score = 1000000000
                __best_cfg = (0, 0, 0)
                __total_counter = 0
                for __p in __p_values:
                    for __d in __d_values:
                        for __q in __q_values:
                            __order = (__p, __d, __q)
                            # Create Break point for Loop
                            if executing == False:
                                __stop_button_text.set("Exit Program")
                                __sub_button_text.set("Submit and Run")
                                __submit_btn["state"] = "normal"
                                __stop_btn["state"] = "disabled"
                                __timing_text_box.delete("1.0", "end")
                                __timing_text_box.insert("end-1c", " ")
                                __progress_text_box.delete("1.0", "end")
                                __progress_text_box.insert("end-1c", " ")
                                tk.messagebox.showwarning(
                                    title="Program Stopped", message=str(
                                        "Program Stopped on Item = "
                                        + str(__item_id_value)
                                        + " and Site = "
                                        + str(__site_id_value))
                                )

                                return
                            try:
                                # Currently Univatiate
                                ## Exog needed for Multivariate
                                __model = ARIMA(
                                    endog=__df_temp["quantity_sold"], order=__order
                                )
                                try:
                                    __model_fit = __model.fit()
                                except:
                                    error_dict[__site_id_value, __item_id_value] = str(
                                        "LU decomposition error with "
                                        + str(__order)
                                        + " | Model failed to fit"
                                        )
                                    continue
                                __output = __model_fit.predict(
                                    start=len(__df_temp),
                                    end=int(len(__df_temp) + __prediction_time_frame - 1),
                                    dynamic=False,
                                )

                                __rmse = sqrt(
                                    mean_squared_error(__test_temp.quantity_sold, __output)
                                )

                                counter = counter + 1
                                __total_counter = (
                                    len(__p_values)
                                    * len(__d_values)
                                    * len(__q_values)
                                    * len(item_list)
                                    * len(site_list)
                                )

                                # Give progress outputs
                                __string_progress_text = str(
                                    "Progress "
                                    + str(counter)
                                    + " models evaluated out of an expected "
```

```
                            + str(__total_counter)
                        )

                        __progress_text_box.delete("1.0", "end")
                        __progress_text_box.insert("end-1c", __string_progress_text)

                        # Update config is RMSE is lower
                        if __rmse < __best_score:
                            __best_score = __rmse
                            __best_cfg = __order
                            holdout_prediction_dict[
                                __site_id_value, __item_id_value
                            ] = __output
                        else:
                            pass
                    except ValueError as VE:
                        if __warning_inp.get() == "Yes":
                            if "A constant trend was included in the model" in str(
                                VE
                            ):
                                tk.messagebox.showerror(
                                    title="ERROR",
                                    message="""Trend Included in the Model. Allow
                                    for gridsearching of Parameter [d]""",
                                )
                            else:
                                pass
                        else:
                            pass
                # if no better rmse is found or loop failed, input values
                if __best_cfg == (0, 0, 0):
                    __best_cfg = __order
                else:
                    pass

                # Create Break Point
                if executing == False:
                    __stop_button_text.set("Exit Program")
                    __sub_button_text.set("Submit and Run")
                    __submit_btn["state"] = "normal"
                    __stop_btn["state"] = "disabled"
                    __timing_text_box.delete("1.0", "end")
                    __timing_text_box.insert("end-1c", " ")
                    __progress_text_box.delete("1.0", "end")
                    __progress_text_box.insert("end-1c", " ")
                    tk.messagebox.showwarning(
                        title="Program Stopped", message=str(
                            "Program Stopped on Item = "
                            + str(__item_id_value)
                            + " and Site = "
                            + str(__site_id_value))
                    )
                    return


                # Create the Main Prediction Model
                ## Final Out-of Sample Holdout evaluation
                ## Show errors if anything goes wrong in the process
                try:
                    __model = ARIMA(endog=__df_temp["quantity_sold"], order=__best_cfg,)
                except ValueError as VE:
                    if __warning_inp.get() == "Yes":
                        if "A constant trend was included in the model" in str(VE):
                            message_List = (
                                "Trend Included in the Model."
                                + "\n"
```

```
                        + "Allow for gridsearching of Parameter [d]"
                        + "\n"
                        + "\n"
                        + "----- Model Quitting -----"
                    )

                    tk.messagebox.showerror(title="ERROR", message=message_List)
                    continue
                elif "zero-size array" in str(VE):
                    message_List = (
                        "Zero-Size Array Error"
                        + "\n"
                        + "Not enough historical data present"
                        + "\n"
                        + "\n"
                        + "----- Model Quitting -----"
                    )

                    tk.messagebox.showerror(
                        title="ERROR",
                        message="""Trend Included in the Model.
                        Allow for gridsearching of Parameter [d]""",
                    )

                    continue
            elif __warning_inp.get() != "Yes":
                if "A constant trend was included in the model" in str(VE):
                    trend_error = 1
                    error_dict[__site_id_value, __item_id_value] = str(
                        "Trend Error with "
                        + str(__order)
                        + " | Allow for Gridsearching of parameter d"
                    )

                elif "zero-size array" in str(VE):
                    array_error = 1
                    error_dict[
                        __site_id_value, __item_id_value
                    ] = "Array Error | Item too new to predict"
            else:
                continue
        # Fit the model and cooerce errors if needed
        try:
            __model_fit = __model.fit()
        except:
            error_dict[__site_id_value, __item_id_value] = str(
                "LU decomposition error with "
                + str(__order)
                + " | Model failed to fit")
            continue

        # Predict Hold-Out with the created models and prep the output
        __output = __model_fit.predict(
            start=len(__df_temp),
            end=int(len(__df_temp) + __prediction_time_frame - 1),
            dynamic=False,
        )

        # Fill in variables into dictionaries
        model_sum_dict[__site_id_value, __item_id_value] = __model_fit.summary()
        test_dict[__site_id_value, __item_id_value] = __test_temp.quantity_sold
        score_dict[__site_id_value, __item_id_value, "cfg"] = str(__best_cfg)
        score_dict[__site_id_value, __item_id_value, "rmse"] = round(
            __best_score, 2
        )

```

```
            score_dict[__site_id_value, __item_id_value, "sum"] = ceil(
                sum(__test_temp.quantity_sold)
            )

            # Do the Main Prediction on New data
            __model = ARIMA(endog=__df_temp_bu["quantity_sold"], order=__best_cfg)
            __model_fit = __model.fit()
            __output = __model_fit.predict(
                start=len(__df_temp_bu),
                end=int(len(__df_temp_bu) + __prediction_time_frame - 1),
                dynamic=False,
            )

            # Fill in the outputs into dictionaries
            model_sum_dict[__site_id_value, __item_id_value] = __model_fit.summary()
            test_dict[__site_id_value, __item_id_value] = __test_temp.quantity_sold

            prediction_dict[__site_id_value, __item_id_value] = round(__output, 2)

    # Create Future required variables
    __timeD = time.time()
    out_list = []
    promt_dict = {}

    # Create output Promts
    for i in item_list:
        for x in site_list:
            try:
                sum_val = ceil(sum(prediction_dict[x, i]))
                sum_val2 = score_dict[x, i, "sum"]
                val3 = ceil(sum(holdout_prediction_dict[x, i])) - sum_val2
                item_name = productdict[i]
                string_output = str(
                    "Item_ID = "
                    + str(i)
                    + " | Site = "
                    + str(x)
                    + "\n"
                    + "Item_Name = "
                    + str(item_name)
                    + "\n"
                    + "---Prediction---"
                    + "\n"
                    + prediction_dict[x, i].to_string()
                    + "\n"
                    + "----------------"
                    + "\n"
                    + "Predicted Sum of Quantity "
                    + str(__time_inp.get())
                    + " Days = "
                    + str(sum_val)
                    + "\n"
                    + "Expected Sum of Quantity last "
                    + str(__time_inp.get())
                    + " Days = "
                    + str(sum_val2)
                    + "\n"
                    + "OOS Total Error = "
                    + str(val3)
                    + "\n"
                    + "Best OOS Config = "
                    + str(score_dict[x, i, "cfg"])
                    + "\n"
                    + "Best OOS RMSE = "
                    + str(round(score_dict[x, i, "rmse"], 2))
                    + "\n"
```

```
                    + "\n"
                )

                out_list.append(string_output)
                promt_dict[i, x] = str(string_output)
            except:
                continue

    # Save Output when option is Yes
    if __save_inp.get() == "Yes":
        try:
            pd.DataFrame.from_dict(data=prediction_dict, orient="index").to_csv(
                "prediction_output.csv", header=True
            )

            pd.DataFrame.from_dict(data=test_dict, orient="index").to_csv(
                "test_data.csv", header=True
            )

            pd.DataFrame.from_dict(data=score_dict, orient="index").to_csv(
                "scores_output.csv", header=True
            )

            pd.DataFrame.from_dict(data=promt_dict, orient="index").to_csv(
                "promt_output.csv", header=True
            )

            message_string = "Predictions and Scores Saved at: " + str(os.getcwd())
            tk.messagebox.showinfo(title="Outcome", message=message_string)
        except:
            __stop_button_text.set("Exit Program")
            __sub_button_text.set("Submit and Run")
            __submit_btn["state"] = "normal"
            __stop_btn["state"] = "disabled"
            __timing_text_box.delete("1.0", "end")
            __timing_text_box.insert("end-1c", " ")
            __progress_text_box.delete("1.0", "end")
            __progress_text_box.insert("end-1c", " ")
            tk.messagebox.showerror(
                title="File Output",
                message="Output coercion to files failed, Make sure they are not in
                use.",
            )
            return
    else:
        pass

    # Warn that errors occured if Error Dict has lines in it
    if len(error_dict) > 0:
        try:
            pd.DataFrame.from_dict(data=error_dict, orient="index").to_csv(
                "error_output.csv", header=True
                )
        except:
            __stop_button_text.set("Exit Program")
            __sub_button_text.set("Submit and Run")
            __submit_btn["state"] = "normal"
            __stop_btn["state"] = "disabled"
            __timing_text_box.delete("1.0", "end")
            __timing_text_box.insert("end-1c", " ")
            __progress_text_box.delete("1.0", "end")
            __progress_text_box.insert("end-1c", " ")
            tk.messagebox.showerror(
                title="File Output",
                message="Output coercion to error_output.csv failed, Make sure it is
                not in use.",
```

```
            )
            return

    else:
        pass

    # Warn for Specific encountered Errors
    if (trend_error == 1) & (array_error == 1):
        message_string = (
            "Trend and Array Errors Encountered | Errors Saved at : "
            + "\n"
            + str(os.getcwd())
        )

        tk.messagebox.showwarning(title="Error Warning", message=message_string)
    elif trend_error == 1:
        message_string = (
            "Trend Errors Encountered | Errors Saved at : " + "\n" + str(os.getcwd())
        )

        tk.messagebox.showwarning(title="Error Warning", message=message_string)

    elif array_error == 1:
        message_string = (
            "Array Errors Encountered | Errors Saved at : " + "\n" + str(os.getcwd())
        )

        tk.messagebox.showwarning(title="Error Warning", message=message_string)
    else:
        pass

    # Warn about failure if output is empty
    # Or Show final message
    if len(out_list) == 0:

        __timing_text_box.delete(1.0, "end-1c")
        __timing_text_box.insert("end-1c", " ")
        __progress_text_box.delete(1.0, "end-1c")
        __progress_text_box.insert("end-1c", " ")
        tk.messagebox.showwarning(
            title="No Output",
            message="No combinations were valid so no predictions have been made",
        )
        pass
    else:
        __timediff2 = int(__timeD - __timeC)
        __timing_text_box.delete(1.0, "end-1c")
        __timing_text_box.insert(
            "end-1c", str("Total time elapsed = " + str(__timediff2) + " Seconds")
        )

        __progress_text_box.delete(1.0, "end-1c")
        __progress_text_box.insert("end-1c", str("Models evaluated  = " + str(counter)))
        __output_text_box.delete(1.0, "end-1c")
        __output_text_box.insert("end-1c", out_list)
        pass

    # Reset GUI Buttons to origional State for next run
    running = False
    __stop_button_text.set("Exit Program")
    __sub_button_text.set("Submit and Run")
    __submit_btn["state"] = "normal"
    __stop_btn["state"] = "disabled"


# In[4]:
```

```
1120
1121  # Set_Up GUI requirements
1122  __app = tk.Tk()
1123  __app.geometry()
1124  __app.title("Maverik: Candy Bar prediction")
1125  __app.resizable(width=FALSE, height=FALSE)
1126
1127
1128  __s = ttk.Style()
1129  __s.theme_use("alt")
1130
1131  __sub_button_text = tk.StringVar()
1132  __sub_button_text.set("Submit and Run")
1133
1134  __stop_button_text = tk.StringVar()
1135  __stop_button_text.set("Exit Program")
1136
1137  __browse_button_text = tk.StringVar()
1138  __browse_button_text.set("Browse")
1139
1140  __browse_button_text2 = tk.StringVar()
1141  __browse_button_text2.set("Browse")
1142
1143  __browse_button_text3 = tk.StringVar()
1144  __browse_button_text3.set("Browse")
1145
1146  __header_fontstyle = tkFont.Font(size=15)
1147  __header_fontstyle.configure(underline=True)
1148
1149
1150  sep_ver = Separator(__app, orient="vertical")
1151
1152  # Download Needed Images
1153  urllib.request.urlretrieve(
1154      "https://www.dropbox.com/s/hph99elpmvwjjjl/logo_maverick.gif?dl=1",
1155      "logo_maverick.gif",
1156  )
1157
1158  urllib.request.urlretrieve(
1159      "https://www.dropbox.com/s/2o85xau93qu53wa/maverick_icon.ico?dl=1",
1160      "maverick_icon.ico",
1161  )
1162
1163
1164  # Create Variables
1165  __df_folder_path = StringVar()
1166  __df_folder_path2 = StringVar()
1167  __df_folder_path3 = StringVar()
1168  __string_progress_text = StringVar()
1169  __string_progress_text = StringVar()
1170
1171  # Create first Line with Logo
1172  __logo = tk.PhotoImage(file="logo_maverick.gif", master=__app,)
1173
1174  __label00 = tk.Label(__app, text="")
1175  __label00.grid(column=1, row=0)
1176
1177  w1 = tk.Label(__app, image=__logo).grid(column=0, row=1, columnspan=9, sticky="ew")
1178
1179  # Add Padding
1180  __label00 = tk.Label(__app, text="    ")
1181  __label00.grid(column=6, row=0)
1182  __label01 = tk.Label(__app, text="    ")
1183  __label01.grid(column=0, row=0)
1184
1185
```

```
    sep_hor1 = Separator(__app, orient="horizontal")
    sep_hor1.grid(column=1, row=2, columnspan=8, sticky="ew")

    # Header
    tk.Label(__app, text="Files and Directory", font=__header_fontstyle).grid(
        row=3, column=0, columnspan=7
    )

    # Dataframe
    tk.Label(__app, text="Main Dataframe").grid(row=4, column=1, sticky=E)
    __browse_btn_1 = Button(textvariable=__browse_button_text, command=browse_button).grid(
        row=4, column=2, columnspan=4, sticky=EW
    )


    # Product_df
    tk.Label(__app, text="Product Dataframe").grid(row=5, column=1, sticky=E)
    __browse_btn_2 = Button(
        textvariable=__browse_button_text2, command=browse_button2
    ).grid(row=5, column=2, columnspan=4, sticky=EW)


    # Directory
    tk.Label(__app, text="Output Directory").grid(row=6, column=1, sticky=E)
    __browse_btn_3 = Button(
        textvariable=__browse_button_text3, command=browse_button3
    ).grid(row=6, column=2, columnspan=4, sticky=EW)


    # Padding
    __label02 = tk.Label(__app, text="      ")
    __label02.grid(column=0, row=7)

    sep_hor2 = Separator(__app, orient="horizontal")
    sep_hor2.grid(column=1, row=8, columnspan=5, sticky="ew")

    __label03 = tk.Label(__app, text="      ")
    __label03.grid(column=0, row=9)

    # Header
    tk.Label(__app, text="General Inputs", font=__header_fontstyle).grid(
        row=9, column=0, columnspan=7
    )

    # Item_id
    tk.Label(__app, text="Item_ID").grid(row=10, column=1, sticky=E)
    __item_id_inp = tk.Entry(__app)
    __item_id_inp.insert(END, "10")
    __item_id_inp.grid(row=10, column=2)

    __submit_btn_4 = Button(__app, text="Check", width=10, command=check_item_id)
    __submit_btn_4.grid(row=10, column=4)

    __item_id_inp_text_box = tk.Text(__app, width=15, height=1)
    __item_id_inp_text_box.grid(row=10, column=5, columnspan=1)
    __item_id_inp_text_box.insert("end-1c", "    ")

    # Site_id
    tk.Label(__app, text="Site_ID").grid(row=11, column=1, sticky=E)
    __site_id_inp = tk.Entry(__app)
    __site_id_inp.grid(row=11, column=2)

    __submit_btn_5 = Button(__app, text="Check", width=10, command=check_site_id)
    __submit_btn_5.grid(row=11, column=4)

    __site_id_text_box = tk.Text(__app, width=15, height=1)
```

```
__site_id_text_box.grid(row=11, column=5, columnspan=1)
__site_id_text_box.insert("end-1c", "    ")

# Cleaning_Std
tk.Label(__app, text="Outlier Std.Dev.").grid(row=12, column=1, sticky=E)
__std_dev_inp = tk.Entry(__app)
__std_dev_inp.insert(END, "4")
__std_dev_inp.grid(row=12, column=2)

__submit_btn_3 = Button(__app, text="Check", width=10, command=check_standard_dev)
__submit_btn_3.grid(row=12, column=4)

__std_dev_inp_text_box = tk.Text(__app, width=15, height=1)
__std_dev_inp_text_box.grid(row=12, column=5, columnspan=1)
__std_dev_inp_text_box.insert("end-1c", "    ")

# Time
tk.Label(__app, text="Days to Predict").grid(row=13, column=1, sticky=E)
__time_inp = tk.Entry(__app)
__time_inp.insert(END, "10")
__time_inp.grid(row=13, column=2)

__submit_btn_9 = Button(__app, text="Check", width=10, command=check_time)
__submit_btn_9.grid(row=13, column=4)

__time_inp_text_box = tk.Text(__app, width=15, height=1)
__time_inp_text_box.grid(row=13, column=5, columnspan=1)
__time_inp_text_box.insert("end-1c", "    ")


# Padding
__label04 = tk.Label(__app, text="    ")
__label04.grid(column=0, row=14)

sep_hor3 = Separator(__app, orient="horizontal")
sep_hor3.grid(column=1, row=15, columnspan=5, sticky="ew")

ver_hor3 = Separator(__app, orient="horizontal")
ver_hor3.grid(column=3, row=17, rowspan=3, sticky="ns")

# Header
tk.Label(__app, text="Parameter Inputs", font=__header_fontstyle).grid(
    row=16, column=0, columnspan=7
)

# P
tk.Label(__app, text="Starting p").grid(row=17, column=1, sticky=E)
__p_inp = tk.Entry(__app)
__p_inp.insert(END, "7")
__p_inp.grid(row=17, column=2)


tk.Label(__app, text="p Grid Mult").grid(row=17, column=4, sticky=E)
__p_gs_list_inp = tk.Entry(__app)
__p_gs_list_inp.insert(END, "0,1,2")
__p_gs_list_inp.grid(row=17, column=5)

# d
tk.Label(__app, text="Starting d").grid(row=18, column=1, sticky=E)
__d_inp = tk.Entry(__app)
__d_inp.insert(END, "0")
__d_inp.grid(row=18, column=2)

tk.Label(__app, text="d Grid Sum").grid(row=18, column=4, sticky=E)
__d_gs_list_inp = tk.Entry(__app)
__d_gs_list_inp.insert(END, "0,1,2")
```

```
__d_gs_list_inp.grid(row=18, column=5)

# q
tk.Label(__app, text="Starting q").grid(row=19, column=1, sticky=E)
__q_inp = tk.Entry(__app)
__q_inp.insert(END, "1")
__q_inp.grid(row=19, column=2)

tk.Label(__app, text="q Grid Mult").grid(row=19, column=4, sticky=E)
__q_gs_list_inp = tk.Entry(__app)
__q_gs_list_inp.insert(END, "0,1,2")
__q_gs_list_inp.grid(row=19, column=5)

# Padding
__label05 = tk.Label(__app, text="   ")
__label05.grid(column=0, row=20)

sep_hor4 = Separator(__app, orient="horizontal")
sep_hor4.grid(column=1, row=21, columnspan=5, sticky="ew")

# Header
tk.Label(__app, text="Options", font=__header_fontstyle).grid(
    row=22, column=0, columnspan=7
)

# Input
__label4 = tk.Label(__app, text="Gridsearch All")
__label4.grid(column=1, row=23, sticky=E)

__gridsearch_inp = ttk.Combobox(__app, values=["Yes", "No"])
__gridsearch_inp.insert(END, "No")
__gridsearch_inp.grid(column=2, row=23)

# Gridsearch 2
__label5 = tk.Label(__app, text="Gridsearch d only")
__label5.grid(column=4, row=23, sticky=E)

__gridsearch_d_inp = ttk.Combobox(__app, values=["Yes", "No"])
__gridsearch_d_inp.insert(END, "Yes")
__gridsearch_d_inp.grid(column=5, row=23)

# Save
__label7 = tk.Label(__app, text="Save Outcome")
__label7.grid(column=1, row=24, sticky=E)

__save_inp = ttk.Combobox(__app, values=["Yes", "No"])
__save_inp.insert(END, "No")
__save_inp.grid(column=2, row=24)

# Warnings
__label8 = tk.Label(__app, text="Show Warnings")
__label8.grid(column=4, row=24, sticky=E)

__warning_inp = ttk.Combobox(__app, values=["Yes", "No"])
__warning_inp.insert(END, "Yes")
__warning_inp.grid(column=5, row=24)

# Padding
__label05 = tk.Label(__app, text="   ")
__label05.grid(column=0, row=26, sticky=E)

sep_hor5 = Separator(__app, orient="horizontal")
sep_hor5.grid(column=1, row=27, columnspan=5, sticky="ew")

## Submit
__label06 = tk.Label(__app, text="   ")
```

```
__label06.grid(column=0, row=28, sticky=E)

__submit_btn = tk.Button(
    __app,
    textvariable=__sub_button_text,
    command=button_click,
    height=2,
    width=20,
    bg="#cc0000",
    fg="white",
)

# __submit_btn.grid(row=29, column=2, columnspan=3, padx=10, pady=25)
__submit_btn.grid(row=29, column=1, columnspan=2, padx=10, pady=25)

__stop_btn = tk.Button(
    __app,
    textvariable=__stop_button_text,
    command=stop,
    height=2,
    width=20,
    bg="#cc0000",
    fg="white",
    state=DISABLED,
)

__stop_btn.grid(row=29, column=4, columnspan=2, padx=10, pady=25)


# Padding
__label07 = tk.Label(__app, text="")
__label07.grid(column=1, row=30)

# Progress Boxes
__timing_text_box = tk.Text(__app, height=1, width=30)
__timing_text_box.tag_configure("center", justify="center")
__timing_text_box.grid(row=31, column=1, columnspan=5, sticky="ew")
__timing_text_box.insert("end-1c", " ")

__progress_text_box = tk.Text(__app, height=1, width=30)
__progress_text_box.tag_configure("center", justify="center")
__progress_text_box.grid(row=32, column=1, columnspan=5, sticky="ew")
__progress_text_box.insert("end-1c", " ")

# Padding
__label08 = tk.Label(__app, text="")
__label08.grid(column=1, row=33)

__label09 = tk.Label(__app, text="    ")
__label09.grid(column=9, row=4)

# Output Box

tk.Label(__app, text="Output", font=__header_fontstyle).grid(row=3, column=7)

__output_text_box = tk.Text(__app, width=50)
__output_text_box.grid(row=4, column=7, rowspan=29, sticky="ns")


yscroll = tk.Scrollbar(command=__output_text_box.yview, orient=tk.VERTICAL)
yscroll.grid(row=4, column=8, rowspan=29, sticky="ns")
__output_text_box.configure(yscrollcommand=yscroll.set)

sep_hor5 = Separator(__app, orient="horizontal")
sep_hor5.grid(column=1, row=34, columnspan=8, sticky="ew")

```

```
1450    __label10 = tk.Label(__app, text="")
1451    __label10.grid(column=1, row=35)
1452
1453    #set Title Bar Icon
1454    __app.iconbitmap('maverick_icon.ico')
1455
1456    # Call Main Loop GUI
1457    __app.mainloop()
1458
```