

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Apr 11 14:49:28 2021
4
5  @author: Willem van der Schans
6
7  Related Github Directory:
8  https://github.com/Kydoimos97/CapstoneMSBA2020
9  """
10
11  #!/usr/bin/env python
12  # coding: utf-8
13
14
15  # In[1]:
16
17  # Math
18  from math import ceil, floor, sqrt
19
20  # Plotting
21  import matplotlib
22
23  matplotlib.use("Agg")
24  import matplotlib.pyplot as plt
25
26  # Numpy & Pandas
27  import numpy as np
28  import pandas as pd
29
30  # Stats models
31  import statsmodels.api as sm
32  from matplotlib.pyplot import figure
33
34  # Linear Imputation
35  from scipy.interpolate import interp1d
36
37  # Machine Learning
38  from sklearn.metrics import mean_squared_error
39  from statsmodels.tsa.arima.model import ARIMA
40
41  # Augmented Dickey Fuller Test
42  from statsmodels.tsa.stattools import adfuller
43
44  # Options
45  pd.options.display.max_rows = 2500
46
47  # Low Verbose Warning Coercion
48  import warnings
49  warnings.filterwarnings("ignore",
50                          "statsmodels.tsa.arima_model.ARMA", FutureWarning)
51  warnings.filterwarnings("ignore",
52                          "statsmodels.tsa.arima_model.ARIMA", FutureWarning)
53  warnings.filterwarnings("ignore")
54
55  # Set working Directory In code
56  import os
57
58  # MultiThreading
59  import subprocess as sub
60
61  # Dependency Import
62  import sys
63
64  # Time Tracking
65  import time
66
```

```

67 # First GUI element Imports
68 import tkinter as tk
69 import tkinter.font as tkFont
70 import tkinter.messagebox
71
72 # Download Files from the Web
73 import urllib
74
75 #Multi Threading
76 from threading import Thread
77
78 # Second GUI element Imports
79 from tkinter import *
80 from tkinter import filedialog, ttk
81 from tkinter.ttk import Separator, Style
82
83
84 # In[2]:
85 # Create all supporting Functions to be called in the GUI
86
87 def check_standard_dev():
88     value = __std_dev_inp.get()
89     if value.isdigit():
90         __std_dev_inp_text_box.delete(1.0, "end-1c")
91         __std_dev_inp_text_box.insert("end-1c", u"\u2714")
92     else:
93         __std_dev_inp_text_box.delete(1.0, "end-1c")
94         __std_dev_inp_text_box.insert("end-1c", u"\u2717")
95
96
97 def check_item_id():
98     value = __item_id_inp.get()
99     value_str = str(value)
100     if ("," in str(value)) and (value_str.startswith("-")):
101         __item_id_inp_text_box.delete(1.0, "end-1c")
102         __item_id_inp_text_box.insert("end-1c", "Item_ID List \u2714")
103     elif value.isdigit():
104         __item_id_inp_text_box.delete(1.0, "end-1c")
105         __item_id_inp_text_box.insert("end-1c", "Top_N \u2714")
106     elif value_str.startswith("-"):
107         __item_id_inp_text_box.delete(1.0, "end-1c")
108         __item_id_inp_text_box.insert("end-1c", "Item_ID \u2714")
109     else:
110         __item_id_inp_text_box.delete(1.0, "end-1c")
111         __item_id_inp_text_box.insert("end-1c", u"\u2717")
112
113
114 def check_site_id():
115     value = __site_id_inp.get()
116     if value == "":
117         __site_id_text_box.delete(1.0, "end-1c")
118         __site_id_text_box.insert("end-1c", "All_Sites \u2714")
119     elif value.isdigit():
120         if len(value) == 3:
121             __site_id_text_box.delete(1.0, "end-1c")
122             __site_id_text_box.insert("end-1c", "Site_ID \u2714")
123         else:
124             pass
125     elif "," in str(value):
126         __site_id_text_box.delete(1.0, "end-1c")
127         __site_id_text_box.insert("end-1c", "Site_ID List \u2714")
128     else:
129         __site_id_text_box.delete(1.0, "end-1c")
130         __site_id_text_box.insert("end-1c", u"\u2717")
131
132

```

```

133 def check_time():
134     value = __time_inp.get()
135     if value.isdigit():
136         __time_inp_text_box.delete(1.0, "end-1c")
137         __time_inp_text_box.insert("end-1c", u"\u2714")
138     else:
139         __time_inp_text_box.delete(1.0, "end-1c")
140         __time_inp_text_box.insert("end-1c", u"\u2717")
141
142
143 def browse_button():
144     # Allow user to select a directory and store it in global var
145     # called folder_path
146     global __df_folder_path
147     __filename = filedialog.askopenfilename()
148     __df_folder_path.set(str(__filename))
149     __browse_button_text.set("/".join(str(__filename).split("/", -1)[-3:]))
150
151
152 def browse_button2():
153     # Allow user to select a directory and store it in global var
154     # called folder_path
155     global __df_folder_path2
156     __filename2 = filedialog.askopenfilename()
157     __df_folder_path2.set(str(__filename2))
158     __browse_button_text2.set("/".join(str(__filename2).split("/", -1)[-3:]))
159
160
161 def browse_button3():
162     # Allow user to select a directory and store it in global var
163     # called folder_path
164     global __df_folder_path2
165     __filename3 = filedialog.askdirectory()
166     __df_folder_path3.set(str(__filename3))
167     __browse_button_text3.set("/".join(str(__filename3).split("/", -1)[-3:]))
168
169
170 def duplicate_remover(x):
171     return list(dict.fromkeys(x))
172
173 # Main Starting Button
174 def button_click():
175     global executing # create global
176     executing = True
177
178     # Create new thread
179     t = Thread(target=getInput)
180     # Start new thread
181     t.start()
182
183 # Stop Button
184 def stop():
185     global executing # create global
186     executing = False
187     # Disable Button to prevent backlog of clicks
188     __stop_btn["state"] = "disabled"
189     __timing_text_box.delete("1.0", "end")
190     __timing_text_box.insert("end-1c", " ")
191     __progress_text_box.delete("1.0", "end")
192     __progress_text_box.insert("end-1c", " ")
193
194
195 running = True # Global flag
196
197
198 # 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     )

```

```

331     temp = temp.drop(
332         columns=[
333             "location_id",
334             "open_date",
335             "sq_footage",
336             "locale",
337             "maxtemp",
338             "mintemp",
339             "fiscal_period",
340             "periodic_gbv",
341             "current_gbv",
342             "mpds",
343         ]
344     )
345     index_list.append(temp.iloc[0, 0])
346
347     ## Reduce Dimensions and Ram Usage
348     df.rename(columns={df.columns[0]: "index_set"}, inplace=True)
349
350     df = df[~df.index_set.isin(index_list)]
351
352     df.drop(
353         [
354             "index_set",
355             "location_id",
356             "current_gbv",
357             "product_name",
358             "open_date",
359             "fiscal_period",
360             "project",
361             "price",
362         ],
363         axis=1,
364         inplace=True,
365     )
366
367     ### Recode Locale Variable
368     locale_string = list(df.locale.unique())
369     locale_replace = list(range(1, len(locale_string) + 1))
370     df.locale.replace(locale_string, locale_replace, inplace=True)
371
372     df = df[
373         [
374             "site_id",
375             "item_id",
376             "date",
377             "sq_footage",
378             "mpds",
379             "locale",
380             "periodic_gbv",
381             "maxtemp",
382             "mintemp",
383             "quantity_sold",
384             "sales",
385         ]
386     ]
387
388     ## Create data set for item_id top N Input
389     __df_agg = (
390         df.groupby(["item_id"])
391         .agg(
392             total_sales=pd.NamedAgg(column="sales", aggfunc=sum),
393             days_sold=pd.NamedAgg(column="sales", aggfunc="count"),
394         )
395         .reset_index()
396         .sort_values("total_sales", ascending=False)

```

```

397 )
398
399 ## Get Item and Site Inputs and create lists
400 item_list = []
401 site_list = []
402
403 value = __item_id_inp.get()
404 value_str = str(value)
405 if ("," in str(value)) and (value_str.startswith("-")):
406     value_str = value_str.replace(" ", "")
407     item_list = value_str.split(",")
408     for __i in range(0, len(item_list)):
409         item_list[__i] = int(item_list[__i])
410 elif value.isdigit():
411     item_list = list(__df_agg.item_id.head(int(value)))
412
413 elif value_str.startswith("-"):
414     item_list = [int(__item_id_inp.get())]
415 else:
416     item_list = list(__df_agg.item_id.unique())
417
418 value = __site_id_inp.get()
419 if value == "":
420     site_list = list(df.site_id.unique())
421 elif value.isdigit():
422     if len(value) == 3:
423         site_list = [int(__site_id_inp.get())]
424     else:
425         pass
426 elif "," in str(value):
427     site_list = str(value).replace(" ", "").split(",")
428     for __i in range(0, len(site_list)):
429         site_list[__i] = int(site_list[__i])
430
431 ## Create further lists and dictionaries
432 date_list = list(df.sort_values("date").date.unique())
433 __app_list = []
434 prediction_dict = {}
435 holdout_prediction_dict = {}
436 test_dict = {}
437 model_sum_dict = {}
438
439 ## Create a dictionary for none changing values and fill it
440 df_dict = {}
441
442 for __i in site_list:
443     df_dict[int(__i)] = []
444
445 for __i in range(0, len(site_list)):
446     __site_id_value = site_list[__i]
447
448     __df_temp = (
449         df.loc[(df["site_id"] == __site_id_value)]
450         .reset_index(drop=True)
451         .sort_values("date")
452     )
453
454     df_dict[__site_id_value].append(int(__df_temp.sq_footage.mode()))
455     df_dict[__site_id_value].append(int(__df_temp.mpsds.mode()))
456     df_dict[__site_id_value].append(int(__df_temp.locale.mode()))
457     df_dict[__site_id_value].append(int(__df_temp.periodic_gbv.mode()))
458
459 except:
460     __stop_button_text.set("Exit Program")
461     __sub_button_text.set("Submit and Run")

```

```

463     __submit_btn["state"] = "normal"
464     __stop_btn["state"] = "disabled"
465     tk.messagebox.showerror(
466         title="Program Stopped",
467         message="Supplied inputs or files are not compatible",
468     )
469     return
470
471 # Start Timing
472 __timeC = time.time()
473
474 # Prep the Parameter Inputs
475 __p_values = []
476 __d_values = []
477 __q_values = []
478
479 # Read the Input
480 value = __p_gs_list_inp.get()
481 value_str = str(value)
482 if "," in str(value):
483     value_str = value_str.replace(" ", "")
484     __p_gs_list = value_str.split(",")
485     for __i in range(0, len(__p_gs_list)):
486         __p_gs_list[__i] = int(__p_gs_list[__i])
487 else:
488     __p_gs_list = [0, 1, 2]
489
490 value = __d_gs_list_inp.get()
491 value_str = str(value)
492 if "," in str(value):
493     value_str = value_str.replace(" ", "")
494     __d_gs_list = value_str.split(",")
495     for __i in range(0, len(__d_gs_list)):
496         __d_gs_list[__i] = int(__d_gs_list[__i])
497 else:
498     __d_gs_list = [0, 1, 2]
499
500 value = __q_gs_list_inp.get()
501 value_str = str(value)
502 if "," in str(value):
503     value_str = value_str.replace(" ", "")
504     __q_gs_list = value_str.split(",")
505     for __i in range(0, len(__q_gs_list)):
506         __q_gs_list[__i] = int(__q_gs_list[__i])
507 else:
508     __q_gs_list = [0, 1, 2]
509
510 # Create Grid Searching Lists
511 if __gridsearch_inp.get() == "Yes":
512     if int(__p_inp.get()) == 0:
513         if int(__q_inp.get()) == 0:
514             for __p in __p_gs_list:
515                 __p_values.append(int(int(__p_inp.get()) + int(__p)))
516             for __d in __d_gs_list:
517                 __d_values.append(int(int(__d_inp.get()) + int(__d)))
518             for __q in __q_gs_list:
519                 __q_values.append(int(int(__q_inp.get()) + int(__q)))
520         else:
521             for __p in __p_gs_list:
522                 __p_values.append(int(int(__p_inp.get()) + int(__p)))
523             for __d in __d_gs_list:
524                 __d_values.append(int(int(__d_inp.get()) + int(__d)))
525             for __q in __q_gs_list:
526                 __q_values.append(int(int(__q_inp.get()) * int(__q)))
527         elif int(__q_inp.get()) == 0:
528             for __p in __p_gs_list:

```



```

529         __p_values.append(int(int(__p_inp.get()) * int(__p)))
530     for __d in __d_gs_list:
531         __d_values.append(int(int(__d_inp.get()) + int(__d)))
532     for __q in __q_gs_list:
533         __q_values.append(int(int(__q_inp.get()) + int(__q)))
534     else:
535         for __p in __p_gs_list:
536             __p_values.append(int(int(__p_inp.get()) * int(__p)))
537         for __d in __d_gs_list:
538             __d_values.append(int(int(__d_inp.get()) + int(__d)))
539         for __q in __q_gs_list:
540             __q_values.append(int(int(__q_inp.get()) * int(__q)))
541     else:
542         __p_values = [int(__p_inp.get())]
543         __q_values = [int(__q_inp.get())]
544         if __gridsearch_d_inp.get() == "Yes":
545             for __d in __d_gs_list:
546                 __d_values.append(int(int(__d_inp.get()) + int(__d)))
547         else:
548             __d_values = [int(__d_inp.get())]
549
550     # Remove Potential Duplicate values to limit runtime.
551     __p_values = duplicate_remover(__p_values)
552     __d_values = duplicate_remover(__d_values)
553     __q_values = duplicate_remover(__q_values)
554
555     # Create Machine Learning Variables and dictionaries
556     __best_score = 1000000000
557     __best_cfg = 0
558     counter = 0
559     __timeA = time.time()
560     __timeB = time.time()
561     __site_id_value = "None"
562
563     # Create Machine Inputs and Dictionaries
564     __prediction_time_frame = int(__time_inp.get())
565     score_dict = {}
566     error_dict = {}
567
568     # Set Binary Indicators for Errors
569     trend_error = 0
570     array_error = 0
571     combination_error = 0
572
573
574
575     # Start Main Algorithm Loop
576     __stop_btn["state"] = "normal"
577     for __i in range(0, len(item_list)):
578         __item_id_value = item_list[__i]
579         # Create a point to stop the loop
580         if executing == False:
581             __stop_button_text.set("Exit Program")
582             __sub_button_text.set("Submit and Run")
583             __submit_btn["state"] = "normal"
584             __stop_btn["state"] = "disabled"
585             __timing_text_box.delete("1.0", "end")
586             __timing_text_box.insert("end-1c", " ")
587             __progress_text_box.delete("1.0", "end")
588             __progress_text_box.insert("end-1c", " ")
589             tk.messagebox.showwarning(
590                 title="Program Stopped", message=str(
591                     "Program Stopped on Item = "
592                     + str(__item_id_value)
593                     + " and Site = "
594                     + str(__site_id_value))

```

```

595         )
596
597         return
598     elif __i > 0:
599         # Update timing text box
600         __timeB = time.time()
601         __timediff = __timeB - __timeA
602         __string_timing_text = ""
603         __string_timing_text = str(
604             "Item "
605             + str(__i)
606             + "/"
607             + str(len(item_list))
608             + " | Runtime = "
609             + str(int(__timediff))
610             + " Sec | Runtime Left = "
611             + str(int((__timediff / __i) * (len(item_list) - __i)))
612             + " Sec"
613         )
614         __timing_text_box.delete("1.0", "end")
615         __timing_text_box.insert("end-1c", __string_timing_text)
616     else:
617         __timing_text_box.delete("1.0", "end")
618         __timing_text_box.insert("end-1c", "Initializing")
619     for __x in range(0, len(site_list)):
620         # Nested Loop site_ID under Item_ID
621         __site_id_value = site_list[__x]
622
623         __df_temp = (
624             df.loc[
625                 (df["site_id"] == __site_id_value)
626                 & (df["item_id"] == __item_id_value)
627             ]
628             .reset_index(drop=True)
629             .sort_values("date")
630         )
631
632         # Check if item and site combination exist
633         try:
634             __index_pos = date_list.index(__df_temp.date.unique()[0])
635         except:
636             combination_error = 1
637             error_dict[
638                 __item_id_value, __site_id_value
639             ] = "Combination Does not Exist"
640             continue
641
642         # Create a date list of differences to impute zero sales
643         __trunc_date_list = list(date_list[__index_pos : len(date_list)])
644
645         __diff_list = list(
646             set(__trunc_date_list) - set(list(__df_temp.date.unique()))
647         )
648
649         if len(__diff_list) > 0:
650
651             for __y in range(0, len(__diff_list)):
652                 __app_list = []
653                 __app_list.extend(
654                     (
655                         __site_id_value,
656                         __item_id_value,
657                         __diff_list[__y],
658                         df_dict[__site_id_value][0],
659                         df_dict[__site_id_value][1],
660                         df_dict[__site_id_value][2],

```

```

661         df_dict[__site_id_value][3],
662         np.nan,
663         np.nan,
664         0,
665         0,
666     )
667 )
668 __app_series = pd.Series(__app_list, index=__df_temp.columns)
669 __df_temp = __df_temp.append(__app_series, ignore_index=True)
670
671 else:
672     pass
673
674 # sort newly created df by sate
675 __df_temp = __df_temp.sort_values("date")
676 # impute missing temperature values with linear regression
677 __df_temp["mintemp"].interpolate(method="linear", inplace=True)
678 __df_temp["maxtemp"].interpolate(method="linear", inplace=True)
679 # Overwrite index to date
680 __df_temp["dateind"] = __df_temp["date"]
681 __df_temp.set_index("dateind", inplace=True)
682 # Set Frequency for Arima Model
683 __df_temp.index = pd.DatetimeIndex(
684     __df_temp.index.values, freq=__df_temp.index.inferred_freq
685 )
686
687 # Hold-Out Evaluation
688 ## Create Test Dataframe
689 __test_temp = __df_temp.tail(__prediction_time_frame)
690 __df_temp_bu = __df_temp
691 ## Create Train Dataframe
692 __df_temp = __df_temp.drop(__df_temp.tail(__prediction_time_frame).index)
693
694 # Item and Site Combination First Sale Check
695 if int(len(__df_temp)) < int(__time_inp.get()):
696     if __warning_inp.get() == "Yes":
697         message_list = (
698             "Not enough data For Item_ID = "
699             + str(__item_id_value)
700             + " & Site_ID = "
701             + str(__site_id_value)
702             + "\n"
703             + "Total Data Points = "
704             + str(len(__df_temp_bu))
705             + "\n"
706             + "Not enough historical data present for this combination"
707             + "\n"
708             + "No Prediction can be made at this time"
709         )
710
711 tk.messagebox.showerror(title="Warning", message=message_list)
712
713 array_error = 1
714 error_dict[
715     __item_id_value, __site_id_value
716 ] = "No Historical Data | Item too new for the store"
717 elif __warning_inp.get() != "Yes":
718     array_error = 1
719     error_dict[
720         __item_id_value, __site_id_value
721     ] = "No Historical Data | Item too new for the store"
722
723     continue
724 else:
725     pass
726

```

```

727 # Gridsearching Loop
728 __best_score = 1000000000
729 __best_cfg = (0, 0, 0)
730 __total_counter = 0
731 for __p in __p_values:
732     for __d in __d_values:
733         for __q in __q_values:
734             __order = (__p, __d, __q)
735             # Create Break point for Loop
736             if executing == False:
737                 __stop_button_text.set("Exit Program")
738                 __sub_button_text.set("Submit and Run")
739                 __submit_btn["state"] = "normal"
740                 __stop_btn["state"] = "disabled"
741                 __timing_text_box.delete("1.0", "end")
742                 __timing_text_box.insert("end-1c", " ")
743                 __progress_text_box.delete("1.0", "end")
744                 __progress_text_box.insert("end-1c", " ")
745                 tk.messagebox.showwarning(
746                     title="Program Stopped", message=str(
747                         "Program Stopped on Item = "
748                         + str(__item_id_value)
749                         + " and Site = "
750                         + str(__site_id_value))
751                 )
752
753             return
754         try:
755             # Currently Univariate
756             ## Exog needed for Multivariate
757             __model = ARIMA(
758                 endog=__df_temp["quantity_sold"], order=__order
759             )
760             try:
761                 __model_fit = __model.fit()
762             except:
763                 error_dict[__site_id_value, __item_id_value] = str(
764                     "LU decomposition error with "
765                     + str(__order)
766                     + " | Model failed to fit"
767                 )
768                 continue
769             __output = __model_fit.predict(
770                 start=len(__df_temp),
771                 end=int(len(__df_temp) + __prediction_time_frame - 1),
772                 dynamic=False,
773             )
774
775             __rmse = sqrt(
776                 mean_squared_error(__test_temp.quantity_sold, __output)
777             )
778
779             counter = counter + 1
780             __total_counter = (
781                 len(__p_values)
782                 * len(__d_values)
783                 * len(__q_values)
784                 * len(item_list)
785                 * len(site_list)
786             )
787
788             # Give progress outputs
789             __string_progress_text = str(
790                 "Progress "
791                 + str(counter)
792                 + " models evaluated out of an expected "

```

```

793         + str(__total_counter)
794     )
795
796     __progress_text_box.delete("1.0", "end")
797     __progress_text_box.insert("end-1c", __string_progress_text)
798
799     # Update config is RMSE is lower
800     if __rmse < __best_score:
801         __best_score = __rmse
802         __best_cfg = __order
803         holdout_prediction_dict[
804             __site_id_value, __item_id_value
805         ] = __output
806     else:
807         pass
808 except ValueError as VE:
809     if __warning_inp.get() == "Yes":
810         if "A constant trend was included in the model" in str(
811             VE
812         ):
813             tk.messagebox.showerror(
814                 title="ERROR",
815                 message="""Trend Included in the Model. Allow
816                             for gridsearching of Parameter [d]""",
817             )
818         else:
819             pass
820     else:
821         pass
822
823     # if no better rmse is found or loop failed, input values
824     if __best_cfg == (0, 0, 0):
825         __best_cfg = __order
826     else:
827         pass
828
829     # Create Break Point
830     if executing == False:
831         __stop_button_text.set("Exit Program")
832         __sub_button_text.set("Submit and Run")
833         __submit_btn["state"] = "normal"
834         __stop_btn["state"] = "disabled"
835         __timing_text_box.delete("1.0", "end")
836         __timing_text_box.insert("end-1c", " ")
837         __progress_text_box.delete("1.0", "end")
838         __progress_text_box.insert("end-1c", " ")
839         tk.messagebox.showwarning(
840             title="Program Stopped", message=str(
841                 "Program Stopped on Item = "
842                 + str(__item_id_value)
843                 + " and Site = "
844                 + str(__site_id_value))
845         )
846     return
847
848     # Create the Main Prediction Model
849     ## Final Out-of Sample Holdout evaluation
850     ## Show errors if anything goes wrong in the process
851     try:
852         __model = ARIMA(endog=__df_temp["quantity_sold"], order=__best_cfg,)
853     except ValueError as VE:
854         if __warning_inp.get() == "Yes":
855             if "A constant trend was included in the model" in str(VE):
856                 message_List = (
857                     "Trend Included in the Model."
858                     + "\n"

```

```

858         + "Allow for gridsearching of Parameter [d]"
859         + "\n"
860         + "\n"
861         + "----- Model Quitting -----"
862     )
863
864     tk.messagebox.showerror(title="ERROR", message=message_List)
865     continue
866     elif "zero-size array" in str(VE):
867         message_List = (
868             "Zero-Size Array Error"
869             + "\n"
870             + "Not enough historical data present"
871             + "\n"
872             + "\n"
873             + "----- Model Quitting -----"
874         )
875
876         tk.messagebox.showerror(
877             title="ERROR",
878             message="Trend Included in the Model. Allow for gridsearching of Parameter [d]",
879         )
880
881         continue
882     elif __warning_inp.get() != "Yes":
883         if "A constant trend was included in the model" in str(VE):
884             trend_error = 1
885             error_dict[__site_id_value, __item_id_value] = str(
886                 "Trend Error with "
887                 + str(__order)
888                 + " | Allow for Gridsearching of parameter d"
889             )
890
891         elif "zero-size array" in str(VE):
892             array_error = 1
893             error_dict[
894                 __site_id_value, __item_id_value
895             ] = "Array Error | Item too new to predict"
896
897         else:
898             continue
899         # Fit the model and coerce errors if needed
900         try:
901             __model_fit = __model.fit()
902         except:
903             error_dict[__site_id_value, __item_id_value] = str(
904                 "LU decomposition error with "
905                 + str(__order)
906                 + " | Model failed to fit"
907             )
908             continue
909
910         # Predict Hold-Out with the created models and prep the output
911         __output = __model_fit.predict(
912             start=len(__df_temp),
913             end=int(len(__df_temp) + __prediction_time_frame - 1),
914             dynamic=False,
915         )
916
917         # Fill in variables into dictionaries
918         model_sum_dict[__site_id_value, __item_id_value] = __model_fit.summary()
919         test_dict[__site_id_value, __item_id_value] = __test_temp.quantity_sold
920         score_dict[__site_id_value, __item_id_value, "cfg"] = str(__best_cfg)
921         score_dict[__site_id_value, __item_id_value, "rmse"] = round(
922             __best_score, 2
923         )

```

```

924     score_dict[__site_id_value, __item_id_value, "sum"] = ceil(
925         sum(__test_temp.quantity_sold)
926     )
927
928     # Do the Main Prediction on New data
929     __model = ARIMA(endog=__df_temp_bu["quantity_sold"], order=__best_cfg)
930     __model_fit = __model.fit()
931     __output = __model_fit.predict(
932         start=len(__df_temp_bu),
933         end=int(len(__df_temp_bu) + __prediction_time_frame - 1),
934         dynamic=False,
935     )
936
937     # Fill in the outputs into dictionaries
938     model_sum_dict[__site_id_value, __item_id_value] = __model_fit.summary()
939     test_dict[__site_id_value, __item_id_value] = __test_temp.quantity_sold
940
941     prediction_dict[__site_id_value, __item_id_value] = round(__output, 2)
942
943     # Create Future required variables
944     __timeD = time.time()
945     out_list = []
946     prompt_dict = {}
947
948     # Create output Prompts
949     for i in item_list:
950         for x in site_list:
951             try:
952                 sum_val = ceil(sum(prediction_dict[x, i]))
953                 sum_val2 = score_dict[x, i, "sum"]
954                 val3 = ceil(sum(holdout_prediction_dict[x, i])) - sum_val2
955                 item_name = productdict[i]
956                 string_output = str(
957                     "Item_ID = "
958                     + str(i)
959                     + " | Site = "
960                     + str(x)
961                     + "\n"
962                     + "Item_Name = "
963                     + str(item_name)
964                     + "\n"
965                     + "---Prediction---"
966                     + "\n"
967                     + prediction_dict[x, i].to_string()
968                     + "\n"
969                     + "-----"
970                     + "\n"
971                     + "Predicted Sum of Quantity "
972                     + str(__time_inp.get())
973                     + " Days = "
974                     + str(sum_val)
975                     + "\n"
976                     + "Expected Sum of Quantity last "
977                     + str(__time_inp.get())
978                     + " Days = "
979                     + str(sum_val2)
980                     + "\n"
981                     + "OOS Total Error = "
982                     + str(val3)
983                     + "\n"
984                     + "Best OOS Config = "
985                     + str(score_dict[x, i, "cfg"])
986                     + "\n"
987                     + "Best OOS RMSE = "
988                     + str(round(score_dict[x, i, "rmse"], 2))
989                     + "\n"

```

```

990         + "\n"
991     )
992
993     out_list.append(string_output)
994     prompt_dict[i, x] = str(string_output)
995 except:
996     continue
997
998 # Save Output when option is Yes
999 if __save_inp.get() == "Yes":
1000     try:
1001         pd.DataFrame.from_dict(data=prediction_dict, orient="index").to_csv(
1002             "prediction_output.csv", header=True
1003         )
1004
1005         pd.DataFrame.from_dict(data=test_dict, orient="index").to_csv(
1006             "test_data.csv", header=True
1007         )
1008
1009         pd.DataFrame.from_dict(data=score_dict, orient="index").to_csv(
1010             "scores_output.csv", header=True
1011         )
1012
1013         pd.DataFrame.from_dict(data=prompt_dict, orient="index").to_csv(
1014             "prompt_output.csv", header=True
1015         )
1016
1017         message_string = "Predictions and Scores Saved at: " + str(os.getcwd())
1018         tk.messagebox.showinfo(title="Outcome", message=message_string)
1019     except:
1020         __stop_button_text.set("Exit Program")
1021         __sub_button_text.set("Submit and Run")
1022         __submit_btn["state"] = "normal"
1023         __stop_btn["state"] = "disabled"
1024         __timing_text_box.delete("1.0", "end")
1025         __timing_text_box.insert("end-1c", " ")
1026         __progress_text_box.delete("1.0", "end")
1027         __progress_text_box.insert("end-1c", " ")
1028         tk.messagebox.showerror(
1029             title="File Output",
1030             message="Output coercion to files failed, Make sure they are not in
1031             use.",
1032         )
1033         return
1034     else:
1035         pass
1036
1037 # Warn that errors occurred if Error Dict has lines in it
1038 if len(error_dict) > 0:
1039     try:
1040         pd.DataFrame.from_dict(data=error_dict, orient="index").to_csv(
1041             "error_output.csv", header=True
1042         )
1043     except:
1044         __stop_button_text.set("Exit Program")
1045         __sub_button_text.set("Submit and Run")
1046         __submit_btn["state"] = "normal"
1047         __stop_btn["state"] = "disabled"
1048         __timing_text_box.delete("1.0", "end")
1049         __timing_text_box.insert("end-1c", " ")
1050         __progress_text_box.delete("1.0", "end")
1051         __progress_text_box.insert("end-1c", " ")
1052         tk.messagebox.showerror(
1053             title="File Output",
1054             message="Output coercion to error_output.csv failed, Make sure it is
1055             not in use.",

```



```

1054         )
1055         return
1056
1057     else:
1058         pass
1059
1060     # Warn for Specific encountered Errors
1061     if (trend_error == 1) & (array_error == 1):
1062         message_string = (
1063             "Trend and Array Errors Encountered | Errors Saved at : "
1064             + "\n"
1065             + str(os.getcwd())
1066         )
1067
1068         tk.messagebox.showwarning(title="Error Warning", message=message_string)
1069     elif trend_error == 1:
1070         message_string = (
1071             "Trend Errors Encountered | Errors Saved at : " + "\n" + str(os.getcwd())
1072         )
1073
1074         tk.messagebox.showwarning(title="Error Warning", message=message_string)
1075
1076     elif array_error == 1:
1077         message_string = (
1078             "Array Errors Encountered | Errors Saved at : " + "\n" + str(os.getcwd())
1079         )
1080
1081         tk.messagebox.showwarning(title="Error Warning", message=message_string)
1082     else:
1083         pass
1084
1085     # Warn about failure if output is empty
1086     # Or Show final message
1087     if len(out_list) == 0:
1088
1089         __timing_text_box.delete(1.0, "end-1c")
1090         __timing_text_box.insert("end-1c", " ")
1091         __progress_text_box.delete(1.0, "end-1c")
1092         __progress_text_box.insert("end-1c", " ")
1093         tk.messagebox.showwarning(
1094             title="No Output",
1095             message="No combinations were valid so no predictions have been made",
1096         )
1097         pass
1098     else:
1099         __timediff2 = int(__timeD - __timeC)
1100         __timing_text_box.delete(1.0, "end-1c")
1101         __timing_text_box.insert(
1102             "end-1c", str("Total time elapsed = " + str(__timediff2) + " Seconds")
1103         )
1104
1105         __progress_text_box.delete(1.0, "end-1c")
1106         __progress_text_box.insert("end-1c", str("Models evaluated = " + str(counter)))
1107         __output_text_box.delete(1.0, "end-1c")
1108         __output_text_box.insert("end-1c", out_list)
1109         pass
1110
1111     # Reset GUI Buttons to original State for next run
1112     running = False
1113     __stop_button_text.set("Exit Program")
1114     __sub_button_text.set("Submit and Run")
1115     __submit_btn["state"] = "normal"
1116     __stop_btn["state"] = "disabled"
1117
1118
1119 # 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/hph99elpmvwj1/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

```

```

1186 sep_hor1 = Separator(__app, orient="horizontal")
1187 sep_hor1.grid(column=1, row=2, columnspan=8, sticky="ew")
1188
1189 # Header
1190 tk.Label(__app, text="Files and Directory", font=__header_fontstyle).grid(
1191     row=3, column=0, columnspan=7
1192 )
1193
1194 # Dataframe
1195 tk.Label(__app, text="Main Dataframe").grid(row=4, column=1, sticky=E)
1196 __browse_btn_1 = Button(textvariable=__browse_button_text, command=browse_button).grid(
1197     row=4, column=2, columnspan=4, sticky=EW
1198 )
1199
1200
1201 # Product_df
1202 tk.Label(__app, text="Product Dataframe").grid(row=5, column=1, sticky=E)
1203 __browse_btn_2 = Button(
1204     textvariable=__browse_button_text2, command=browse_button2
1205 ).grid(row=5, column=2, columnspan=4, sticky=EW)
1206
1207
1208 # Directory
1209 tk.Label(__app, text="Output Directory").grid(row=6, column=1, sticky=E)
1210 __browse_btn_3 = Button(
1211     textvariable=__browse_button_text3, command=browse_button3
1212 ).grid(row=6, column=2, columnspan=4, sticky=EW)
1213
1214
1215 # Padding
1216 __label02 = tk.Label(__app, text="")
1217 __label02.grid(column=0, row=7)
1218
1219 sep_hor2 = Separator(__app, orient="horizontal")
1220 sep_hor2.grid(column=1, row=8, columnspan=5, sticky="ew")
1221
1222 __label03 = tk.Label(__app, text="")
1223 __label03.grid(column=0, row=9)
1224
1225 # Header
1226 tk.Label(__app, text="General Inputs", font=__header_fontstyle).grid(
1227     row=9, column=0, columnspan=7
1228 )
1229
1230 # Item_id
1231 tk.Label(__app, text="Item_ID").grid(row=10, column=1, sticky=E)
1232 __item_id_inp = tk.Entry(__app)
1233 __item_id_inp.insert(END, "10")
1234 __item_id_inp.grid(row=10, column=2)
1235
1236 __submit_btn_4 = Button(__app, text="Check", width=10, command=check_item_id)
1237 __submit_btn_4.grid(row=10, column=4)
1238
1239 __item_id_inp_text_box = tk.Text(__app, width=15, height=1)
1240 __item_id_inp_text_box.grid(row=10, column=5, columnspan=1)
1241 __item_id_inp_text_box.insert("end-1c", " ")
1242
1243 # Site_id
1244 tk.Label(__app, text="Site_ID").grid(row=11, column=1, sticky=E)
1245 __site_id_inp = tk.Entry(__app)
1246 __site_id_inp.grid(row=11, column=2)
1247
1248 __submit_btn_5 = Button(__app, text="Check", width=10, command=check_site_id)
1249 __submit_btn_5.grid(row=11, column=4)
1250
1251 __site_id_text_box = tk.Text(__app, width=15, height=1)

```

```

1252 __site_id_text_box.grid(row=11, column=5, columnspan=1)
1253 __site_id_text_box.insert("end-1c", " ")
1254
1255 # Cleaning Std
1256 tk.Label(__app, text="Outlier Std.Dev.").grid(row=12, column=1, sticky=E)
1257 __std_dev_inp = tk.Entry(__app)
1258 __std_dev_inp.insert(END, "4")
1259 __std_dev_inp.grid(row=12, column=2)
1260
1261 __submit_btn_3 = Button(__app, text="Check", width=10, command=check_standard_dev)
1262 __submit_btn_3.grid(row=12, column=4)
1263
1264 __std_dev_inp_text_box = tk.Text(__app, width=15, height=1)
1265 __std_dev_inp_text_box.grid(row=12, column=5, columnspan=1)
1266 __std_dev_inp_text_box.insert("end-1c", " ")
1267
1268 # Time
1269 tk.Label(__app, text="Days to Predict").grid(row=13, column=1, sticky=E)
1270 __time_inp = tk.Entry(__app)
1271 __time_inp.insert(END, "10")
1272 __time_inp.grid(row=13, column=2)
1273
1274 __submit_btn_9 = Button(__app, text="Check", width=10, command=check_time)
1275 __submit_btn_9.grid(row=13, column=4)
1276
1277 __time_inp_text_box = tk.Text(__app, width=15, height=1)
1278 __time_inp_text_box.grid(row=13, column=5, columnspan=1)
1279 __time_inp_text_box.insert("end-1c", " ")
1280
1281
1282 # Padding
1283 __label04 = tk.Label(__app, text=" ")
1284 __label04.grid(column=0, row=14)
1285
1286 sep_hor3 = Separator(__app, orient="horizontal")
1287 sep_hor3.grid(column=1, row=15, columnspan=5, sticky="ew")
1288
1289 ver_hor3 = Separator(__app, orient="horizontal")
1290 ver_hor3.grid(column=3, row=17, rowspan=3, sticky="ns")
1291
1292 # Header
1293 tk.Label(__app, text="Parameter Inputs", font=__header_fontstyle).grid(
1294     row=16, column=0, columnspan=7
1295 )
1296
1297 # P
1298 tk.Label(__app, text="Starting p").grid(row=17, column=1, sticky=E)
1299 __p_inp = tk.Entry(__app)
1300 __p_inp.insert(END, "7")
1301 __p_inp.grid(row=17, column=2)
1302
1303
1304 tk.Label(__app, text="p Grid Mult").grid(row=17, column=4, sticky=E)
1305 __p_gs_list_inp = tk.Entry(__app)
1306 __p_gs_list_inp.insert(END, "0,1,2")
1307 __p_gs_list_inp.grid(row=17, column=5)
1308
1309 # d
1310 tk.Label(__app, text="Starting d").grid(row=18, column=1, sticky=E)
1311 __d_inp = tk.Entry(__app)
1312 __d_inp.insert(END, "0")
1313 __d_inp.grid(row=18, column=2)
1314
1315 tk.Label(__app, text="d Grid Sum").grid(row=18, column=4, sticky=E)
1316 __d_gs_list_inp = tk.Entry(__app)
1317 __d_gs_list_inp.insert(END, "0,1,2")

```

```

1318 __d_gs_list_inp.grid(row=18, column=5)
1319
1320 # q
1321 tk.Label(__app, text="Starting q").grid(row=19, column=1, sticky=E)
1322 __q_inp = tk.Entry(__app)
1323 __q_inp.insert(END, "1")
1324 __q_inp.grid(row=19, column=2)
1325
1326 tk.Label(__app, text="q Grid Mult").grid(row=19, column=4, sticky=E)
1327 __q_gs_list_inp = tk.Entry(__app)
1328 __q_gs_list_inp.insert(END, "0,1,2")
1329 __q_gs_list_inp.grid(row=19, column=5)
1330
1331 # Padding
1332 __label05 = tk.Label(__app, text=" ")
1333 __label05.grid(column=0, row=20)
1334
1335 sep_hor4 = Separator(__app, orient="horizontal")
1336 sep_hor4.grid(column=1, row=21, columnspan=5, sticky="ew")
1337
1338 # Header
1339 tk.Label(__app, text="Options", font=__header_fontstyle).grid(
1340     row=22, column=0, columnspan=7
1341 )
1342
1343 # Input
1344 __label4 = tk.Label(__app, text="Gridsearch All")
1345 __label4.grid(column=1, row=23, sticky=E)
1346
1347 __gridsearch_inp = ttk.Combobox(__app, values=["Yes", "No"])
1348 __gridsearch_inp.insert(END, "No")
1349 __gridsearch_inp.grid(column=2, row=23)
1350
1351 # Gridsearch 2
1352 __label5 = tk.Label(__app, text="Gridsearch d only")
1353 __label5.grid(column=4, row=23, sticky=E)
1354
1355 __gridsearch_d_inp = ttk.Combobox(__app, values=["Yes", "No"])
1356 __gridsearch_d_inp.insert(END, "Yes")
1357 __gridsearch_d_inp.grid(column=5, row=23)
1358
1359 # Save
1360 __label7 = tk.Label(__app, text="Save Outcome")
1361 __label7.grid(column=1, row=24, sticky=E)
1362
1363 __save_inp = ttk.Combobox(__app, values=["Yes", "No"])
1364 __save_inp.insert(END, "No")
1365 __save_inp.grid(column=2, row=24)
1366
1367 # Warnings
1368 __label8 = tk.Label(__app, text="Show Warnings")
1369 __label8.grid(column=4, row=24, sticky=E)
1370
1371 __warning_inp = ttk.Combobox(__app, values=["Yes", "No"])
1372 __warning_inp.insert(END, "Yes")
1373 __warning_inp.grid(column=5, row=24)
1374
1375 # Padding
1376 __label05 = tk.Label(__app, text=" ")
1377 __label05.grid(column=0, row=26, sticky=E)
1378
1379 sep_hor5 = Separator(__app, orient="horizontal")
1380 sep_hor5.grid(column=1, row=27, columnspan=5, sticky="ew")
1381
1382 ## Submit
1383 __label06 = tk.Label(__app, text=" ")

```

```

1384 __label06.grid(column=0, row=28, sticky=E)
1385
1386 __submit_btn = tk.Button(
1387     __app,
1388     textvariable=__sub_button_text,
1389     command=button_click,
1390     height=2,
1391     width=20,
1392     bg="#cc0000",
1393     fg="white",
1394 )
1395
1396 # __submit_btn.grid(row=29, column=2, columnspan=3, padx=10, pady=25)
1397 __submit_btn.grid(row=29, column=1, columnspan=2, padx=10, pady=25)
1398
1399 __stop_btn = tk.Button(
1400     __app,
1401     textvariable=__stop_button_text,
1402     command=stop,
1403     height=2,
1404     width=20,
1405     bg="#cc0000",
1406     fg="white",
1407     state=DISABLED,
1408 )
1409
1410 __stop_btn.grid(row=29, column=4, columnspan=2, padx=10, pady=25)
1411
1412
1413 # Padding
1414 __label07 = tk.Label(__app, text="")
1415 __label07.grid(column=1, row=30)
1416
1417 # Progress Boxes
1418 __timing_text_box = tk.Text(__app, height=1, width=30)
1419 __timing_text_box.tag_configure("center", justify="center")
1420 __timing_text_box.grid(row=31, column=1, columnspan=5, sticky="ew")
1421 __timing_text_box.insert("end-1c", " ")
1422
1423 __progress_text_box = tk.Text(__app, height=1, width=30)
1424 __progress_text_box.tag_configure("center", justify="center")
1425 __progress_text_box.grid(row=32, column=1, columnspan=5, sticky="ew")
1426 __progress_text_box.insert("end-1c", " ")
1427
1428 # Padding
1429 __label08 = tk.Label(__app, text="")
1430 __label08.grid(column=1, row=33)
1431
1432 __label09 = tk.Label(__app, text=" ")
1433 __label09.grid(column=9, row=4)
1434
1435 # Output Box
1436
1437 tk.Label(__app, text="Output", font=__header_fontstyle).grid(row=3, column=7)
1438
1439 __output_text_box = tk.Text(__app, width=50)
1440 __output_text_box.grid(row=4, column=7, rowspan=29, sticky="ns")
1441
1442
1443 yscroll = tk.Scrollbar(command=__output_text_box.yview, orient=tk.VERTICAL)
1444 yscroll.grid(row=4, column=8, rowspan=29, sticky="ns")
1445 __output_text_box.configure(yscrollcommand=yscroll.set)
1446
1447 sep_hor5 = Separator(__app, orient="horizontal")
1448 sep_hor5.grid(column=1, row=34, columnspan=8, sticky="ew")
1449

```

```
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
```