

```

1  from pandas import set_option,concat,DataFrame,Series,to_numeric,to_datetime,options
2  from numpy import concatenate,random,unique,round
3  from pyodbc import connect
4  from warnings import filterwarnings
5  from matplotlib.pyplot import subplots,tight_layout,savefig,close
6  from sklearn.metrics import
accuracy_score,make_scorer,confusion_matrix,f1_score,roc_curve,auc
7  from sklearn.metrics import
precision_recall_curve,precision_score,recall_score,roc_auc_score
8  from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
9  from sklearn.dummy import DummyClassifier
10 #from sklearn.utils import shuffle
11 from sklearn.tree import DecisionTreeClassifier#,plot_tree
12 from sklearn.model_selection import GridSearchCV#,cross_val_score,train_test_split
13 from sklearn.preprocessing import
StandardScaler,LabelBinarizer,label_binarize,MaxAbsScaler
14 #from sklearn.feature_extraction import FeatureHasher
15 from catboost import CatBoostClassifier
16 from xgboost import XGBClassifier
17 from lightgbm import LGBMClassifier
18 from IPython.display import clear_output
19 from sklearn.neighbors import *
20 from re import search#,match
21 from random import randint
22 from time import time
23 from smtplib import SMTP_SSL
24 from ssl import create_default_context
25 from os import path,walk
26 from zipfile import ZipFile, ZIP_DEFLATED
27 from time import time#,sleep
28
29 filterwarnings('ignore')
30 set_option('display.max_rows', None)
31 set_option('display.max_columns', None)
32
33 def full_score_report_multi_class(model, features, target, predictions,image_name):
34     # Calculate recall and precision for weighted average
35     recall = round(recall_score(target, predictions, average='weighted'), 3)
36     precision = round(precision_score(target, predictions, average='weighted'), 3)
37
38     # Binarize the targets for multiclass ROC and Precision-Recall curves
39     classes = unique(target)
40     target_binarized = label_binarize(target, classes=classes)
41     n_classes = len(classes)
42
43     # Get the predicted probabilities
44     probabilities = model.predict_proba(features)
45
46     # Initialize dictionaries for ROC and Precision-Recall
47     fpr = dict()
48     tpr = dict()
49     roc_auc = dict()
50     precision_curve = dict()
51     recall_curve = dict()
52     pr_auc = dict()
53
54     for i in range(n_classes):
55         fpr[i], tpr[i], _ = roc_curve(target_binarized[:, i], probabilities[:, i])
56         roc_auc[i] = auc(fpr[i], tpr[i])
57         precision_curve[i], recall_curve[i], _ =
precision_recall_curve(target_binarized[:, i], probabilities[:, i])
58         pr_auc[i] = auc(recall_curve[i], precision_curve[i])
59
60     # Plotting the ROC and Precision-Recall curves
61     fig, ax = subplots(1, 2, figsize=(14, 7))
62
63     # Plot ROC curve
64     for i in range(n_classes):
65         ax[0].plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC =

```

```

        {roc_auc[i]:.2f}))')
66 ax[0].plot([0, 1], [0, 1], 'r--')
67 ax[0].set_xlim([0.0, 1.0])
68 ax[0].set_ylim([0.0, 1.0])
69 ax[0].set_xlabel('False Positive Rate')
70 ax[0].set_ylabel('True Positive Rate')
71 ax[0].set_title('ROC Curve')
72 ax[0].legend(loc='lower right')
73
74 # Plot Precision-Recall curve
75 for i in range(n_classes):
76     ax[1].plot(recall_curve[i], precision_curve[i], lw=2, label=f'Class {classes[i]}
        (AUC = {pr_auc[i]:.2f}))')
77 ax[1].set_xlim([0.0, 1.0])
78 ax[1].set_ylim([0.0, 1.0])
79 ax[1].set_xlabel('Recall')
80 ax[1].set_ylabel('Precision')
81 ax[1].set_title('Precision-Recall Curve')
82 ax[1].legend(loc='lower left')
83
84 tight_layout()
85 savefig(f"{image_name}.png")
86 close()
87 # show()
88
89 def full_score_report_binary_class(model, features, target, predictions, image_name):
90     # Calculate recall and precision for weighted average
91     recall = round(recall_score(target, predictions, average='weighted'), 3)
92     precision = round(precision_score(target, predictions, average='weighted'), 3)
93
94     # Binarize the targets for multiclass ROC and Precision-Recall curves
95     classes = unique(target)
96     target_binarized = label_binarize(target, classes=classes)
97     n_classes = len(classes)
98
99     # Get the predicted probabilities
100    if hasattr(model, "predict_proba"):
101        probabilities = model.predict_proba(features)
102    else:
103        probabilities = model.decision_function(features)
104        probabilities = (probabilities - probabilities.min()) / (probabilities.max() -
        probabilities.min())
105
106    # Initialize dictionaries for ROC and Precision-Recall
107    fpr = dict()
108    tpr = dict()
109    roc_auc = dict()
110    precision_curve = dict()
111    recall_curve = dict()
112    pr_auc = dict()
113
114    if n_classes == 2:
115        # Handle binary classification
116        fpr[0], tpr[0], _ = roc_curve(target, probabilities[:, 1])
117        roc_auc[0] = auc(fpr[0], tpr[0])
118        precision_curve[0], recall_curve[0], _ = precision_recall_curve(target,
        probabilities[:, 1])
119        pr_auc[0] = auc(recall_curve[0], precision_curve[0])
120    else:
121        for i in range(n_classes):
122            fpr[i], tpr[i], _ = roc_curve(target_binarized[:, i], probabilities[:, i])
123            roc_auc[i] = auc(fpr[i], tpr[i])
124            precision_curve[i], recall_curve[i], _ =
        precision_recall_curve(target_binarized[:, i], probabilities[:, i])
125            pr_auc[i] = auc(recall_curve[i], precision_curve[i])
126
127    # Plotting the ROC and Precision-Recall curves
128    fig, ax = subplots(1, 2, figsize=(14, 7))
129

```

```

130     # Plot ROC curve
131     if n_classes == 2:
132         ax[0].plot(fpr[0], tpr[0], lw=2, label=f'Class {classes[1]} (AUC =
            {roc_auc[0]:.2f})')
133     else:
134         for i in range(n_classes):
135             ax[0].plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC =
                {roc_auc[i]:.2f})')
136     ax[0].plot([0, 1], [0, 1], 'r--')
137     ax[0].set_xlim([0.0, 1.0])
138     ax[0].set_ylim([0.0, 1.0])
139     ax[0].set_xlabel('False Positive Rate')
140     ax[0].set_ylabel('True Positive Rate')
141     ax[0].set_title('ROC Curve')
142     ax[0].legend(loc='lower right')
143
144     # Plot Precision-Recall curve
145     if n_classes == 2:
146         ax[1].plot(recall_curve[0], precision_curve[0], lw=2, label=f'Class {classes[1]}
            (AUC = {pr_auc[0]:.2f})')
147     else:
148         for i in range(n_classes):
149             ax[1].plot(recall_curve[i], precision_curve[i], lw=2, label=f'Class
                {classes[i]} (AUC = {pr_auc[i]:.2f})')
150     ax[1].set_xlim([0.0, 1.0])
151     ax[1].set_ylim([0.0, 1.0])
152     ax[1].set_xlabel('Recall')
153     ax[1].set_ylabel('Precision')
154     ax[1].set_title('Precision-Recall Curve')
155     ax[1].legend(loc='lower left')
156
157     tight_layout()
158     savefig(f"{image_name}.png")
159     close()
160     # show()
161
162 def get_scores(model, features, target, predictions):
163     accuracy = accuracy_score(target, predictions)
164     f1 = f1_score(target, predictions, average='weighted')
165     matrix = confusion_matrix(target, predictions)
166     recall = recall_score(target, predictions, average='weighted')
167     precision = precision_score(target, predictions, average='weighted')
168     if (target.nunique() > 1):
169         probabilities = model.predict_proba(features)
170         # auc_roc = roc_auc_score(target,
            probabilities, average='weighted', multi_class='ovr', labels=)
171         # Binarize the true labels
172         target_binarized = label_binarize(target, classes=[0,1,2,3])
173
174         # Calculate the ROC AUC score using the 'ovr' (one-vs-rest) strategy
175         roc_auc_ovr = roc_auc_score(target_binarized, probabilities, multi_class='ovr')
176
177         # Calculate the ROC AUC score using the 'ovo' (one-vs-one) strategy
178         roc_auc_ovo = roc_auc_score(target_binarized, probabilities, multi_class='ovo')
179         return [accuracy, recall, precision, f1, roc_auc_ovr, roc_auc_ovo, matrix]
180     else:
181         return [accuracy, recall, precision, f1, -1, -1, matrix]
182
183 def empty_string_to_null(string: str):
184     if (len(str(string)) == 0 or len(str(string).replace(' ', '')) == 0):
185         return "None"
186     return string
187
188 def object_to_int(original, lookup_table):
189     for row in lookup_table.index:
190         for col in lookup_table.columns:
191             if (original == lookup_table[col][row]):
192                 return int(col)
193

```

```

194 def get_knn(df, n, k, metric, feature_names):
195     """
196     Returns k nearest neighbors
197
198     :param df: pandas DataFrame used to find similar objects within
199     :param n: object no for which the nearest neighbours are looked for
200     :param k: the number of the nearest neighbours to return
201     :param metric: name of distance metric
202     """
203
204
205     nbrs = NearestNeighbors(n_neighbors=k, algorithm='ball_tree',
206                             metric=metric).fit(df[feature_names].to_numpy())
207     nbrs_distances, nbrs_indices = nbrs.kneighbors([df.iloc[n][feature_names]], k,
208                                                    return_distance=True)
209
210     df_res = concat([
211         df.iloc[nbrs_indices[0]],
212         DataFrame(nbrs_distances.T, index=nbrs_indices[0], columns=['distance'])
213     ], axis=1)
214
215     return df_res
216
217 def build_knc(random_state, train, target, test, n_neighbors):
218     random.seed(random_state)
219     knc = KNeighborsClassifier(n_neighbors=n_neighbors)
220     knc.fit(train, target)
221     y_pred = knc.predict(test)
222     return knc, y_pred
223
224 def random_int(min_val, max_val):
225     # Get the current time in microseconds
226     current_time = int(time() * 1000000)
227
228     # Use the current time as a seed and perform some operations to get more randomness
229     seed = (current_time ^ (current_time >> randint(1,20))) & 0xFFFFFFFF
230     seed = (seed ^ (seed << randint(1,20))) & 0xFFFFFFFF
231     seed = (seed ^ (seed >> randint(1,20))) & 0xFFFFFFFF
232
233     # Scale the seed to the desired range
234     random_value = min_val + (seed % (max_val - min_val + 1))
235
236     return random_value
237
238 def random_features(columns):
239     subset = []
240     columns = list(columns)
241     while(len(subset)<50):
242         random_index = random_int(0,len(list(columns))-1)
243         while(columns[random_index] in subset):
244             random_index = random_int(0,len(list(columns))-1)
245         subset.append(columns[random_index])
246     return subset
247
248 def send_email_to_self(subject: str, body: str):
249     port = 465 # For SSL
250     smtp_server = "smtp.gmail.com"
251     sender_email = "micpowers98@gmail.com" # Enter your address
252     receiver_email = "micpowers98@gmail.com" # Enter receiver address
253     password = 'efex cwhv gppq ueob'
254     message = "Subject: "+subject+"\n"+body
255
256     context = create_default_context()
257     with SMTP_SSL(smtp_server, port, context=context) as server:
258         server.login(sender_email, password)
259         server.sendmail(sender_email, receiver_email, message)
260
261 def get_train_data(columns: list):
262     cursor_1.execute("""

```

```

261 select
262     case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
263         'N' end as IsClaimFlg,
264         li.Checked,
265         li.[ATG_Ref],
266         li.[ATG_LI_Ref],
267         li.[BatchNbr],
268         li.[CATEGORY_ATG],
269         li.[ClaimType],
270         li.[AP_VndNbr],
271         li.[APVndrName],
272         li.[OOB_ATG],
273         li.[ItemNbr],
274         li.[UPCNbr],
275         li.[UPCUnit],
276         li.[ItemDescription],
277         li.[ItemShipPack],
278         li.[PoNbr],
279         li.[PODate],
280         li.[ReceivingDate],
281         li.[TurnRatio_ATG],
282         li.[TurnQty_ATG],
283         li.[OrdQty],
284         li.[PdQty_ATG],
285         li.[PdGross_ATG],
286         li.[PdOI_ATG],
287         li.[PdBB_ATG],
288         li.[PdNet_ATG],
289         li.[DealNbr],
290         li.[OrdStartDate_ATG],
291         li.[OrdEndDate_ATG],
292         li.[DateStartArrival_ATG],
293         li.[DateEndArrival_ATG],
294         li.[DLAmtOI],
295         li.[DLAmtBB]--,
296 from
297     [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
298     join [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
299         ds
300     on (li.BatchNbr = ds.BatchNbr)
301     and (li.DLVendorNbr = ds.DLVendorNbr)
302     and (li.DealNbr = ds.DealNbr)
303     and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
304     and (li.ClaimType = ds.ClaimType_ATG)
305 where
306     li.ClaimType = 'IN DEAL'
307     and
308     ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
309     and
310     li.checked='true'
311     and
312     ds.checked='claim'
313
314 union
315
316 select
317     case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
318         'N' end as IsClaimFlg, li.Checked,
319         li.[ATG_Ref],
320         li.[ATG_LI_Ref],
321         li.[BatchNbr],
322         li.[CATEGORY_ATG],
323         li.[ClaimType],
324         li.[AP_VndNbr],
325         li.[APVndrName],
326         li.[OOB_ATG],
327         li.[ItemNbr],
328         li.[UPCNbr],
329         li.[UPCUnit],

```

```

327         li.[ItemDescription],
328         li.[ItemShipPack],
329         li.[PoNbr],
330         li.[PODate],
331         li.[ReceivingDate],
332         li.[TurnRatio_ATG],
333         li.[TurnQty_ATG],
334         li.[OrdQty],
335         li.[PdQty_ATG],
336         li.[PdGross_ATG],
337         li.[PdOI_ATG],
338         li.[PdBB_ATG],
339         li.[PdNet_ATG],
340         li.[DealNbr],
341         li.[OrdStartDate_ATG],
342         li.[OrdEndDate_ATG],
343         li.[DateStartArrival_ATG],
344         li.[DateEndArrival_ATG],
345         li.[DLAmtOI],
346         li.[DLAmtBB]--,
347     from
348         [MOE].[prod>WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
349         join [MOE].[prod>WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
350             ds
351         on (li.BatchNbr = ds.BatchNbr)
352         and (li.DLVendorNbr = ds.DLVendorNbr)
353         and (li.DealNbr = ds.DealNbr)
354         and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
355         and (li.ClaimType = ds.ClaimType_ATG)
356     where
357         li.ClaimType = 'IN DEAL'
358         and
359         ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
360         and
361         li.checked='true'
362         and
363         ds.checked='x'
364
365     union
366
367     select top(30000)
368         case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
369             'N' end as IsClaimFlg, li.Checked,
370         li.[ATG_Ref],
371         li.[ATG_LI_Ref],
372         li.[BatchNbr],
373         li.[CATEGORY_ATG],
374         li.[ClaimType],
375         li.[AP_VndNbr],
376         li.[APVndrName],
377         li.[OOB_ATG],
378         li.[ItemNbr],
379         li.[UPCNbr],
380         li.[UPCUnit],
381         li.[ItemDescription],
382         li.[ItemShipPack],
383         li.[PoNbr],
384         li.[PODate],
385         li.[ReceivingDate],
386         li.[TurnRatio_ATG],
387         li.[TurnQty_ATG],
388         li.[OrdQty],
389         li.[PdQty_ATG],
390         li.[PdGross_ATG],
391         li.[PdOI_ATG],
392         li.[PdBB_ATG],
393         li.[PdNet_ATG],
394         li.[DealNbr],
395         li.[OrdStartDate_ATG],

```

```

394         li.[OrdEndDate_ATG],
395         li.[DateStartArrival_ATG],
396         li.[DateEndArrival_ATG],
397         li.[DLAmtOI],
398         li.[DLAmtBB]--,
399     from
400         [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
401     join     [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
402             ds
403     on (li.BatchNbr = ds.BatchNbr)
404     and (li.DLVendorNbr = ds.DLVendorNbr)
405     and (li.DealNbr = ds.DealNbr)
406     and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
407     and (li.ClaimType = ds.ClaimType_ATG)
408     where
409         li.ClaimType = 'IN DEAL'
410         and
411         ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
412         and
413         li.checked='x'
414         and
415         ds.checked='x'
416     """)
417 train_data = cursor_1.fetchall()
418 train_data_list = []
419 for index in range(len(train_data)):
420     train_data_list.append(list(train_data[index]))
421 del train_data
422 return DataFrame(data=train_data_list,columns=columns)
423
424 def get_test_data(batch_number: int, columns: list):
425     cursor_1.execute(f"""
426         select
427             case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
428             'N' end as IsClaimFlg, li.Checked,
429             li.[ATG_Ref],
430             li.[ATG_LI_Ref],
431             li.[BatchNbr],
432             li.[CATEGORY_ATG],
433             li.[ClaimType],
434             li.[AP_VndNbr],
435             li.[APVndrName],
436             li.[OOB_ATG],
437             li.[ItemNbr],
438             li.[UPCNbr],
439             li.[UPCUnit],
440             li.[ItemDescription],
441             li.[ItemShipPack],
442             li.[PoNbr],
443             li.[PODate],
444             li.[ReceivingDate],
445             li.[TurnRatio_ATG],
446             li.[TurnQty_ATG],
447             li.[OrdQty],
448             li.[PdQty_ATG],
449             li.[PdGross_ATG],
450             li.[PdOI_ATG],
451             li.[PdBB_ATG],
452             li.[PdNet_ATG],
453             li.[DealNbr],
454             li.[OrdStartDate_ATG],
455             li.[OrdEndDate_ATG],
456             li.[DateStartArrival_ATG],
457             li.[DateEndArrival_ATG],
458             li.[DLAmtOI],
459             li.[DLAmtBB]--,
460     from
461         [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
462     join     [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]

```

```

461         ds
462         on (li.BatchNbr = ds.BatchNbr)
463         and (li.DLVendorNbr = ds.DLVendorNbr)
464         and (li.DealNbr = ds.DealNbr)
465         and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
466         and (li.ClaimType = ds.ClaimType_ATG)
467     where
468         li.BatchNbr={batch_number}
469         and
470         li.ClaimType = 'IN DEAL'
471         and
472         ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
473         and
474         li.checked='x'
475         and
476         ds.checked='x'
477
478 union
479
480 select top(1)
481     case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
482     'N' end as IsClaimFlg, li.Checked,
483     li.[ATG_Ref],
484     li.[ATG_LI_Ref],
485     li.[BatchNbr],
486     li.[CATEGORY_ATG],
487     li.[ClaimType],
488     li.[AP_VndNbr],
489     li.[APVndrName],
490     li.[OOB_ATG],
491     li.[ItemNbr],
492     li.[UPCNbr],
493     li.[UPCUnit],
494     li.[ItemDescription],
495     li.[ItemShipPack],
496     li.[PoNbr],
497     li.[PODate],
498     li.[ReceivingDate],
499     li.[TurnRatio_ATG],
500     li.[TurnQty_ATG],
501     li.[OrdQty],
502     li.[PdQty_ATG],
503     li.[PdGross_ATG],
504     li.[PdOI_ATG],
505     li.[PdBB_ATG],
506     li.[PdNet_ATG],
507     li.[DealNbr],
508     li.[OrdStartDate_ATG],
509     li.[OrdEndDate_ATG],
510     li.[DateStartArrival_ATG],
511     li.[DateEndArrival_ATG],
512     li.[DLAmtOI],
513     li.[DLAmtBB]--,
514
515 from
516     [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
517 join    [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
518     ds
519     on (li.BatchNbr = ds.BatchNbr)
520     and (li.DLVendorNbr = ds.DLVendorNbr)
521     and (li.DealNbr = ds.DealNbr)
522     and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
523     and (li.ClaimType = ds.ClaimType_ATG)
524
525 where
526     li.ClaimType = 'IN DEAL'
527     and
528     ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
529     and
530     li.checked='true'
531     and

```



```

527         ds.checked='claim'
528     """)
529     test_data = cursor_1.fetchall()
530     test_data_list = []
531     for index in range(len(test_data)):
532         test_data_list.append(list(test_data[index]))
533     del test_data
534     return DataFrame(data=test_data_list, columns=columns)
535
536 def compress_folder_to_zip(folder_path, output_zip_file):
537     # Create a ZipFile object
538     with ZipFile(output_zip_file, 'w', ZIP_DEFLATED) as zipf:
539         # Walk through the folder
540         for root, dirs, files in walk(folder_path):
541             for file in files:
542                 # Create the full file path
543                 file_path = path.join(root, file)
544                 # Add file to zip, using relative path to maintain folder structure
545                 arcname = path.relpath(file_path, start=folder_path)
546                 zipf.write(file_path, arcname=arcname)
547
548 columns: list = ['S_DealNbr', 'S_ATG_IR', 'S_ClaimType_ATG', 'S_OrdStartDate_ATG',
549 'S_OrdEndDate_ATG', 'S_InDeal_Due', 'S_Shoulder_Due', 'S_LargeBuy_Due',
550 'S_TTL_OIDue', 'S_TTL_BBDue', 'S_NoBuy_Due', 'S_Facility', 'S_DLVendorNbr',
551 'S_MultiDeal_ATG', 'S_DealOrigin_ATG', 'S_AddDate',
552 'S_DealStatus_ATG', 'S_ClaimNumber', 'S_ClaimDate', 'S_ATG_IR_SourceFile',
553 'S_ClmBatchNbr_ATG', 'S_ItemCount', 'S_PA_Claimed', 'S_AR_AdjTyp',
554 'S_AR_Amount', 'S_AltTABLE2_ATG', 'S_DQ_Reason_ATG', 'S_QtyClaimed',
555 'S_QtyExample', 'S_LeadQTY', 'S_InDealQTY', 'S_PostQTY', 'S_SellUnitQty',
556 'S_UnitQtyOverSold_ATG', 'S_DaysBefore_ATG', 'S_DaysAfter_ATG',
557 'S_MultiVendor_ATG', 'S_DealVendorName', 'S_AP_VndNbr', 'S_PurVndrNbr',
558 'S_PurVndrNbr2', 'S_PurVndrName', 'S_PurVndrName2', 'S_PurVndCount_ATG',
559 'S_DateStartArrival_ATG', 'S_DateEndArrival_ATG', 'S_PromoStartDate_ATG',
560 'S_PromoEndDate_ATG', 'S_OCCURS', 'S_CATEGORY_ATG', 'S_ATG_ID', 'S_checked',
561 'S_BatchNbr', 'S_ATG_Ref', 'S_ClaimActivityID_ATGSYS',
562 'S_ClaimActivityCount_ATGSYS', 'E_BatchNbr', 'E_PODate', 'E_ReceivingDate',
563 'E_InvoicedDate', 'E_OIDue_ATG', 'E_BBDue_ATG', 'E_ClaimType',
564 'E_Shoulder_ATG', 'E_DtMatch_ATG', 'E_ItemNbr', 'E_ItemDescription',
565 'E_ItemShipPack', 'E_PoNbr', 'E_DealNbr', 'E_ATG_IR', 'E_OrdStartDate_ATG',
566 'E_OrdEndDate_ATG', 'E_DateStartArrival_ATG', 'E_DateEndArrival_ATG',
567 'E_AddDate', 'E_DLAmtOI', 'E_DLAmtBB', 'E_TurnRatio_ATG', 'E_OrdQty',
568 'E_PdQty_ATG', 'E_PdNetSB_ATG', 'E_PdNet_ATG', 'E_PdOI_ATG', 'E_PdBB_ATG',
569 'E_PdGross_ATG', 'E_ListcostAtStart', 'E_ListCostSB_ATG', 'E_BestOI_ATG',
570 'E_BestBB_ATG', 'E_DQ_Reason_ATG', 'E_ClaimedAmt', 'E_AR_AdjTyp',
571 'E_AR_Amount', 'E_AR_InvNbr', 'E_AR_InvNbr2', 'E_UPCNbr', 'E_UPCUnit',
572 'E_TurnQty_ATG', 'E_TruckSize_ATG', 'E_PdUpDn_ATG', 'E_BalFlag_ATG',
573 'E_OOB_ATG', 'E_MiscAdj_ATG', 'E_DLVendorNbr', 'E_DealVendorName',
574 'E_PurVndrNbr',
575 'E_PurVndrName', 'E_Facility', 'E_ReceiptNbr', 'E_ReceiptSfx',
576 'E_VndrInvNbr', 'E_TotWght', 'E_AP_VndNbr', 'E_APVndrName', 'E_BuyrNbr',
577 'E_BuyrName',
578 'E_Contact', 'E_AP_CheckNbr', 'E_AP_CheckDate', 'E_AP_GrossAmt',
579 'E_AP_DiscAmt', 'E_VendorCmnt', 'E_TTLVendorCmnt_ATG', 'E_PORemarks',
580 'E_MultiDeal_ATG', 'E_DealOrigin_ATG', 'E_AbsDays_ATG', 'E_ATG_Ref',
581 'E_ATG_DL_Ref', 'E_ATG_LI_Ref', 'E_ATG_HDR_Ref', 'E_DedBBInd', 'E_checked',
582 'E_CATEGORY_ATG', 'E_TrnCde_ATG', 'E_Dept', 'E_DLPctOI', 'E_DLPctBB',
583 'E_DLIncvPctOI', 'E_DLIncvPctBB', 'E_ClaimActivityID_ATGSYS',
584 'E_ClaimActivityCount_ATGSYS', 'E_ClaimActivityTypeID_ATGSYS']
585
586 columns: list = [
587     'IsClaimFlg',
588     'Checked',
589     'ATG_Ref',
590     'ATG_LI_Ref',
591     'BatchNbr',
592     'CATEGORY_ATG',
593     'ClaimType',
594     'AP_VndNbr',
595     'APVndrName',

```

```

577     'OOB_ATG',
578     'ItemNbr',
579     'UPCNbr',
580     'UPCUnit',
581     'ItemDescription',
582     'ItemShipPack',
583     'PoNbr',
584     'PODate',
585     'ReceivingDate',
586     'TurnRatio_ATG',
587     'TurnQty_ATG',
588     'OrdQty',
589     'PdQty_ATG',
590     'PdGross_ATG',
591     'PdOI_ATG',
592     'PdBB_ATG',
593     'PdNet_ATG',
594     'DealNbr',
595     'OrdStartDate_ATG',
596     'OrdEndDate_ATG',
597     'DateStartArrival_ATG',
598     'DateEndArrival_ATG',
599     'DLAmtOI',
600     'DLAmtBB'
601 ]
602
603 # try:
604 for batch_number in range(80,81):
605     start = time()
606     options.display.float_format = '{:.10f}'.format
607     server = 'troy'
608     database = 'prod_Costco_RecoverNow'
609     connectionString = f'DRIVER={{ODBC Driver 17 for SQL
Server}};SERVER={server};DATABASE={database};Integrated
Security={True};Autocommit={True};Trusted_Connection=yes;'
610     conn = connect(connectionString)
611     cursor_1 = conn.cursor()
612     train: DataFrame = get_train_data(columns=columns)
613     test: DataFrame = get_test_data(batch_number=batch_number,columns=columns)
614     subject = 'Data Collected'
615     body = f"""
616         {len(test):,} rows of data in batch {batch_number}.
617         """
618     send_email_to_self(subject,body)
619
620     empty_col = []
621     for col in test.columns:
622         if(test[col].dtype=='object'):
623             try:
624                 train[col] = train[col].astype(float)
625                 test[col] = test[col].astype(float)
626             except Exception as e:
627                 print(f"{str(e)}: {col}")
628     for col in train.columns:
629         if train[col].dtype == 'object':
630             if(train[col].nunique()>1000):
631                 empty_col.append(col)
632     train = train.drop(empty_col,axis=1)
633     test = test.drop(empty_col,axis=1)
634     for col in train.columns:
635         if train[col].dtype == 'object' and not(col=='checked'):
636             try:
637                 train[col] = train[col].astype(int)
638             except:
639                 try:
640                     train[col] = train[col].astype(float)
641                 except Exception as e: pass
642     for col in test.columns:
643         if test[col].dtype == 'object' and not(col=='checked'):

```

```

644         try:
645             test[col] = test[col].astype(int)
646         except:
647             try:
648                 test[col] = test[col].astype(float)
649             except Exception as e: pass
650     for col in train.columns:
651         if train[col].dtype == 'object':
652             train[col] = train[col].fillna('None')
653         elif train[col].dtype == 'int64':
654             train[col] = train[col].fillna(-1)
655         elif train[col].dtype == 'float64':
656             train[col] = train[col].fillna(-1)
657     for col in test.columns:
658         if test[col].dtype == 'object':
659             test[col] = test[col].fillna('None')
660         elif test[col].dtype == 'int64':
661             test[col] = test[col].fillna(-1)
662         elif test[col].dtype == 'float64':
663             test[col] = test[col].fillna(-1)
664     for col in train.columns:
665         if train[col].dtype == 'object' and not(col=='checked'):
666             try:
667                 train[col] = train[col].astype(int)
668             except:
669                 try:
670                     train[col] = train[col].astype(float)
671                 except Exception as e: pass
672     for col in test.columns:
673         if test[col].dtype == 'object' and not(col=='checked'):
674             try:
675                 test[col] = test[col].astype(int)
676             except:
677                 try:
678                     test[col] = test[col].astype(float)
679                 except Exception as e: pass
680     category_values = {}
681     category_new_values = {}
682     for col in train.columns:
683         if train[col].dtype == 'object' and train[col].nunique() <= 1000:
684             train[col] = train[col].apply(empty_string_to_null)
685             test[col] = test[col].apply(empty_string_to_null)
686             # categorical_columns.append(col)
687             category_values[col] =
                Series(list(train[col].unique()) + list(test[col].unique())).unique()
688     max_length = 0
689     for key, item in category_values.items():
690         if len(item) > max_length:
691             max_length = len(item)
692     for key in category_values:
693         category_values[key] = concatenate((category_values[key], ['None'] * (max_length
        - len(category_values[key]))))
694     lookup_table = DataFrame.from_dict(category_values, orient='index')
695     lookup_table.columns = range(max_length)
696     for col in lookup_table.columns:
697         lookup_table[col] = lookup_table[col].astype(str)
698
699     lookup_table.to_csv(f'C:/Code/Python/Machine_Learning_AI/Lookup_Table_Batch_{batch_number}.csv')
700     del lookup_table
701     subject = 'Lookup Table Created'
702     body = f"""
703         Starting long category_new_values section.
704         """
705     #send_email_to_self(subject, body)
706
707     category_new_values = {}
708     count = 0
709     for key, item in category_values.items():

```

```

709         clear_output(wait=False)
710         # print(f"Categories Read: {count}\nCategories Left:
711         {len(category_values)-count}")
712         count += 1
713         for new_value in item:
714             try:
715                 category_new_values[f"{key} : {new_value}"] =
716                 list(Series(list(train[str(key)].unique())+list(test[str(key)].unique()))
717                 .unique()).index(new_value)
718             except:
719                 category_new_values[f"{key} : {new_value}"] =
720                 len(list(Series(list(train[str(key)].unique())+list(test[str(key)].unique
721                 ())).unique()))
722             break
723 del category_values
724 count = 0
725 for key,item in category_new_values.items():
726     full_value = f"{key} : {item}"
727     # print(f"Categories Read: {count}\nCategories Left:
728     {len(category_new_values)-count}\n{full_value}\n")
729     count += 1
730     col = search(r"[A-Za-z\_\\s]{1,}:",full_value).group()
731     col = full_value.split(':')[0]
732     # print(f"{col[:-1]}")
733     col_value =
734     search(":.+:",full_value).group()#search(r":[A-Za-z0-9\_\\s\\-\\/\\<\\>\\=\\'\\.\\+\\,\\&]{1
735     ,}:",full_value).group()
736     col_value = full_value.split(':')[1]
737     # print(col_value[1:-1])
738     new_value = search(r": [0-9]{1,}",full_value).group().replace('
739     ',',').replace(':',',')
740     # print(new_value)
741     # print()
742     train[col[:-1]] = train[col[:-1]].replace({col_value[1:-1]: new_value})
743     test[col[:-1]] = test[col[:-1]].replace({col_value[1:-1]: new_value})
744 del category_new_values
745 subject = 'New categories completed'
746 body = f"""
747     About to prepare for machine learning testing.
748     """
749 #send_email_to_self(subject,body)
750 for col in train.columns:
751     if train[col].dtype == 'object':# and not(col=='checked'):
752         try:
753             train[col] = train[col].astype(int)
754         except Exception as e:pass
755         # train = DataFrame(train.drop(col,axis=1))
756         # test = DataFrame(test.drop(col,axis=1))
757         # print(f"{col}: {str(e)}")
758 for col in test.columns:
759     if test[col].dtype == 'object':# and not(col=='checked'):
760         try:
761             test[col] = test[col].astype(int)
762         except Exception as e:pass
763         # train = DataFrame(train.drop(col,axis=1))
764         # test = DataFrame(test.drop(col,axis=1))
765         # print(f"{col}: {str(e)}")
766 datetime_columns = []
767 for col in train.columns:
768     if train[col].dtype == 'datetime64[ns]':
769         datetime_columns.append(col)
770 for col in datetime_columns:
771     train[col] = to_numeric(train[col])
772 datetime_columns = []
773 for col in test.columns:
774     if test[col].dtype == 'datetime64[ns]':
775         datetime_columns.append(col)
776 for col in datetime_columns:
777     test[col] = to_numeric(test[col])

```

```

769 del datetime_columns
770
771 random_state = randint(1,4294967295)
772 model_scores =
DataFrame(None,columns=['Accuracy','Recall','Precision','F1','ROC_OVR','ROC_OVO'])
773 accuracy_scorer = make_scorer(f1_score)
774 options.display.float_format = '{:.10f}'.format
775 additional_excluded_columns =
['S_ATG_Ref','E_ATG_Ref','S_ClaimDate','S_BatchNbr','E_BatchNbr','E_ATG_HDR_Ref','S_P
urVndrName','E_PurVndrName','E_PODate',
776
'E_Dept','S_Facility','E_Facility','E_AP_CheckDate','E_UP
CUnit','E_UPCNbr','S_ClmBatchNbr_ATG','S_ATG_IR','E_ATG_I
R',
777
'S_ClaimActivityCount_ATGSYS','E_ClaimActivityTypeID_ATGS
YS','E_ATG_DL_Ref','S_ATG_ID','E_ATG_LI_Ref','E_ClaimActi
vityID_ATGSYS',
778
'E_ReceiptNbr','E_DQ_Reason_ATG','S_DealNbr','S_ClaimType
_ATG',
779
'E_ReceivingDate','E_ClaimActivityCount_ATGSYS','E_Invoic
edDate','E_PoNbr','E_ClaimedAmt','S_QtyClaimed','S_Should
er_Due']
780 additional_excluded_columns = [
781     'IsClaimFlg',
782     'Checked',
783     'ATG_Ref',
784     'ATG_LI_Ref',
785     'BatchNbr',
786     'CATEGORY_ATG',
787     'ClaimType',
788     'AP_VndNbr',
789     'APVndrName',
790     'OOB_ATG',
791     'ItemNbr',
792     'UPCNbr',
793     'UPCUnit',
794     'ItemDescription',
795     'ItemShipPack',
796     'PoNbr',
797     'PODate',
798     'ReceivingDate',
799     'TurnRatio_ATG',
800     'DealNbr',
801     'OrdStartDate_ATG',
802     'OrdEndDate_ATG',
803     'DateStartArrival_ATG',
804     'DateEndArrival_ATG',
805     'DLAmtOI',
806     'DLAmtBB'
807 ]
808 #additional_excluded_columns.extend(empty_col)
809 unique_excluded_columns = []
810 for col in additional_excluded_columns:
811     if col in unique_excluded_columns or not(col in list(test.columns)):
812         pass
813     else:
814         unique_excluded_columns.append(col)
815 additional_excluded_columns = unique_excluded_columns.copy()
816 del unique_excluded_columns
817 excluded_features = test[additional_excluded_columns]
818
819 scaler = StandardScaler()
820 scaler.fit(train.drop(['Checked','IsClaimFlg']+additional_excluded_columns,axis=1))
821 features_train_scaled =
DataFrame(scaler.transform(train.drop(['Checked','IsClaimFlg']+additional_excluded_co
lumn, axis=1)), columns=list(train.drop(['Checked','IsClaimFlg']+additional_excluded_c

```

```

columns,axis=1).columns))
822 target_train_E = train['Checked']
823 target_train_S = train['IsClaimFlg']
824 features_test_scaled =
DataFrame(scaler.transform(test.drop(['Checked','IsClaimFlg']+additional_excluded_col
umns,axis=1)),columns=list(test.drop(['Checked','IsClaimFlg']+additional_excluded_col
umns,axis=1).columns))
825 target_test_E = test['Checked']
826 target_test_S = test['IsClaimFlg']
827
828 subject = 'Starting Dummy Classifier'
829 body = f"""
830 """
831 #send_email_to_self(subject,body)
832 dc_e = DummyClassifier(random_state=random_state,strategy='most_frequent')
833 dc_e.fit(features_train_scaled,target_train_E)
834 dc_predictions = dc_e.predict(features_test_scaled)
835
full_score_report_binary_class(dc_e,features_test_scaled,target_test_E,dc_predictions
,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Dummy_Exceptions_ROC_Re
call_Precision_Curves')
836 dc_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,dc_predictions,labels=[0,1]),columns=['Predi
cted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
837 model_scores.loc['Dummy_Exceptions'] =
get_scores(dc_e,features_test_scaled,target_test_E,dc_predictions)[:1]
838 dc_e_importances = DataFrame([[1/len(features_train_scaled.columns) for _ in
range(len(features_train_scaled.columns))]],columns=list(features_train_scaled.column
s),index=['Dummy_Exceptions']).T
839
840 dc_s = DummyClassifier(random_state=random_state,strategy='most_frequent')
841 dc_s.fit(features_train_scaled,target_train_S)
842 dc_predictions = dc_s.predict(features_test_scaled)
843
full_score_report_binary_class(dc_s,features_test_scaled,target_test_S,dc_predictions
,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Dummy_Summary_ROC_Recall_
Precision_Curves')
844 dc_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S,dc_predictions,labels=[0,1]),columns=['Predi
cted_x','Predicted_claim'],index=['Actual_x','Actual_claim'])
845 model_scores.loc['Dummy_Summary'] =
get_scores(dc_s,features_test_scaled,target_test_S,dc_predictions)[:1]
846 dc_s_importances = DataFrame([[1/len(features_train_scaled.columns) for _ in
range(len(features_train_scaled.columns))]],columns=list(features_train_scaled.column
s),index=['Dummy_Summary']).T
847
848 subject = 'Starting DecisionTreeClassifier Exceptions'
849 body = f"""
850 """
851 #send_email_to_self(subject,body)
852 dt_e_parameters = {
853     'random_state':[random_state],
854     'max_depth':[2,3,4,5,6,7,8],
855     'splitter':['best','random']
856 }
857 dt_e = DecisionTreeClassifier(random_state=random_state,max_depth=None)
858 dt_e
=GridSearchCV(DecisionTreeClassifier(),dt_e_parameters,verbose=10,cv=5,refit=True,err
or_score='raise',return_train_score=True)
859 dt_e.fit(features_train_scaled,target_train_E)
860 dt_e_best = DecisionTreeClassifier(**dt_e.best_params_)
861 dt_e_best.fit(features_train_scaled,target_train_E)
862 dt_e_predictions = dt_e.predict(features_test_scaled)
863
full_score_report_binary_class(dt_e,features_test_scaled,target_test_E,dt_e_predictio
ns,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Decision_Tree_Except
ions_ROC_Recall_Precision_Curves')
864 dt_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,dt_e_predictions,labels=[0,1]),columns=['Pre

```

```

dicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
865 model_scores.loc['Decision_Tree_Exceptions'] =
get_scores(dt_e, features_test_scaled, target_test_E, dt_e_predictions)[: -1]
866 dt_e_importances =
DataFrame([dt_e_best.feature_importances_], columns=features_train_scaled.columns, index=['Decision_Tree_Exceptions']).T

867
868 subject = 'Starting DecisionTreeClassifier Summary'
869 body = f"""
870 """
871 #send_email_to_self(subject, body)
872 dt_s_parameters = {
873     'random_state': [random_state],
874     'max_depth': [2, 3, 4, 5, 6, 7, 8],
875     'splitter': ['best', 'random']
876 }
877 dt_s = DecisionTreeClassifier(random_state=random_state, max_depth=None)
878 dt_s
=GridSearchCV(DecisionTreeClassifier(), dt_s_parameters, verbose=10, cv=5, refit=True, error_score='raise', return_train_score=True)
879 dt_s.fit(features_train_scaled, target_train_S)
880 dt_s_best = DecisionTreeClassifier(**dt_s.best_params_)
881 dt_s_best.fit(features_train_scaled, target_train_S)
882 dt_s_predictions = dt_e.predict(features_test_scaled)
883 if(target_test_S.nunique() > 1):
884
full_score_report_binary_class(dt_s, features_test_scaled, target_test_S, dt_s_predictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Decision_Tree_Summary_ROC_Recall_Precision_Curves')
885 dt_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S, dt_s_predictions, labels=[0, 1]), columns=['Predicted_x', 'Predicted_claim'], index=['Actual_x', 'Actual_claim'])
886 model_scores.loc['Decision_Tree_Summary'] =
get_scores(dt_s, features_test_scaled, target_test_S, dt_s_predictions)[: -1]
887 dt_s_importances =
DataFrame([dt_s_best.feature_importances_], columns=features_train_scaled.columns, index=['Decision_Tree_Summary']).T

888
889 subject = 'Starting RandomForestClassifier Exceptions'
890 body = f"""
891 """
892 #send_email_to_self(subject, body)
893 rf_e_parameters = {
894     'random_state': [random_state],
895     'max_depth': [5], # [2, 3, 5],
896     'n_estimators': [50], # [50, 100, 150, 200],
897     'max_features': [None],
898     'warm_start': [True]
899 }
900 rf_e =
RandomForestClassifier(n_estimators=500, random_state=random_state, warm_start=True, max_depth=None, verbose=10)
901 rf_e =
GridSearchCV(RandomForestClassifier(), rf_e_parameters, verbose=10, cv=2, refit=True, error_score='raise', return_train_score=True)
902 rf_e.fit(features_train_scaled, target_train_E)
903 rf_e_best = RandomForestClassifier(**rf_e.best_params_)
904 rf_e_best.fit(features_train_scaled, target_train_E)
905 rf_e_predictions = rf_e.predict(features_test_scaled)
906
full_score_report_binary_class(rf_e_best, features_test_scaled, target_test_E, rf_e_predictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Random_Forest_Exceptions_ROC_Recall_Precision_Curves')
907 rf_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E, rf_e_predictions, labels=[0, 1]), columns=['Predicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
908 model_scores.loc['Random_Forest_Exceptions'] =
get_scores(rf_e_best, features_test_scaled, target_test_E, rf_e_predictions)[: -1]
909 rf_e_importances =

```

```

DataFrame([rf_e_best.feature_importances_], columns=features_train_scaled.columns, index=['Random_Forest_Exceptions']).T
910
911 subject = 'Starting RandomForestClassifier Summary'
912 body = f"""
913 """
914 #send_email_to_self(subject,body)
915 rf_s_parameters = {
916     'random_state':[random_state],
917     'max_depth':[5],#[2,3,5],
918     'n_estimators':[50],#[50,100,150,200],
919     'max_features':[None],
920     'warm_start':[True]
921 }
922 rf_s =
RandomForestClassifier(n_estimators=500,random_state=random_state,warm_start=True,max
_depth=None,verbose=10)
923 rf_s =
GridSearchCV(RandomForestClassifier(),rf_s_parameters,verbose=10,cv=2,refit=True,error
_score='raise',return_train_score=True)
924 rf_s.fit(features_train_scaled,target_train_S)
925 rf_s_best = RandomForestClassifier(**rf_s.best_params_)
926 rf_s_best.fit(features_train_scaled,target_train_S)
927 rf_s_predictions = rf_s.predict(features_test_scaled)
928
full_score_report_binary_class(rf_s_best,features_test_scaled,target_test_S,rf_s_pred
ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Random_Forest_Summ
ary_ROC_Recall_Precision_Curves')
929 rf_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S,rf_s_predictions,labels=[0,1]),columns=['Pre
dicted_x','Predicted_claim'],index=['Actual_x','Actual_claim'])
930 model_scores.loc['Random_Forest_Summary'] =
get_scores(rf_s_best,features_test_scaled,target_test_S,rf_s_predictions)[:1]
931 rf_s_importances =
DataFrame([rf_s_best.feature_importances_],columns=features_train_scaled.columns,inde
x=['Random_Forest_Summary']).T
932
933 subject = 'Starting Gradient Boost Exceptions'
934 body = f"""
935 """
936 #send_email_to_self(subject,body)
937 gb_e_parameters = {
938     'random_state':[random_state],
939     'n_estimators':[50,100]#[150,200,250]
940 }
941 gb_e =
GridSearchCV(GradientBoostingClassifier(),gb_e_parameters,verbose=10,cv=3,refit=True,
error_score='raise',return_train_score=True)
942 gb_e.fit(features_train_scaled,target_train_E)
943 gb_e_best = GradientBoostingClassifier(**gb_e.best_params_)
944 gb_e_best.fit(features_train_scaled,target_train_E)
945 gb_e_predictions = gb_e_best.predict(features_test_scaled)
946
full_score_report_binary_class(gb_e_best,features_test_scaled,target_test_E,gb_e_pred
ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/GradientBoost_E
xceptions_ROC_Recall_Precision_Curves')
947 gb_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,gb_e_predictions,labels=[0,1]),columns=['Pre
dicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
948 model_scores.loc['GradientBoost_Exceptions'] =
get_scores(gb_e_best,features_test_scaled,target_test_E,gb_e_predictions)[:1]
949 gb_e_importances =
DataFrame([gb_e_best.feature_importances_],columns=features_train_scaled.columns,inde
x=['GradientBoost_Exceptions']).T
950
951 subject = 'Starting Gradient Boost Summary'
952 body = f"""
953 """
954 #send_email_to_self(subject,body)

```



```

955 gb_s_parameters = {
956     'random_state':[random_state],
957     'n_estimators':[50,100]#,150,200,250]
958 }
959 gb_s =
GridSearchCV(GradientBoostingClassifier(),gb_s_parameters,verbose=10,cv=3,refit=True,
error_score='raise',return_train_score=True)
960 gb_s.fit(features_train_scaled,target_train_E)
961 gb_s_best = GradientBoostingClassifier(**gb_s.best_params_)
962 gb_s_best.fit(features_train_scaled,target_train_E)
963 gb_s_predictions = gb_s_best.predict(features_test_scaled)
964
full_score_report_binary_class(gb_s_best,features_test_scaled,target_test_E,gb_s_pred
ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/GradientBoost_S
ummary_ROC_Recall_Precision_Curves')
965 gb_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,gb_s_predictions,labels=[0,1]),columns=['Pre
dicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
966 model_scores.loc['GradientBoost_Summary'] =
get_scores(gb_s_best,features_test_scaled,target_test_E,gb_s_predictions)[:1]
967 gb_s_importances =
DataFrame([gb_s_best.feature_importances_],columns=features_train_scaled.columns,inde
x=['GradientBoost_Summary']).T
968
969 subject = 'Starting LGBM Exceptions'
970 body = f"""
971 """
972 #send_email_to_self(subject,body)
973 lgbm_e_parameters = {
974     'random_state':[random_state],
975     'n_estimators':[50,100]#,150,200,250]
976 }
977 lgbm_e = LGBMClassifier(verbosity=3,n_estimators=200)
978 lgbm_e =
GridSearchCV(LGBMClassifier(),lgbm_e_parameters,verbose=10,cv=3,refit=True,error_scor
e='raise',return_train_score=True)
979 lgbm_e.fit(features_train_scaled,target_train_E)
980 lgbm_e_best = LGBMClassifier(**lgbm_e.best_params_)
981 lgbm_e_best.fit(features_train_scaled,target_train_E)
982 lgbm_e_predictions = lgbm_e_best.predict(features_test_scaled)
983
full_score_report_binary_class(lgbm_e_best,features_test_scaled,target_test_E,lgbm_e
predictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/LGBM_Except
ions_ROC_Recall_Precision_Curves')
984 lgbm_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,lgbm_e_predictions,labels=[0,1]),columns=['P
redicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
985 model_scores.loc['LGBM_Exceptions'] =
get_scores(lgbm_e_best,features_test_scaled,target_test_E,lgbm_e_predictions)[:1]
986 lgbm_e_importances =
DataFrame([lgbm_e_best.feature_importances_],columns=features_train_scaled.columns,in
dex=['LGBM_Exceptions']).T
987 lgbm_e_importances['LGBM_Exceptions'] = lgbm_e_importances['LGBM_Exceptions']/3000
988
989 subject = 'Starting LGBM Summary'
990 body = f"""
991 """
992 #send_email_to_self(subject,body)
993 lgbm_s_parameters = {
994     'random_state':[random_state],
995     'n_estimators':[50,100]#,150,200,250]
996 }
997 lgbm_s = LGBMClassifier(verbosity=3,n_estimators=200)
998 lgbm_s =
GridSearchCV(LGBMClassifier(),lgbm_s_parameters,verbose=10,cv=3,refit=True,error_scor
e='raise',return_train_score=True)
999 lgbm_s.fit(features_train_scaled,target_train_E)
1000 lgbm_s_best = LGBMClassifier(**lgbm_s.best_params_)
1001 lgbm_s_best.fit(features_train_scaled,target_train_E)

```

```

1002     lgbm_s_predictions = lgbm_s_best.predict(features_test_scaled)
1003
1004     full_score_report_binary_class(lgbm_s_best, features_test_scaled, target_test_E, lgbm_s_
    predictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/lgbm_sxception
    s_ROC_Recall_Precision_Curves')
1005     lgbm_s_confusion_matrix =
    DataFrame(confusion_matrix(target_test_E, lgbm_s_predictions, labels=[0,1]), columns=['P
    redicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
1006     model_scores.loc['LGBM_Summary'] =
    get_scores(lgbm_s_best, features_test_scaled, target_test_E, lgbm_s_predictions)[: -1]
1007     lgbm_s_importances =
    DataFrame([lgbm_s_best.feature_importances_], columns=features_train_scaled.columns, in
    dex=['LGBM_Summary']).T
1008     lgbm_s_importances['LGBM_Summary'] = lgbm_s_importances['LGBM_Summary']/3000
1009
1010     subject = 'Starting XGBoost Exceptions'
1011     body = f"""
1012         """
1013     #send_email_to_self(subject, body)
1014     xgb_e_parameters = {
1015         'random_state': [random_state],
1016         'n_estimators': [50, 100, 150, 200, 250]
1017     }
1018     xgb_e = XGBClassifier(verbosity=3, n_estimators=200)
1019     xgb_e =
    GridSearchCV(XGBClassifier(), xgb_e_parameters, verbose=10, cv=5, refit=True, error_score=
    'raise', return_train_score=True)
1020     xgb_e.fit(features_train_scaled, target_train_E)
1021     xgb_e_best = XGBClassifier(**xgb_e.best_params_)
1022     xgb_e_best.fit(features_train_scaled, target_train_E)
1023     xgb_e_predictions = xgb_e_best.predict(features_test_scaled)
1024
1025     full_score_report_binary_class(xgb_e_best, features_test_scaled, target_test_E, xgb_e_pr
    edictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/XGBoost_Excep
    tions_ROC_Recall_Precision_Curves')
1026     xgb_e_confusion_matrix =
    DataFrame(confusion_matrix(target_test_E, xgb_e_predictions, labels=[0,1]), columns=['Pr
    edicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
1027     model_scores.loc['XGBoost_Exceptions'] =
    get_scores(xgb_e_best, features_test_scaled, target_test_E, xgb_e_predictions)[: -1]
1028     xgb_e_importances =
    DataFrame([xgb_e_best.feature_importances_], columns=features_train_scaled.columns, ind
    ex=['XGBoost_Exceptions']).T
1029
1030     subject = 'Starting XGBoost Summary'
1031     body = f"""
1032         """
1033     #send_email_to_self(subject, body)
1034     xgb_s_parameters = {
1035         'random_state': [random_state],
1036         'n_estimators': [50, 100, 150, 200, 250]
1037     }
1038     xgb_s = XGBClassifier(verbosity=3, n_estimators=200)
1039     xgb_s =
    GridSearchCV(XGBClassifier(), xgb_s_parameters, verbose=10, cv=5, refit=True, error_score=
    'raise', return_train_score=True)
1040     xgb_s.fit(features_train_scaled, target_train_S)
1041     xgb_s_best = XGBClassifier(**xgb_s.best_params_)
1042     xgb_s_best.fit(features_train_scaled, target_train_S)
1043     xgb_s_predictions = xgb_s_best.predict(features_test_scaled)
1044
1045     full_score_report_binary_class(xgb_s_best, features_test_scaled, target_test_S, xgb_s_pr
    edictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/XGBoostBoost_Sum
    mary_ROC_Recall_Precision_Curves')
1046     xgb_s_confusion_matrix =
    DataFrame(confusion_matrix(target_test_S, xgb_s_predictions, labels=[0,1]), columns=['Pr
    edicted_x', 'Predicted_claim'], index=['Actual_x', 'Actual_claim'])
1047     model_scores.loc['XGBoost_Summary'] =
    get_scores(xgb_s_best, features_test_scaled, target_test_S, xgb_s_predictions)[: -1]

```

```

1045 xgb_s_importances =
DataFrame([xgb_s_best.feature_importances_], columns=features_train_scaled.columns, index=['XGBoost_Summary']).T

1046
1047 subject = 'Starting CatBoost Exceptions'
1048 body = f"""
1049 """
1050 #send_email_to_self(subject,body)
1051 cb_e_parameters = {
1052     'iterations':[1500],#[500,750,1000,1500],
1053     'random_state':[random_state],
1054     'learning_rate':[0.005],#[0.005,0.0075,0.01],
1055     'depth':[5],#[2,5],
1056     'verbose':[0],
1057     'early_stopping_rounds':[3]#[3,5,10]
1058 }
1059 cb_e =
CatBoostClassifier(iterations=10000, learning_rate=0.0075, random_state=random_state, depth=7, verbose=10)
1060 cb_e =
GridSearchCV(CatBoostClassifier(), param_grid=cb_e_parameters, verbose=10, refit=True, cv=2, error_score='raise', return_train_score=True)
1061 cb_e.fit(features_train_scaled, target_train_E)
1062 cb_e_best = CatBoostClassifier(**cb_e.best_params_)
1063 cb_e_best.fit(features_train_scaled, target_train_E)
1064 cb_e_predictions = cb_e_best.predict(features_test_scaled)
1065
full_score_report_binary_class(cb_e_best, features_test_scaled, target_test_E, cb_e_predictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Cat_Boost_Exceptions_ROC_Recall_Precision_Curves')
1066 cb_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E, cb_e_predictions, labels=[0,1]), columns=['Predicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
1067 model_scores.loc['Cat_Boost_Exceptions'] =
get_scores(cb_e_best, features_test_scaled, target_test_E, cb_e_predictions)[-1]
1068 cb_e_importances =
DataFrame([cb_e_best.feature_importances_], columns=features_train_scaled.columns, index=['Cat_Boost_Exceptions']).T
1069 cb_e_importances['Cat_Boost_Exceptions'] =
cb_e_importances['Cat_Boost_Exceptions']/100
1070
1071 subject = 'Starting CatBoost Summary'
1072 body = f"""
1073 """
1074 #send_email_to_self(subject,body)
1075 cb_s_parameters = {
1076     'iterations':[1500],#[500,750,1000,1500],
1077     'random_state':[random_state],
1078     'learning_rate':[0.005],#[0.005,0.0075,0.01],
1079     'depth':[4],#[2,5],
1080     'verbose':[0],
1081     'early_stopping_rounds':[5]#[3,5,10]
1082 }
1083 cb_s =
CatBoostClassifier(iterations=10000, learning_rate=0.0075, random_state=random_state, depth=7, verbose=10)
1084 cb_s =
GridSearchCV(CatBoostClassifier(), param_grid=cb_s_parameters, verbose=10, refit=True, cv=2, error_score='raise', return_train_score=True)
1085 cb_s.fit(features_train_scaled, target_train_S)
1086 cb_s_best = CatBoostClassifier(**cb_s.best_params_)
1087 cb_s_best.fit(features_train_scaled, target_train_S)
1088 cb_s_predictions = cb_s_best.predict(features_test_scaled)
1089
full_score_report_binary_class(cb_s_best, features_test_scaled, target_test_S, cb_s_predictions, 'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Cat_Boost_Summary_ROC_Recall_Precision_Curves')
1090 cb_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S, cb_s_predictions, labels=[0,1]), columns=['Pre

```

```

dicted_x', 'Predicted_claim'], index=['Actual_x', 'Actual_claim'])
1091 model_scores.loc['Cat_Boost_Summary'] =
get_scores(cb_s_best, features_test_scaled, target_test_S, cb_s_predictions)[: -1]
1092 cb_s_importances =
DataFrame([cb_s_best.feature_importances_], columns=features_train_scaled.columns, index=['Cat_Boost_Summary']).T
1093 cb_s_importances['Cat_Boost_Summary'] = cb_s_importances['Cat_Boost_Summary']/100
1094
1095 subject = 'Starting K-NearestNeighbors'
1096 body = f"""
1097 """
1098 #send_email_to_self(subject, body)
1099 knnc_e, knnc_e_predictions =
build_knc(random_state=random_state, train=features_train_scaled, target=target_train_E,
n_neighbors=3, test=features_test_scaled)
1100
full_score_report_binary_class(model=knnc_e, features=features_test_scaled, target=target_test_E,
predictions=knnc_e_predictions, image_name='C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/KNN_Exceptions_ROC_Recall_Precision_Curves')
1101 knnc_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E, knnc_e_predictions, labels=[0, 1]), columns=['Predicted_x', 'Predicted_TRUE'], index=['Actual_x', 'Actual_TRUE'])
1102 model_scores.loc['K_Neighbors_Exceptions'] =
get_scores(knnc_e, features_test_scaled, target_test_E, knnc_e_predictions)[: -1]
1103
1104 knnc_s, knnc_s_predictions =
build_knc(random_state=random_state, train=features_train_scaled, target=target_train_S,
n_neighbors=3, test=features_test_scaled)
1105
full_score_report_binary_class(model=knnc_s, features=features_test_scaled, target=target_test_S,
predictions=knnc_s_predictions, image_name='C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/KNN_Summary_ROC_Recall_Precision_Curves')
1106 knnc_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S, knnc_s_predictions, labels=[0, 1]), columns=['Predicted_x', 'Predicted_claim'], index=['Actual_x', 'Actual_claim'])
1107 model_scores.loc['K_Neighbors_Summary'] =
get_scores(knnc_s, features_test_scaled, target_test_S, knnc_s_predictions)[: -1]
1108
1109 final_result: DataFrame = get_test_data(batch_number=batch_number, columns=columns)
1110 final_result['DecisionTree_Model_Checked_Predictions'] =
Series(dt_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1111 final_result['RandomForest_Model_Checked_Predictions'] =
Series(rf_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1112 final_result['LGBM_Model_Checked_Predictions'] =
Series(lgbm_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1113 final_result['GradientBoost_Model_Checked_Predictions'] =
Series(gb_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1114 final_result['XGBoost_Model_Checked_Predictions'] =
Series(xgb_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1115 final_result['CatBoost_Model_Checked_Predictions'] =
Series(cb_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1116 final_result['KNeighbors_Model_Checked_Predictions'] =
Series(knnc_e_predictions).reset_index(drop=True).replace({0: 'x', 1: 'TRUE'})
1117
1118 final_result.to_csv('C:/Code/Python/Machine_Learning_AI/Data_With_Model_Predictions.csv', index=False)
1119
1120 print("DONE STOP STOP")
1121
1122 subject = 'Analyzing Models'
1123 body = f"""
1124 """
1125 #send_email_to_self(subject, body)
1126
model_scores.iloc[[0, 2, 4, 6, 8, 10]].to_csv('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/All_Model_Scores_Exceptions.csv')
1127 combined_importances_e =
concat([dc_e_importances, dt_e_importances, rf_e_importances, gb_e_importances, lgbm_e_im

```

```

portances,xgb_e_importances,cb_e_importances],axis=1)
1128 combined_importances_e['SUM_Exceptions'] =
combined_importances_e['Dummy_Exceptions']+combined_importances_e['Decision_Tree_Exce
ptions']+combined_importances_e['Random_Forest_Exceptions']+combined_importances_e['G
radientBoost_Exceptions']+combined_importances_e['LGBM_Exceptions']+combined_importan
ces_e['XGBoost_Exceptions']+combined_importances_e['Cat_Boost_Exceptions']
1129
combined_importances_e.to_csv('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exce
ptions/Feature_Importance_Exceptions_All_Models.csv')
1130 confusion_e_matrices = [dc_e_confusion_matrix, dt_e_confusion_matrix,
rf_e_confusion_matrix, gb_e_confusion_matrix, lgbm_e_confusion_matrix,
xgb_e_confusion_matrix, cb_e_confusion_matrix, knn_e_confusion_matrix]
1131 titles_e = ['Dummy_Exceptions', 'Decision_Tree_Exceptions',
'Random_Forest_Exceptions', 'GradientBoost_Exceptions',
1132 'LGBM_Exceptions', 'XGBoost_Exceptions',
'Cat_Boost_Exceptions','K_Neighbors_Exceptions']
1133 table_e_rows = [len(tbl) for tbl in confusion_e_matrices]
1134 # Create a figure and a set of subplots
1135 fig_e, axs_e = subplots(ncols=1, nrows=8,gridspec_kw={'height_ratios':
table_e_rows},figsize=(6,10))
1136 for ax, dfs, title in zip(axs_e, confusion_e_matrices, titles_e):
1137     # Hide the axes
1138     # ax.axis('tight')
1139     df_with_index_e = dfs.copy()
1140     df_with_index_e.insert(0, '', dfs.index)
1141     ax.axis('off')
1142     # Create the table
1143     table = ax.table(cellText=df_with_index_e.values,
collLabels=df_with_index_e.columns, cellLoc='center', loc='center')
1144     # Add title
1145     ax.set_title(title)
1146 tight_layout()
1147
savefig('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Confused_Matric
es_Exceptions.pdf',format='pdf')
1148
savefig('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Exceptions/Con
fused_Matrices_Exceptions.pdf',format='pdf')
1149 close()
1150
1151 confusion_s_matrices = [dc_s_confusion_matrix, dt_s_confusion_matrix,
rf_s_confusion_matrix, gb_s_confusion_matrix, lgbm_s_confusion_matrix,
xgb_s_confusion_matrix, cb_s_confusion_matrix, knn_s_confusion_matrix]
1152 titles_s = ['Dummy_Summary', 'Decision_Tree_Summary', 'Random_Forest_Summary',
'GradientBoost_Summary',
1153 'LGBM_Summary', 'XGBoost_Summary',
'Cat_Boost_Summary','K_Neighbors_Summary']
1154 table_s_rows = [len(tbl) for tbl in confusion_s_matrices]
1155
model_scores.iloc[[1,3,5,7,9,11]].to_csv('C:/Code/Python/Machine_Learning_AI/Model_An
alysis/Summary/All_Model_Scores_Summary.csv')
1156 combined_importances_s =
concat([dc_s_importances,dt_s_importances,rf_s_importances,gb_s_importances,lgbm_s_im
portances,xgb_s_importances,cb_s_importances],axis=1)
1157 combined_importances_s['SUM_Summary'] =
combined_importances_s['Dummy_Summary']+combined_importances_s['Decision_Tree_Summary
']+combined_importances_s['Random_Forest_Summary']+combined_importances_s['GradientBo
ost_Summary']+combined_importances_s['LGBM_Summary']+combined_importances_s['XGBoost
_Summary']+combined_importances_s['Cat_Boost_Summary']
1158
combined_importances_s.to_csv('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summ
ary/Feature_Importance_Summary_All_Models.csv')
1159 fig_s, axs_s = subplots(ncols=1, nrows=8,gridspec_kw={'height_ratios':
table_s_rows},figsize=(6,10))
1160 for ax, dfs, title in zip(axs_s, confusion_s_matrices, titles_s):
1161     # Hide the axes
1162     # ax.axis('tight')
1163     df_with_index_s = dfs.copy()
1164     df_with_index_s.insert(0, '', dfs.index)

```

```

1165         ax.axis('off')
1166         # Create the table
1167         table = ax.table(cellText=df_with_index_s.values,
1168                        collabels=df_with_index_s.columns, cellLoc='center', loc='center')
1169         # Add title
1170         ax.set_title(title)
1171         tight_layout()

1172     savefig('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Confused_Matrices_Summary.pdf',format='pdf')

1173     savefig('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Summary/Confused_Matrices_Summary.pdf',format='pdf')
1174     close()
1175     subject = 'Analysis Complete'
1176     body = f"""\
1177         Summary Scores:\n{model_scores.iloc[[1,3,5,7,9,11]]}\n
1178         Exceptions Scores:\n{model_scores.iloc[[0,2,4,6,8,10]]}
1179     """
1180     #send_email_to_self(subject,body)

1181     subject = 'Reconstructing Original Data'
1182     body = f"""\
1183     """
1184     #send_email_to_self(subject,body)
1185     options.display.float_format = '{:.2f}'.format
1186     features_test_with_excluded_features =
1187     DataFrame(features_test_scaled.copy(),columns=features_test_scaled.columns)
1188     features_test_with_excluded_features =
1189     DataFrame(round(scaler.inverse_transform(features_test_with_excluded_features),2).astype(float),columns=features_test_scaled.columns)
1190     features_test_with_excluded_features[list(excluded_features.columns)] =
1191     excluded_features.loc[features_test_scaled.index]
1192     if True:
1193         df_with_discrepancies =
1194         DataFrame(features_test_with_excluded_features,columns=features_test_with_excluded_features.columns).reset_index(drop=True)
1195         df_with_discrepancies['E_checked'] =
1196         Series(target_test_E).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1197         df_with_discrepancies['DecisionTree_E_checked'] =
1198         Series(dt_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1199         df_with_discrepancies['RandomForest_E_checked'] =
1200         Series(rf_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1201         df_with_discrepancies['GradientBoost_E_checked'] =
1202         Series(gb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1203         df_with_discrepancies['LGBM_E_checked'] =
1204         Series(lgbm_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1205         df_with_discrepancies['XGBoost_E_checked'] =
1206         Series(xgb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1207         df_with_discrepancies['CatBoost_E_checked'] =
1208         Series(cb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1209         df_with_discrepancies['KNeighbors_E_checked'] =
1210         Series(knn_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1211         df_with_discrepancies['E_OrdStartDate_ATG'] =
1212         to_datetime(df_with_discrepancies['E_OrdStartDate_ATG'])
1213         df_with_discrepancies['E_OrdEndDate_ATG'] =
1214         to_datetime(df_with_discrepancies['E_OrdEndDate_ATG'])
1215         df_with_discrepancies['E_ReceivingDate'] =
1216         to_datetime(df_with_discrepancies['E_ReceivingDate'])
1217         df_with_discrepancies['E_InvoicedDate'] =
1218         to_datetime(df_with_discrepancies['E_InvoicedDate'])
1219         df_with_discrepancies['E_DateStartArrival_ATG'] =

```

```

1204     to_datetime(df_with_discrepancies['E_DateStartArrival_ATG'])
1205     df_with_discrepancies['E_DateEndArrival_ATG'] =
1206     to_datetime(df_with_discrepancies['E_DateEndArrival_ATG'])
1207     df_with_discrepancies['E_AddDate'] =
1208     to_datetime(df_with_discrepancies['E_AddDate'])
1209     df_with_discrepancies['E_PODate'] =
1210     to_datetime(df_with_discrepancies['E_PODate'])
1211     df_with_discrepancies['E_AP_CheckDate'] =
1212     to_datetime(df_with_discrepancies['E_AP_CheckDate'])
1213     df_with_discrepancies['E_Dept'] = df_with_discrepancies['E_Dept'].astype(int)
1214 if True:
1215     df_with_discrepancies['S_checked'] =
1216     Series(target_test_E).reset_index(drop=True).replace({0:'x',1:'claim'})
1217     df_with_discrepancies['DecisionTree_S_checked'] =
1218     Series(dt_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
1219     im'})
1220     df_with_discrepancies['RandomForest_S_checked'] =
1221     Series(rf_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
1222     im'})
1223     df_with_discrepancies['GradientBoost_S_checked'] =
1224     Series(gb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
1225     im'})
1226     df_with_discrepancies['LGBM_S_checked'] =
1227     Series(lgbm_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'c
1228     laim'})
1229     df_with_discrepancies['XGBoost_S_checked'] =
1230     Series(xgb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cl
1231     aim'})
1232     df_with_discrepancies['CatBoost_S_checked'] =
1233     Series(cb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
1234     im'})
1235     df_with_discrepancies['KNeighbors_S_checked'] =
1236     Series(knn_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cl
1237     aim'})
1238     df_with_discrepancies['S_OrdStartDate_ATG'] =
1239     to_datetime(df_with_discrepancies['S_OrdStartDate_ATG'])
1240     df_with_discrepancies['S_OrdEndDate_ATG'] =
1241     to_datetime(df_with_discrepancies['S_OrdEndDate_ATG'])
1242     df_with_discrepancies['S_DateStartArrival_ATG'] =
1243     to_datetime(df_with_discrepancies['S_DateStartArrival_ATG'])
1244     df_with_discrepancies['S_DateEndArrival_ATG'] =
1245     to_datetime(df_with_discrepancies['S_DateEndArrival_ATG'])
1246     df_with_discrepancies['S_AddDate'] =
1247     to_datetime(df_with_discrepancies['S_AddDate'])
1248     df_with_discrepancies['S_ClaimDate'] =
1249     to_datetime(df_with_discrepancies['S_ClaimDate'])
1250     df_with_discrepancies['S_PromoStartDate_ATG'] =
1251     to_datetime(df_with_discrepancies['S_PromoStartDate_ATG'])
1252     df_with_discrepancies['S_PromoEndDate_ATG'] =
1253     to_datetime(df_with_discrepancies['S_PromoEndDate_ATG'])
1254
1255 df_with_discrepancies[~(df_with_discrepancies['E_checked']==df_with_discrepancies['Ca
1256 tBoost_E_checked']) |
1257
1258     ~(df_with_discrepancies['S_checked']==df_with_discrepancies['CatB
1259 oost_S_checked'])]].head()
1260
1261 subject = 'Posting Discrepancies'
1262 body = f"""
1263     """
1264 #send_email_to_self(subject,body)
1265
1266 df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepanc
1267 ies['CatBoost_E_checked'].values) |
1268
1269     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
1270 s['XGBoost_E_checked'].values) |
1271
1272     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie

```

```

1236         s['LGBM_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1237     s['GradientBoost_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1238     s['RandomForest_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1239     s['DecisionTree_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1240     s['KNeighbors_E_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1241     s['CatBoost_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1242     s['XGBoost_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1243     s['LGBM_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1244     s['GradientBoost_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1245     s['RandomForest_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1246     s['DecisionTree_S_checked'].values) |
~(df_with_discrepancies['S_checked'].values==df_with_discrepancies[
1247     s['KNeighbors_S_checked'].values)
1248     ] [
['S_ATG_Ref', 'E_ATG_Ref', 'E_checked', 'DecisionTree_E_checked', 'RandomForest_E_checked',
1249     'GradientBoost_E_checked', 'LGBM_E_checked',
'S_XGBoost_E_checked', 'CatBoost_E_checked', 'KNeighbors_E_checked',
1250     'S_checked', 'DecisionTree_S_checked', 'RandomForest_S_checked', 'GradientBoost_S_checked',
'LGBM_S_checked',
1251     'XGBoost_S_checked', 'CatBoost_S_checked', 'KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/ATG_Refs_Only/Discrepancies_Between_Original_All_Checked_and_Model_Predictions.csv', index=False)
1252 df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1253     s['CatBoost_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1254     s['XGBoost_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1255     s['RandomForest_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1256     s['DecisionTree_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1257     s['LGBM_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1258     s['GradientBoost_E_checked'].values) |
~(df_with_discrepancies['E_checked'].values==df_with_discrepancies[
1259     s['KNeighbors_E_checked'].values)
1260     ] [

```



```

1261     ['S_ATG_Ref', 'E_ATG_Ref', 'E_checked', 'DecisionTree_E_checked', 'RandomForest_E_checked', 'GradientBoost_E_checked', 'LGBM_E_checked',
1262     'XGBoost_E_checked', 'CatBoost_E_checked', 'KNeighbors_E_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/ATG_Refs_Only/Exceptions/Discrepancies_Between_Original_E_Checked_and_Model_Predictions.csv', index=False)
1263
1264     df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values) |
1265     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values) |
1266     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values) |
1267     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values) |
1268     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['LGBM_S_checked'].values) |
1269     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['GradientBoost_S_checked'].values) |
1270     ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)
1271     ] [
1272     ['S_ATG_Ref', 'E_ATG_Ref', 'S_checked', 'DecisionTree_S_checked', 'RandomForest_S_checked', 'GradientBoost_S_checked', 'LGBM_S_checked',
1273     'XGBoost_S_checked', 'CatBoost_S_checked', 'KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/ATG_Refs_Only/Summary/Discrepancies_Between_Original_S_Checked_and_Model_Predictions.csv', index=False)
1274
1275     df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['CatBoost_E_checked'].values) |
1276     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['XGBoost_E_checked'].values) |
1277     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['RandomForest_E_checked'].values) |
1278     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['DecisionTree_E_checked'].values) |
1279     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['LGBM_E_checked'].values) |
1280     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['GradientBoost_E_checked'].values) |
1281     ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)
1282     ] [
1283     list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions', ascending=False).index[:15])+[
1284     'E_checked', 'DecisionTree_E_checked', 'RandomForest_E_checked',
1285     'XGBoost_E_checked', 'CatBoost_E_checked', 'KNeighbors_E_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_Features_Only/Exceptions/Discrepancies_Between_Original_E_Checked_and_Model_Predictions.csv', index=False)

```

```
df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values)]
```

```
1284         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values)|
1285         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values)|
1286         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values)#|
1287         #~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)
1288     ]|
1289     list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary',ascending=False).index[:15])+[
1290         'S_checked','DecisionTree_S_checked','RandomForest_S_checked','GradientBoost_E_checked','LGBM_E_checked',
1291         'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_Features_Only/Summary/Discrepancies_Between_Original_S_Checked_and_Model_Predictions.csv',index=False)
1292 df_with_discrepancies[(df_with_discrepancies['E_checked']=='x')&
1293         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['CatBoost_E_checked'].values)|
1294         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['XGBoost_E_checked'].values)|
1295         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['RandomForest_E_checked'].values)|
1296         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['DecisionTree_E_checked'].values)|
1297         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)]|
1298     list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',ascending=False).index[:15])+[
1299         'E_checked','DecisionTree_E_checked','RandomForest_E_checked','GradientBoost_E_checked','LGBM_E_checked',
1300         'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_Features_Only/Exceptions/x_Discrepancies_Between_Original_E_Checked_and_Model_Predictions.csv',index=False)
1301 df_with_discrepancies[(df_with_discrepancies['S_checked']=='x')&
1302         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values)|
1303         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values)|
1304         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values)|
1305         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values)|
1306         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)]|
```

```

1307         list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary',asc
1308         ending=False).index[:15])+[
1309         'S_checked','DecisionTree_S_checked','RandomForest_S_checked',
1310         'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
1311         eatures_Only/Summary/x_Discrepancies_Between_Original_S_Checked_and_Model_Predict
1312         ions.csv',index=False)
1313
1314 df_with_discrepancies[(df_with_discrepancies['E_checked']=='x')&(df_with_discrepancie
1315 s['S_checked']=='x')
1316
1317         &
1318         (
1319
1320             ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1321             ncies['CatBoost_E_checked'].values)|
1322
1323             ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1324             ncies['XGBoost_E_checked'].values)|
1325
1326             ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1327             ncies['RandomForest_E_checked'].values)|
1328
1329             ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1330             ncies['DecisionTree_E_checked'].values)|
1331
1332             ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1333             ncies['KNeighbors_E_checked'].values)
1334         )
1335         &
1336         (
1337
1338             ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
1339             ncies['CatBoost_S_checked'].values)|
1340
1341             ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
1342             ncies['XGBoost_S_checked'].values)|
1343
1344             ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
1345             ncies['RandomForest_S_checked'].values)|
1346
1347             ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
1348             ncies['DecisionTree_S_checked'].values)|
1349
1350             ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
1351             ncies['KNeighbors_S_checked'].values)
1352         )][
1353
1354         list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',
1355         ascending=False).index[:15])+[
1356         'E_checked','DecisionTree_E_checked','RandomForest_E_checked',
1357         'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked',
1358         'S_checked','DecisionTree_S_checked','RandomForest_S_checked',
1359         'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
1360         eatures_Only/x_Discrepancies_Between_Original_BOTH_Checked_and_Model_Predictions.
1361         csv',index=False)
1362 df_with_discrepancies[(df_with_discrepancies['S_checked']=='claim')&
1363
1364         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
1365         s['CatBoost_S_checked'].values)|
1366
1367         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
1368         s['XGBoost_S_checked'].values)|
1369
1370         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
1371         s['RandomForest_S_checked'].values)|

```

```

1336         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values) |
1337
1338         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)] [
1339
1340     list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary', ascending=False).index[:15])+[
1341     'S_checked', 'DecisionTree_S_checked', 'RandomForest_S_checked',
1342
1343     'XGBoost_S_checked', 'CatBoost_S_checked', 'KNeighbors_S_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_Features_Only/Summary/Claim_Discrepancies_Between_Original_S_Checked_and_Model_Predictions.csv', index=False)
1344
1345 df_with_discrepancies[(df_with_discrepancies['E_checked']=='TRUE') &
1346
1347         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['CatBoost_E_checked'].values) |
1348
1349         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['XGBoost_E_checked'].values) |
1350
1351         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['RandomForest_E_checked'].values) |
1352
1353         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['DecisionTree_E_checked'].values) |
1354
1355         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)] [
1356
1357     list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions', ascending=False).index[:15])+[
1358     'E_checked', 'DecisionTree_E_checked', 'RandomForest_E_checked',
1359
1360     'XGBoost_E_checked', 'CatBoost_E_checked', 'KNeighbors_E_checked']].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_Features_Only/Exceptions/True_Discrepancies_Between_Original_E_Checked_and_Model_Predictions.csv', index=False)
1361
1362 df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['CatBoost_E_checked'].values) |
1363
1364         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['XGBoost_E_checked'].values) |
1365
1366         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['RandomForest_E_checked'].values) |
1367
1368         ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['DecisionTree_E_checked'].values) |
1369
1370         #~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)
1371
1372     ].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Exceptions/Discrepancies_Between_Original_E_Checked_and_Model_Predictions_ALL_FEATURES.csv', index=False)
1373
1374 df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values) |
1375
1376         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values) |
1377
1378         ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values) |

```

```

1359 ~ (df_with_discrepancies['S_checked'].values==df_with_discrepanci
1360 s['DecisionTree_S_checked'].values)#|
1361
1362 #~ (df_with_discrepancies['S_checked'].values==df_with_discrepanci
1363 es['KNeighbors_S_checked'].values)
1364
1365 ].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_
AI/Discrepancies/All_Features/Summary/Discrepancies_Between_Origina
l_S_Checked_and_Model_Predictions_ALL_FEATURES.csv',index=False
)
1366
1367 server = 'barney'
1368 database = 'sandbox_mp'
1369 connectionString = f'DRIVER={{ODBC Driver 17 for SQL
Server}};SERVER={server};DATABASE={database};Integrated
Security={True};Autocommit={True};Trusted_Connection=yes;'
1370 conn = connect(connectionString)
1371 cursor_2 = conn.cursor()
1372 insert_sql = """
1373     INSERT INTO Weis_Market_Claim_Discrepancies_With_Context_1 (
1374         ATG_Deal_Summary_ATG_Ref,
1375         ATG_DealLI_Exceptions_ATG_Ref,
1376         ATG_DealLI_Exceptions_DL_ATG_Ref,
1377         ATG_DealLI_Exceptions_LI_ATG_Ref,
1378         ATG_DealLI_Exceptions_HDR_ATG_Ref,
1379         ATG_Deal_Summary_BatchNbr,
1380         ATG_DealLI_Exceptions_BatchNbr,
1381         ATG_Deal_Summary_DealNbr,
1382         ATG_Deal_Summary_OrdStartDate_ATG,
1383         ATG_Deal_Summary_OrdeNDDate_ATG,
1384         ATG_Deal_Summary_DLVendorNbr,
1385         ATG_Deal_Summary_AddDate,
1386         ATG_Deal_Summary_ClaimType,
1387         ATG_Deal_Summary_DealVendorName,
1388         ATG_Deal_Summary_AP_VndNbr,
1389         ATG_Deal_Summary_PurVndrNbr,
1390         ATG_Deal_Summary_PurVndrName,
1391         ATG_DealLI_Exceptions_ItemNbr,
1392         ATG_DealLI_Exceptions_PODate,
1393         ATG_DealLI_Exceptions_ReceivingDate,
1394         ATG_DealLI_Exceptions_InvoicedDate,
1395         ATG_DealLI_Exceptions_ClaimType,
1396         ATG_DealLI_Exceptions_UPCNbr,
1397         ATG_DealLI_Exceptions_UPCUnit,
1398         ATG_DealLI_Exceptions_PurVndrName,
1399         ATG_DealLI_Exceptions_ReceiptNbr,
1400         ATG_DealLI_Exceptions_Checked_Before_AI,
1401         DecisionTree_Model_Predictions_ATG_DealLI_Exceptions_checked,
1402         RandomForest_Model_Predictions_ATG_DealLI_Exceptions_checked,
1403         GradientBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
1404         LGBM_Model_Predictions_ATG_DealLI_Exceptions_checked,
1405         XGBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
1406         CatBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
1407         KNeighbors_Model_Predictions_ATG_DealLI_Exceptions_checked,
1408         ATG_Deal_Summary_Checked_Before_AI,
1409         DecisionTree_Model_Predictions_ATG_Deal_Summary_checked,
1410         RandomForest_Model_Predictions_ATG_Deal_Summary_checked,
1411         GradientBoost_Model_Predictions_ATG_Deal_Summary_checked,
1412         LGBM_Model_Predictions_ATG_Deal_Summary_checked,
1413         XGBoost_Model_Predictions_ATG_Deal_Summary_checked,
1414         CatBoost_Model_Predictions_ATG_Deal_Summary_checked,
1415         KNeighbors_Model_Predictions_ATG_Deal_Summary_checked
1416     ) VALUES
1417     (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,
1418     ?,?,?,?,?,?);
1419 """
1420 df_with_discrepancies[
1421     (

```

```

1416         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1417         repancies['CatBoost_E_checked'].values) &
1418
1419         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1420         repancies['XGBoost_E_checked'].values) &
1421
1422         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1423         repancies['LGBM_E_checked'].values) &
1424
1425         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1426         repancies['GradientBoost_E_checked'].values) &
1427
1428         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1429         repancies['RandomForest_E_checked'].values) &
1430
1431         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1432         repancies['DecisionTree_E_checked'].values) &
1433
1434         ~(df_with_discrepancies['E_checked'].values==df_with_disc
1435         repancies['KNeighbors_E_checked'].values)
1436     )
1437 |
1438 (
1439     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1440     repancies['CatBoost_S_checked'].values) &
1441
1442     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1443     repancies['XGBoost_S_checked'].values) &
1444
1445     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1446     repancies['LGBM_S_checked'].values) &
1447
1448     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1449     repancies['GradientBoost_S_checked'].values) &
1450
1451     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1452     repancies['RandomForest_S_checked'].values) &
1453
1454     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1455     repancies['DecisionTree_S_checked'].values) &
1456
1457     ~(df_with_discrepancies['S_checked'].values==df_with_disc
1458     repancies['KNeighbors_S_checked'].values)
1459 )
1460
1461     ].to_csv('C:/Code/Python/Machine_Learning_AI/TESTING_DISCREPANCIE
1462     S.csv')
1463 data = df_with_discrepancies[(
1464
1465     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1466     ncies['CatBoost_E_checked'].values) &
1467
1468     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1469     ncies['XGBoost_E_checked'].values) &
1470
1471     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1472     ncies['LGBM_E_checked'].values) &
1473
1474     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1475     ncies['GradientBoost_E_checked'].values) &
1476
1477     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1478     ncies['RandomForest_E_checked'].values) &
1479
1480     ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
1481     ncies['DecisionTree_E_checked'].values) &
1482

```

```

1443         ~ (df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)
1444     )
1445     |
1446     (
1447         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values) &
1448         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values) &
1449         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['LGBM_S_checked'].values) &
1450         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['GradientBoost_S_checked'].values) &
1451         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values) &
1452         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values) &
1453         ~ (df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)
1454     )
1455 ] [
1456     ['S_ATG_Ref', 'E_ATG_Ref', 'E_ATG_DL_Ref', 'E_ATG_LI_Ref', 'E_ATG_HDR_Ref', 'S_BatchNbr', 'E_BatchNbr',
1457     'S_DealNbr', 'S_OrdStartDate_ATG', 'S_OrdEndDate_ATG', 'S_DLVendorNbr', 'S_AddDate', 'S_ClaimType_ATG',
1458     'S_DealVendorName', 'S_AP_VndNbr', 'S_PurVndrNbr', 'S_PurVndrName',
1459     'E_ItemNbr', 'E_PODate', 'E_ReceivingDate', 'E_InvoicedDate', 'E_ClaimType',
1460     'E_UPCNbr', 'E_UPCUnit', 'E_PurVndrName', 'E_ReceiptNbr',
1461     'E_checked', 'DecisionTree_E_checked', 'RandomForest_E_checked', 'GradientBoost_E_checked',
1462     'LGBM_E_checked', 'XGBoost_E_checked', 'CatBoost_E_checked', 'KNeighbors_E_checked',
1463     'S_checked', 'DecisionTree_S_checked', 'RandomForest_S_checked', 'GradientBoost_S_checked',
1464     'LGBM_S_checked', 'XGBoost_S_checked', 'CatBoost_S_checked', 'KNeighbors_S_checked']
1465 ].reset_index(drop=True)
1466 for col in data.columns:
1467     data[col] = data[col].astype(str)
1468 for row in data.index:
1469     current_row = list(data.iloc[int(row)])
1470     cursor_2.execute(insert_sql, current_row)
1471 conn.commit()
1472
1473 folder_path = "C:/Code/Python/Machine_Learning_AI" # replace with your folder path
1474 output_zip_file =
1475 f'C:/Code/Python/Machine_Learning_AI_Batch{batch_number}_Completed.zip' # replace
1476 with your desired zip file name
1477 compress_folder_to_zip(folder_path, output_zip_file)
1478
1479 port = 465 # For SSL
1480 smtp_server = "smtp.gmail.com"
1481 sender_email = "micpowers98@gmail.com" # Enter your address
1482 receiver_email = "micpowers98@gmail.com" # Enter receiver address
1483 password = 'efex cwhv gppq ueob'
1484 elapsed = int(round(time()-start))

```

```
1482     m, s = divmod(elapsed, 60)
1483     h, m = divmod(m, 60)
1484     print(f'Batch {batch_number} had {len(test):,} test rows and took
        {h:d}:{m:02d}:{s:02d}.')
1485     message = f"""\
1486     Subject: Batch {batch_number} Completed
1487
1488     Batch {batch_number} had {len(test):,} test rows and took {h:d}:{m:02d}:{s:02d}."""
1489     context = create_default_context()
1490     with SMTP_SSL(smtp_server, port, context=context) as server:
1491         server.login(sender_email, password)
1492         server.sendmail(sender_email, receiver_email, message)
1493 # except Exception as e:
1494 #     send_email_to_self(subject='ERROR',body=f"Here's the error: {str(e)}. GET BACK TO
1495 #     print(str(e))
1496 #     WORK NOW!!!")
```