```
from pandas import set_option,concat,DataFrame,Series,to_numeric,to_datetime,options
from numpy import concatenate,random,unique,round
from pyodbc import connect
from warnings import filterwarnings
from matplotlib.pyplot import subplots,tight_layout,savefig,close
from sklearn.metrics import
accuracy_score,make_scorer,confusion_matrix,f1_score,roc_curve,auc
from sklearn.metrics import
precision_recall_curve,precision_score,recall_score,roc_auc_score
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.dummy import DummyClassifier
#from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier#,plot_tree
from sklearn.model_selection import GridSearchCV#,cross_val_score,train_test_split
from sklearn.preprocessing import
StandardScaler,LabelBinarizer,label_binarize,MaxAbsScaler
#from sklearn.feature_extraction import FeatureHasher
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from IPython.display import clear_output
from sklearn.neighbors import *
from re import search#,match
from random import randint
from time import time
from smtplib import SMTP_SSL
from ssl import create_default_context
from os import path,walk
from zipfile import ZipFile, ZIP_DEFLATED
from time import time#,sleep

filterwarnings('ignore')
set_option('display.max_rows', None)
set_option('display.max_columns', None)

def full_score_report_multi_class(model, features, target, predictions,image_name):
    # Calculate recall and precision for weighted average
    recall = round(recall_score(target, predictions, average='weighted'), 3)
    precision = round(precision_score(target, predictions, average='weighted'), 3)

    # Binarize the targets for multiclass ROC and Precision-Recall curves
    classes = unique(target)
    target_binarized = label_binarize(target, classes=classes)
    n_classes = len(classes)

    # Get the predicted probabilities
    probabilities = model.predict_proba(features)

    # Initialize dictionaries for ROC and Precision-Recall
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    precision_curve = dict()
    recall_curve = dict()
    pr_auc = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(target_binarized[:, i], probabilities[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
        precision_curve[i], recall_curve[i], _ =
        precision_recall_curve(target_binarized[:, i], probabilities[:, i])
        pr_auc[i] = auc(recall_curve[i], precision_curve[i])

    # Plotting the ROC and Precision-Recall curves
    fig, ax = subplots(1, 2, figsize=(14, 7))

    # Plot ROC curve
    for i in range(n_classes):
        ax[0].plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC =
```

```
            {roc_auc[i]:.2f})')
        ax[0].plot([0, 1], [0, 1], 'r--')
        ax[0].set_xlim([0.0, 1.0])
        ax[0].set_ylim([0.0, 1.0])
        ax[0].set_xlabel('False Positive Rate')
        ax[0].set_ylabel('True Positive Rate')
        ax[0].set_title('ROC Curve')
        ax[0].legend(loc='lower right')

        # Plot Precision-Recall curve
        for i in range(n_classes):
            ax[1].plot(recall_curve[i], precision_curve[i], lw=2, label=f'Class {classes[i]}
            (AUC = {pr_auc[i]:.2f})')
        ax[1].set_xlim([0.0, 1.0])
        ax[1].set_ylim([0.0, 1.0])
        ax[1].set_xlabel('Recall')
        ax[1].set_ylabel('Precision')
        ax[1].set_title('Precision-Recall Curve')
        ax[1].legend(loc='lower left')

        tight_layout()
        savefig(f"{image_name}.png")
        close()
        # show()

    def full_score_report_binary_class(model, features, target, predictions, image_name):
        # Calculate recall and precision for weighted average
        recall = round(recall_score(target, predictions, average='weighted'), 3)
        precision = round(precision_score(target, predictions, average='weighted'), 3)

        # Binarize the targets for multiclass ROC and Precision-Recall curves
        classes = unique(target)
        target_binarized = label_binarize(target, classes=classes)
        n_classes = len(classes)

        # Get the predicted probabilities
        if hasattr(model, "predict_proba"):
            probabilities = model.predict_proba(features)
        else:
            probabilities = model.decision_function(features)
            probabilities = (probabilities - probabilities.min()) / (probabilities.max() -
            probabilities.min())

        # Initialize dictionaries for ROC and Precision-Recall
        fpr = dict()
        tpr = dict()
        roc_auc = dict()
        precision_curve = dict()
        recall_curve = dict()
        pr_auc = dict()

        if n_classes == 2:
            # Handle binary classification
            fpr[0], tpr[0], _ = roc_curve(target, probabilities[:, 1])
            roc_auc[0] = auc(fpr[0], tpr[0])
            precision_curve[0], recall_curve[0], _ = precision_recall_curve(target,
            probabilities[:, 1])
            pr_auc[0] = auc(recall_curve[0], precision_curve[0])
        else:
            for i in range(n_classes):
                fpr[i], tpr[i], _ = roc_curve(target_binarized[:, i], probabilities[:, i])
                roc_auc[i] = auc(fpr[i], tpr[i])
                precision_curve[i], recall_curve[i], _ =
                precision_recall_curve(target_binarized[:, i], probabilities[:, i])
                pr_auc[i] = auc(recall_curve[i], precision_curve[i])

        # Plotting the ROC and Precision-Recall curves
        fig, ax = subplots(1, 2, figsize=(14, 7))

```

```
        # Plot ROC curve
        if n_classes == 2:
            ax[0].plot(fpr[0], tpr[0], lw=2, label=f'Class {classes[1]} (AUC =
            {roc_auc[0]:.2f})')
        else:
            for i in range(n_classes):
                ax[0].plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC =
                {roc_auc[i]:.2f})')
        ax[0].plot([0, 1], [0, 1], 'r--')
        ax[0].set_xlim([0.0, 1.0])
        ax[0].set_ylim([0.0, 1.0])
        ax[0].set_xlabel('False Positive Rate')
        ax[0].set_ylabel('True Positive Rate')
        ax[0].set_title('ROC Curve')
        ax[0].legend(loc='lower right')

        # Plot Precision-Recall curve
        if n_classes == 2:
            ax[1].plot(recall_curve[0], precision_curve[0], lw=2, label=f'Class {classes[1]}
            (AUC = {pr_auc[0]:.2f})')
        else:
            for i in range(n_classes):
                ax[1].plot(recall_curve[i], precision_curve[i], lw=2, label=f'Class
                {classes[i]} (AUC = {pr_auc[i]:.2f})')
        ax[1].set_xlim([0.0, 1.0])
        ax[1].set_ylim([0.0, 1.0])
        ax[1].set_xlabel('Recall')
        ax[1].set_ylabel('Precision')
        ax[1].set_title('Precision-Recall Curve')
        ax[1].legend(loc='lower left')

        tight_layout()
        savefig(f"{image_name}.png")
        close()
        # show()

    def get_scores(model, features, target, predictions):
        accuracy = accuracy_score(target, predictions)
        f1 = f1_score(target,predictions,average='weighted')
        matrix = confusion_matrix(target,predictions)
        recall = recall_score(target,predictions,average='weighted')
        precision = precision_score(target,predictions,average='weighted')
        if(target.nunique()>1):
            probabilities = model.predict_proba(features)
            #auc_roc = roc_auc_score(target,
            probabilities,average='weighted',multi_class='ovr',labels=)
            # Binarize the true labels
            target_binarized = label_binarize(target, classes=[0,1,2,3])

            # Calculate the ROC AUC score using the 'ovr' (one-vs-rest) strategy
            roc_auc_ovr = roc_auc_score(target_binarized, probabilities, multi_class='ovr')

            # Calculate the ROC AUC score using the 'ovo' (one-vs-one) strategy
            roc_auc_ovo = roc_auc_score(target_binarized, probabilities, multi_class='ovo')
            return [accuracy,recall,precision,f1,roc_auc_ovr,roc_auc_ovo,matrix]
        else:
            return [accuracy,recall,precision,f1,-1,-1,matrix]

    def empty_string_to_null(string: str):
        if(len(str(string))==0 or len(str(string).replace(' ',''))==0):
            return "None"
        return string

    def object_to_int(original,lookup_table):
        for row in lookup_table.index:
            for col in lookup_table.columns:
                if(original==lookup_table[col][row]):
                    return int(col)

```

```
def get_knn(df, n, k, metric, feature_names):

    """
    Returns k nearest neighbors

    :param df: pandas DataFrame used to find similar objects within
    :param n: object no for which the nearest neighbours are looked for
    :param k: the number of the nearest neighbours to return
    :param metric: name of distance metric
    """

    nbrs = NearestNeighbors(n_neighbors=k, algorithm='ball_tree',
    metric=metric).fit(df[feature_names].to_numpy())
    nbrs_distances, nbrs_indices = nbrs.kneighbors([df.iloc[n][feature_names]], k,
    return_distance=True)

    df_res = concat([
        df.iloc[nbrs_indices[0]],
        DataFrame(nbrs_distances.T, index=nbrs_indices[0], columns=['distance'])
        ], axis=1)

    return df_res

def build_knc(random_state, train, target, test, n_neighbors):
    random.seed(random_state)
    knc = KNeighborsClassifier(n_neighbors=n_neighbors)
    knc.fit(train, target)
    y_pred = knc.predict(test)
    return knc,y_pred

def random_int(min_val, max_val):
    # Get the current time in microseconds
    current_time = int(time() * 1000000)

    # Use the current time as a seed and perform some operations to get more randomness
    seed = (current_time ^ (current_time >> randint(1,20))) & 0xFFFFFFFF
    seed = (seed ^ (seed << randint(1,20))) & 0xFFFFFFFF
    seed = (seed ^ (seed >> randint(1,20))) & 0xFFFFFFFF

    # Scale the seed to the desired range
    random_value = min_val + (seed % (max_val - min_val + 1))

    return random_value

def random_features(columns):
    subset = []
    columns = list(columns)
    while(len(subset)<50):
        random_index = random_int(0,len(list(columns))-1)
        while(columns[random_index] in subset):
            random_index = random_int(0,len(list(columns))-1)
        subset.append(columns[random_index])
    return subset

def send_email_to_self(subject: str,body: str):
    port = 465  # For SSL
    smtp_server = "smtp.gmail.com"
    sender_email = "micpowers98@gmail.com"  # Enter your address
    receiver_email = "micpowers98@gmail.com"  # Enter receiver address
    password = 'efex cwhv gppq ueob'
    message = "Subject: "+subject+"\n"+body

    context = create_default_context()
    with SMTP_SSL(smtp_server, port, context=context) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message)

def get_train_data(columns: list):
    cursor_1.execute("""
```

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

```
                    li.[ItemDescription],
                    li.[ItemShipPack],
                    li.[PoNbr],
                    li.[PODate],
                    li.[ReceivingDate],
                    li.[TurnRatio_ATG],
                    li.[TurnQty_ATG],
                    li.[OrdQty],
                    li.[PdQty_ATG],
                    li.[PdGross_ATG],
                    li.[PdOI_ATG],
                    li.[PdBB_ATG],
                    li.[PdNet_ATG],
                    li.[DealNbr],
                    li.[OrdStartDate_ATG],
                    li.[OrdEndDate_ATG],
                    li.[DateStartArrival_ATG],
                    li.[DateEndArrival_ATG],
                    li.[DLAmtOI],
                    li.[DLAmtBB]--,
                from
                    [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
                    join  [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
                    ds
                    on (li.BatchNbr = ds.BatchNbr)
                    and (li.DLVendorNbr = ds.DLVendorNbr)
                    and (li.DealNbr = ds.DealNbr)
                    and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
                    and (li.ClaimType = ds.ClaimType_ATG)
                where
                    li.ClaimType = 'IN DEAL'
                    and
                    ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
                    and
                    li.checked='true'
                    and
                    ds.checked='x'

            union

            select top(30000)
                case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
                'N' end as IsClaimFlg, li.Checked,
                    li.[ATG_Ref],
                    li.[ATG_LI_Ref],
                    li.[BatchNbr],
                    li.[CATEGORY_ATG],
                    li.[ClaimType],
                    li.[AP_VndNbr],
                    li.[APVndrName],
                    li.[OOB_ATG],
                    li.[ItemNbr],
                    li.[UPCNbr],
                    li.[UPCUnit],
                    li.[ItemDescription],
                    li.[ItemShipPack],
                    li.[PoNbr],
                    li.[PODate],
                    li.[ReceivingDate],
                    li.[TurnRatio_ATG],
                    li.[TurnQty_ATG],
                    li.[OrdQty],
                    li.[PdQty_ATG],
                    li.[PdGross_ATG],
                    li.[PdOI_ATG],
                    li.[PdBB_ATG],
                    li.[PdNet_ATG],
                    li.[DealNbr],
                    li.[OrdStartDate_ATG],
```

```
                    li.[OrdEndDate_ATG],
                    li.[DateStartArrival_ATG],
                    li.[DateEndArrival_ATG],
                    li.[DLAmtOI],
                    li.[DLAmtBB]--,
                from
                    [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
                    join  [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
                    ds
                    on (li.BatchNbr = ds.BatchNbr)
                    and (li.DLVendorNbr = ds.DLVendorNbr)
                    and (li.DealNbr = ds.DealNbr)
                    and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
                    and (li.ClaimType = ds.ClaimType_ATG)
                where
                    li.ClaimType = 'IN DEAL'
                    and
                    ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
                    and
                    li.checked='x'
                    and
                    ds.checked='x'
        """)
    train_data = cursor_1.fetchall()
    train_data_list = []
    for index in range(len(train_data)):
        train_data_list.append(list(train_data[index]))
    del train_data
    return DataFrame(data=train_data_list,columns=columns)

def get_test_data(batch_number: int, columns: list):
    cursor_1.execute(f"""
            select
                case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
                'N' end as IsClaimFlg, li.Checked,
                    li.[ATG_Ref],
                    li.[ATG_LI_Ref],
                    li.[BatchNbr],
                    li.[CATEGORY_ATG],
                    li.[ClaimType],
                    li.[AP_VndNbr],
                    li.[APVndrName],
                    li.[OOB_ATG],
                    li.[ItemNbr],
                    li.[UPCNbr],
                    li.[UPCUnit],
                    li.[ItemDescription],
                    li.[ItemShipPack],
                    li.[PoNbr],
                    li.[PODate],
                    li.[ReceivingDate],
                    li.[TurnRatio_ATG],
                    li.[TurnQty_ATG],
                    li.[OrdQty],
                    li.[PdQty_ATG],
                    li.[PdGross_ATG],
                    li.[PdOI_ATG],
                    li.[PdBB_ATG],
                    li.[PdNet_ATG],
                    li.[DealNbr],
                    li.[OrdStartDate_ATG],
                    li.[OrdEndDate_ATG],
                    li.[DateStartArrival_ATG],
                    li.[DateEndArrival_ATG],
                    li.[DLAmtOI],
                    li.[DLAmtBB]--,
                from
                    [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
                    join  [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
```

```
                    ds
                    on (li.BatchNbr = ds.BatchNbr)
                    and (li.DLVendorNbr = ds.DLVendorNbr)
                    and (li.DealNbr = ds.DealNbr)
                    and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
                    and (li.ClaimType = ds.ClaimType_ATG)
                where
                    li.BatchNbr={batch_number}
                    and
                    li.ClaimType = 'IN DEAL'
                    and
                    ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
                    and
                    li.checked='x'
                    and
                    ds.checked='x'

            union

            select top(1)
                case when (li.checked = 'true' and ds.checked = 'claim') then 'Y' else
                'N' end as IsClaimFlg, li.Checked,
                    li.[ATG_Ref],
                    li.[ATG_LI_Ref],
                    li.[BatchNbr],
                    li.[CATEGORY_ATG],
                    li.[ClaimType],
                    li.[AP_VndNbr],
                    li.[APVndrName],
                    li.[OOB_ATG],
                    li.[ItemNbr],
                    li.[UPCNbr],
                    li.[UPCUnit],
                    li.[ItemDescription],
                    li.[ItemShipPack],
                    li.[PoNbr],
                    li.[PODate],
                    li.[ReceivingDate],
                    li.[TurnRatio_ATG],
                    li.[TurnQty_ATG],
                    li.[OrdQty],
                    li.[PdQty_ATG],
                    li.[PdGross_ATG],
                    li.[PdOI_ATG],
                    li.[PdBB_ATG],
                    li.[PdNet_ATG],
                    li.[DealNbr],
                    li.[OrdStartDate_ATG],
                    li.[OrdEndDate_ATG],
                    li.[DateStartArrival_ATG],
                    li.[DateEndArrival_ATG],
                    li.[DLAmtOI],
                    li.[DLAmtBB]--,
                from
                    [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_DealLI_Exceptions] li
                    join  [MOE].[prod_WeisMarkets_RecoverNow].[dbo].[ATG_Deal_Summary]
                    ds
                    on (li.BatchNbr = ds.BatchNbr)
                    and (li.DLVendorNbr = ds.DLVendorNbr)
                    and (li.DealNbr = ds.DealNbr)
                    and (li.CATEGORY_ATG = ds.CATEGORY_ATG)
                    and (li.ClaimType = ds.ClaimType_ATG)
            where
                li.ClaimType = 'IN DEAL'
                and
                ds.CATEGORY_ATG = 'SAME VENDOR - AMT DEALS'
                and
                li.checked='true'
                and
```

```
                ds.checked='claim'
        """)
    test_data = cursor_1.fetchall()
    test_data_list = []
    for index in range(len(test_data)):
        test_data_list.append(list(test_data[index]))
    del test_data
    return DataFrame(data=test_data_list,columns=columns)

def compress_folder_to_zip(folder_path, output_zip_file):
    # Create a ZipFile object
    with ZipFile(output_zip_file, 'w', ZIP_DEFLATED) as zipf:
        # Walk through the folder
        for root, dirs, files in walk(folder_path):
            for file in files:
                # Create the full file path
                file_path = path.join(root, file)
                # Add file to zip, using relative path to maintain folder structure
                arcname = path.relpath(file_path, start=folder_path)
                zipf.write(file_path, arcname=arcname)

columns: list = ['S_DealNbr', 'S_ATG_IR', 'S_ClaimType_ATG', 'S_OrdStartDate_ATG',
'S_OrdEndDate_ATG', 'S_InDeal_Due', 'S_Shoulder_Due', 'S_LargeBuy_Due',
           'S_TTL_OIDue', 'S_TTL_BBDue', 'S_NoBuy_Due', 'S_Facility', 'S_DLVendorNbr',
           'S_MultiDeal_ATG', 'S_DealOrigin_ATG', 'S_AddDate',
           'S_DealStatus_ATG', 'S_ClaimNumber', 'S_ClaimDate', 'S_ATG_IR_SourceFile',
           'S_ClmBatchNbr_ATG', 'S_ItemCount', 'S_PA_Claimed', 'S_AR_AdjTyp',
           'S_AR_Amount', 'S_AltTABLE2_ATG', 'S_DQ_Reason_ATG', 'S_QtyClaimed',
           'S_QtyExample', 'S_LeadQTY', 'S_InDealQTY', 'S_PostQTY', 'S_SellUnitQty',
           'S_UnitQtyOverSold_ATG', 'S_DaysBefore_ATG', 'S_DaysAfter_ATG',
           'S_MultiVendor_ATG', 'S_DealVendorName', 'S_AP_VndNbr', 'S_PurVndrNbr',
           'S_PurVndrNbr2', 'S_PurVndrName', 'S_PurVndrName2', 'S_PurVndCount_ATG',
           'S_DateStartArrival_ATG', 'S_DateEndArrival_ATG', 'S_PromoStartDate_ATG',
           'S_PromoEndDate_ATG', 'S_OCCURS', 'S_CATEGORY_ATG', 'S_ATG_ID', 'S_checked',
           'S_BatchNbr', 'S_ATG_Ref', 'S_ClaimActivityID_ATGSYS',
           'S_ClaimActivityCount_ATGSYS', 'E_BatchNbr', 'E_PODate', 'E_ReceivingDate',
           'E_InvoicedDate', 'E_OIDue_ATG', 'E_BBDue_ATG', 'E_ClaimType',
           'E_Shoulder_ATG', 'E_DtMatch_ATG', 'E_ItemNbr', 'E_ItemDescription',
           'E_ItemShipPack', 'E_PoNbr', 'E_DealNbr', 'E_ATG_IR', 'E_OrdStartDate_ATG',
           'E_OrdEndDate_ATG', 'E_DateStartArrival_ATG', 'E_DateEndArrival_ATG',
           'E_AddDate', 'E_DLAmtOI', 'E_DLAmtBB', 'E_TurnRatio_ATG', 'E_OrdQty',
           'E_PdQty_ATG', 'E_PdNetSB_ATG', 'E_PdNet_ATG', 'E_PdOI_ATG', 'E_PdBB_ATG',
           'E_PdGross_ATG', 'E_ListcostAtStart', 'E_ListCostSB_ATG', 'E_BestOI_ATG',
           'E_BestBB_ATG', 'E_DQ_Reason_ATG', 'E_ClaimedAmt', 'E_AR_AdjTyp',
           'E_AR_Amount', 'E_AR_InvNbr', 'E_AR_InvNbr2', 'E_UPCNbr', 'E_UPCUnit',
           'E_TurnQty_ATG', 'E_TruckSize_ATG', 'E_PdUpDn_ATG', 'E_BalFlag_ATG',
           'E_OOB_ATG', 'E_MiscAdj_ATG', 'E_DLVendorNbr', 'E_DealVendorName',
           'E_PurVndrNbr',
           'E_PurVndrName', 'E_Facility', 'E_ReceiptNbr', 'E_ReceiptSfx',
           'E_VndrInvNbr', 'E_TotWght', 'E_AP_VndNbr', 'E_APVndrName', 'E_BuyrNbr',
           'E_BuyrName',
           'E_Contact', 'E_AP_CheckNbr', 'E_AP_CheckDate', 'E_AP_GrossAmt',
           'E_AP_DiscAmt', 'E_VendorCmmnt', 'E_TTLVendorCmmnt_ATG', 'E_PORemarks',
           'E_MultiDeal_ATG', 'E_DealOrigin_ATG', 'E_AbsDays_ATG', 'E_ATG_Ref',
           'E_ATG_DL_Ref', 'E_ATG_LI_Ref', 'E_ATG_HDR_Ref', 'E_DedBBInd', 'E_checked',
           'E_CATEGORY_ATG', 'E_TrnCde_ATG', 'E_Dept', 'E_DLPctOI', 'E_DLPctBB',
           'E_DLIncvPctOI', 'E_DLIncvPctBB', 'E_ClaimActivityID_ATGSYS',
           'E_ClaimActivityCount_ATGSYS', 'E_ClaimActivityTypeID_ATGSYS']

columns: list = [
    'IsClaimFlg',
    'Checked',
    'ATG_Ref',
    'ATG_LI_Ref',
    'BatchNbr',
    'CATEGORY_ATG',
    'ClaimType',
    'AP_VndNbr',
    'APVndrName',
```

```
    'OOB_ATG',
    'ItemNbr',
    'UPCNbr',
    'UPCUnit',
    'ItemDescription',
    'ItemShipPack',
    'PoNbr',
    'PODate',
    'ReceivingDate',
    'TurnRatio_ATG',
    'TurnQty_ATG',
    'OrdQty',
    'PdQty_ATG',
    'PdGross_ATG',
    'PdOI_ATG',
    'PdBB_ATG',
    'PdNet_ATG',
    'DealNbr',
    'OrdStartDate_ATG',
    'OrdEndDate_ATG',
    'DateStartArrival_ATG',
    'DateEndArrival_ATG',
    'DLAmtOI',
    'DLAmtBB'
]

# try:
for batch_number in range(80,81):
    start = time()
    options.display.float_format = '{:.10f}'.format
    server = 'troy'
    database = 'prod_Costco_RecoverNow'
    connectionString = f'DRIVER={{ODBC Driver 17 for SQL
    Server}};SERVER={server};DATABASE={database};Integrated
    Security={True};Autocommit={True};Trusted_Connection=yes;'
    conn = connect(connectionString)
    cursor_1 = conn.cursor()
    train: DataFrame = get_train_data(columns=columns)
    test: DataFrame = get_test_data(batch_number=batch_number,columns=columns)
    subject = 'Data Collected'
    body = f"""
            {len(test):,} rows of data in batch {batch_number}.
        """
    send_email_to_self(subject,body)

    empty_col = []
    for col in test.columns:
        if(test[col].dtype=='object'):
            try:
                train[col] = train[col].astype(float)
                test[col] = test[col].astype(float)
            except Exception as e:
                print(f"{str(e)}: {col}")
    for col in train.columns:
        if train[col].dtype == 'object':
            if(train[col].nunique()>1000):
                empty_col.append(col)
    train = train.drop(empty_col,axis=1)
    test = test.drop(empty_col,axis=1)
    for col in train.columns:
        if train[col].dtype == 'object' and not(col=='checked'):
            try:
                train[col] = train[col].astype(int)
            except:
                try:
                    train[col] = train[col].astype(float)
                except Exception as e: pass
    for col in test.columns:
        if test[col].dtype == 'object' and not(col=='checked'):
```

```
            try:
                test[col] = test[col].astype(int)
            except:
                try:
                    test[col] = test[col].astype(float)
                except Exception as e: pass
    for col in train.columns:
        if train[col].dtype == 'object':
            train[col] = train[col].fillna('None')
        elif train[col].dtype == 'int64':
            train[col] = train[col].fillna(-1)
        elif train[col].dtype == 'float64':
            train[col] = train[col].fillna(-1)
    for col in test.columns:
        if test[col].dtype == 'object':
            test[col] = test[col].fillna('None')
        elif test[col].dtype == 'int64':
            test[col] = test[col].fillna(-1)
        elif test[col].dtype == 'float64':
            test[col] = test[col].fillna(-1)
    for col in train.columns:
        if train[col].dtype == 'object' and not(col=='checked'):
            try:
                train[col] = train[col].astype(int)
            except:
                try:
                    train[col] = train[col].astype(float)
                except Exception as e: pass
    for col in test.columns:
        if test[col].dtype == 'object' and not(col=='checked'):
            try:
                test[col] = test[col].astype(int)
            except:
                try:
                    test[col] = test[col].astype(float)
                except Exception as e: pass
    category_values = {}
    category_new_values = {}
    for col in train.columns:
        if train[col].dtype == 'object' and train[col].nunique()<=1000:
            train[col] = train[col].apply(empty_string_to_null)
            test[col] = test[col].apply(empty_string_to_null)
            # categorical_columns.append(col)
            category_values[col] =
            Series(list(train[col].unique())+list(test[col].unique())).unique()
    max_length = 0
    for key,item in category_values.items():
        if(len(item)>max_length):
            max_length = len(item)
    for key in category_values:
        category_values[key] = concatenate((category_values[key],['None'] * (max_length
        - len(category_values[key]))))
    lookup_table = DataFrame.from_dict(category_values, orient='index')
    lookup_table.columns = range(max_length)
    for col in lookup_table.columns:
        lookup_table[col] = lookup_table[col].astype(str)

    lookup_table.to_csv(f'C:/Code/Python/Machine_Learning_AI/Lookup_Table_Batch_{batch_nu
    mber}.csv')
    del lookup_table
    subject = 'Lookup Table Created'
    body = f"""
            Starting long category_new_values section.
        """
    #send_email_to_self(subject,body)

    category_new_values = {}
    count = 0
    for key,item in category_values.items():
```

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

```
olumns,axis=1).columns))
target_train_E = train['Checked']
target_train_S = train['IsClaimFlg']
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))
target_test_E = test['Checked']
target_test_S = test['IsClaimFlg']

subject = 'Starting Dummy Classifier'
body = f"""
    """
#send_email_to_self(subject,body)
dc_e = DummyClassifier(random_state=random_state,strategy='most_frequent')
dc_e.fit(features_train_scaled,target_train_E)
dc_predictions = dc_e.predict(features_test_scaled)

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_R
ecall_Precision_Curves')
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'])
model_scores.loc['Dummy_Exceptions'] =
get_scores(dc_e,features_test_scaled,target_test_E,dc_predictions)[:-1]
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

dc_s = DummyClassifier(random_state=random_state,strategy='most_frequent')
dc_s.fit(features_train_scaled,target_train_S)
dc_predictions = dc_s.predict(features_test_scaled)

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')
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'])
model_scores.loc['Dummy_Summary'] =
get_scores(dc_s,features_test_scaled,target_test_S,dc_predictions)[:-1]
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

subject = 'Starting DecisionTreeClassifier Exceptions'
body = f"""
    """
#send_email_to_self(subject,body)
dt_e_parameters = {
    'random_state':[random_state],
    'max_depth':[2,3,4,5,6,7,8],
    'splitter':['best','random']
}
dt_e = DecisionTreeClassifier(random_state=random_state,max_depth=None)
dt_e
=GridSearchCV(DecisionTreeClassifier(),dt_e_parameters,verbose=10,cv=5,refit=True,err
or_score='raise',return_train_score=True)
dt_e.fit(features_train_scaled,target_train_E)
dt_e_best = DecisionTreeClassifier(**dt_e.best_params_)
dt_e_best.fit(features_train_scaled,target_train_E)
dt_e_predictions = dt_e.predict(features_test_scaled)

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')
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,inde
        x=['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,err
        or_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_predi
            ctions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Decision_Tree_S
            ummary_ROC_Recall_Precision_Curves')
885     dt_s_confusion_matrix =
        DataFrame(confusion_matrix(target_test_S,dt_s_predictions,labels=[0,1]),columns=['Pre
        dicted_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,inde
        x=['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,erro
        r_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_pred
        ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Random_Forest_E
        xceptions_ROC_Recall_Precision_Curves')
907     rf_e_confusion_matrix =
        DataFrame(confusion_matrix(target_test_E,rf_e_predictions,labels=[0,1]),columns=['Pre
        dicted_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

subject = 'Starting RandomForestClassifier Summary'
body = f"""
    """
#send_email_to_self(subject,body)
rf_s_parameters = {
    'random_state':[random_state],
    'max_depth':[5],#[2,3,5],
    'n_estimators':[50],#[50,100,150,200],
    'max_features':[None],
    'warm_start':[True]
}
rf_s = RandomForestClassifier(n_estimators=500,random_state=random_state,warm_start=True,max_depth=None,verbose=10)
rf_s = GridSearchCV(RandomForestClassifier(),rf_s_parameters,verbose=10,cv=2,refit=True,error_score='raise',return_train_score=True)
rf_s.fit(features_train_scaled,target_train_S)
rf_s_best = RandomForestClassifier(**rf_s.best_params_)
rf_s_best.fit(features_train_scaled,target_train_S)
rf_s_predictions = rf_s.predict(features_test_scaled)

full_score_report_binary_class(rf_s_best,features_test_scaled,target_test_S,rf_s_predictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Random_Forest_Summary_ROC_Recall_Precision_Curves')
rf_s_confusion_matrix = DataFrame(confusion_matrix(target_test_S,rf_s_predictions,labels=[0,1]),columns=['Predicted_x','Predicted_claim'],index=['Actual_x','Actual_claim'])
model_scores.loc['Random_Forest_Summary'] = get_scores(rf_s_best,features_test_scaled,target_test_S,rf_s_predictions)[:-1]
rf_s_importances = DataFrame([rf_s_best.feature_importances_],columns=features_train_scaled.columns,index=['Random_Forest_Summary']).T

subject = 'Starting Gradient Boost Exceptions'
body = f"""
    """
#send_email_to_self(subject,body)
gb_e_parameters = {
    'random_state':[random_state],
    'n_estimators':[50,100]#,150,200,250]
}
gb_e = GridSearchCV(GradientBoostingClassifier(),gb_e_parameters,verbose=10,cv=3,refit=True,error_score='raise',return_train_score=True)
gb_e.fit(features_train_scaled,target_train_E)
gb_e_best = GradientBoostingClassifier(**gb_e.best_params_)
gb_e_best.fit(features_train_scaled,target_train_E)
gb_e_predictions = gb_e_best.predict(features_test_scaled)

full_score_report_binary_class(gb_e_best,features_test_scaled,target_test_E,gb_e_predictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/GradientBoost_Exceptions_ROC_Recall_Precision_Curves')
gb_e_confusion_matrix = DataFrame(confusion_matrix(target_test_E,gb_e_predictions,labels=[0,1]),columns=['Predicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
model_scores.loc['GradientBoost_Exceptions'] = get_scores(gb_e_best,features_test_scaled,target_test_E,gb_e_predictions)[:-1]
gb_e_importances = DataFrame([gb_e_best.feature_importances_],columns=features_train_scaled.columns,index=['GradientBoost_Exceptions']).T

subject = 'Starting Gradient Boost Summary'
body = f"""
    """
#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)
```

```
lgbm_s_predictions = lgbm_s_best.predict(features_test_scaled)

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_sxceptions_ROC_Recall_Precision_Curves')
lgbm_s_confusion_matrix = DataFrame(confusion_matrix(target_test_E,lgbm_s_predictions,labels=[0,1]),columns=['Predicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
model_scores.loc['LGBM_Summary'] = get_scores(lgbm_s_best,features_test_scaled,target_test_E,lgbm_s_predictions)[:-1]
lgbm_s_importances = DataFrame([lgbm_s_best.feature_importances_],columns=features_train_scaled.columns,index=['LGBM_Summary']).T
lgbm_s_importances['LGBM_Summary'] = lgbm_s_importances['LGBM_Summary']/3000

subject = 'Starting XGBoost Exceptions'
body = f"""
    """
#send_email_to_self(subject,body)
xgb_e_parameters = {
    'random_state':[random_state],
    'n_estimators':[50,100,150,200,250]
}
xgb_e = XGBClassifier(verbosity=3,n_estimators=200)
xgb_e = GridSearchCV(XGBClassifier(),xgb_e_parameters,verbose=10,cv=5,refit=True,error_score='raise',return_train_score=True)
xgb_e.fit(features_train_scaled,target_train_E)
xgb_e_best = XGBClassifier(**xgb_e.best_params_)
xgb_e_best.fit(features_train_scaled,target_train_E)
xgb_e_predictions = xgb_e_best.predict(features_test_scaled)

full_score_report_binary_class(xgb_e_best,features_test_scaled,target_test_E,xgb_e_predictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/XGBoost_Exceptions_ROC_Recall_Precision_Curves')
xgb_e_confusion_matrix = DataFrame(confusion_matrix(target_test_E,xgb_e_predictions,labels=[0,1]),columns=['Predicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
model_scores.loc['XGBoost_Exceptions'] = get_scores(xgb_e_best,features_test_scaled,target_test_E,xgb_e_predictions)[:-1]
xgb_e_importances = DataFrame([xgb_e_best.feature_importances_],columns=features_train_scaled.columns,index=['XGBoost_Exceptions']).T

subject = 'Starting XGBoost Summary'
body = f"""
    """
#send_email_to_self(subject,body)
xgb_s_parameters = {
    'random_state':[random_state],
    'n_estimators':[50,100,150,200,250]
}
xgb_s = XGBClassifier(verbosity=3,n_estimators=200)
xgb_s = GridSearchCV(XGBClassifier(),xgb_s_parameters,verbose=10,cv=5,refit=True,error_score='raise',return_train_score=True)
xgb_s.fit(features_train_scaled,target_train_S)
xgb_s_best = XGBClassifier(**xgb_s.best_params_)
xgb_s_best.fit(features_train_scaled,target_train_S)
xgb_s_predictions = xgb_s_best.predict(features_test_scaled)

full_score_report_binary_class(xgb_s_best,features_test_scaled,target_test_S,xgb_s_predictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/XGBoostBoost_Summary_ROC_Recall_Precision_Curves')
xgb_s_confusion_matrix = DataFrame(confusion_matrix(target_test_S,xgb_s_predictions,labels=[0,1]),columns=['Predicted_x','Predicted_claim'],index=['Actual_x','Actual_claim'])
model_scores.loc['XGBoost_Summary'] = get_scores(xgb_s_best,features_test_scaled,target_test_S,xgb_s_predictions)[:-1]
```

```
    xgb_s_importances =
DataFrame([xgb_s_best.feature_importances_],columns=features_train_scaled.columns,ind
ex=['XGBoost_Summary']).T

    subject = 'Starting CatBoost Exceptions'
    body = f"""
        """
    #send_email_to_self(subject,body)
    cb_e_parameters = {
        'iterations':[1500],#[500,750,1000,1500],
        'random_state':[random_state],
        'learning_rate':[0.005],#[0.005,0.0075,0.01],
        'depth':[5],#[2,5],
        'verbose':[0],
        'early_stopping_rounds':[3]#[3,5,10]
    }
    cb_e =
CatBoostClassifier(iterations=10000,learning_rate=0.0075,random_state=random_state,de
pth=7,verbose=10)
    cb_e =
GridSearchCV(CatBoostClassifier(),param_grid=cb_e_parameters,verbose=10,refit=True,cv
=2,error_score='raise',return_train_score=True)
    cb_e.fit(features_train_scaled,target_train_E)
    cb_e_best = CatBoostClassifier(**cb_e.best_params_)
    cb_e_best.fit(features_train_scaled,target_train_E)
    cb_e_predictions = cb_e_best.predict(features_test_scaled)

    full_score_report_binary_class(cb_e_best,features_test_scaled,target_test_E,cb_e_pred
ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Cat_Boost_Excep
tions_ROC_Recall_Precision_Curves')
    cb_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,cb_e_predictions,labels=[0,1]),columns=['Pre
dicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
    model_scores.loc['Cat_Boost_Exceptions'] =
get_scores(cb_e_best,features_test_scaled,target_test_E,cb_e_predictions)[:-1]
    cb_e_importances =
DataFrame([cb_e_best.feature_importances_],columns=features_train_scaled.columns,inde
x=['Cat_Boost_Exceptions']).T
    cb_e_importances['Cat_Boost_Exceptions'] =
cb_e_importances['Cat_Boost_Exceptions']/100

    subject = 'Starting CatBoost Summary'
    body = f"""
        """
    #send_email_to_self(subject,body)
    cb_s_parameters = {
        'iterations':[1500],#[500,750,1000,1500],
        'random_state':[random_state],
        'learning_rate':[0.005],#[0.005,0.0075,0.01],
        'depth':[4],#[2,5],
        'verbose':[0],
        'early_stopping_rounds':[5]#[3,5,10]
    }
    cb_s =
CatBoostClassifier(iterations=10000,learning_rate=0.0075,random_state=random_state,de
pth=7,verbose=10)
    cb_s =
GridSearchCV(CatBoostClassifier(),param_grid=cb_s_parameters,verbose=10,refit=True,cv
=2,error_score='raise',return_train_score=True)
    cb_s.fit(features_train_scaled,target_train_S)
    cb_s_best = CatBoostClassifier(**cb_s.best_params_)
    cb_s_best.fit(features_train_scaled,target_train_S)
    cb_s_predictions = cb_s_best.predict(features_test_scaled)

    full_score_report_binary_class(cb_s_best,features_test_scaled,target_test_S,cb_s_pred
ictions,'C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Cat_Boost_Summary_
ROC_Recall_Precision_Curves')
    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'])
model_scores.loc['Cat_Boost_Summary'] =
get_scores(cb_s_best,features_test_scaled,target_test_S,cb_s_predictions)[:-1]
cb_s_importances =
DataFrame([cb_s_best.feature_importances_],columns=features_train_scaled.columns,inde
x=['Cat_Boost_Summary']).T
cb_s_importances['Cat_Boost_Summary'] = cb_s_importances['Cat_Boost_Summary']/100

subject = 'Starting K-NearestNeighbors'
body = f"""
    """
#send_email_to_self(subject,body)
knc_e,knn_e_predictions =
build_knc(random_state=random_state,train=features_train_scaled,target=target_train_E
,n_neighbors=3,test=features_test_scaled)

full_score_report_binary_class(model=knc_e,features=features_test_scaled,target=targe
t_test_E,predictions=knn_e_predictions,image_name='C:/Code/Python/Machine_Learning_AI
/Model_Analysis/Exceptions/KNN_Exceptions_ROC_Recall_Precision_Curves')
knn_e_confusion_matrix =
DataFrame(confusion_matrix(target_test_E,knn_e_predictions,labels=[0,1]),columns=['Pr
edicted_x','Predicted_TRUE'],index=['Actual_x','Actual_TRUE'])
model_scores.loc['K_Neighors_Exceptions'] =
get_scores(knc_e,features_test_scaled,target_test_E,knn_e_predictions)[:-1]

knc_s,knn_s_predictions =
build_knc(random_state=random_state,train=features_train_scaled,target=target_train_S
,n_neighbors=3,test=features_test_scaled)

full_score_report_binary_class(model=knc_s,features=features_test_scaled,target=targe
t_test_S,predictions=knn_s_predictions,image_name='C:/Code/Python/Machine_Learning_AI
/Model_Analysis/Summary/KNN_Summary_ROC_Recall_Precision_Curves')
knn_s_confusion_matrix =
DataFrame(confusion_matrix(target_test_S,knn_s_predictions,labels=[0,1]),columns=['Pr
edicted_x','Predicted_claim'],index=['Actual_x','Actual_claim'])
model_scores.loc['K_Neighors_Summary'] =
get_scores(knc_s,features_test_scaled,target_test_S,knn_s_predictions)[:-1]

final_result: DataFrame = get_test_data(batch_number=batch_number,columns=columns)
final_result['DecisionTree_Model_Checked_Predictions'] =
Series(dt_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['RandomForest_Model_Checked_Predictions'] =
Series(rf_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['LGBM_Model_Checked_Predictions'] =
Series(lgbm_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['GradientBoost_Model_Checked_Predictions'] =
Series(gb_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['XGBoost_Model_Checked_Predictions'] =
Series(xgb_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['CatBoost_Model_Checked_Predictions'] =
Series(cb_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})
final_result['KNeighbors_Model_Checked_Predictions'] =
Series(knn_e_predictions).reset_index(drop=True).replace({0:'x',1:'TRUE'})


final_result.to_csv('C:/Code/Python/Machine_Learning_AI/Data_With_Model_Predictions.c
sv',index=False)

print("DONE STOP STOP")

subject = 'Analyzing Models'
body = f"""
    """
#send_email_to_self(subject,body)

model_scores.iloc[[0,2,4,6,8,10]].to_csv('C:/Code/Python/Machine_Learning_AI/Model_An
alysis/Exceptions/All_Model_Scores_Exceptions.csv')
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)
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']

combined_importances_e.to_csv('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exce
ptions/Feature_Importance_Exceptions_All_Models.csv')
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]
titles_e = ['Dummy_Exceptions', 'Decision_Tree_Exceptions',
'Random_Forest_Exceptions', 'GradientBoost_Exceptions',
            'LGBM_Exceptions', 'XGBoost_Exceptions',
            'Cat_Boost_Exceptions','K_Neighbors_Exceptions']
table_e_rows = [len(tbl) for tbl in confusion_e_matrices]
# Create a figure and a set of subplots
fig_e, axs_e = subplots(ncols=1, nrows=8,gridspec_kw={'height_ratios':
table_e_rows},figsize=(6,10))
for ax, dfs, title in zip(axs_e, confusion_e_matrices, titles_e):
    # Hide the axes
    # ax.axis('tight')
    df_with_index_e = dfs.copy()
    df_with_index_e.insert(0, '', dfs.index)
    ax.axis('off')
    # Create the table
    table = ax.table(cellText=df_with_index_e.values,
    colLabels=df_with_index_e.columns, cellLoc='center', loc='center')
    # Add title
    ax.set_title(title)
tight_layout()

savefig('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Exceptions/Confused_Matric
es_Exceptions.pdf',format='pdf')

savefig('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Exceptions/Con
fused_Matrices_Exceptions.pdf',format='pdf')
close()

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]
titles_s = ['Dummy_Summary', 'Decision_Tree_Summary', 'Random_Forest_Summary',
'GradientBoost_Summary',
            'LGBM_Summary', 'XGBoost_Summary',
            'Cat_Boost_Summary','K_Neighbors_Summary']
table_s_rows = [len(tbl) for tbl in confusion_s_matrices]

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')
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)
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']

combined_importances_s.to_csv('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summ
ary/Feature_Importance_Summary_All_Models.csv')
fig_s, axs_s = subplots(ncols=1, nrows=8,gridspec_kw={'height_ratios':
table_s_rows},figsize=(6,10))
for ax, dfs, title in zip(axs_s, confusion_s_matrices, titles_s):
    # Hide the axes
    # ax.axis('tight')
    df_with_index_s = dfs.copy()
    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,
                colLabels=df_with_index_s.columns, cellLoc='center', loc='center')
1168            # Add title
1169            ax.set_title(title)
1170        tight_layout()
1171
                savefig('C:/Code/Python/Machine_Learning_AI/Model_Analysis/Summary/Confused_Matrices_
                Summary.pdf',format='pdf')
1172
                savefig('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Summary/Confus
                ed_Matrices_Summary.pdf',format='pdf')
1173        close()
1174        subject = 'Analysis Complete'
1175        body = f"""\
1176                Summary Scores:\n{model_scores.iloc[[1,3,5,7,9,11]]}\n
1177                Exceptions Scores:\n{model_scores.iloc[[0,2,4,6,8,10]]}
1178            """
1179        #send_email_to_self(subject,body)
1180
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 =
                DataFrame(features_test_scaled.copy(),columns=features_test_scaled.columns)
1187        features_test_with_excluded_features =
                DataFrame(round(scaler.inverse_transform(features_test_with_excluded_features),2).ast
                ype(float),columns=features_test_scaled.columns)
1188        features_test_with_excluded_features[list(excluded_features.columns)] =
                excluded_features.loc[features_test_scaled.index]
1189        if True:
1190            df_with_discrepancies =
                    DataFrame(features_test_with_excluded_features,columns=features_test_with_exclude
                    d_features.columns).reset_index(drop=True)
1191            df_with_discrepancies['E_checked'] =
                    Series(target_test_E).reset_index(drop=True).replace({0:'x',1:'TRUE'})
1192            df_with_discrepancies['DecisionTree_E_checked'] =
                    Series(dt_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRU
                    E'})
1193            df_with_discrepancies['RandomForest_E_checked'] =
                    Series(rf_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRU
                    E'})
1194            df_with_discrepancies['GradientBoost_E_checked'] =
                    Series(gb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRU
                    E'})
1195            df_with_discrepancies['LGBM_E_checked'] =
                    Series(lgbm_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'T
                    RUE'})
1196            df_with_discrepancies['XGBoost_E_checked'] =
                    Series(xgb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TR
                    UE'})
1197            df_with_discrepancies['CatBoost_E_checked'] =
                    Series(cb_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TRU
                    E'})
1198            df_with_discrepancies['KNeighbors_E_checked'] =
                    Series(knn_e_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'TR
                    UE'})
1199            df_with_discrepancies['E_OrdStartDate_ATG'] =
                    to_datetime(df_with_discrepancies['E_OrdStartDate_ATG'])
1200            df_with_discrepancies['E_OrdEndDate_ATG'] =
                    to_datetime(df_with_discrepancies['E_OrdEndDate_ATG'])
1201            df_with_discrepancies['E_ReceivingDate'] =
                    to_datetime(df_with_discrepancies['E_ReceivingDate'])
1202            df_with_discrepancies['E_InvoicedDate'] =
                    to_datetime(df_with_discrepancies['E_InvoicedDate'])
1203            df_with_discrepancies['E_DateStartArrival_ATG'] =
```

```
        to_datetime(df_with_discrepancies['E_DateStartArrival_ATG'])
1204        df_with_discrepancies['E_DateEndArrival_ATG'] =
            to_datetime(df_with_discrepancies['E_DateEndArrival_ATG'])
1205        df_with_discrepancies['E_AddDate'] =
            to_datetime(df_with_discrepancies['E_AddDate'])
1206        df_with_discrepancies['E_PODate'] =
            to_datetime(df_with_discrepancies['E_PODate'])
1207        df_with_discrepancies['E_AP_CheckDate'] =
            to_datetime(df_with_discrepancies['E_AP_CheckDate'])
1208        df_with_discrepancies['E_Dept'] = df_with_discrepancies['E_Dept'].astype(int)
1209    if True:
1210        df_with_discrepancies['S_checked'] =
            Series(target_test_E).reset_index(drop=True).replace({0:'x',1:'claim'})
1211        df_with_discrepancies['DecisionTree_S_checked'] =
            Series(dt_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
            im'})
1212        df_with_discrepancies['RandomForest_S_checked'] =
            Series(rf_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
            im'})
1213        df_with_discrepancies['GradientBoost_S_checked'] =
            Series(gb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
            im'})
1214        df_with_discrepancies['LGBM_S_checked'] =
            Series(lgbm_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'c
            laim'})
1215        df_with_discrepancies['XGBoost_S_checked'] =
            Series(xgb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cl
            aim'})
1216        df_with_discrepancies['CatBoost_S_checked'] =
            Series(cb_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cla
            im'})
1217        df_with_discrepancies['KNeighbors_S_checked'] =
            Series(knn_s_predictions.reshape(-1)).reset_index(drop=True).replace({0:'x',1:'cl
            aim'})
1218        df_with_discrepancies['S_OrdStartDate_ATG'] =
            to_datetime(df_with_discrepancies['S_OrdStartDate_ATG'])
1219        df_with_discrepancies['S_OrdEndDate_ATG'] =
            to_datetime(df_with_discrepancies['S_OrdEndDate_ATG'])
1220        df_with_discrepancies['S_DateStartArrival_ATG'] =
            to_datetime(df_with_discrepancies['S_DateStartArrival_ATG'])
1221        df_with_discrepancies['S_DateEndArrival_ATG'] =
            to_datetime(df_with_discrepancies['S_DateEndArrival_ATG'])
1222        df_with_discrepancies['S_AddDate'] =
            to_datetime(df_with_discrepancies['S_AddDate'])
1223        df_with_discrepancies['S_ClaimDate'] =
            to_datetime(df_with_discrepancies['S_ClaimDate'])
1224        df_with_discrepancies['S_PromoStartDate_ATG'] =
            to_datetime(df_with_discrepancies['S_PromoStartDate_ATG'])
1225        df_with_discrepancies['S_PromoEndDate_ATG'] =
            to_datetime(df_with_discrepancies['S_PromoEndDate_ATG'])
1226
    df_with_discrepancies[~(df_with_discrepancies['E_checked']==df_with_discrepancies['Ca
    tBoost_E_checked'])|
1227
                          ~(df_with_discrepancies['S_checked']==df_with_discrepancies['CatB
                          oost_S_checked'])].head()
1228
1229    subject = 'Posting Discrepancies'
1230    body = f"""
1231        """
1232    #send_email_to_self(subject,body)
1233
    df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepanc
    ies['CatBoost_E_checked'].values)|
1234
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['XGBoost_E_checked'].values)|
1235
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
```

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

```
        ['S_ATG_Ref','E_ATG_Ref','E_checked','DecisionTree_E_checked','RandomForest_E_che
        cked','GradientBoost_E_checked','LGBM_E_checked',
1261
        'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked']].reset_index(dro
        p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/ATG_Refs_Only/Ex
        ceptions/Discrepancies_Between_Original_E_Checked_and_Model_Predictions.csv',inde
        x=False)
1262
    df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepanc
    ies['CatBoost_S_checked'].values)|
1263
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['XGBoost_S_checked'].values)|
1264
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['RandomForest_S_checked'].values)|
1265
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['DecisionTree_S_checked'].values)|
1266
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['LGBM_S_checked'].values)|
1267
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['GradientBoost_S_checked'].values)|
1268
                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['KNeighbors_S_checked'].values)
1269                      ][
1270
        ['S_ATG_Ref','E_ATG_Ref','S_checked','DecisionTree_S_checked','RandomForest_S_che
        cked','GradientBoost_S_checked','LGBM_S_checked',
1271
        'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(dro
        p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/ATG_Refs_Only/Su
        mmary/Discrepancies_Between_Original_S_Checked_and_Model_Predictions.csv',index=F
        alse)
1272
    df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepanc
    ies['CatBoost_E_checked'].values)|
1273
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['XGBoost_E_checked'].values)|
1274
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['RandomForest_E_checked'].values)|
1275
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['DecisionTree_E_checked'].values)|
1276
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['LGBM_E_checked'].values)|
1277
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['GradientBoost_E_checked'].values)|
1278
                          ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                          s['KNeighbors_E_checked'].values)
1279                      ][
1280
        list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',
        ascending=False).index[:15])+[
1281      'E_checked','DecisionTree_E_checked','RandomForest_E_checked',
1282
        'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked']].reset_index(dro
        p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
        eatures_Only/Exceptions/Discrepancies_Between_Original_E_Checked_and_Model_Predic
        tions.csv',index=False)
1283
```

```
df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values)#|
                      #~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)
                      ][
    list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary',ascending=False).index[:15])+[
    'S_checked','DecisionTree_S_checked','RandomForest_S_checked','GradientBoost_E_checked','LGBM_E_checked',
    '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)
df_with_discrepancies[(df_with_discrepancies['E_checked']=='x')&
                      ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['CatBoost_E_checked'].values)|
                      ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['XGBoost_E_checked'].values)|
                      ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['RandomForest_E_checked'].values)|
                      ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['DecisionTree_E_checked'].values)|
                      ~(df_with_discrepancies['E_checked'].values==df_with_discrepancies['KNeighbors_E_checked'].values)][
    list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',ascending=False).index[:15])+[
    'E_checked','DecisionTree_E_checked','RandomForest_E_checked','GradientBoost_E_checked','LGBM_E_checked',
    '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)
df_with_discrepancies[(df_with_discrepancies['S_checked']=='x')&
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['CatBoost_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['XGBoost_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['RandomForest_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values)|
                      ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)][
```

```
        list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary',asc
        ending=False).index[:15])+[
        'S_checked','DecisionTree_S_checked','RandomForest_S_checked',

        'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(dro
        p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
        eatures_Only/Summary/x_Discrepancies_Between_Original_S_Checked_and_Model_Predict
        ions.csv',index=False)

    df_with_discrepancies[(df_with_discrepancies['E_checked']=='x')&(df_with_discrepancie
    s['S_checked']=='x')
                          &
                          (

                              ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                              ncies['CatBoost_E_checked'].values)|

                              ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                              ncies['XGBoost_E_checked'].values)|

                              ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                              ncies['RandomForest_E_checked'].values)|

                              ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                              ncies['DecisionTree_E_checked'].values)|

                              ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                              ncies['KNeighbors_E_checked'].values)
                          )
                          &
                          (

                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                              ncies['CatBoost_S_checked'].values)|

                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                              ncies['XGBoost_S_checked'].values)|

                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                              ncies['RandomForest_S_checked'].values)|

                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                              ncies['DecisionTree_S_checked'].values)|

                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                              ncies['KNeighbors_S_checked'].values)
                          )][

        list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',
        ascending=False).index[:15])+[
        'E_checked','DecisionTree_E_checked','RandomForest_E_checked',
        'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked',
        'S_checked','DecisionTree_S_checked','RandomForest_S_checked',

        'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(dro
        p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
        eatures_Only/x_Discrepancies_Between_Original_BOTH_Checked_and_Model_Predictions.
        csv',index=False)
    df_with_discrepancies[(df_with_discrepancies['S_checked']=='claim')&

                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['CatBoost_S_checked'].values)|

                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['XGBoost_S_checked'].values)|

                          ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                          s['RandomForest_S_checked'].values)|
```

```
                        ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                        s['DecisionTree_S_checked'].values)|
                        ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                        s['KNeighbors_S_checked'].values)][
    list(excluded_features)+list(combined_importances_s.sort_values('SUM_Summary',asc
    ending=False).index[:15])+[
    'S_checked','DecisionTree_S_checked','RandomForest_S_checked',
    'XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_checked']].reset_index(dro
    p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
    eatures_Only/Summary/Claim_Discrepancies_Between_Original_S_Checked_and_Model_Pre
    dictions.csv',index=False)
df_with_discrepancies[(df_with_discrepancies['E_checked']=='TRUE')&
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['CatBoost_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['XGBoost_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['RandomForest_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['DecisionTree_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['KNeighbors_E_checked'].values)][
    list(excluded_features)+list(combined_importances_e.sort_values('SUM_Exceptions',
    ascending=False).index[:15])+[
    'E_checked','DecisionTree_E_checked','RandomForest_E_checked',
    'XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_checked']].reset_index(dro
    p=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/Most_Important_F
    eatures_Only/Exceptions/True_Discrepancies_Between_Original_E_Checked_and_Model_P
    redictions.csv',index=False)
df_with_discrepancies[~(df_with_discrepancies['E_checked'].values==df_with_discrepanc
ies['CatBoost_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['XGBoost_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['RandomForest_E_checked'].values)|
                        ~(df_with_discrepancies['E_checked'].values==df_with_discrepancie
                        s['DecisionTree_E_checked'].values)#|
                        #~(df_with_discrepancies['E_checked'].values==df_with_discrepanci
                        es['KNeighbors_E_checked'].values)
                        ].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_
                        AI/Discrepancies/All_Features/Exceptions/Discrepancies_Between_Or
                        iginal_E_Checked_and_Model_Predictions_ALL_FEATURES.csv',index=Fa
                        lse)
df_with_discrepancies[~(df_with_discrepancies['S_checked'].values==df_with_discrepanc
ies['CatBoost_S_checked'].values)|
                        ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                        s['XGBoost_S_checked'].values)|
                        ~(df_with_discrepancies['S_checked'].values==df_with_discrepancie
                        s['RandomForest_S_checked'].values)|
```

```
                              ~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['DecisionTree_S_checked'].values)#|
                              #~(df_with_discrepancies['S_checked'].values==df_with_discrepancies['KNeighbors_S_checked'].values)
                              ].reset_index(drop=True).to_csv('C:/Code/Python/Machine_Learning_AI/Discrepancies/All_Features/Summary/Discrepancies_Between_Original_S_Checked_and_Model_Predictions_ALL_FEATURES.csv',index=False)

    server = 'barney'
    database = 'sandbox_mp'
    connectionString = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Integrated Security={True};Autocommit={True};Trusted_Connection=yes;'
    conn = connect(connectionString)
    cursor_2 = conn.cursor()
    insert_sql = """
            INSERT INTO Weis_Market_Claim_Discrepancies_With_Context_1 (
                ATG_Deal_Summary_ATG_Ref,
                ATG_DealLI_Exceptions_ATG_Ref,
                ATG_DealLI_Exceptions_DL_ATG_Ref,
                ATG_DealLI_Exceptions_LI_ATG_Ref,
                ATG_DealLI_Exceptions_HDR_ATG_Ref,
                ATG_Deal_Summary_BatchNbr,
                ATG_DealLI_Exceptions_BatchNbr,
                ATG_Deal_Summary_DealNbr,
                ATG_Deal_Summary_OrdStartDate_ATG,
                ATG_Deal_Summary_OrdeNDDate_ATG,
                ATG_Deal_Summary_DLVendorNbr,
                ATG_Deal_Summary_AddDate,
                ATG_Deal_Summary_ClaimType,
                ATG_Deal_Summary_DealVendorName,
                ATG_Deal_Summary_AP_VndNbr,
                ATG_Deal_Summary_PurVndrNbr,
                ATG_Deal_Summary_PurVndrName,
                ATG_DealLI_Exceptions_ItemNbr,
                ATG_DealLI_Exceptions_PODate,
                ATG_DealLI_Exceptions_ReceivingDate,
                ATG_DealLI_Exceptions_InvoicedDate,
                ATG_DealLI_Exceptions_ClaimType,
                ATG_DealLI_Exceptions_UPCNbr,
                ATG_DealLI_Exceptions_UPCUnit,
                ATG_DealLI_Exceptions_PurVndrName,
                ATG_DealLI_Exceptions_ReceiptNbr,
                ATG_DealLI_Exceptions_Checked_Before_AI,
                DecisionTree_Model_Predictions_ATG_DealLI_Exceptions_checked,
                RandomForest_Model_Predictions_ATG_DealLI_Exceptions_checked,
                GradientBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
                LGBM_Model_Predictions_ATG_DealLI_Exceptions_checked,
                XGBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
                CatBoost_Model_Predictions_ATG_DealLI_Exceptions_checked,
                KNeighbors_Model_Predictions_ATG_DealLI_Exceptions_checked,
                ATG_Deal_Summary_Checked_Before_AI,
                DecisionTree_Model_Predictions_ATG_Deal_Summary_checked,
                RandomForest_Model_Predictions_ATG_Deal_Summary_checked,
                GradientBoost_Model_Predictions_ATG_Deal_Summary_checked,
                LGBM_Model_Predictions_ATG_Deal_Summary_checked,
                XGBoost_Model_Predictions_ATG_Deal_Summary_checked,
                CatBoost_Model_Predictions_ATG_Deal_Summary_checked,
                KNeighbors_Model_Predictions_ATG_Deal_Summary_checked
            ) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);
        """
    df_with_discrepancies[
                              (
```

```
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['CatBoost_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['XGBoost_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['LGBM_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['GradientBoost_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['RandomForest_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['DecisionTree_E_checked'].values)&
                                    ~(df_with_discrepancies['E_checked'].values==df_with_disc
                                    repancies['KNeighbors_E_checked'].values)
                                )
                                    |
                                (
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['CatBoost_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['XGBoost_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['LGBM_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['GradientBoost_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['RandomForest_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['DecisionTree_S_checked'].values)&
                                    ~(df_with_discrepancies['S_checked'].values==df_with_disc
                                    repancies['KNeighbors_S_checked'].values)
                                )

                            ].to_csv('C:/Code/Python/Machine_Learning_AI/TESTING_DISCREPANCIE
                            S.csv')
    data = df_with_discrepancies[(
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['CatBoost_E_checked'].values)&
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['XGBoost_E_checked'].values)&
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['LGBM_E_checked'].values)&
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['GradientBoost_E_checked'].values)&
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['RandomForest_E_checked'].values)&
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['DecisionTree_E_checked'].values)&
```

```
                                ~(df_with_discrepancies['E_checked'].values==df_with_discrepa
                                ncies['KNeighbors_E_checked'].values)
                            )
                            |
                            (

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['CatBoost_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['XGBoost_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['LGBM_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['GradientBoost_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['RandomForest_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['DecisionTree_S_checked'].values)&

                                ~(df_with_discrepancies['S_checked'].values==df_with_discrepa
                                ncies['KNeighbors_S_checked'].values)
                            )
                            ][
        ['S_ATG_Ref','E_ATG_Ref','E_ATG_DL_Ref','E_ATG_LI_Ref','E_ATG_HDR_Ref','S_BatchNb
        r','E_BatchNbr',

            'S_DealNbr','S_OrdStartDate_ATG','S_OrdEndDate_ATG','S_DLVendorNbr','S_AddDat
            e','S_ClaimType_ATG',
            'S_DealVendorName','S_AP_VndNbr','S_PurVndrNbr','S_PurVndrName',
            'E_ItemNbr','E_PODate','E_ReceivingDate','E_InvoicedDate','E_ClaimType',
            'E_UPCNbr','E_UPCUnit','E_PurVndrName','E_ReceiptNbr',

            'E_checked','DecisionTree_E_checked','RandomForest_E_checked','GradientBoost_
            E_checked',

            'LGBM_E_checked','XGBoost_E_checked','CatBoost_E_checked','KNeighbors_E_check
            ed',

            'S_checked','DecisionTree_S_checked','RandomForest_S_checked','GradientBoost_
            S_checked',

            'LGBM_S_checked','XGBoost_S_checked','CatBoost_S_checked','KNeighbors_S_check
            ed'
        ]].reset_index(drop=True)
    for col in data.columns:
        data[col] = data[col].astype(str)
    for row in data.index:
        current_row = list(data.iloc[int(row)])
        cursor_2.execute(insert_sql, current_row)
    conn.commit()

    folder_path = "C:/Code/Python/Machine_Learning_AI"  # replace with your folder path
    output_zip_file =
    f'C:/Code/Python/Machine_Learning_AI_Batch{batch_number}_Completed.zip'  # replace
    with your desired zip file name
    compress_folder_to_zip(folder_path, output_zip_file)

    port = 465  # For SSL
    smtp_server = "smtp.gmail.com"
    sender_email = "micpowers98@gmail.com"  # Enter your address
    receiver_email = "micpowers98@gmail.com"  # Enter receiver address
    password = 'efex cwhv gppq ueob'
    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
        WORK NOW!!!")
1495    #     print(str(e))
```