**APPEND COL IN NEW FILE WHILE FILTERING ALSO READING ALL SHEETS**

import pandas as pd  
import openpyxl  
  
testcase\_file = "Payments.xlsx"  
theme\_file = "THEME\_DASHBOARD\_OUTPUT.xlsx"  
  
def expectedcol\_append(theme,test):  
 all\_sheets = pd.read\_excel(theme,sheet\_name=None)  
 df\_testcase = pd.read\_excel(test,sheet\_name='Sheet1')  
  
 updated\_sheets = {}  
 for sheet\_name,df in list(all\_sheets.items())[1:]:  
 df['Expected\_Results'] = ""  
  
 for index,row in df.iterrows():  
 testcase\_id = row['TestCaseID']  
  
 matching\_rows = df\_testcase[df\_testcase['TestCaseNum']==testcase\_id]  
  
 if not matching\_rows.empty:  
 last\_row = matching\_rows.iloc[-1]  
 sentence = last\_row['Sentence']  
  
 df.at[index,'Expected\_Results'] = sentence  
  
 updated\_sheets[sheet\_name] = df  
  
 with pd.ExcelWriter('UPADTED\_THEME.xlsx') as writer:  
 for sheet\_name,df in updated\_sheets.items():  
 df.to\_excel(writer,sheet\_name=sheet\_name,index=False)  
 print(f'{sheet\_name} appended to updated excel file')  
  
  
expectedcol\_append(theme\_file,testcase\_file)

**APPEND MANY EXCEL OF SAME SIZE DF INTO ONE**

def append\_to\_excel(excel\_files):  
 df\_list =[]  
  
 for excel in excel\_files:  
 df = pd.read\_excel(excel)  
 df\_list.append(df)  
  
 combined\_df = pd.concat(df\_list,ignore\_index=True)  
 combined\_df.to\_excel("Payments.xlsx")

# Here pd concat by default concatenates the df vertically

**UTIL FUNCTIONS NLP**

def count\_and\_sort\_words(word\_list):  
 word\_counts = Counter(word\_list)  
 sorted\_word\_counts = dict(sorted(word\_counts.items(), key=lambda item: item[1], reverse=True))  
 return sorted\_word\_counts  
  
def clean\_sentence(sentence, stop\_words):  
 words = word\_tokenize(sentence)  
 filtered\_sentence = [word for word in words if word.lower() not in stop\_words]  
 return ' '.join(filtered\_sentence)  
  
def remove\_stop\_words(sentences):  
 stop\_words = set(stopwords.words('english'))  
 cleaned\_sentences = [clean\_sentence(sentence, stop\_words) for sentence in sentences]  
 return cleaned\_sentences  
  
def sentences\_to\_json(sentences):  
 data = {f"sentence\_{i + 1}": sentence for i, sentence in enumerate(sentences)}  
 with open('unique\_testcase','w') as json\_file:  
 json\_data = json.dump(data,json\_file, indent=4)  
  
 return json\_data  
  
def remove\_punctuation(sentence):  
 translator = str.maketrans('', '', string.punctuation)  
 return sentence.translate(translator)  
  
  
def remove\_stopwords(sentences):  
 stop\_words = set(stopwords.words('english'))  
 filtered\_sentences = []  
 for sentence in sentences:  
 words = word\_tokenize(sentence)  
 filtered\_sentence = [word for word in words if word.lower() not in stop\_words]  
 filtered\_sentences.append(' '.join(filtered\_sentence))  
 return filtered\_sentences  
  
def removestopfrom\_sentence(sentence):  
 stop\_words = set(stopwords.words('english'))  
 words = word\_tokenize(sentence)  
 filter\_sentence = [word for word in words if word.lower() not in stop\_words]  
 return ' '.join(filter\_sentence)  
  
from textblob import TextBlob  
  
def correct\_spelling(text):  
 blob = TextBlob(text)  
 correct\_text = blob.correct()  
 return str(correct\_text)  
  
def remove\_stop\_words\_and\_single\_chars(word\_list):  
 stop\_words = set(stopwords.words('english'))  
 filtered\_words = [word for word in word\_list if word not in stop\_words and len(word) > 1]  
  
 return filtered\_words  
  
def cleansen(sentence):  
 return sentence.replace(r'\n',' ')  
  
def remove\_duplicates(sentences):  
 unique\_sentences = list(set(sentences))  
 return unique\_sentences  
  
def remove\_empty\_sentences(sentences):  
 return [sentence for sentence in sentences if sentence.strip()]  
  
def remove\_specific\_words(word\_list, words\_to\_remove):  
 removal\_set = set(words\_to\_remove)  
 filtered\_words = [word for word in word\_list if word not in removal\_set]  
  
 return filtered\_words  
  
def convert\_to\_individual\_words(sentences):  
 words = []  
 for sentence in sentences:  
 words.extend(word\_tokenize(sentence))  
 return words  
  
def word\_frequencies(words):  
 frequency = defaultdict(int)  
 for word in words:  
 frequency[word.lower()] += 1  
 return dict(frequency)

**EXTRACTION STEPS WITH INDEX LABEL**

def steps\_to\_list(step\_summary):  
 steps\_list = re.split(r'\d+[\.)]\s\*', step\_summary)  
 steps\_list = [step.replace('\n', '').strip() for step in steps\_list if step.strip()]

the step.strip() strips the trailing whitespaces so if there’s only whitespaces then it will return empty string so that’s why the condition  
  
 return steps\_list

**REMOVE PUNCTUATION**

translator=str.maketrans('','',string.punctuation)

**CHANGING DF VALUES BASED ON OTHER VAL**

df.loc[df['Sentence'].str.contains('contact page',case=False,na=False),'ScreenName'] = 'contact'

df.loc[(df['Sentence'].str.contains('on CS', case=False, na=False))&(df['ScreenName'] == ''), 'ScreenName'] = 'CS'

df['ExpectedResults'] = df['Action'].fillna('') + ' ' + df['FieldName'].fillna('')  
df['ExpectedResults'] = df['ExpectedResults'].str.strip()  
df['ScreenName'] = df['ScreenName'].fillna(method='ffill') // imp  
df = df.fillna("<EMPTY>")

df['ScreenName'] = df.apply(lambda row: extractPatter(row), axis=1)

def extractPatter(row):  
 if pd.isna(row['ScreenName']) or row['ScreenName'] == "" or row['ScreenName'] == None:  
 mysent = str(row['Sentence'])  
 if mysent.rfind(">>")!=-1:  
 x = mysent.rfind(">>")  
 if mysent[x+2:].strip() not in ['Add','Approve','Reject']:  
 return mysent[x+2:].strip()  
 else:  
 y = find\_second\_last(mysent,">>")  
 return mysent[y+2:x].strip()  
 return row['ScreenName']  
 return row['ScreenName']

def find\_second\_last(text, pattern):  
 return text.rfind(pattern, 0, text.rfind(pattern))

duplicate\_condition = df.duplicated(subset=['Action', 'FieldName', 'DataField','ScreenName'], keep=False) & ~df.duplicated(  
 subset=['Action', 'FieldName', 'DataField', 'Sentence','ScreenName'], keep=False)  
df.loc[duplicate\_condition, 'Flag'] = 'Duplicate'

def remove\_actions\_from\_excel(input\_file, output\_file, sheet\_name, column, actions\_to\_remove):  
 df = pd.read\_excel(input\_file, sheet\_name=sheet\_name)  
 df[column] = df[column].apply(lambda x: ','.join([action for action in str(x).split(',') if action.lower() not in actions\_to\_remove]))  
 df.to\_excel(output\_file, sheet\_name=sheet\_name, index=False)

def screenbyTest(input\_excel):  
 df = pd.read\_excel(input\_excel)  
 result = df.groupby('TestCaseNum')['ScreenName'].agg(lambda x: list(x.unique())).reset\_index()  
 result['unique\_screen\_count'] = result['ScreenName'].apply(len)  
  
 print(result)  
 result.to\_excel('ScreenSeq.xlsx')