Data Wrangling Mini Project

For my Capstone Project #1, I downloaded a data with approximately 200.000 observations in 100 columns in which has texts in some of them. I am following the explained steps below:

1. The website had a limitation of 5000 observation for each download. In the query page, I chose date criteria for all data to be downloaded. The first file was 2019. Then I chose January-June in the first chunk, and July-December in the second chunk each year. As a result, except 2019, 1999, 1995 and 1988, all years have two files with approximately 3500 observations. 1999 and 1995 have three files. In total 64 csv files obtained with exact same formatting.

2. All files were combined by using following commands with a for loop and can be executed like;

```
 filenames=glob('database/*.csv')
data=[]
for f in filenames:
   df=pd.read_csv(f, header=[0,1], delimiter=',')
   data.append(df)
final_df = pd.concat(data, axis=0, ignore_index=True)
final_df.shape
```

3. I got rid of the duplicates and unnecessary columns with the following command;

```
df_nodup=final_df.drop_duplicates()
df_nodup.shape
df1=df_nodup.drop(columns=['Work Environment Factor','RVR.Single Value','Aircraft Zone', 'Maintenance Status.Maintenance Deferred','Maintenance Status.Records Complete', 'Maintenance Status.Released For Service','Maintenance Status.Required / Correct Doc On Board','Maintenance Status.Maintenance Type','Maintenance Status.Maintenance Items Involved', 'Cabin Lighting','Crew Size Flight Attendant.Number Of Crew','Callback','When Detected'], level=1)

df1=df_nodup.drop(columns=['Report 2','Unnamed'], level=0)
```

4. In order to classify the data frame according to “Events” column, I created multiple data frames. The classification of the data is as follows:
    a. Airspace conflict issues,
    b. Abnormal equipment/activity due to emergency situation,
    c. Flight cabin event,
    d. Ground issues.

    In events column 55 different explanation word taxonomy used. Each taxonomy with different combination falls under one of the classifications above. I wrote a function for this categorization as below and applied it for Anomaly column:

```
Types=['Ground','Emergency','Conflict','Cabin']
def categorize(dataframe,column_name,new_column='Type'):
    dataframe[new_column]='Not Categorized'
    dataframe.index.fillna('Empty')
    dataframe.fillna('Empty')
    for index, row in dataframe.iterrows():
        if 'Equipment' in dataframe[column_name][index]:
            dataframe[new_column][index]='Emergency'
        elif 'Ground' in dataframe[column_name][index]:
            dataframe[new_column][index]='Ground'
        elif 'Clearence' in dataframe[column_name][index]:
            dataframe[new_column][index]='Conflict'
        elif 'Conflict' in dataframe[column_name][index]:
            dataframe[new_column][index]='Conflict'
        elif 'Deviation' in dataframe[column_name][index]:
            dataframe[new_column][index]='Conflict'
        elif 'Cabin' in dataframe[column_name][index]:
            dataframe[new_column][index]='Cabin'
        else:
            dataframe[new_column][index]='Not Categorized'
```

5. I created a new dataframe with only “conflict” kind of safety issues which are including air traffic conflicts.

Conflict	91669
Emergency	65320
Ground	23084
Not Categorized	20544
Cabin	1908

6. In order to get rid of two level column names and get more useful column names, I applied the following code:

```
column_names=list(df_c1.columns)
del column_names[-1]
del column_names[-1]
df_new={}
for l1,l2 in column_names:
    if df_c1[l1,l2].dtype=='object':
        df_t=df_c1[l1,l2].fillna('None')
        column_t=str(l1)+'_'+str(l2)
        df_new[column_t]=df_t
    if df_c1[l1,l2].dtype=='float64' or df_c1[l1,l2].dtype=='int64':
        df_t=df_c1[l1,l2].fillna(0)
        column_t=str(l1)+'_'+str(l2)
        df_new[column_t]=df_t
len(df_new)
column_new_names=['_No','Month', 'Time', 'Place_refer', 'State', 'Radial', 'Distance', 'AGL',
'MSL', 'MCondition', 'Visibility','Light', 'Ceiling', 'AC1_ATC', 'AC1_Operator', 'AC1_Model',
```

```
'AC1_Crew', 'AC1_Rule', 'AC1_FP', 'AC1_Mission', 'AC1_Nav', 'AC1_Phase', 'AC1_Route',
'AC1_Airspace', 'AC1_Seats','AC1_Passengers','AC2_ATC','AC2_Operator', 'AC2_Model',
'AC2_Crew', 'AC2_Rule', 'AC2_FP', 'AC2_Mission', 'AC2_Nav', 'AC2_Phase',
'AC2_Route','AC2_Airspace','AC2_Seats', 'AC2_Passengers', 'P1_Loc', 'P1_Org', 'P1_Func',
'P1_Qual','P1_Experience','P1_HumaFactor','P2_Loc','P2_Org', 'P2_Func', 'P2_Qual',
'P2_Experience', 'Anomaly', 'Detector', 'Result', 'Other_Factors', 'Pri_Problem', 'Narrative',
'Synopsis']
data_clean.columns=column_new_names
data_clean.head()
```

7. For missing values; I used the “categorize” function that I defined in 6th bullet. If the Dtype of the column is object then the empty cells will be filled with “None”. It the column is float or integer the empty cells are filled with 0. The concerning lines are as follows:

```
if df_c1[l1,l2].dtype=='object':
        df_t=df_c1[l1,l2].fillna('None')
        column_t=str(l1)+'_'+str(l2)
        df_new[column_t]=df_t
    if df_c1[l1,l2].dtype=='float64' or df_c1[l1,l2].dtype=='int64':
        df_t=df_c1[l1,l2].fillna(0)
        column_t=str(l1)+'_'+str(l2)
        df_new[column_t]=df_t
```

8. For each column I applied different codes to clean data. Outliers of each column treated differently. If there is an error, it needs to be checked individually. Examples of some columns are as follows:

```
empty_time=['None','ZZZ']
for item in empty_time:
  for index,row in df.iterrows():
    if df.Time[index]==item:
      if df.Light[index]=='Daylight':
        df.Time[index]=='0601-1200'
      elif df.Light[index]=='Night':
        df.Time[index]=='0001-0600'
…….
df.AGL=pd.to_numeric(df.AGL,errors='coerce')
df.MSL=pd.to_numeric(df.MSL,errors='coerce')
…….
df.State=df.State.str.upper()
df[df.State=='US.AIRPORT']['State']
df.loc[40984, 'State']='US'
…….
position=df[~((df.Distance==0) & (df.Radial==0))][['Distance','Radial','Altitude']]
…….
Phase_df=df.AC1_Phase.str.split(';')
Phase_df=Phase_df.str.get(0)
df['AC1_Phase']=Phase_df
df.AC1_Phase.value_counts().head(11)
```

```
……..
Airspace=df[~(df.AC1_Airspace=='None')]['AC1_Airspace'].value_counts()
#Airspace.index=Airspace.index.str.split('')
Airspace.index=Airspace.index.str.get(6)
Airspace=Airspace.groupby(Airspace.index).sum().sort_values(ascending=False).head(6)
Airspace.index='Class '+Airspace.index
Airspace= Airspace.sort_index()
```