```
def dep_queue_time(rwy='20R', rhp=['20R_W'], speed_at_queue=0.5):

    import sys
    import pandas as pd
    import numpy as np
    from datetime import datetime, timedelta

    # import script for geofencing
    sys.path.insert(0, 'D:/CAG/Projects/_Database/Ad hoc codes/if_inside_polygon')
    from if_inside_polygon import is_inside_polygon

    twy_coor_full = pd.read_csv("TWY Locality.csv")
    rwy_to_include = rwy
    rhp_to_include = rhp

    # Get take off direction locality
    rwy_coor = twy_coor_full[twy_coor_full['taxiway']==rwy_to_include]

    lst_rwy_coor = []
    for index, rows in rwy_coor.iterrows():
        temp =[rows.taxiway, rows.X1, rows.Y1, rows.X2, rows.Y2, rows.X3, rows.Y3, rows.X4, rows.Y4]
        lst_rwy_coor.append(temp)

    # Match coordinates to take off direction
    acft_takeoff_trail = acft_trail_trimmed[(acft_trail_trimmed['ACTIVITY']=='TAKEOFF')]
    acft_takeoff_trail['TWY'] = "ERR"

    for twy in lst_rwy_coor:
        twy_boundary = [(twy[1], twy[2]),
                        (twy[3], twy[4]),
                        (twy[5], twy[6]),
                        (twy[7], twy[8])]
        twy_name = twy[0]

        acft_takeoff_trail['TWY'] = [twy_name if is_inside_polygon(points=twy_boundary, p=(x, y))
                                     else z
                                     for x,y,z
                                     in zip(acft_takeoff_trail['LAT'],
                                            acft_takeoff_trail['LON'],
                                            acft_takeoff_trail['TWY'])]

    lst_acft_takeoff = acft_takeoff_trail[acft_takeoff_trail['TWY']!='ERR'].URNO.unique().tolist()

    # Get RHP locality
    rhp_coor = twy_coor_full[twy_coor_full['taxiway'].isin(rhp_to_include)]

    lst_rhp_coor = []
    for index, rows in rhp_coor.iterrows():
        temp =[rows.taxiway, rows.X1, rows.Y1, rows.X2, rows.Y2, rows.X3, rows.Y3, rows.X4, rows.Y4]
        lst_rhp_coor.append(temp)

    # Work only on aircrafts taking off from selected take off direction
    acft_dep_trail = acft_trail_trimmed[(acft_trail_trimmed['ACTIVITY']=='DEPTAXI')]
    acft_dep_trail = acft_dep_trail[acft_dep_trail['URNO'].isin(lst_acft_takeoff)]

```

```
    acft_dep_trail['TWY'] = "ERR"

    # Match coordinates to RHP queue area
    for twy in lst_rhp_coor:
        twy_boundary = [(twy[1], twy[2]),
                        (twy[3], twy[4]),
                        (twy[5], twy[6]),
                        (twy[7], twy[8])]
        twy_name = twy[0]

        acft_dep_trail['TWY'] = [twy_name if is_inside_polygon(points=twy_boundary,
p=(x, y))
                                 else z
                                 for x,y,z
                                 in zip(acft_dep_trail['LAT'],
                                        acft_dep_trail['LON'],
                                        acft_dep_trail['TWY'])]

    # Get list of aircrafts queued (speed <= 0.3) at the RHP queue area
    acft_queue_trail = acft_dep_trail[
        (acft_dep_trail['TWY']!='ERR') &
        (acft_dep_trail['SPEED']<=0.3)
    ].sort_values(['SPEED']).drop_duplicates('URNO')

    acft_queue_trail = acft_queue_trail[['URNO','LASTUPDATE']]
    acft_queue_trail.columns = ['URNO','queue_start']

    # Compute queue time of each aircraft
    queue_time =
acft_dep_trail_assigned.groupby(['URNO']).agg({'LASTUPDATE':max}).reset_index()
    queue_time = queue_time.merge(right=acft_queue_trail, on='URNO', how='left')
    queue_time['queue_time'] = (queue_time['LASTUPDATE'] -
queue_time['queue_start']).astype('timedelta64[s]')

    queue_time.fillna(0, inplace=True)

    # plot histogram of queue time
    bin_size = 10
    x_limit = round(queue_time['queue_time'].max()-1) + bin_size

    g = sns.displot(data=queue_time[queue_time['queue_time']>0]['queue_time'],
                    bins=[x for x in range(0, x_limit, bin_size)],
                    kind= 'hist',
                    height=5, aspect=2)

    return g

def txy_avg_volume(filename = "2021-05.csv"):

    import sys
    import pandas as pd
    import numpy as np
    from datetime import datetime, timedelta

    # import script for geofencing
    sys.path.insert(0, 'D:/CAG/Projects/_Database/Ad hoc codes/if_inside_polygon')
    from if_inside_polygon import is_inside_polygon

    path = "_raw data\\"
    acft_trail_trimmed = data_cleaning(filename = path + filename)
```

```
113
114     # Analyse only aircraft on ground
115     acft_ground_trail = acft_trail_trimmed[~
((acft_trail_trimmed['ACTIVITY']=='AIRBORNE') |
116
(acft_trail_trimmed['ACTIVITY']=='TAKEOFF'))]
117
118     # Get TWY locality
119     twy_coor_full = pd.read_csv("TWY Locality.csv")
120     twy_to_exclude = [
121         'Exclude',
122         'RET W1', 'RET W2', 'RET W3', 'RET W4', 'RET W5',
123         'RET W6', 'RET W7', 'RET W8', 'RET W9', 'RET W10',
124         '02L_R', '02L', '20R_W', '20R', '02L_V']
125     twy_coor = twy_coor_full[~(twy_coor_full['taxiway'].isin(twy_to_exclude))]
126
127     lst_twy_coor = []
128     for index, rows in twy_coor.iterrows():
129         temp =[rows.taxiway, rows.X1, rows.Y1, rows.X2, rows.Y2, rows.X3, rows.Y3,
rows.X4, rows.Y4]
130         lst_twy_coor.append(temp)
131
132     # Match coordinates to EXCLUDED taxiway sections
133     acft_ground_trail['TWY'] = "ERR"
134
135     exlude_twy =
twy_coor_full.loc[twy_coor_full['taxiway']=='Exclude'].values.flatten().tolist()
136     exclude_boundary = [(exlude_twy[1], exlude_twy[2]),
137                         (exlude_twy[3], exlude_twy[4]),
138                         (exlude_twy[5], exlude_twy[6]),
139                         (exlude_twy[7], exlude_twy[8])]
140
141     twy_name = exlude_twy[0]
142
143     acft_ground_trail['TWY'] = [twy_name if
is_inside_polygon(points=exclude_boundary, p=(x, y))
144                                 else z
145                                 for x,y,z
146                                 in
zip(acft_ground_trail['LAT'],acft_ground_trail['LON'], acft_ground_trail['TWY'])]
147
148     # Match coordinates to taxiway sections
149     twy_assigned = pd.DataFrame(columns=['ID', 'ACTIVITY', 'ADID', 'ACTYPE', 'CSGN',
'FLNO', 'LASTUPDATE', 'LAT',
150                                          'LON', 'REGN', 'SEGID', 'SPEED', 'STAND',
'TRACKID', 'URNO', 'TWY'])
151
152     for twy in lst_twy_coor:
153         twy_boundary = [(twy[1], twy[2]),
154                         (twy[3], twy[4]),
155                         (twy[5], twy[6]),
156                         (twy[7], twy[8])]
157         twy_name = twy[0]
158
159         df_temp = acft_ground_trail[acft_ground_trail['TWY']=="ERR"]
160
161         df_temp['TWY'] = [twy_name if is_inside_polygon(points=twy_boundary, p=(x,
y))
162                           else z
163                           for x,y,z
```

```
                        in zip(df_temp['LAT'],df_temp['LON'], df_temp['TWY'])]

        twy_assigned_temp = df_temp[df_temp['TWY']!="ERR"]
        twy_assigned = pd.concat([twy_assigned, twy_assigned_temp], sort=False)

    twy_assigned.to_csv('2. Output//twy_assigned.csv')

    # Remove duplicates data from the same segment of twy
    twy_assigned['uniqueID'] = twy_assigned['URNO'].astype(str) +
twy_assigned['TWY'].astype(str)

    twy_assigned_cleaned =
twy_assigned[twy_assigned['TWY']!='ERR'].drop_duplicates('uniqueID')
    twy_assigned_cleaned.drop(['uniqueID'], axis = 1, inplace=True)

    twy_assigned_cleaned.sort_values(['URNO','LASTUPDATE'], ascending=True,
inplace=True)

    # Replace missing aircraft type
    lst_missing =
twy_assigned_cleaned[twy_assigned_cleaned['ACTYPE'].isna()].TRACKID.unique().tolist()

    missing_found = acft_trail_trimmed[(~acft_trail_trimmed['ACTYPE'].isna()) &

(acft_trail_trimmed['TRACKID'].isin(lst_missing))].drop_duplicates('TRACKID')
[['TRACKID','ACTYPE']]
    missing_found.columns = ['TRACKID', 'ACTYPE_temp']

    missing = twy_assigned_cleaned[twy_assigned_cleaned['ACTYPE'].isna()]
    not_missing = twy_assigned_cleaned[~twy_assigned_cleaned['ACTYPE'].isna()]

    merge_found = missing.merge(right=missing_found, on='TRACKID', how='left')
    merge_found.drop_duplicates(inplace=True)
    merge_found['ACTYPE'] = merge_found['ACTYPE_temp']
    merge_found.drop(['ACTYPE_temp'], axis= 1, inplace=True)

    twy_assigned_complete = pd.concat([not_missing, merge_found], sort = False,
ignore_index=True)


 twy_assigned_complete[twy_assigned_complete['ACTYPE'].isna()].TRACKID.unique().tolis
t()

    # Export to csv for visualisation
    twy_assigned_complete.to_csv('2. Output//TWY_Volume.csv', index=False)

    twy_volume_1 = twy_assigned_complete.copy()

    # Compute twy average volume by hour and by day of week
    twy_volume_1['weekday'] = twy_volume_1['LASTUPDATE'].dt.weekday
    twy_volume_1['hour'] = twy_volume_1['LASTUPDATE'].dt.hour
    twy_volume_1['day'] = twy_volume_1['LASTUPDATE'].dt.day

    list_twy = twy_volume_1['TWY'].unique().tolist()

    twy_volume_mean = twy_volume_1.groupby(['weekday', 'hour','TWY']).agg(
        {'TWY':np.count_nonzero}).unstack(2).droplevel(0, axis=1).reset_index()

    day_to_weekday =
twy_volume_1.drop_duplicates(['day']).groupby(['weekday']).agg({'ID':np.count_nonzero
```

```
}).reset_index()
    day_to_weekday.columns = ['weekday', 'count']

    twy_volume_mean_BI = twy_volume_mean.melt(id_vars=['weekday', 'hour']).fillna(0)

    twy_volume_mean_BI = twy_volume_mean_BI.merge(right=day_to_weekday, on='weekday', how='left')
    twy_volume_mean_BI['mean'] = twy_volume_mean_BI['value'] / twy_volume_mean_BI['count']
    twy_volume_mean_BI['mean'] = twy_volume_mean_BI['mean'].round(0)

    twy_volume_mean_BI.to_csv('twy_vol_' + filename, index=False)

    return
```