SCHEDULE METRICS

THOMAS EAPEN

2018

Contents

List of Figures

1 REPORTING & ANALYTICS

1.1 SCHEDULE METRICS

DELIVERABLES

1. Generate schedule metrics such as schedule time, exception time, scheduled vs actual productive time, sign-in%, shrinkage, occupancy

1.1.1 DESIGN

The design is a pipeline staged execution where each stage is in it's own function.

1.1.2 STAGE 1: READ AND CLEAN DATA

The data import and cleaning involved cleaning up the date formats that are generated by GWFM to proper timestamps for time delta calculations in Pandas and Python. An example function in this stage is listed below.

```
def read_and_clean_schedule_isr_data_v1(filename=''):
    df_isr = pd.read_excel(filename)
    df_isr = df_isr[(~df_isr['Schedule State'].str.contains('Day Off')) & (~df_isr['Schedule State'].str.contains('Attrition'))].copy()

    df_isr['Date Filled'] = df_isr['Date']
    df_isr['STC'] = df_isr['Start Time']
    df_isr['ETC'] = df_isr['End Time']

    df_isr['Date Filled'].fillna(method='ffill', inplace=True)
    generic_start_time = datetime.time(hour=9)
    generic_end_time = datetime.time(hour=16, minute=30)
    df_isr['STC'] =  df_isr['STC'].fillna(generic_start_time)
    df_isr['ETC'] =  df_isr['ETC'].fillna(generic_end_time)

    df_isr.sort_values(by=['Agent', 'Start Time'], inplace=True)
    df_isr['STC'] = df_isr['STC'].astype(str)
    df_isr['ETC'] = df_isr['ETC'].astype(str)
    df_isr['Date Filled'] = df_isr['Date Filled'].astype(str)
    #set_trace()
    current_day = str(parse(df_isr.iloc[0]['Date Filled']).date())
    next_day = str((parse(df_isr.iloc[0]['Date Filled']) + datetime.timedelta(days=1)).date())

    #df_isr['STC'] = df_isr['STC'].apply(lambda v: df_isr['Date Filled'] + ' ' + v[1:] if '+' in v else df_isr['Date Filled'] + ' ' + v)
    #df_isr['STC'] = df_isr['STC'].apply(lambda v: df_isr['Date Filled'] + ' ' + '00:00:00' if '+' in v else df_isr['Date Filled'] + ' ' + v)
    df_isr['STC'] = df_isr['STC'].apply(lambda v: next_day + ' ' + '00:00:00' if '+' in v else current_day + ' ' + v)
    #df_isr['ETC'] = df_isr['ETC'].apply(lambda v: df_isr['Date Filled'] + ' ' + v[1:] if '+' in v else df_isr['Date Filled'] + ' ' + v)
    #df_isr['ETC'] = df_isr['ETC'].apply(lambda v: df_isr['Date Filled'] + ' ' + '00:00:00' if '+' in v else df_isr['Date Filled'] + ' ' + v)
    df_isr['ETC'] = df_isr['ETC'].apply(lambda v: next_day + ' ' + '00:00:00' if '+' in v else current_day + ' ' + v)
    df_isr['STC'] = df_isr['STC'].apply(lambda v: parse(v))
    df_isr['ETC'] = df_isr['ETC'].apply(lambda v: parse(v))
    df_isr.loc[df_isr['ETC'] < df_isr['STC'], ['ETC']] = parse(df_isr['Date Filled'][0]) + datetime.timedelta(days=1)
    df_isr['Time Delta'] = df_isr['ETC'] - df_isr['STC']
    #df_isr['Timestamp'] = datetime.datetime.now()
    #df_isr['Code Version'] = 'v1'
    return df_isr.reset_index(drop=True)
```

Listing 1: Code Listing → Schedule Metrics → Stage 1 → Reading and cleaning data example

1.1.3 STAGE 2: CALCULATE SCHEDULE AND ACTUAL PRODUCTIVE TIME

Once the data has been imported and cleaned it is passed through the next stage for group by operations. Once the group by operations are complete, the dataframes are filtered and aggregated to determine scheduled and actual productive data. Since some groups take multiple interactions, overlapping time frames are ignored so as to not provide inflated actual productive times.

```
def scheduled_productive_time(df_isr, df_state, df_state_mapping):
    name_list = []
    schedule_list = []
    exception_list = []
    productive_list = []
    marked_time = ['Banked (due to holiday)', 'Makeup Time', 'Overtime (Banked)', 'Overtime (Paid)', 'Shift Trade']
    agent_groups = df_isr.groupby('Agent')

    state_mapping_array = (df_state_mapping.replace(np.nan, '', regex=True)).as_matrix(columns=None)
    state_mapping_array = state_mapping_array.flatten()
    state_mapping_array = state_mapping_array[state_mapping_array!='']
    safe_matches = [re.escape(m) for m in state_mapping_array]
    search_patterns = '|'.join(safe_matches)

    for name, frame in agent_groups:
        name_list.append(name)
        schedule_frame = frame[frame['Date'].notnull()]
        lunch_frame = frame[frame['Schedule State'].str.contains('Lunch')]
        if lunch_frame.empty == False:
            schedule_list.append(schedule_frame.iloc[0]['Time Delta'] - lunch_frame.iloc[0]['Time Delta'])
        else:
            schedule_list.append(schedule_frame.iloc[0]['Time Delta'])
        exception_frame = frame[(frame['Date'].isnull()) & (~frame['Schedule State'].isin(marked_time))]
        exception_frame = exception_frame[~exception_frame['Schedule State'].str.contains('Lunch')]
        exception_frame = exception_frame[exception_frame['Schedule State'].str.contains(search_patterns)]
        if exception_frame.empty == False:
            exception_list.append(exception_frame['Time Delta'].agg(np.sum))
        else:
            if schedule_frame[schedule_frame['Schedule State'].str.contains(search_patterns)].empty == False:
                exception_list.append(schedule_frame.iloc[0]['Time Delta'])
            else:
                exception_list.append(datetime.timedelta(hours=0, minutes=0, seconds=0))
    df_result = pd.DataFrame({'Agent': name_list, 'Schedule Time': schedule_list, 'Exception Time': exception_list})
    df_result['Scheduled Productive Time'] = df_result['Schedule Time'] - df_result['Exception Time']
    return df_result
```

Listing 2: Code Listing → Stage 2 → Scheduled Productive Data Calculations

```
def actual_productive_time_alt(df_state, df_main_table):
    name_list = []
    actual_productive_list = []
    agent_groups = df_state.groupby('Agent Name')

    for name, frame in agent_groups:
        #set_trace()
        name_list.append(name)
        #pattern = re.compile(".*Timothy.*")
        #if pattern.match(name):
        #    set_trace()
        delta_frame = frame.sort_values(by='STC').reset_index()
        time_delta = datetime.timedelta(seconds=0)
        for index, row in delta_frame.iterrows():
            row_stc = row['STC']
            row_etc = row['ETC']
            if index == 0:
                latest_end_time = row['ETC']
                time_delta = time_delta + (row_etc - row_stc)
            else:
                if row_etc <= latest_end_time:
                    continue
                else:
                    if row_stc <= latest_end_time:
                        row_stc = latest_end_time
                        latest_end_time = row_etc
                    else:
                        latest_end_time = row_etc
                time_delta = time_delta + (row_etc - row_stc)
        actual_productive_list.append(time_delta)
    df_state_result = pd.DataFrame({'Interactive Name': name_list, 'Actual Productive Time': actual_productive_list})
    df_state_result = df_state_result.merge(df_main_table[['Interactive_Insight_Names', 'FullName']], left_on='Interactive Name',
                      right_on='Interactive_Insight_Names', how='left', indicator=True)
    df_state_result = df_state_result.rename(columns={'_merge': '_merge_main_table'})
    return df_state_result
```

Listing 3: Code Listing → Stage 2 → Actual Productive Data Calculations

1.1.4 STAGE 3: COMBINING RESULTS

At this point, combinatorial and merging operations to determine if Agent KPIs are being met such as Actual vs Scheduled Productive Time and Sign-in%

```
def total_scheduled_productive_time(df_isr, df_isr_prev, df_state, df_state_mapping):
    df_result = scheduled_productive_time(df_isr, df_state, df_state_mapping)
    df_result_prev = scheduled_productive_time(df_isr_prev, df_state, df_state_mapping)
    df_result = df_result.append(df_result_prev, ignore_index=True)
    df_result = df_result.sort_values(by='Agent')
    df_result = df_result.groupby(['Agent'], as_index=False).sum()
    return df_result

df_result = total_scheduled_productive_time(df_isr, df_isr_prev, df_state, df_state_mapping)

df_state_result_alt = actual_productive_time_alt(df_state, df_main_table)

df_combined = pd.merge(df_result, df_state_result_alt, left_on='Agent', right_on='FullName', how='left', indicator=True)

df_combined['Sign-in %'] = df_combined['Actual Productive Time'] / df_combined['Scheduled Productive Time']
```

Listing 4: Code Listing → Stage 3 → Combinatorial and Merging operations

1.1.5 MULTI-DAY MODIFICATION

DELIVERABLES

1. Modify schedule metrics to process multi-day data rather than just single-day data

The data processing pipeline was built with dealing with a single day's worth of raw data. In response to a request from the Reporting Team to generate stats for an entire month, the functions of the different stages were modified to handle multi-day data. Some functions are flexible enough to work in a single-day and multi-day environment.

SINGLE-DAY FUNCTION	MULTI-DAY FUNCTION
read_and_clean_state_date_v2	read_and_clean_state_data_v3
scheduled_productive_time	schedule_productive_time_multi_day
total_scheduled_productive_time	total_scheduled_productive_time_multi_day

FLEXIBLE FUNCTION (WORKS IN BOTH SCENARIOS)

read_and_clean_schedule_isr_data_v1
read_and_clean_schedule_isr_previous_data_v1
actual_productive_time_alt