COMMAND LINE FRAMEWORK

THOMAS EAPEN

2018

Contents

List of Figures

1 REPORTING & ANALYTICS

1.1 COMMAND LINE FRAMEWORK

DELIVERABLES

1. Develop a command line framework to be used by end users once the programs/scripts are developed. The command line framework ca pass options and arguments as well as have defaults that end users can modify as per needs

1.1.1 DESIGN

From the start, I investigated a few command line frameworks in python (of which there are many). A command line framework provides most of the heavy lifting of parsing argument and command line options as well providing automated help features.Based on the initial investigation, I decided to go with the `click_` framework `http://click.pocoo.org` which is a python package for creating command line interfaces in a composable manner. It is included in the Anaconda distribution which is another advantage. The code in listing below shows a command creation with valid options created, default arguments arguments, help features in built and command groups with 1 or more commands.

```python
@click.group()
def cli():
    pass

@click.command()
@click.option('--fileout', default='output.xlsx', help='The name of the output file including the xlsx extension (Optional: default -> output.xlsx)')
@click.option('--isr', default='isr.xlsx', help='Individual Schedule Report for the current day (Optional: default -> isr.xlsx)')
@click.option('--previous_isr', default='isr_prev.xlsx', help='Individual Schedule Report for the previous day (Optional: default -> isr_prev.xlsx)')
@click.option('--state_details', default='state_details.xlsx', help='State Detail Report for the current day (Optional: default -> state_details.xlsx)')
@click.option('--main_table', default='main_table.xlsx', help='Main Table for mapping between GWFM/GI2 (Optional: default -> main_table.xlsx)')
@click.option('--state_mapping', default='state_mapping.xlsx', help='Schedule State Code Mapping (Optional: default -> state_mapping.xlsx)')
@click.option('--group_mapping', default='group_mapping.xlsx', help='Group Mapping (Optional: default -> group_mapping.xlsx)')
def run_script(fileout, isr, previous_isr, state_details, main_table, state_mapping, group_mapping):
    df_isr = read_and_clean_schedule_isr_data_v1(filename=isr)
    click.echo('ISR Data read and cleaned')
    df_isr_prev = read_and_clean_schedule_isr_previous_data_v1(previous_isr)
    click.echo('Previous day ISR Data read and cleaned')
    df_state = read_and_clean_state_data_v1(state_details)
    click.echo('State Detail Data read and cleaned')
    df_main_table = read_and_clean_main_table_v1(filename=main_table)
    click.echo('Main Table Data read and cleaned')
    df_state_mapping = read_and_clean_state_mapping_v1(filename=state_mapping)
    click.echo('Schedule State Data read and cleaned')
    df_group_mapping = read_and_clean_group_mapping_v1(filename=group_mapping)
    click.echo('Group Data read and cleaned')
    df_result = total_scheduled_productive_time(df_isr, df_isr_prev, df_state, df_state_mapping)
    click.echo('Schedule Productive Calculated')
    df_state_result_alt = actual_productive_time_alt(df_state, df_main_table)
    click.echo('Actual Productive Calculated')
    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']
    click.echo('Sign-in % Generated')
    df_combined.to_excel(fileout)
    click.echo('Output file written: %s' % fileout)

cli.add_command(run_script())

if __name__ == '__main__':
    cli()
```

Listing 1: Code Listing → click_ command line framework