Task 1:

1. Fetch the page containing the location wise datasets for that year. (Bash Operator with wget or curl command)

```
fetch_page_task = BashOperator(
    task_id='fetch_page',
    bash_command=f'curl -s {BASE_URL}{YEAR}/ > {DOWNLOAD_DIR}index_{YEAR}.html',
    dag=dag,
)
```

2. Based on the required number of data files, select the data files randomly from the available list of files. (Python Operator)

```
def parse_and_select_files(**kwargs):
    # Parse the saved webpage
    with open(f'{DOWNLOAD_DIR}index_{YEAR}.html', 'r') as file:
        soup = BeautifulSoup(file, 'html.parser')

    # Extract file links
    links = [link.get('href') for link in soup.find_all('a')]
    data_file_links = [link for link in links if link.endswith('.csv')]

    # Randomly select the required number of data file links
    selected_files = random.sample(data_file_links, NUM_FILES_REQUIRED)
    # Push selected files to XCom for the next task to use
    kwargs['ti'].xcom_push(key='selected_files', value=selected_files)

select_files_task = PythonOperator(
    task_id='select_files',
    python_callable=parse_and_select_files,
    provide_context=True,
    dag=dag,
)
```

3. Fetch the individual data files (Bash or Python Operator)

```
def download_files(**kwargs):
    # Retrieve selected files
    ti = kwargs['ti']
    selected_files = ti.xcom_pull(task_ids='select_files', key='selected_files')

    # Download selected file
    for file_url in selected_files:
        response = requests.get(file_url)
        filename = file_url.split('/')[-1]
        with open(f'{DOWNLOAD_DIR}{filename}', 'wb') as file:
            file.write(response.content)

download_files_task = PythonOperator(
    task_id='download_files',
    python_callable=download_files,
    provide_context=True,
    dag=dag,
)
```

4. Zip them into an archive. (Python Operator)

```
def zip_files(**kwargs):
    # Retrieve selected files
    ti = kwargs['ti']
    selected_files = ti.xcom_pull(task_ids='select_files', key='selected_files')

    with zipfile.ZipFile(f'{DOWNLOAD_DIR}data_{YEAR}.zip', 'w') as zipf:
        for filename in selected_files:
            zipf.write(f'{DOWNLOAD_DIR}{filename}', arcname=filename)
            # We are zipping

zip_files_task = PythonOperator(
    task_id='zip_files',
    python_callable=zip_files,
    provide_context=True,
    dag=dag,
)
```

5. Place the archive at a required location. (Bash/Python Operator)

```
move_archive_task = BashOperator(
    task_id='move_archive',
    bash_command=f'mv {DOWNLOAD_DIR}data_{YEAR}.zip /newloc/',
    dag=dag,
)
```

Task 2:

1. Wait for the archive to be available (with a timeout of 5 secs) at the destined location. If the wait has timed out, stop the pipeline. (FileSensor)

```
wait_for_archive = FileSensor(
    task_id='wait_for_archive',
    filepath=f'/newloc/data_{YEAR}.zip',
    timeout=5, #ourtimeout
    poke_interval=1,
    mode='poke',
    dag=dag,
)
```

2. Upon the availability (status=success), check if the file is a valid archive followed by unzip the contents into individual CSV files. (BashOperator)

```
unzip_archive = BashOperator(
    task_id='unzip_archive',
    bash_command=f'unzip -o /newloc/data_{YEAR}.zip -d /unzipped/',
    trigger_rule='all_success'
    dag=dag,
)
```

3. Extract the contents of the CSV into a data frame and filter the dataframe based on the required fields such Windspeed or BulbTemperature, etc. Extract also the Lat/Long values from the CSV to create a tuple of the form <Lat, Lon, [[ArrayOfHourlyDataOfTheReqFields]]>. (PythonOperator using Apache Beam)

```
def process_csv_files():

    options = PipelineOptions(
        runner='DirectRunner',
        temp_location='/temp/',
        job_name='process-csv-files'
    )


    with beam.Pipeline(options=options) as p:
        def parse_csv(line):
            # The fields were are interested is HourlyWindspeed and HourlyDryBulbTemperature
            parts = line.split(',')
            return {'Lat': parts[2], 'Lon': parts[3], 'HourlyWindSpeed': float(parts[23]), 'HourlyDryBulbTemperature
        #filtering
        def filter_records(record):
            return record['HourlyWindSpeed'] > 3 and record['HourlyDryBulbTemperature'] > -10
        #returning the desired output
        def to_lat_lon_key(record):
            return ((record['Lat'], record['Lon']), [record['HourlyWindSpeed'], record['HourlyDryBulbTemperature']])

        (p
         | 'Read' >> ReadFromText('/unzipped/*.csv', skip_header_lines=1)
         | 'Parse' >> beam.Map(parse_csv)
         | 'Filter' >> beam.Filter(filter_records)
         | 'ToLatLonKey' >> beam.Map(to_lat_lon_key)
         | 'GroupByKey' >> beam.GroupByKey()
         | 'Write' >> WriteToText('/process/output')
        )

process_task = PythonOperator(
    task_id='process_csv',
    python_callable=process_csv_files,
    dag=dag
)
```

4. Setup another PythonOperator over ApacheBeam to compute the monthly averages of the required fields. The output will be of the form <Lat, Long, [[Avg_11, ..., Avg_1N] .. [Avg_M1, ..., Avg_MN]]> for N fields and M months.

```
def compute_monthly_averages():
    options = PipelineOptions(
        runner='DirectRunner',
        temp_location='/temp/',
        job_name='compute-monthly-averages'
    )

    with beam.Pipeline(options=options) as p:
        def parse_csv(line):
            parts = line.split(',')
             def parse_csv(line):

                parts = line.split(',')
                return {'Date':parts[1],'Lat': parts[2], 'Lon': parts[3], 'HourlyWindSpeed': float(parts[23]), 'Hour

        def to_monthly_key(record):
            date, lat, lon, dry_bulb_temp, wind_temp = (
                record['Date'], record['Lat'], record['Lon'],
                record['HourlyDryBulbTemperature'], record['HourlyWindTemperature']
            )
            month = date.split('-')[1]
            return ((lat, lon, month), (dry_bulb_temp, wind_temp))

        def compute_monthly_averages(kv_pair):
            key, values = kv_pair
            dry_bulb_temps = [value[0] for value in values]
            wind_temps = [value[1] for value in values]
            avg_dry_bulb_temp = sum(dry_bulb_temps) / len(dry_bulb_temps)
            avg_wind_temp = sum(wind_temps) / len(wind_temps)
            return (key, (avg_dry_bulb_temp, avg_wind_temp))

        (p
         | 'Read' >> beam.io.ReadFromText('/unzipped/*.csv', skip_header_lines=1)
         | 'Parse' >> beam.Map(parse_csv)
         | 'ToMonthlyKey' >> beam.Map(to_monthly_key)
         | 'GroupByKey' >> beam.GroupByKey()
         | 'ComputeMonthlyAverages' >> beam.Map(compute_monthly_averages)
         | 'WriteMonthlyAverages' >> beam.io.WriteToText('/monthly/output')
        )

compute_monthly_averages_task = PythonOperator(
    task_id='compute_monthly_averages',
    python_callable=compute_monthly_averages,
    dag=dag
)
```

5. Using ‘geopandas’ and ‘geodatasets’ packages, create a visualization where you plot the required fields (one per field) using heatmaps at different lat/lon positions. Export the plots to PNG. (PythonOperator using ApacheBeam)

```
def create_visualizations():
    data=pd.read_csv('monthly/averages/output-*')
    geometry = [Point(lon, lat) for lon, lat in zip(data['lon'], data['lat'])]
    gdf = gpd.GeoDataFrame(data, geometry=geometry)

    fig, ax = plt.subplots(figsize=(10, 10))
    gdf.plot(column='Average', ax=ax, legend=True, cmap='viridis')
    plt.title('Monthly Average Visualization')
    plt.savefig('/visualization/monthly_average.png')

create_visualizations_task = PythonOperator(
    task_id='create_visualizations',
    python_callable=create_visualizations,
    dag=dag
)
```

6. Upon successful completion, delete the CSV file from the destined location.

```
cleanup_csv = BashOperator(
    task_id='cleanup_csv',
    bash_command='rm -f /process/output/*.csv',
    dag=dag
)
```