DATA ENGINEER PYTHON TEST

Problem Statement -

Convert the weather data into parquet format. Set the raw group to appropriate value you see fit for this data. The converted data should be queryable to answer the following question.

- Which date was the hottest day?
- What was the temperature on that day?
- In which region was the hottest day?

Please provide the source code, tests, documentations and any assumptions you have made. Note: We are looking for the candidate's "Data Engineering" ability not just the Python programming skills.

Assumptions/Guidelines -

To test the code, Input data should have the same data structure like same column names and same data types

Missing value treatment - Since there are many missing values in the weather dataset we'd not be able to get the correct values for the queries like getting maximum temperature etc. Hence we need to impute those values appropriately (for numeric values - Mean and for categorical - Dictionary mapping)

ObservationDate - is of Integer type. to get the exact date, we need to convert this from Integer to Date format. Using inital 10 character, we can obtain the Date.

Source Code with Documentation and Tests:

Installing all the required library using pip command

In [10]:

```
# In order to achieve the given results we need to install below packages
pip install pandas
pip install pyarrow
```

Importing libraries

In [2]:

```
# Doing Dataframe related operation like reading from csv and describing the data we need to load Pandas
import pandas as pd
# Import library to help convert from csv to parquet format
import pyarrow
```

In [3]:

```
# path for the weather.csv file in the system
csv_path = './weather.20160201.csv'
```

In [4]:

```
# Read the csv file into python using python
weather_pd = pd.read_csv(csv_path)
```

In [5]:

```
# Sample of weather data
weather_pd.head(5)
```

Out[5]:

	ForecastSiteCode	ObservationTime	ObservationDate	WindDirection	WindSpeed	WindGust	Visibility	ScreenTemperature	Pressure	SignificantWeatherC
0	3002	0	2016-02-01T00:00:00	12	8	NaN	30000.0	2.1	997.0	
1	3005	0	2016-02-01T00:00:00	10	2	NaN	35000.0	0.1	997.0	
2	3008	0	2016-02-01T00:00:00	8	6	NaN	50000.0	2.8	997.0	
3	3017	0	2016-02-01T00:00:00	6	8	NaN	40000.0	1.6	996.0	
4	3023	0	2016-02-01T00:00:00	10	30	37.0	2600.0	9.8	991.0	

In [6]:

```
# Check the dimension of the weather data [row, columns]
weather_pd.shape
```

Out[6]:

```
(93255, 15)
```

In [7]:

```
# Describe the data to see the distribution of each variable
print(weather_pd.describe())
```

```
       ForecastSiteCode  ObservationTime  WindDirection     WindSpeed  \
count      93255.000000     93255.000000   93255.000000  93255.000000   
mean        4724.903673        11.520412       9.142695      9.817297   
std        11058.434533         6.940482       4.268251     21.316042   
min         3002.000000         0.000000       0.000000    -99.000000   
25%         3166.000000         5.000000       7.000000      6.000000   
50%         3385.000000        12.000000      10.000000     11.000000   
75%         3740.000000        18.000000      12.000000     17.000000   
max        99214.000000        23.000000      16.000000    105.000000   

           WindGust    Visibility  ScreenTemperature      Pressure  \
count  27093.000000  80542.000000       93255.000000  86556.000000   
mean      38.181781  26029.960890           3.005259   1006.854487   
std       11.116764  14635.844332          12.109166     14.499151   
min        0.000000     20.000000         -99.000000    961.000000   
25%       31.000000  15000.000000           1.900000    997.000000   
50%       36.000000  25000.000000           4.300000   1009.000000   
75%       43.000000  35000.000000           6.600000   1017.000000   
max      149.000000  75000.000000          15.600000   1036.000000   

       SignificantWeatherCode      Latitude     Longitude  
count            93255.000000  93255.000000  93255.000000  
mean                -7.116315     53.815688     -2.764828  
std                 35.121523      2.416302      2.094725  
min                -99.000000     49.913000     -7.577000  
25%                  1.000000     51.680000     -4.149000  
50%                  7.000000     53.307000     -2.800000  
75%                  8.000000     55.311000     -1.183000  
max                 28.000000     60.749000      1.348000  
```

Missing values - Treatment

In [8]:

```
# check how many missing values are there in the weather dataset
weather_pd.isna().sum()
```

Out[8]:

```
ForecastSiteCode              0
ObservationTime               0
ObservationDate               0
WindDirection                 0
WindSpeed                     0
WindGust                  66162
Visibility                12713
ScreenTemperature             0
Pressure                   6699
SignificantWeatherCode        0
SiteName                      0
Latitude                      0
Longitude                     0
Region                        0
Country                   13101
dtype: int64
```

Replacing missing values with sensible values

In [9]:

```
# Imputing WindGust with it's mean value
weather_pd['WindGust'].fillna(weather_pd['WindGust'].mean(), inplace=True)
```

In [10]:

```
# Imputing Visibility with it's mean value
weather_pd['Visibility'].fillna(weather_pd['Visibility'].mean(), inplace=True)
```

In [11]:

```
# Imputing Pressure with it's mean value
weather_pd['Pressure'].fillna(weather_pd['Pressure'].mean(), inplace=True)
```

In [12]:

```
# Country is the categorical variable which we can fill using it's correspoding region
# Since, Region to country mapping is already present in the weather data, use it to form a dictionary

# Let's drop all rows with missing country to avoid getting any NAN's and to get proper mapping between Region with country
pd_without_NAN = weather_pd[['Region','Country']].dropna()
# Create the dictionary using zip
region_to_country_dict = dict(zip(pd_without_NAN.Region,pd_without_NAN.Country))
# Show the dictionary
region_to_country_dict
```

Out[12]:

```
{'Orkney & Shetland': 'SCOTLAND',
 'Highland & Eilean Siar': 'SCOTLAND',
 'Grampian': 'SCOTLAND',
 'Strathclyde': 'SCOTLAND',
 'Central Tayside & Fife': 'SCOTLAND',
 'Dumfries, Galloway': 'SCOTLAND',
 'Northern Ireland': 'NORTHERN IRELAND',
 'Wales': 'WALES',
 'North West England': 'ENGLAND',
 'North East England': 'ENGLAND',
 'Yorkshire & Humber': 'ENGLAND',
 'West Midlands': 'ENGLAND',
 'East Midlands': 'ENGLAND',
 'East of England': 'ENGLAND',
 'South West England': 'ENGLAND',
 'London & South East England': 'ENGLAND'}
```

In [13]:

```
# Replace missing countries based on above Region to country mapping using map
weather_pd['Country'] = weather_pd['Region'].map(region_to_country_dict)
```

In [14]:

```
# Check for any missing values, it should be 0 by now.
weather_pd.isna().sum()
```

Out[14]:

```
ForecastSiteCode          0
ObservationTime           0
ObservationDate           0
WindDirection             0
WindSpeed                 0
WindGust                  0
Visibility                0
ScreenTemperature         0
Pressure                  0
SignificantWeatherCode    0
SiteName                  0
Latitude                  0
Longitude                 0
Region                    0
Country                   0
dtype: int64
```

Convert the csv file to Parquet format using pandas and pyarrow

In [15]:

```
# Saving the processed data
weather_pd.to_parquet('weather.parquet')
```

Reading from parquet file

In [16]:

```
weather_pd_parquet = pd.read_parquet('weather.parquet', engine='pyarrow')
print('Rows and columns of the weather parquet data :',weather_pd_parquet.shape)

print('\nColumn name of the weather parquet data :', weather_pd_parquet.columns.values)
```

```
Rows and columns of the weather parquet data : (93255, 15)

Column name of the weather parquet data : ['ForecastSiteCode' 'ObservationTime' 'ObservationDate' 'WindDirection'
 'WindSpeed' 'WindGust' 'Visibility' 'ScreenTemperature' 'Pressure'
 'SignificantWeatherCode' 'SiteName' 'Latitude' 'Longitude' 'Region'
 'Country']
```

Queires for the following questions

In [17]:

```
# Converting Datetime to Date format by extracting first 10 characters of this integer
# Apply function help achieve this for all the rows at the same time
weather_pd_parquet['ObservationDate'] = weather_pd_parquet.ObservationDate.apply(lambda x: x[0:10])
```

In [18]:

```
# Which date was the hottest day?
# Using the hottest day index we can extract its ObservationDate
hottest_day = weather_pd_parquet.loc[weather_pd_parquet[['ScreenTemperature']].idxmax(), ['ObservationDate']].values
print('Which date was the hottest day? ',hottest_day)
```

```
Which date was the hottest day?  [['2016-02-21']]
```

In [19]:

```
# What was the temperature on that day?
# Using the hottest day index we can extract its ScreenTemperature
hottest_temp = weather_pd_parquet.loc[weather_pd_parquet[['ScreenTemperature']].idxmax(), ['ScreenTemperature']].values
print('What was the temperature on that day? ',hottest_temp)
```

```
What was the temperature on that day?  [[15.6]]
```

In [20]:

```
# In which region was the hottest day?
# Using the hottest day index we can extract its Region
hottest_region = weather_pd_parquet.loc[weather_pd_parquet[['ScreenTemperature']].idxmax(), ['Region']].values
print('In which region was the hottest day? ',hottest_region)
```

```
In which region was the hottest day?  [['South West England']]
```

In [21]:

```
#  To get all the answers in a single line of code -
weather_pd_parquet.loc[weather_pd_parquet[['ScreenTemperature']].idxmax(), ['ObservationDate','ScreenTemperature','Region']]
```

Out[21]:

	ObservationDate	ScreenTemperature	Region
65916	2016-02-21	15.6	South West England