Data Ingestion

In this report I have discussed different methods that can be used to read data in python. It also includes the demonstration of a sample data pipeline where I ingest some new data and then check whether it satisfies the requirements mentioned in the config file.

Methods for reading Data:

I tried using Dask, Pandas and Modin for reading the large data file (2.13 GB). It took dask 0.0119 seconds to read the data while pandas took 46.4 seconds.

Using Dask:

```
In [9]: import dask.dataframe as dd
        start = time.time()
        df = dd.read_csv("../Datasets/airlineData.csv",dtype={'CANCELLATION_CODE': 'object'})
        end = time.time()
        print(end - start)
```

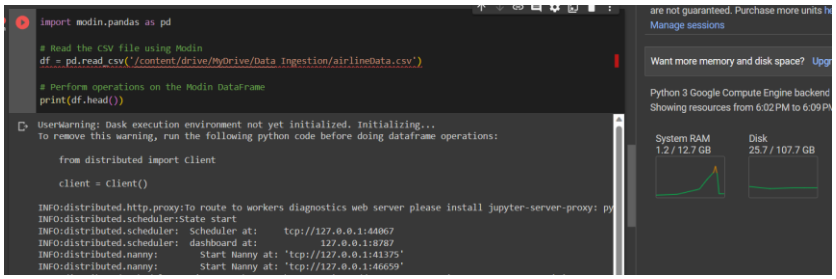
```
0.011973142623901367
```

Using Pandas:

```
In [14]: import pandas as pd
         start = time.time()
         df = pd.read_csv("../Datasets/airlineData.csv",dtype={'CANCELLATION_CODE': 'object'})
         end = time.time()
         print(end - start)
```

```
46.40069341659546
```

Using Modin:



Data Ingestion Pipeline:

Dataset Used: Airline Delay and cancellation data 2015 – 2018

Original Column Names:

```
df.columns
```

```
Index(['FL_DATE', 'OP_CARRIER', 'OP_CARRIER_FL_NUM', 'ORIGIN', 'DEST',
       'CRS_DEP_TIME', 'DEP_TIME', 'DEP_DELAY', 'TAXI_OUT', 'WHEELS_OFF',
       'WHEELS_ON', 'TAXI_IN', 'CRS_ARR_TIME', 'ARR_TIME', 'ARR_DELAY',
       'CANCELLED', 'CANCELLATION_CODE', 'DIVERTED', 'CRS_ELAPSED_TIME',
       'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'CARRIER_DELAY',
       'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY'],
      dtype='object')
```

Step 1: Validation on the columns

In this step, the script performs validates the columns of the input file and removes any unwanted special characters or white spaces from the names.

Import the utility file and Cleaning the Columns

```
In [4]: import testUtil

In [9]: df.columns = testUtil.cleanColumnHeaders(df.columns)

In [10]: df.head(2)
```

Out[10]:

	fl_date	op_carrier	op_carrier_fl_num	origin	dest	crs_dep_time	dep_time	dep_delay	taxi_out	wheels_off	...	diverted	crs_elapsed_time	actual_elapsed_t
0	2015-01-01	NK	195	MCO	FLL	2147	2143.0	-4.0	15.0	2158.0	...	0.0	63.0	
1	2015-01-01	NK	197	LGA	FLL	1050	1104.0	14.0	20.0	1124.0	...	0.0	194.0	1

2 rows × 27 columns

Step 2: Creating the config YAML file

Using the cleaned column names, we then write a YAML file to define the configuration settings for the dataset. The YAML file contains information about the columns expected in the dataset, the input and output file formats and name to the incoming dataset allowing for flexible and easy configuration of the script.

Writing the Yaml file

```
In [11]: %%writefile config.yaml
file_type: csv
dataset_name: airlineDelayData
file_name: airline_2014
inbound_delimiter: ","
outbound_delimiter: "|"
columns:
  - fl_date
  - op_carrier
  - op_carrier_fl_num
  - origin
  - dest
  - crs_dep_time
  - dep_time
  - dep_delay
  - taxi_out
  - wheels_off
  - wheels_on
  - taxi_in
```

Step 3: Reading new data

We then read the new airline delay data for the year 2014. We utilize the Dask dataFrame to efficiently handle the large datasets. The data is loaded into a Dask DataFrame for further processing. In this step, the script also performs validation on the columns of the dataset to ensure they match the expected columns specified in the config YAML file. This validation ensures that the new data has the required columns for further processing.

Validate the new data file

```
In [3]: testUtil.validate_dataset(config_data,"../Datasets/airline_2014.csv")

        column name and column length validation failed
        Following File columns are not in the YAML file ['unnamed_27']
        Following YAML columns are not in the file uploaded []

Out[3]: 0
```

We can see that the validation has failed due to an extra column in the new dataset. We therefore remove the column before ingestion.

Remove the unwanted column and test again

```
In [9]: tempData = dd.read_csv("../Datasets/airline_2014.csv")

In [10]: tempData = tempData.drop(['unnamed: 27'],axis=1)

In [12]: tempData.to_csv("../Datasets/airline_2014_updated.csv", single_file=True, index=False)
Out[12]: ['D:\\DG Internship\\Datasets\\airline_2014_updated.csv']
```

Once the column is removed, the dataset is ready to be ingested.

Note: Here I have read the dataset, then removed the column and then created a new csv file. In the real-world scenario, the dataset may be discarded or the column be removed manually through excel.

Step 4: Reading updated data

We then read the updated dataset. We can see that there is no error in the validation now.

Validating the updated Dataset

```
In [14]: testUtil.validate_dataset(config_data,"../Datasets/airline_2014_updated.csv")
         column name and column length validation passed
Out[14]: 1
```

Reading the new Data using dask

```
In [15]: airline_2014 = dd.read_csv("../Datasets/airline_2014_updated.csv")

In [16]: airline_2014.info()
         <class 'dask.dataframe.core.DataFrame'>
         Columns: 27 entries, FL_DATE to LATE_AIRCRAFT_DELAY
         dtypes: object(5), float64(19), int64(3)
```

Step 5: Merging Data and writing to new file in pipe-separated format

The script merges the new data for 2014 with the existing data from 2015 to 2018 based on common columns. It combines the datasets into a single dataset, taking care to align the columns correctly. The merged data is then written to a new file in a pipe-separated (|) format, creating a consolidated dataset.

Merging the two Datasets

```
In [16]: fullData = dd.concat([df,airline_2014])

In [18]: fullData.shape[0].compute()
Out[18]: 24526957
```

The final file has 27 columns and 24526957 rows.

Writing the data to a | seperated gz file

```
In [ ]: config_data["outbound_delimiter"]

In [ ]: config_data["outbound_file_type"]

In [ ]: config_data["output_file"]

In [ ]: df.to_csv(config_data["output_file"], single_file=True, sep=config_data["outbound_delimiter"],
                  compression=config_data["outbound_file_type"], index=False)
```

By following these steps, the script validates, merges, and summarizes the airline delay data from 2014 and the existing data from 2015 to 2018, creating a comprehensive dataset for further analysis and exploration.