Analysis of Aeroprice Channel Data

GitHub: https://github.com/alishakar/Travix_Data_Analysis

1. Task 1

To compile two Zip folders which has 92 csv and 92 xlsx files into one file i.e. daily_clicks and daily_transaction zip folder where the final output is a single CSV file.

Steps that I have followed to do the Task-1:

I have chosen Spark to do the parallel processing of files as the numbers were huge i.e.92 csv files and again 92 excel files, and looking at future where more files would come in , I thought this would be more faster as it will process the files pararllely instead one by one (which can be done writing a For Loop) which again is time consuming. Though I tried writing in both ways and found that parallel processing is much faster. Lets say for daily_clicks folder when I was processing each file writing a for loop time consumed for the whole process was 11.9 s and when I did the same using parallel processing it took just 4.53 s.

1. Import all the necessary packages required

```
import os

import zipfile

from pyspark.sql.functions import lit
```

```
import pandas as pd

from pyspark.sql import SparkSession

from pyspark.sql.functions import input_file_name, regexp_extract

from pyspark.sql.types import StructType, StructField, StringType

import re

import shutil

import argparse
```

2. Then I am storing zipped folder paths in variable

```
daily_clicks = "daily_clicks"

daily_transaction = "daily_transaction"


# List of zipped folder paths on the local system

zipped_folder_paths = "/Users/alishakar/Downloads/business_case_travix"
```

3. I wrote a function to extract the files from zipped folder

```
# Define the function to extract files

def extract_files(zip_file_path, extract_to_dir):

   with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:

       zip_ref.extractall(extract_to_dir)
```

Now I was facing a issue like when the daily_transaction was getting unzipped ,I was getting two erroneous files as below where you can see the first two files starting with $sign:

To avoid this case I have created a function called “filter and remove files folder” as below snippet:

```
def filter_and_remove_files(folder):

   pattern = re.compile(r'\d{4}-\d{2}-\d{2}\.(csv|xlsx)')

   for file in os.listdir(folder):

       if not pattern.match(file):

           os.remove(os.path.join(folder, file))
```

Inside the function, it uses the `re` module to compile a regular expression pattern. This pattern looks for filenames that follow a specific format:

- The pattern `\d{4}-\d{2}-\d{2}\.(csv|xlsx)` matches filenames that have a date in the format `YYYY-MM-DD`, followed by either `.csv` or `.xlsx`.
- For example, `2023-06-01.csv` or `2022-12-25.xlsx`.

So I have used a “If” condition where it will do a pattern match, so like if the defined pattern does not match with file name it will remove the same.

4. Finally I am writing a function to process each zip file

```
# Function to process each zip file

def process_zip_file(input_folder, file_name_prefix):

   file_name = file_name_prefix + ".zip"

   zip_path = os.path.join(input_folder, file_name)

   # Extract files to the directory where the zip file is located

   extract_files(zip_path, input_folder)



   extracted_folder = os.path.join(input_folder, file_name_prefix)

   filter_and_remove_files(extracted_folder)

   return extracted_folder
```

5. Now I am doing the work where I am combining all files inside daily_clicks to one csv file. Here I am processing the files parallellly so that it is faster

```
def process_daily_clicks_data(spark, input_folder):

    # Directory containing the CSV files

    csv_data = os.path.join(input_folder, daily_clicks)



    output_dir = os.path.join(input_folder, "output")

    output_csv = os.path.join(output_dir, daily_clicks + ".csv")



    # Read all CSV files from the directory

    df = spark.read.option("header", "true").csv(f"{csv_data}/*.csv")



    # Extract the date from the file path

    df = df.withColumn("file_path", input_file_name())

    df = df.withColumn("date", regexp_extract("file_path", r'(\d{4}-\d{2}-\d{2})', 1))



    # Drop the file_path column as it is no longer needed

    df = df.drop("file_path")



    # Order the DataFrame by date

    df = df.orderBy("date")



    # Combine all partitions into a single partition

    df = df.coalesce(1)
```

```python
# Temporary output directory
temp_output_dir = os.path.join(output_dir, 'temp_output')


# Save the combined DataFrame to a new CSV file
df.write.option("header", "true").mode('overwrite').csv(temp_output_dir)


# Find the generated part file and rename it to hello.csv
for file_name in os.listdir(temp_output_dir):
    if file_name.startswith('part-') and file_name.endswith('.csv'):
        os.rename(os.path.join(temp_output_dir, file_name), output_csv)
        break


# Clean up temporary directory
for file_name in os.listdir(temp_output_dir):
    os.remove(os.path.join(temp_output_dir, file_name))


os.rmdir(temp_output_dir)


print(f"Combined CSV file saved as {output_csv}")
```

Define Paths:

- `csv_data`: Path to the folder with daily click CSV files.
- `output_dir`: Path to the folder where the output will be saved.

- `output_csv`: Path for the final combined CSV file.
- Load all CSV files from `csv_data` into a Spark DataFrame with headers.
- Sort the DataFrame by the extracted date.
- Combine all data into a single partition.
- Write the combined DataFrame to a temporary output directory as a single CSV file.
- Locate the generated CSV part file in the temporary directory and rename it to the desired output file name.
- Remove all files in the temporary directory.
- Delete the temporary directory.
- A final print message

6. Similarly I am doing it for daily_transaction where we have all xlsx files stored and combining it to a single .csv file format.

Creating a function to read the excel file:

```
# Function to read an Excel file and add a "date" column

def process_excel(file_path):

   # Extract the date from the filename (assuming filename format is
"YYYY-MM-DD.xlsx")

   date = os.path.splitext(os.path.basename(file_path))[0]

   # Read the Excel file

   df = pd.read_excel(file_path)

   # Add the "date" column

   df['date'] = date

   return df
```

Then a function to do all the processing for combining files into one csv file

```
def process_daily_transaction_data(spark, input_folder):

   # Directory containing the Excel files

   excel_data = os.path.join(input_folder, daily_transaction)
```

```
    output_dir = os.path.join(input_folder, "output")

    output_excel = os.path.join(output_dir, daily_transaction + ".csv")



    # List all Excel files in the directory

    excel_files = [os.path.join(excel_data, f) for f in os.listdir(excel_data) if
f.endswith('.xlsx')]



    # Read and process all Excel files in parallel

    # Create an RDD of the file paths

    rdd = spark.sparkContext.parallelize(excel_files)



    # Map each file path to the processed DataFrame, then collect the results

    processed_dataframes = rdd.map(lambda file:
process_excel(file).astype(str)).collect()



    # Combine all the Pandas DataFrames into a single DataFrame

    combined_df = pd.concat(processed_dataframes, ignore_index=True)



    combined_df.sort_values(by='date', inplace=True)



    # Create the directory if it doesn't exist

    os.makedirs(output_dir, exist_ok=True)



    combined_df.to_csv(output_excel, index=False)
```

```
print(f"Combined excel file saved as {output_excel}")

return
```

- `excel_data`: Creates a path string by joining `input_folder` with a constant string "daily_transaction" (assuming this is the subfolder containing your Excel files).
- `output_dir`: Creates a path string for the output directory by joining `input_folder` with a string "output".
- `output_excel`: Creates a path string for the final output file by joining `output_dir` with "daily_transaction.csv".
- To parallelize the process, `rdd`: Creates a Spark RDD (Resilient Distributed Dataset) object by parallelizing the `excel_files` list using `spark.sparkContext.parallelize`. This distributes the file paths across Spark workers for parallel processing.
- `map` transformation: Applies the `process_excel` function (assumed to be defined elsewhere) to each file path in the RDD. This function likely reads and processes the Excel file, returning a DataFrame. The `astype(str)` part likely converts the DataFrame columns to strings for potential downstream operations.
- `collect`: Gathers the results of the `map` transformation, bringing the processed DataFrames back to the driver program. This creates a list named `processed_dataframes`.
- `Combined_df`: Uses pandas' `pd.concat` to combine all DataFrames in `processed_dataframes` into a single DataFrame, ignoring the original indices.
- `sort_values`: Sorts the `combined_df` by a column named "date" (assuming this column exists in your Excel data)
- `makedirs`: Creates the `output_dir` if it doesn't already exist, handling potential errors gracefully with `exist_ok=True`.

7. Then I have created another function to cleanup_extracted data:

```python
def cleanup_extracted_data(input_folder):

   shutil.rmtree(os.path.join(input_folder, daily_clicks))

   shutil.rmtree(os.path.join(input_folder, daily_transaction))

   return
```

It uses the `shutil.rmtree` function from the `shutil` module. This function recursively removes a directory and all its contents.

It calls `shutil.rmtree` twice, once for each constructed path (`daily_clicks` and `daily_transaction`). This removes the corresponding subfolders from the `input_folder` if they exist.

The purpose for this function was to clean up potentially temporary data extracted during processing.

8. I have defined a main function where it takes input_folder as argument , within this function I have initialized the Spark Session , processed the above functions like process_daily_clicks_data and process_daily_transaction_data.

```python
def main(input_folder):

   process_zip_file(input_folder, "daily_clicks")

   process_zip_file(zipped_folder_paths, "daily_transaction")



   # Initialize a Spark session

   spark = SparkSession.builder \

       .appName("process daily files") \

       .getOrCreate()



   process_daily_clicks_data(spark, zipped_folder_paths)
```

```
    process_daily_transaction_data(spark, zipped_folder_paths)


    # Stop the Spark session

    spark.stop()


    cleanup_extracted_data(zipped_folder_paths)
```

9. Final code which will perform the core data processing logic.

```
if __name__ == "__main__":


    parser = argparse.ArgumentParser(description='Merge daily data')

    parser.add_argument('-i', '--input', type=str, help='Input folder
containing zip files')


    args = parser.parse_args()

    main(args.input)
```

Calls a function named `main` and passes the value of the `input` argument from `args` as its parameter. This assumes a function named `main` exists elsewhere (likely in the same script or imported from another module). The `main` function would likely process the data based on the provided input folder path.

Basically anyone who runs the code can just type in **Terminal** as below:

python3 final.py --input "/Users/alishakar/Downloads/business_case_travix"

So python3 is my version , final.py the filename then –input and the folder path where the files are stored. This helps anyone to run the code in their system rather than hardcoding it to local path. Just to make it dynamic , I have used this logic so that anyone in team can run this code on their system , just give the specified input folder path in terminal.

How efficient is my method?

Spark and parallel processing is generally efficient, especially when we have large datasets. Spark's distributed computing capabilities enable parallel processing, reducing the time required for data processing tasks like file merging. However, the efficiency also depends on factors like cluster configuration, data size, and resource availability. Overall, parallel processing with Spark is a solid approach for optimizing data merging tasks.

Is it scalable to be used to compile 365 files of a year?

Yes, Spark's parallel processing capabilities make it scalable for compiling 365 files for a year. Spark can efficiently handle large datasets and distribute the workload across multiple nodes in a cluster, allowing for faster processing times compared to traditional single-node solutions. As long as the cluster is appropriately configured to handle the workload and there are enough computing resources available, Spark can effectively scale to handle larger numbers of files without significant performance degradation.

Task 2

Provide an analysis based on the data to answer the following questions:

- **What is the noticeable change in the market across months that impacts profitability (total margin)?**

Month-Year	flight_type	SUM of total_purchase	SUM of total_margin	SUM of channel_fee	SUM of surcharg
Total		0	0	0	0
Apr-2023	Return	871683.1264	10900.25671	-15738.032	2832.09
	One Way	416202.531	-1633.183355	-14179.0755	2731.22
Mar-2023	Return	929748.7025	12614.62825	-17385.9438	4385.32
	One Way	457832.3406	-2787.454322	-16248.1695	3744.32
May-2023	Return	905113.6838	12562.57955	-17235.98	5456.09
	One Way	486705.5124	369.4669599	-16883.9944	5085.73

As per the analysis, when return tickets were booked, for example for the month of May-2023 ,on that purchase order the “channel fee” is around 2%
When one way tickets are booked on that purchase order the Channel fee is around 3.4%.

It might be beneficial to focus marketing efforts on promoting return tickets, as they incur lower channel fees, potentially leading to higher net margins.

Month-Year	carrier	SUM of total_purchase	SUM of total_margin	SUM of channel_fee	SUM of surcharg
Total		0	0	0	0
Apr-2023 Total		1287885.657	9267.073353	-29917.1075	5563.31
Mar-2023 Total		1387581.043	9827.17393	-33634.1133	8129.64
May-2023	AZ	405664.372	-4471.319494	-13597.1869	1601.05
	TK	98149.37402	658.6600413	-1588.4304	510.02
	LA	93775.88444	2805.652378	-1325.6182	2489.57
	FR	77412.98704	289.3818428	-3031.6103	540.79
	QR	71922.69586	2968.095635	-1043.82	168.44
	NO	42603.03979	1692.684558	-895.2574	50.3
	TU	41167.52137	-497.318454	-1443.6444	-11.51
	EK	36735.55874	1467.653287	-374.7296	184.9
	EY	36401.24696	426.6646269	-497.6134	275.12
	KL	26818.63124	-321.9681256	-430.696	137.27
	UA	25895.68023	1514.047392	-317.9186	878.05
	JU	22803.65399	197.2171917	-755.8923	156.49
	BR	22288.18791	102.838381	-383.9156	127.09
	WY	20733.247	214.184334	-244	83.41
	AM	20454.412	240.5901946	-198.4012	43.39
	LH	20410.42	642.4186179	-257	105.98
	IB	18789.55022	55.64779512	-463.0632	78.94

As per the analysis w.r.t “Carrier” we can see “AZ” is big carrier whose total purchase is high in all three months with the data we have for now, Let’s sayyyyyyyy For example in **May-2023 total_purchase for AZ is 4,05,664.372** but the **total_margin is in negative .** We need to analyze more for this carrier to understand the business model with them, whereas “QR” has low total_purchase as compared to “AZ”, generating good total_margin, so we can think of applying QR’s business model to other carriers where total_margin is negative.

Month-Year	route	SUM of total_purchase	SUM of total_margin	SUM of channel_fee	SUM of surcharg
Total		0	0	0	0
Apr-2023 Total		1287885.657	9267.073353	-29917.1075	5563.31
Mar-2023 Total		1387581.043	9827.17393	-33634.1133	8129.64
May-2023	Italy - Italy	345701.8169	-5454.052364	-13496.4969	1303.9
	Italy - United States	150974.5649	2501.711777	-2230.1136	1529.24
	Italy - Brazil	87125.5	2307.145473	-1096.9244	1943.38
	Italy - Tunisia	41810.59	299.0868597	-1458.1196	-13.95
	Italy - Thailand	35667.81583	861.4736939	-360	7.02
	Italy - Indonesia	35354.35935	740.5703125	-360	112.36
	Italy - Spain	33476.74085	241.2311224	-850.468	67.49
	Italy - South Africa	30969.68114	260.1126633	-288	0
	Italy - Turkey	26725.90401	234.6383249	-698.7037	293.29
	United States - Italy	24419.75378	624.6344172	-304.84	360.55
	Tunisia - Italy	23325.72937	-487.2548582	-871.9852	39.48
	Italy - Egypt	22421.67807	200.3003635	-463.8748	-4.94
	Italy - Morocco	21342.4986	-265.0491505	-630.1143	18.35
	Italy - Mexico	21033.41016	444.0687491	-177.2643	30.75
	Turkey - Italy	15484.32893	332.1206487	-529.8731	233.68
	Italy - Vietnam	13492.12664	433.8964912	-198	38.6
	Italy - Australia	12906.3229	955.8627944	-102.6552	22.1
	Italy - Malaysia	11171.54318	288.5870232	-120.9617	17.39
	Italy - Dominican Repu	10621.77787	215.8426654	-121.7296	71.61
	Thailand - Italy	10568.01541	130.2892468	-205.1387	91.16
	Indonesia - Italy	9689.283135	53.49933071	-205.219	51.67
	Italy - Argentina	8961.59	76.2684246	-90	30
	Egypt - Italy	8055.845099	-46.9146447	-263.1524	-3.13
	Italy - Japan	7416.289799	55.39410641	-72	36.5
	Italy - Mauritius	6834.98	-10.502268	-36	0

As per analysis , when we see Route as a feature, let's say for May-2023 , Local **Italy** market the total_purchase amount is high, wherein our total_margin is negative.
The **total purchase** amount for Italy to Italy flights in May 2023 is 345,701.8169 euros. This includes both one-way and return flights, indicating a significant volume of transactions for domestic flights within Italy.
The **total margin** for these flights is negative, amounting to -5,454.052364 euros. This indicates that the flights were not profitable during this period. Negative margins suggest that the costs associated with these flights exceeded the revenues.

The **total channel fees** are significantly negative, at -13,496.4969euros . Channel fees represent costs associated with sales channels or distribution partners. The negative value indicates high costs in this area, which may be a significant factor in the overall negative margin.

The **total surcharge** collected is 1,303.9. Surcharges typically include additional fees that passengers pay, such as fuel surcharges or extra baggage fees. Despite the negative margins and channel fees, surcharges provide some additional revenue.

Recommendations: Investigate the high channel fees and explore opportunities to reduce these costs. This could involve negotiating better terms with distribution partners or finding more cost-effective sales channels**.**
Continuously monitor the financial performance of both one-way and return flights separately to identify specific areas of concern and take targeted actions to improve profitability.

- **Suggest possible reasons for the change (both from the market side and Travix commercial side) and explain how you arrive at the conclusion.**

Market-Side Factors:

1. The data suggests a need to negotiate better terms with distribution partners for one-way tickets to bring down the channel fees.
2. Understanding these fee structures allows for better revenue management. By targeting sales that incur lower fees, the overall profitability of the routes can be enhanced.

Travix Commercial Side Factors :

3. Travix may have different contractual agreements with sales channels(In our case Aero for one-way versus return tickets. Return tickets might be negotiated at lower fees due to higher volume commitments or better terms.
4. Selling return tickets might benefit from economies of scale, reducing the relative cost per ticket for distribution. Travix could be leveraging bulk deals with sales channels that reduce per-ticket fees for return tickets.

The analysis focused on identifying the reasons for the observed drop in total margin (profitability) taking Route, Carrier, flight type, channel fees into account.

The drop in profitability (total margin) can be attributed primarily to the following factors:

1. Disproportionately High Channel Fees for One-Way Tickets
2. Overall Negative Margins

3. Market and Operational Factors

- **What do you think can be done better to improve the profitability?**

  To improve profitability, particularly focusing on the drop in total margin observed in the data, several strategic actions can be considered. Here are recommendations based on different areas of the business:

1. Cost Management

a. Reduce Channel Fees

- Renegotiate contracts with sales channels to reduce fees, particularly for one-way tickets. This could involve seeking volume discounts or leveraging better terms with key partners.
- Focus on optimizing the mix of distribution channels to prioritize those with lower fees. Investing in direct sales platforms (e.g., the Travix website and mobile app) can help reduce reliance on higher-fee third-party channels.

b. Operational Efficiency

- Conduct a detailed review of operational processes to identify inefficiencies and areas for cost reduction. This could include optimizing flight schedules, improving fuel efficiency, and reducing turnaround times.
- Leverage bulk purchasing agreements for essential supplies and services to reduce per-unit costs.

2. Revenue Enhancement

a. Dynamic Pricing and Yield Management

- Utilize dynamic pricing models to adjust ticket prices based on real-time demand, competition, and other market factors, maximizing revenue per seat.
- Apply yield management techniques to ensure the optimal mix of passengers on each flight, enhancing revenue from high-demand periods and routes.

3. Route and Network Optimization

a. Evaluate Route Profitability

- **Route Analysis**: Conduct a detailed analysis of route profitability to identify underperforming routes that may need adjustments or discontinuation.

- **Expand Profitable Routes**: Focus on expanding routes with higher demand and profitability potential.

Data Processing Diagram

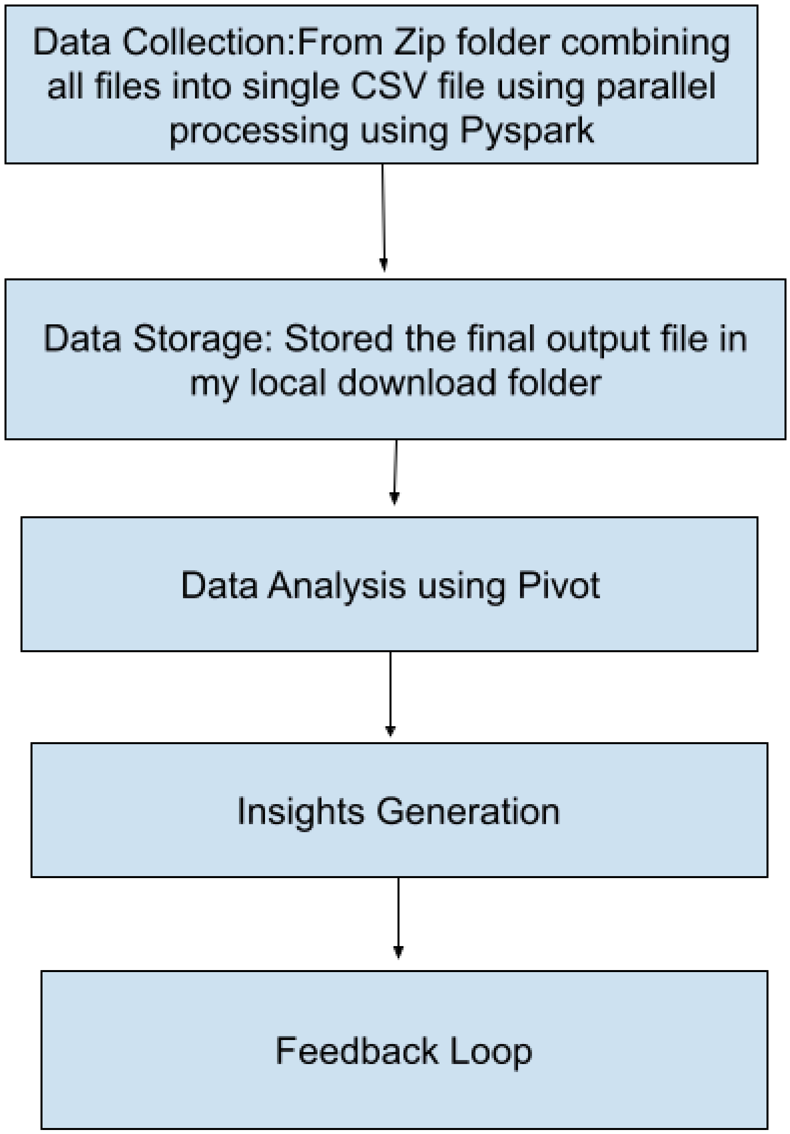
Basically below is a DPD(Data Processing Diagram) that outlines how data is collected, analyzed, and utilized to inform decisions and actions. In the context of improving profitability for Travix, the DPD will illustrate the flow of data through various stages, from initial data collection to actionable insights that lead to strategic decisions.

1. **Data Collection**:
- **Sources**: daily_clicks and daily_transactions zip file
- Travix Database
2. **Data Storage**:
   Pyspark is used to combine all files into one and is stored in local to do analysis.
3. **Data Analysis**:
- Techniques: Descriptive statistics, trend analysis,
- Tools: Data analytics using Google Sheets
4. **Insights Generation**:

Insights were generated taking various field into account using Pivot table in Google sheet.

5. **Feedback Loop**:

- Performance Monitoring: Continuously tracking the impact of implemented strategies.
- Data Collection: Gathering new data to update and refine the analysis and strategies.



Conclusion:

By addressing high channel fees, enhancing revenue management, improving operational efficiency, and optimizing marketing and sales strategies, Travix can work towards improving profitability and mitigating the drop in total margin observed in the data. Implementing these recommendations will help Travix achieve better financial performance and sustainable growth.