Name : Muhammad Fauzy Syaputra
ID : 120190025
Study Program : Industrial Engineering
Course : Data Analytics

TRANSJAKARTA DATA ANALYTICS PROJECT

Topics

The topic to be discussed is related to Customer analytics, specifically focusing on Customer spend analysis. It explains Transactional behavior and customer purchasing behavior to understand patterns and potential expenditures. In this case, user behavior analysis can be conducted to see how users interact with public transportation. This analysis can be used to enhance user satisfaction and reduce operational costs.

Business Description

Transjakarta is a regional government-owned enterprise (BUMD) in Jakarta, operating in the public transportation sector. Established in 2004, Transjakarta aims to provide comfortable public transportation services (service design meeting minimum consumer expectations, especially in travel conditions and stop facilities), safe (adherence to safety and health standards for buses, stops, and travel by operators), and affordable for the people of Jakarta (Indonesia, ranging from IDR 3,500 to IDR 5,000). Transjakarta offers various public transportation services, including bus rapid transit (BRT), bus feeder, microtrans, royaltrans, border feeder, city feeder, and tourist buses.

BRT Transjakarta is the primary service operating in the main corridors of Jakarta (Corridor 1 to 13). The feeder bus service is a supporting service that connects BRT stops with areas not covered by BRT. Mikrotrans is a supporting service operating in areas not served by BRT and feeder buses. Royaltrans is a premium fare Transjakarta bus service (Rp. 20,000) with more exclusive amenities such as more comfortable and spacious seats, free Wi-Fi, USB port, and reclining seat feature. Transjakarta border feeder is a feeder bus service connecting Transjakarta corridors with the buffer regions of DKI Jakarta, such as Depok, Bekasi, Tangerang, and South Tangerang. City feeder Transjakarta is a feeder bus service serving routes within Jakarta operated by Transjakarta and private

companies collaborating with Transjakarta, such as Kopaja, DAMRI, and Trans Swadaya. Transjakarta tourist bus is a tourist bus service provided by PT Transportasi Jakarta (Transjakarta). This tourist bus operates on specific routes in Jakarta, passing through various tourist spots and city landmarks such as Kota Tua, Monas, Ancol, and Pantai Indah Kapuk. Transjakarta has become one of the popular public transportation modes in Jakarta. In 2022, Transjakarta transported more than 400 million passengers.

The goal/problem to be addressed with data analytics.

A. Analyzing Passenger Travel Patterns

Objective: To enhance customer satisfaction and operational efficiency of public transportation systems.

The business objectives/problems can be divided into two sub-objectives, namely:

i. Improving customer satisfaction:

- Reducing passenger waiting time at stops
- Enhancing passenger travel comfort

ii. Increasing operational efficiency:

- Reducing the operating costs of public transportation systems
- Increasing the capacity of public transportation systems

The business objectives/problems can be measured using the following indicators:

1. Customer satisfaction:

- Decrease in customer complaints by 10-20%
- Achieving a target satisfaction survey score of 4.40

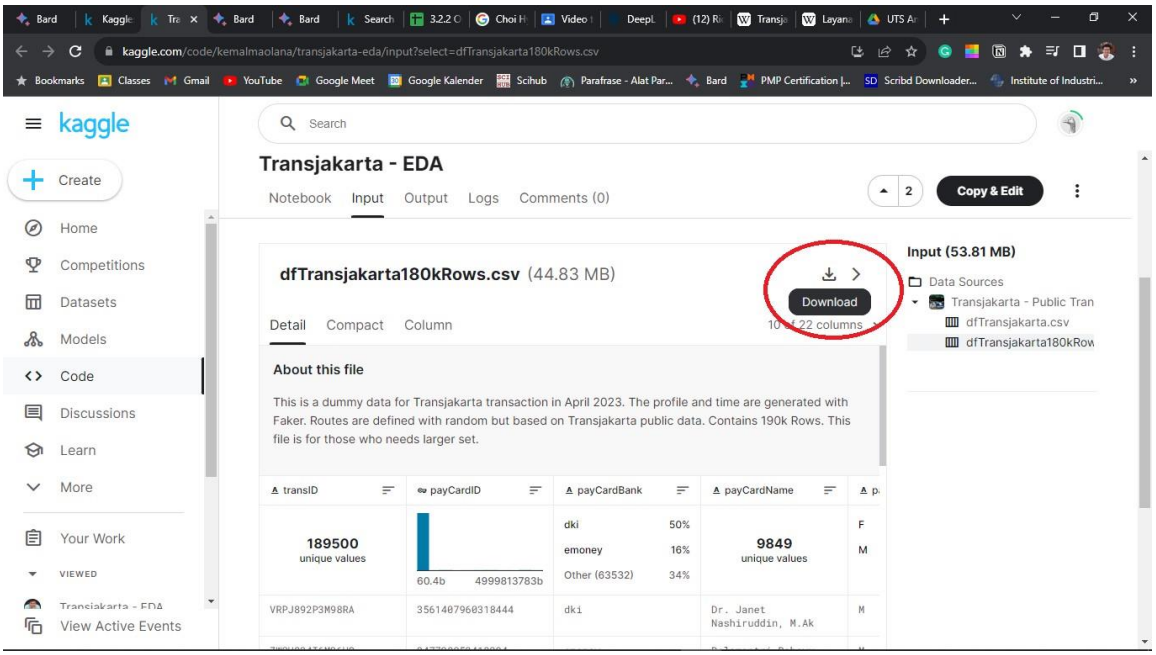
2. Operational efficiency:

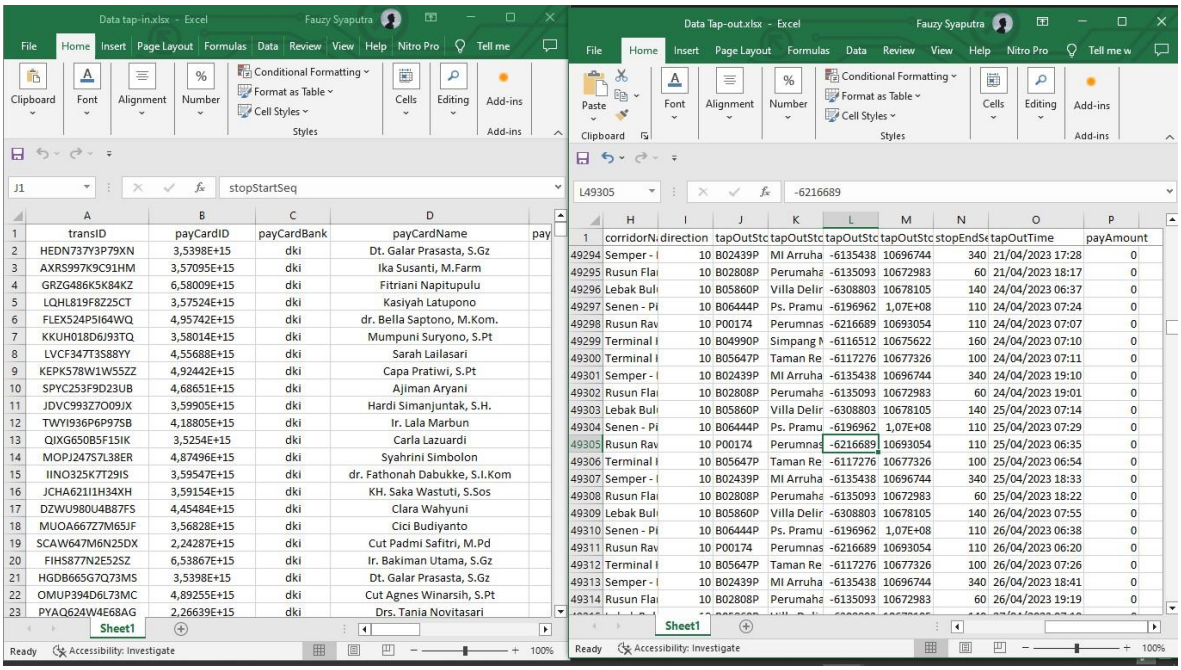
- Decrease in operating costs per passenger, indicated by the cost recovery ratio (Accumulated realization of PSO revenue and non-PSO revenue to operating costs) target of 26.83.

- Increase in the capacity of the public transportation system, indicated by the ratio of the number of passengers to the distance traveled in kilometers, with a target of 0.94.

Obtaining Data and Data Description

The dataset used was obtained from the Kaggle dataset provider website https://www.kaggle.com/. The dataset was acquired from [https://www.kaggle.com/code/kemalmaolana/transjakarta-eda/notebook], and then divided into two datasets for data acquisition purposes. The data was separated into two based on passenger tap-in and tap-out data in the form of an Excel workbook. The separated data can be accessed on the following page https://drive.google.com/drive/folders/12qD2TapfsnuLcgCI9bRzE77vcKLDVFX_?usp=sharing.





Data consist of:

transID: Transaction ID, a unique number used to identify each passenger travel transaction.

payCardID: Payment card ID, the payment card number used by passengers.

payCardBank: Issuer bank of payment cards.

payCardName: Owner's name of the payment card.

payCardSex: Gender of the cardholder.,

payCardBirthDate: Date of birth of the payment card owner

corridorID: ID corridor, the public transportation corridor number used by passengers.

corridorName: The name of the public transportation corridor used by passengers.

direction: Passenger direction or travel direction.

tapInStops: The stop number where passengers tap in.

tapInStopsName: The name of the stop where passengers tap in.

tapInStopsLat: The latitude coordinates of the stop where passengers tap in.

tapInStopsLon: The longitude coordinates of the stop where passengers tap in.

stopStartSeq: The sequence order of the tap in stop in the passenger's journey.

tapInTime: The time when the passenger taps in.

tapOutStops: The stop number where passengers tap out.

tapOutStopsName: The name of the stop where passengers tap out.

tapOutStopsLat: The latitude coordinates of the stop where passengers tap out.

tapOutStopsLon: The longitude coordinates of the stop where passengers tap out.

stopEndSeq: The sequence order of the tap out stop in the passenger's journey.

tapOutTime: The time when the passenger taps out.

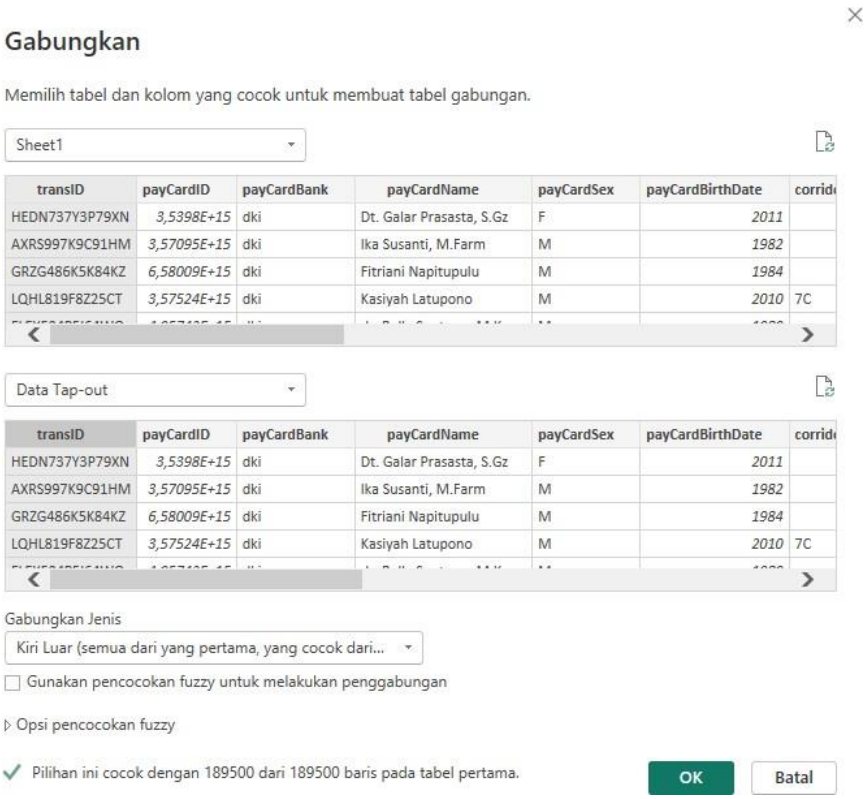
payAmount: The amount of money paid by the passenger.

By analyzing this data, patterns of passenger travel can be identified, such as: which Transjakarta corridors are most frequently used by passengers, the most popular travel directions, the average distance traveled by passengers, the average travel time of passengers, and the frequency of passenger travel.

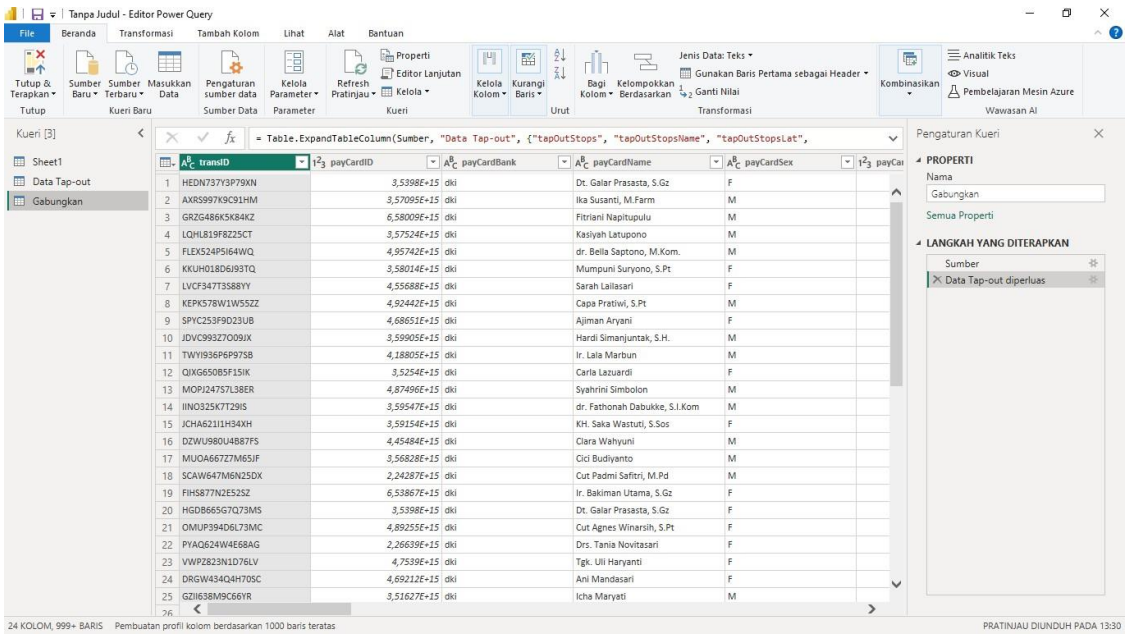
This information can be utilized to enhance customer satisfaction by: increasing the frequency and capacity of Transjakarta services in the most frequently used corridors. Additionally, this information can be used to improve the operational efficiency of the public transportation system by: scheduling and routing Transjakarta services more effectively (headway corresponding to passenger arrival rates), managing public transportation fleets more efficiently (allocating fleets in proportion to passenger demand for each corridor), predicting Transjakarta service demand. Moreover, it can help identify fraud in the Transjakarta payment system.

Data Acquisition

1. Data acquisition is performed from tap-in data in .xlsx format and tap-out data in .csv format using Power BI software.
2. Then, after that, you can use the 'Get Data' feature in Power BI and select from excel workbook (tap-in) and csv/text (tap-out).
    a) Excel workbook
        - Select the .xlsx file dataset used, in this case, it is tap-in.xlsx data.
        - Choose the sheet used, which is sheet 1.
        - Then click 'Load.'
    b) Csv
        - Select the .csv file dataset used, in this case, it is tap-out.csv data.
        - A preview table will be displayed, then click 'Load.'
        - Access the 'Transform Data' feature in Power BI.
3. The entire data will be displayed in tabular form in the query. In this case, there are two queries, namely sheet 1 (tap-in) and tap-out data. At this stage, data combination is performed using the combination feature > Merge Queries > Merge Queries as New (as we want to combine two datasets without altering the existing queries).
4. Then select the merged queries, which are sheet 1 and tap-out data. After that, choose the columns to be combined, in this case, the column 'transID.' Then press 'ok’.



5. After that, there will be one column titled 'Data tap-out,' which contains the combined data from the second query and needs to be expanded. Column expansion is done by clicking the expand icon, and options will appear for which columns to be expanded. Select the columns tapOutStops, tapOutStopsName, tapOutStopsLat, tapOutStopsLon, stopEndSeq, tapOutTime, payAmount, then press 'OK.'
6. Akuisisi data telah selesai di kombinasikan antara .csv dan .xlsx. Data yang berhasil digabungkan dapat dilihat pada kueri ‘Gabungkan’.

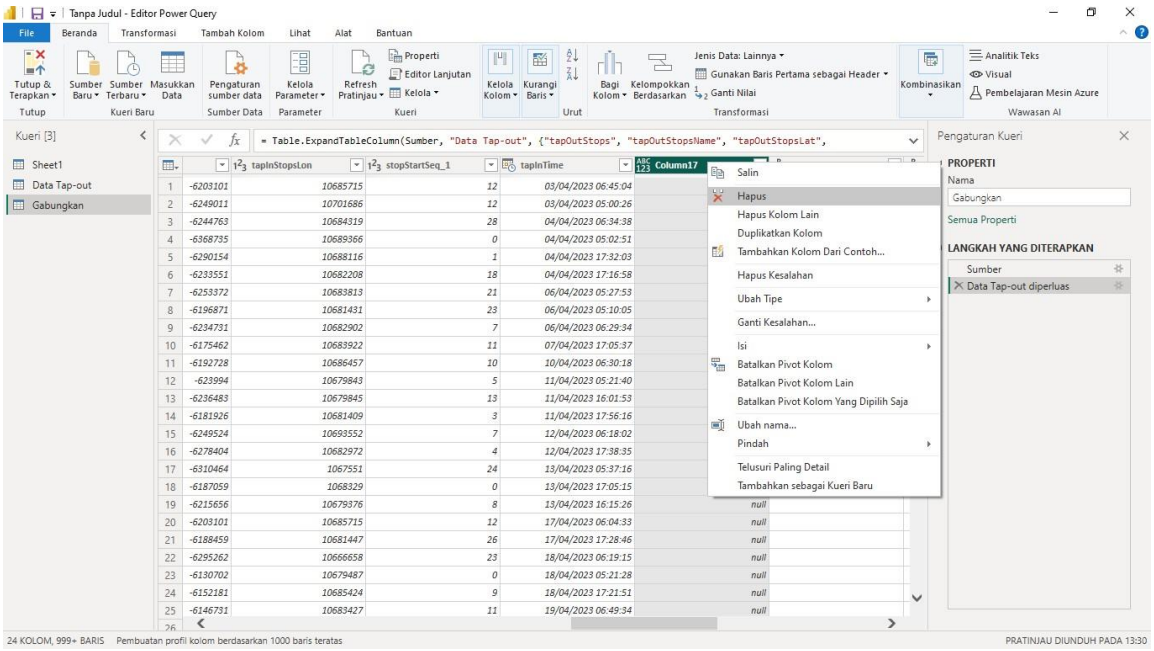


	transID	payCardID	payCardBank	payCardName	payCardSex
1	HEDN737Y3P79XN	3,5398E+15	dki	Dt. Galar Prasasta, S.Gz	F
2	AXRS997K9C91HM	3,57095E+15	dki	Ika Susanti, M.Farm	M
3	GRZG486K5K84KZ	6,58009E+15	dki	Fitriani Napitupulu	M
4	LQHL819F8Z25CT	3,57524E+15	dki	Kasiyah Latupono	M
5	FLEX524P5I64WQ	4,95742E+15	dki	dr. Bella Saptono, M.Kom.	M
6	KKUH018D6J93TQ	3,58014E+15	dki	Mumpuni Suryono, S.Pt	F
7	LVCF347T3S88YY	4,55688E+15	dki	Sarah Lailasari	F
8	KEPK578W1W55ZZ	4,92442E+15	dki	Capa Pratiwi, S.Pt	M
9	SPYC253F9D23UB	4,68651E+15	dki	Ajiman Aryani	F
10	JDVC993Z7O09JX	3,59905E+15	dki	Hardi Simanjuntak, S.H.	M
11	TWYI936P6P97SB	4,18805E+15	dki	Ir. Lala Marbun	M
12	QIXG650B5F15IK	3,5254E+15	dki	Carla Lazuardi	F
13	MOPJ24757L38ER	4,87496E+15	dki	Syahrini Simbolon	M
14	IINO325K7T29IS	3,59547E+15	dki	dr. Fathonah Dabukke, S.I.Kom	M
15	JCHA621I1H34XH	3,59154E+15	dki	KH. Saka Wastuti, S.Sos	F
16	DZWU980U4B87FS	4,45484E+15	dki	Clara Wahyuni	M
17	MUOA667Z7M65JF	3,56828E+15	dki	Cici Budiyanto	M
18	SCAW647M6N25DX	2,24287E+15	dki	Cut Padmi Safitri, M.Pd	M
19	FIHS877N2E52SZ	6,53867E+15	dki	Ir. Bakiman Utama, S.Gz	F
20	HGDB665G7Q73MS	3,5398E+15	dki	Dt. Galar Prasasta, S.Gz	F
21	OMUP394D6L73MC	4,89255E+15	dki	Cut Agnes Winarsih, S.Pt	F
22	PYAQ624W4E68AG	2,26639E+15	dki	Drs. Tania Novitasari	F
23	VWPZ823N1D76LV	4,7539E+15	dki	Tgk. Uli Haryanti	F
24	DRGW434Q4H70SC	4,69212E+15	dki	Ani Mandasari	F
25	GZII638M9C66YR	3,51627E+15	dki	Icha Maryati	M

Perform Data Preparation Process

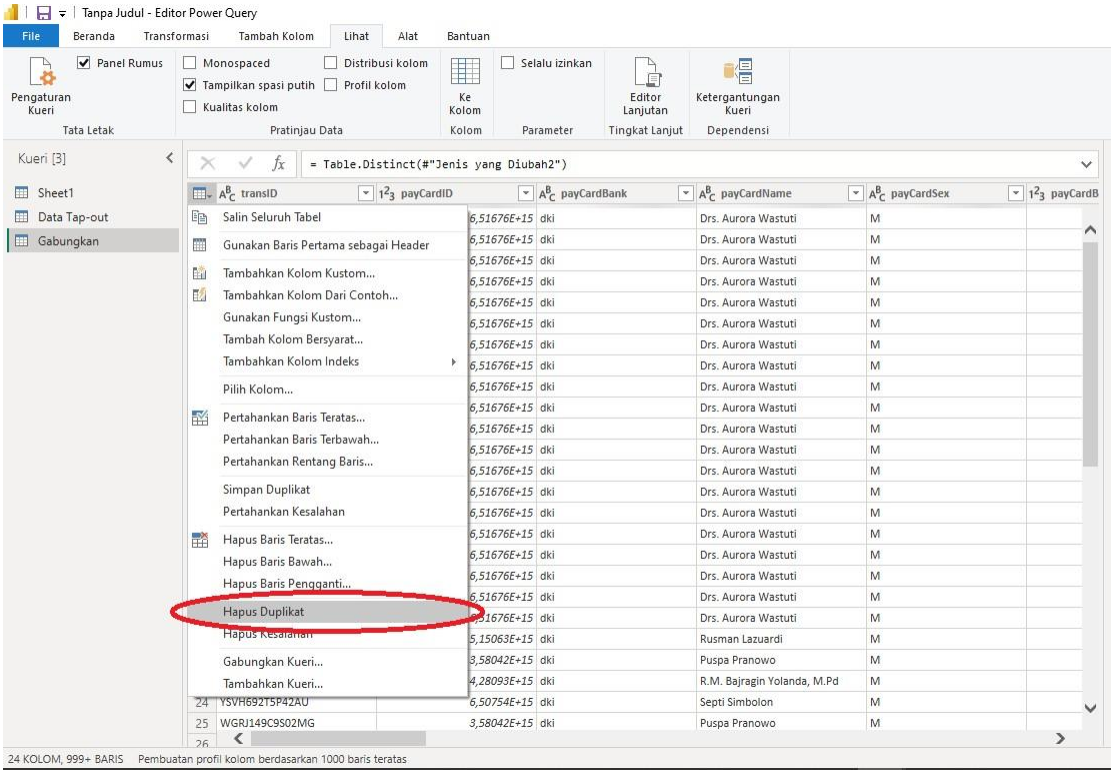
a. Delete the collum

From the Combined dataset, there are two columns, namely column 17, which contains null data. This column is not needed for data analysis and may have resulted from the merging of two queries or may have existed from the initial separation of the dataset. To remove this column, select the column to be deleted, right-click, and choose delete.



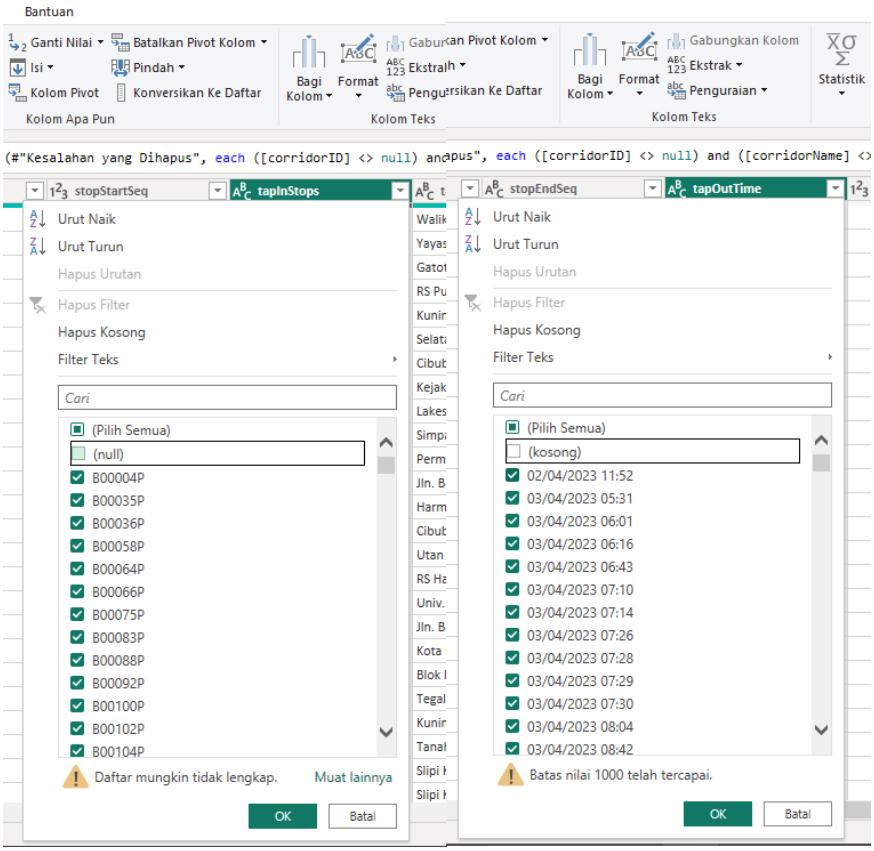
b. Removing Duplicates

Removing duplicates is the process of eliminating identical data from a dataset. Duplicate data can occur for various reasons, such as input data errors, data duplication from different sources, or inaccurate data. In Power BI, this can be done by going to the top-right corner of the table, clicking on the table icon, and selecting 'Remove Duplic*ates.*'



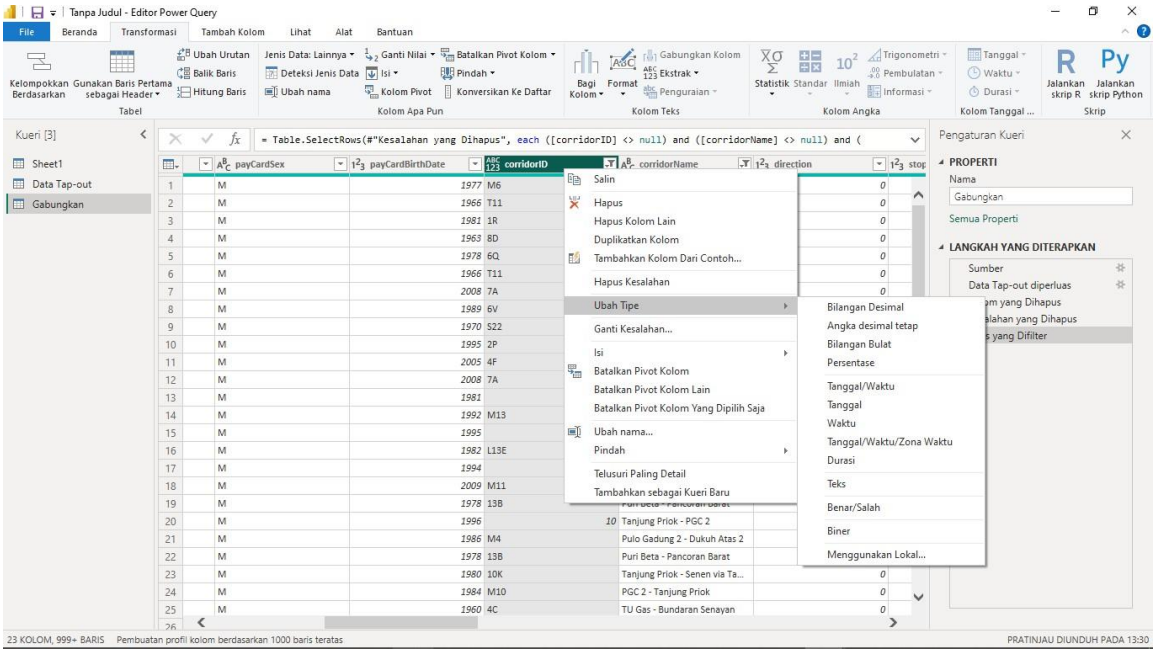
c. Filtering data

Filtering data values is the process of refining data based on specific criteria. This process is carried out to eliminate irrelevant or unnecessary data from the dataset. For data preparation, the columns that need to be filtered include corridorID, corridorName, tapInStops, tapOutStops, tapOutStopsName, tapOutStopsLat, tapOutStopsLon, stopEndSeq, and tapOutTime. This is necessary to filter out these columns to exclude "null" and "missing values." The method for filtering data involves clicking on the down arrow on the right side of the column name, then excluding unwanted values (null and missing values).



d. Changing Data Types

Changing data types values is the process of converting the data type of a value within the data. This process is carried out to ensure that the data has the correct data type for analysis. With Power BI, changing data types involves selecting the column whose data type needs to be changed, right-clicking, selecting 'Change Type,' and choosing the desired data type. In this dataset, the column that is modified is the corridorID column, which is changed to text to address varying values (numbers) within that column.



After the data preparation is completed, don't forget to use the 'Close and Apply' feature to apply all the changes made to the dataset.

Dashboard Descriptive Analytics

1. The dashboard is designed using Power BI.

2. It consists of overall Peak Hours for each BRT corridor, the number of passengers for each BRT corridor; miniTrans; and Jaklingko, the number of users based on the bank name of the electronic card, the total number of passengers per day, and the average number of passengers per day.

   a) To visualize Peak Hours, the following steps are taken:

   - Visualization using split tapInTime data into two, divided into date (tapInTime.1) and time (tapInTime.2).
   - More precisely, visualization is done using tapInTime hour data.
   - Groups are created from the data with a condition for each group to have a range of 1 hour, starting from 05:00 – 22:00.
   - Then, a new measure is created, namely Passenger Arrival = COUNTA(Merge[tapInTime.2 (bin)])

- It is then visualized using an Area Chart with the data.

c) To visualize the number of passengers for each BRT corridor; miniTrans; and Jaklingko, the following steps are taken:
   - Consists of a table with a bar chart for each column. The table includes corridorID, corridorName, Number of Passengers, and Revenue data.
   - The measures used for this section are:

     Number of Passengers = COUNTA('Data Tap-out'[corridorID])

     Revenue = SUMX(Merge; Merge[Number of Passengers]*Merge[payAmount])
   - After that, the data is visualized in the form of a table.
   - Bar chart visualizations are applied to the Number of Passengers and Revenue columns by formatting the chart > visual > cell elements > series > select data to be visualized > Data bars are checked and can be customized as desired.

d) To visualize the number of users based on the electronic card's bank name, the following steps are taken:
   - Create a new measure to see the number of users using electronic cards, with:

     Number of Users = COUNTA(Merge[payCardBank])
   - After that, select the number of users and payCardBank data, then visualize it in the form of a pie chart.

e) To visualize the magnitude of the number of passengers per day, the following steps are taken:
   - Convert the tapInTime.1 data into days by adding a column by selecting the column to be changed, then go to the Add column feature > date > day > day to create a new column containing the day number based on the selected date column. The days will have the following descriptions:

     0 = Sunday

     1 = Monday

     2 = Tuesday

     3 = Wednesday

     4 = Thursday

     5 = Friday

     6 = Saturday
   - After that, use the transformation feature > replace values to replace each value with the corresponding day name.

- Then, after adding the new measure, namely:

  Daily Passenger Count = COUNTA(Merge[Day of the Week])

- Subsequently, visualize the data by selecting Daily Passenger Count and Day of the Week data in the form of a bar chart table.

f) To visualize the average passengers per day, follow these steps:

- Create a new measure, namely:

  Average Customers per day = COUNTA(Merge[payCardID])/30

- Then visualize it using a chart.

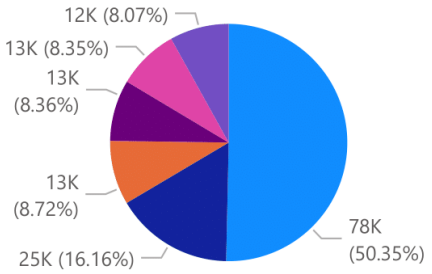
Jumlah penumpang untuk setiap koridor BRT; miniTrans; dan Jaklingko

corridorID	corridorName	Jumlah Penumpang	Pendapatan
12H	Rusun Penjaringan - Penjaringan	1102	3857000
11	Pulo Gebang - Matraman	1052	3682000
13	Ciledug - Tendean	1029	3601500
JAK.39	Kalimalang - Duren Sawit	1006	0
5	Matraman Baru - Ancol	995	3482500
S31	Bintaro - Fatmawati	989	19780000
7	Kampung Rambutan - Bidara Cina	983	3440500
2B	Harapan Indah - ASMI	977	3419500
11C	Rusun Pinus Elok - Rusun Pulo Gebang	975	0
1N	Tanah Abang - Blok M	972	3402000
5D	PGC 1 - Ancol	965	3377500
JAK.14	Tanah Abang - Meruya	954	0
JAK.23	Senen - Pisangan Baru	953	0
JAK.44	Andara - Stasiun Universitas Pancasila	940	0
11N	Rusun Cipinang Muara - Jatinegara	936	0
JAK.11	Tanah Abang - Kebayoran Lama	929	0
Total		**153963**	**411859500**

Jumlah Penumpang dari Hari Bulan April

Hari dalam Seminggu	Jumlah Penumpang hari
Rabu	27863
Kamis	27816
Senin	27808
Minggu	27743
Selasa	27730
Jum'at	7525
Total	**153963**

Jumlah Pengguna by payCardBank



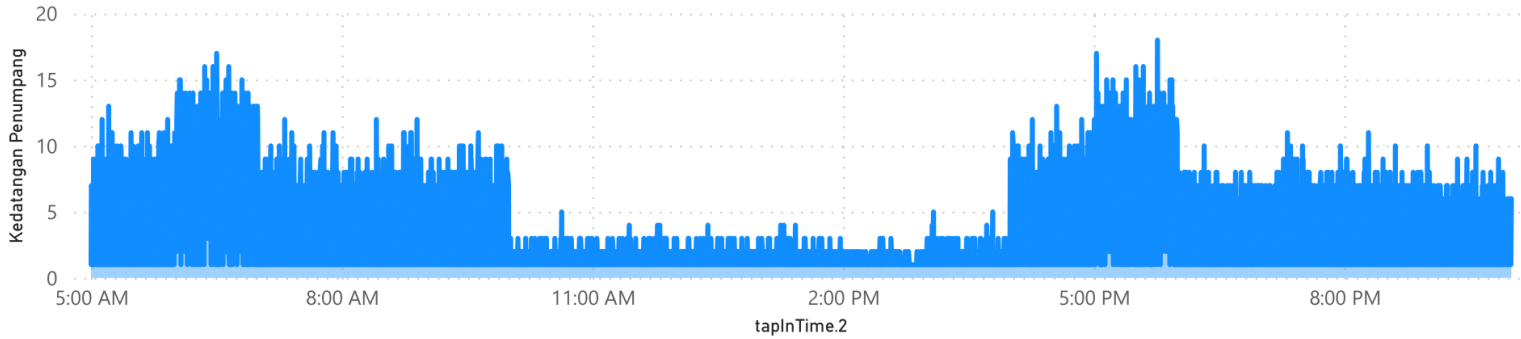
payCardBank
- dki
- emoney
- brizzi
- bni
- online
- flazz

#Integrasi

Tanggal
All

Hari
All

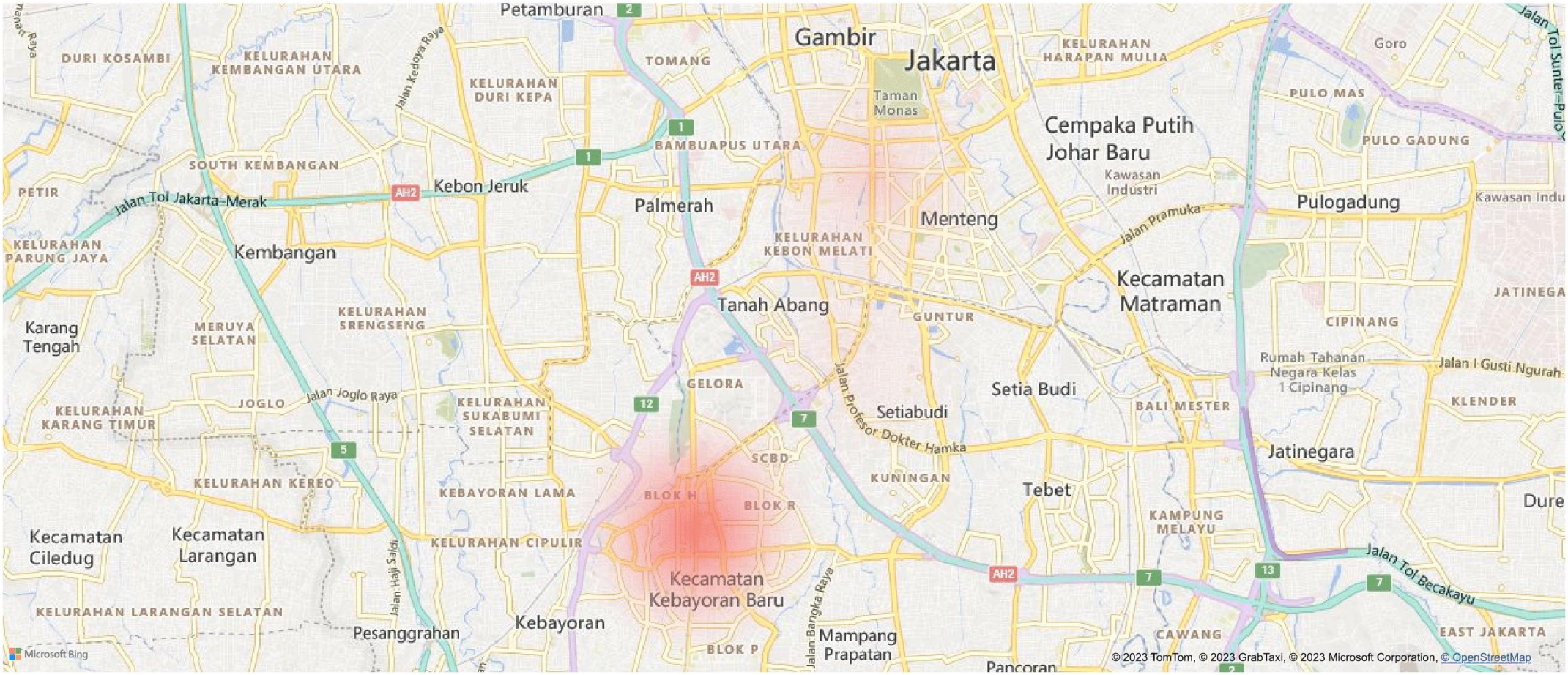
Peak Hours Bulan April



5.13K

Rata-rata jumlah penumpang per hari

jumlah tapin by tapInStopsLat and tapInStopsLon



Mikrotrans

All

Transjakarta

1

Transjabodetabek

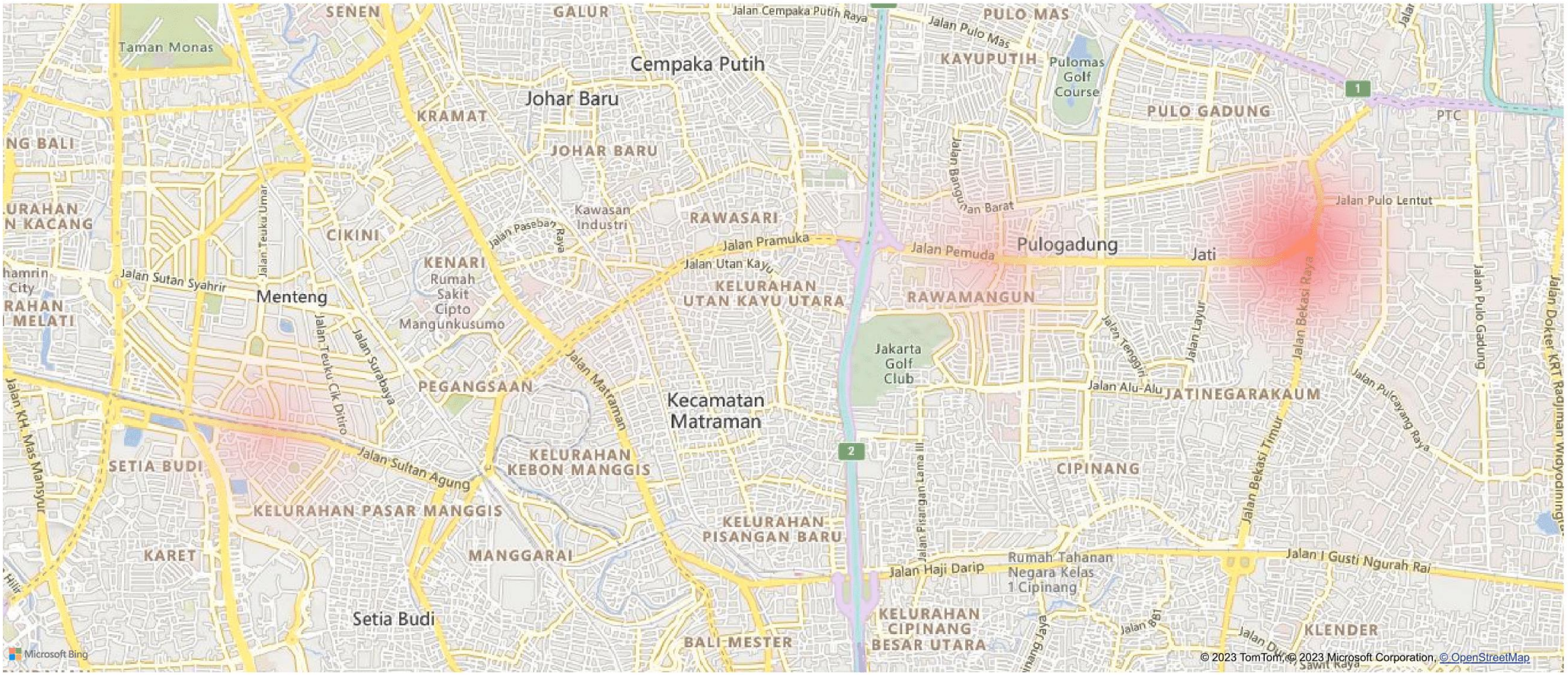
All

Tap-in Rute Transjakarta, Mikrotrans, dan Transjabodetabek Pada Jam Tertentu

Waktu

07:00

jumlah tapout by tapOutStopsLat and tapOutStopsLon



Mikrotrans

All

Transjakarta

4

Transjabodetabek

All

Tap-out Rute Transjakarta, Mikrotrans, dan Transjabodetabek Pada Jam Tertentu

Waktu

18:00

Analytics Process

The results show data related to the passenger arrival patterns in Transjakarta's operations from 05:00 to 22:00. It is found that Transjakarta's peak hours occur from 05:00 to 08:00 in the morning and from 16:00 to 18:00 in the evening. The highest passenger volume is observed on Wednesdays, followed by Thursdays, Mondays, Sundays, Tuesdays, and Fridays. The average number of passengers boarding Transjakarta in April 2023 is 5,130 passengers per day, with the busiest corridor being Transjakarta Corridor 12H from Rusun Penjaringan to Penjaringan. This indicates a significant number of workers from North Jakarta. It is followed by Corridor 11 from Pulogebang to Matraman, suggesting a high number of workers commuting from East Jakarta. These findings support the author's assumption that workers reside on the outskirts of Jakarta and work in Central Jakarta.

Predictive Analytics

Business Description

The business under study is the same as the Analytical Data Midterm Exam project. For the midterm exam, the focus is on Analyzing Passenger Travel Patterns. The data obtained relates to the patterns of passenger arrivals during Transjakarta's operations from 05:00 to 22:00. The results indicate that Transjakarta's peak hours occur from 05:00 to 08:00 in the morning and from 16:00 to 19:00 in the evening. The highest number of passengers is on Wednesdays, followed by Thursdays, Mondays, Sundays, Tuesdays, and Fridays. The average number of passengers boarding Transjakarta in April 2023 is 5,130 passengers per day, with the busiest corridor being Transjakarta 12H from Rusun Penjaringan to Penjaringan. This suggests a significant number of workers commuting from North Jakarta. This is followed by corridor 11, Pulogebang-Matraman, indicating a high number of workers from East Jakarta. This supports the author's assumption that workers reside on the outskirts of Jakarta and work in Central Jakarta.

Predicting the number of passengers for each corridor is necessary to determine the bus frequency category, whether it is good, average, or poor. This is crucial for further evaluation regarding the need to improve bus stop capacity, increase the fleet size, or enhance bus frequency with headway. The author will create a dashboard related to passenger predictions for the entire operational hours from 05:00 to 22:00, busy morning hours, and busy evening hours.

Objectives/Problems

The objective is to predict the number of passengers for Transjakarta, Transjabodetabek, and mikroTrans for the next 10 days based on the attached dataset. The predictions can be categorized based on corridor number, gender, and payment method used. Additionally, this prediction aims to:

- Enhance the effectiveness and efficiency of Transjakarta's operations. By knowing the estimated payamout, Transjakarta can plan the budget needed for its operations, determine the appropriate fare, and improve its services.
- Improve the quality of Transjakarta's services. Knowing the estimated payamout, Transjakarta can adjust the number of fleets and service frequency to meet passenger needs.
- Increase Transjakarta's revenue. By accurately predicting payamout, Transjakarta can target realistic revenue increases.

Decision Making Process

Decision alternatives to be made for predictive analytics are as follows:

1. If the passenger predictions for the overall category show a pattern significantly different from the data (exceeding the Max or Min values of the data with a 95% confidence level), then prescriptive

analytics will utilize the predicted data.

2. If the passenger predictions for the overall category show a pattern significantly different from the data (not exceeding the Max or Min values of the data with a 95% confidence level), then prescriptive analytics will use observational data. This is based on the premise that the predicted results have a pattern not different from the observational data.
3. From the payAmount predictions, estimates can be made regarding the time when passengers will spend money for the categories of Transjakarta, microtrans, and transjabodetabek transportation services..

Data Desctiption

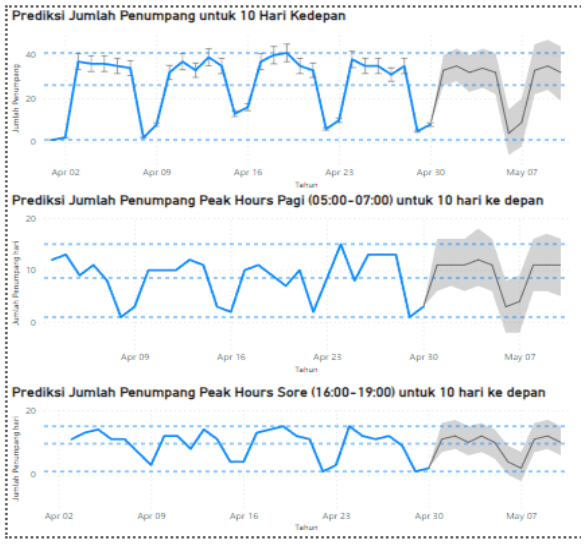
The main data used is the Passenger Count data obtained from measurements in the Data description.

Passenger Count = COUNTA('Tap-out Data'[corridorID])

This passenger count data will be predicted for the next 10 days using exponential smoothing methods or utilizing the built-in forecast feature in Power BI.

Passenger count data is used for prediction because the goal of the predictive analytics case is to forecast the number of passengers. The passenger count data is also used for prediction as per the business description, for the next stage, which is prescriptive analytics, to determine the bus frequency and its category using fuzzy logic for prescribing actions on that corridor. Passenger count data is used for prediction to decide the next stage, using only observational data or combined with predicted data for the next 10 days.

The additional data used includes payCardSex, payCardBirthDate, and payCardBank.

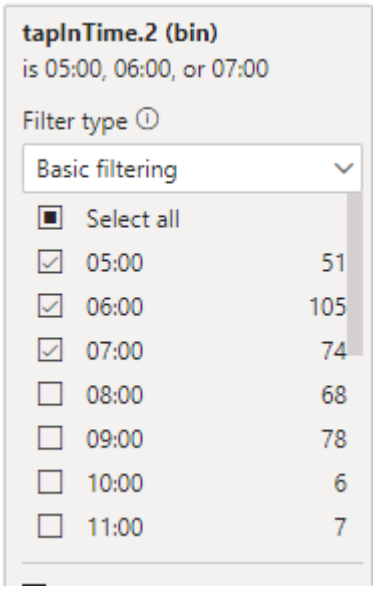


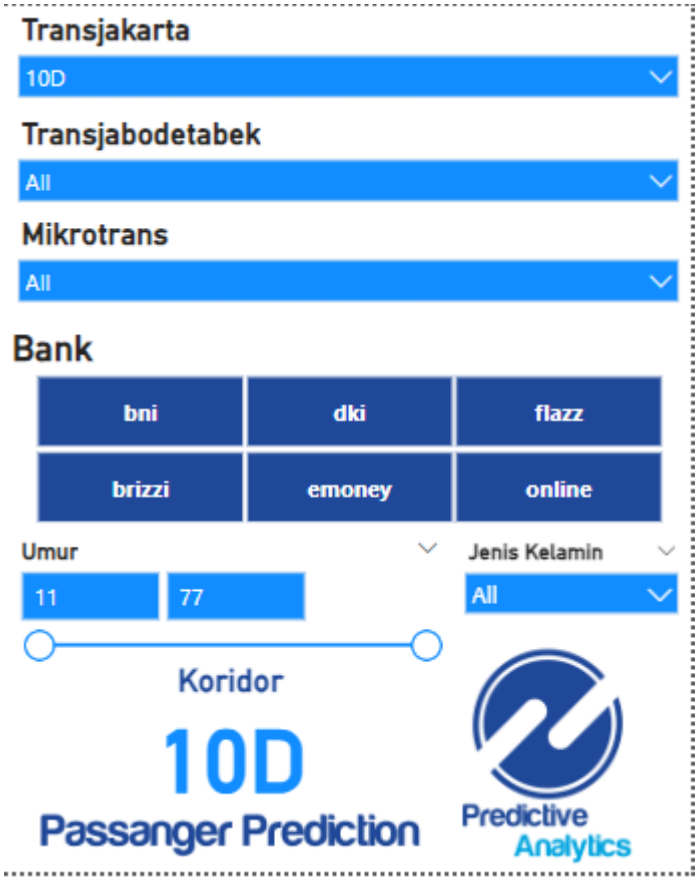
Dashboard Designing Process

1. Predition Graph

This is done to predict the number of passengers. It is performed using visualization features with a line chart. The X-axis represents tapInTime.1 data, and the Y-axis represents the Number of Passengers data. For busy morning and afternoon hours, the filters feature is utilized. Here, I added tapInTime.2 (bin) data in time format to determine when passengers tap in within a 1-hour range (e.g., if a tap-in occurs at 05:30:32, it falls into the 05:00 category). Then, specific hours categories are defined for input.

This is also the same as the rush hour in the evening. Next, filter for corridors, banks, age, and gender. These four elements use the visualization feature called slicer. Add a slicer to Power BI. For example, in the selection of Transjakarta corridors, as there are 3 categories, namely Transjakarta, Transjabodetabek, and MikroTrans, there are 2 data fields in the slicer. For instance, in the Transjakarta corridor slicer selection, there are 2 slicers with advanced filtering to exclude corridors with the name JAK to eliminate MikroTrans corridors. To exclude Transjabodetabek corridors, use basic filtering by selecting only Transjakarta corridors. Actually, it can be directly chosen for basic filtering, but I speed up my work by not selecting each JAK corridor one by one in basic filtering. The process is the same for Transjabodetabek as it is for Transjakarta. For MikroTrans, only one data field is needed, which is to select the corridorID that has JAK.

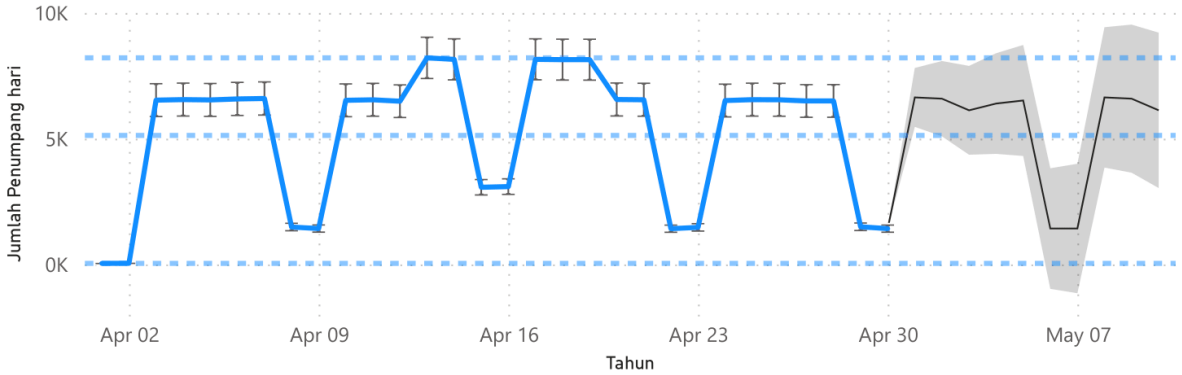




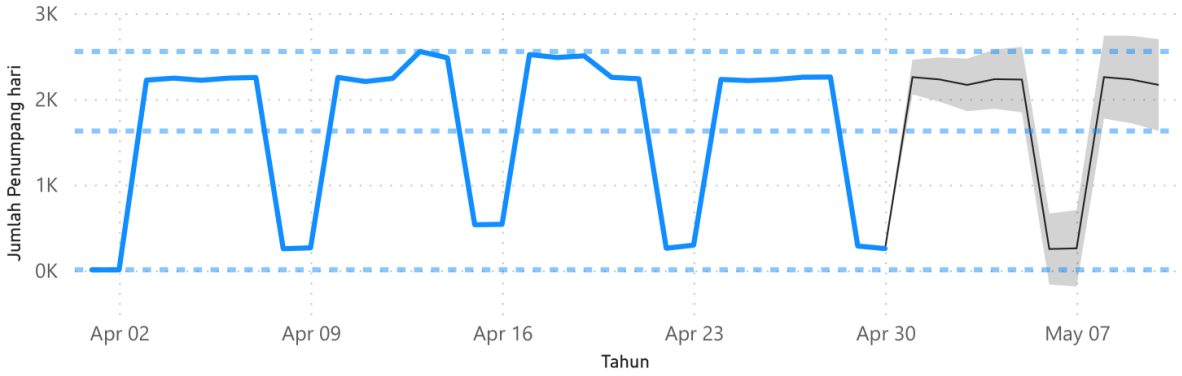
For the bank slicer, utilize the payCardBank data field with a tile-style slicer. For age, employ the payCardBirthDate data field with a 'between' style. For gender, use the payCardSex data with a dropdown.

The creation of the predictive dashboard does not involve the use of specific algorithms in Power BI.

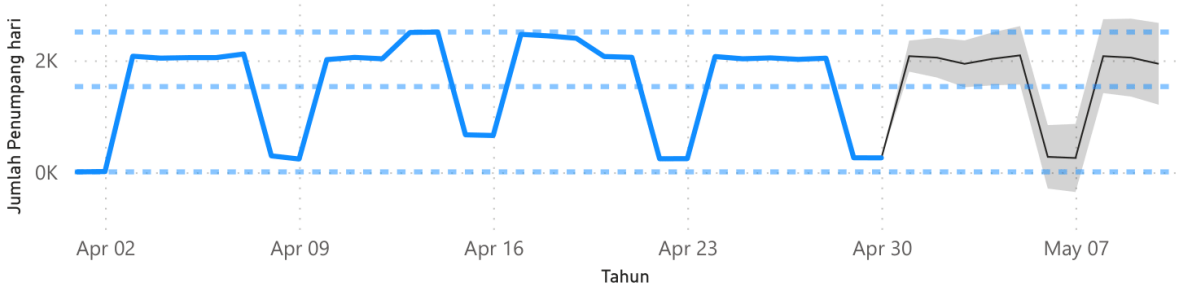
Prediksi Jumlah Penumpang untuk 10 Hari Kedepan



Prediksi Jumlah Penumpang Peak Hours Pagi (05:00-08:00) untuk 10 hari ke depan



Prediksi Jumlah Penumpang Peak Hours Sore (16:00-19:00) untuk 10 hari ke depan



Transjakarta

All

Transjabodetabek

All

Mikrotrans

All

Bank

bni	dki	flazz
brizzi	emoney	online

Umur

11 — 77

Jenis Kelamin

All

Koridor

1

Passanger Prediction

Predictive Analytics

I have included predictive analysis using regression algorithms to forecast the payAmount of passengers, or how much passengers will spend on Transjakarta Transportation services. The dashboard can be accessed on the page: https://www.kaggle.com/code/fauzysyaputra/interactive-dashboard-transjakarta-forecasting/notebook .

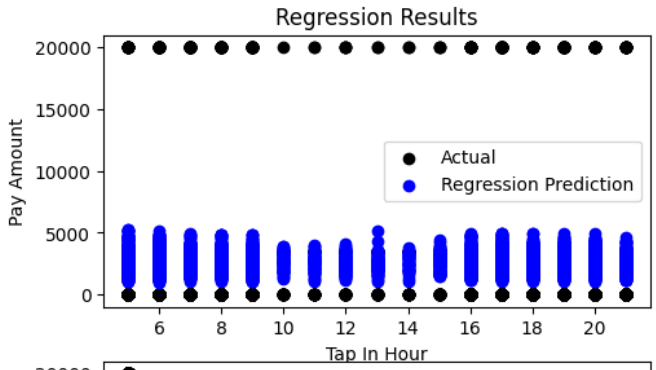
Creation of a prediction analytics dashboard for payAmount using the Python programming language.

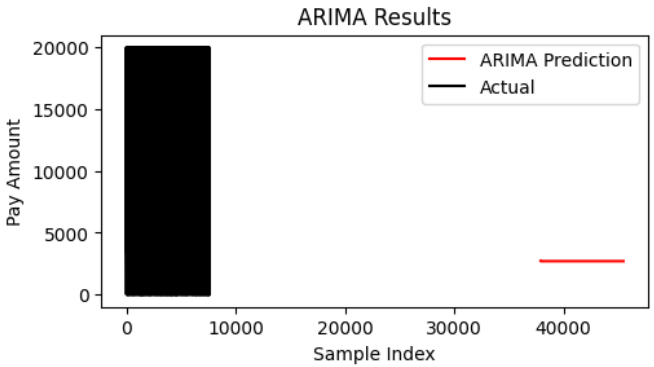
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

# Ganti 'nama_file.csv' dengan nama file CSV yang sesuai
file_path = '/kaggle/input/transjakarta/dfTransjakarta.csv'

# Baca file CSV
df = pd.read_csv(file_path)

# Tampilkan nama kolom
print("Nama Kolom dalam CSV:")
for column in df.columns:
    print(column)


# Tampilkan kolom-kolom dengan tipe data string
string_columns = df.select_dtypes(include='object').columns
print("Kolom-kolom dengan tipe data string:")
print(string_columns)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer

# Ganti 'nama_file.csv' dengan nama file CSV yang sesuai
file_path = '/kaggle/input/transjakarta/dfTransjakarta.csv'

# Baca file CSV
df = pd.read_csv(file_path)

# Tangani nilai NaN dengan menggantinya menggunakan nilai rata-rata
# Tangani kolom numerik
numeric_columns = df.select_dtypes(include=['number']).columns
imputer_numeric = SimpleImputer(strategy='mean')
df[numeric_columns] = imputer_numeric.fit_transform(df[numeric_columns])

# Tangani kolom non-numerik
non_numeric_columns = df.select_dtypes(exclude=['number']).columns
imputer_non_numeric = SimpleImputer(strategy='most_frequent')  # Ganti dengan
strategi yang sesuai
df[non_numeric_columns] =
imputer_non_numeric.fit_transform(df[non_numeric_columns])
```

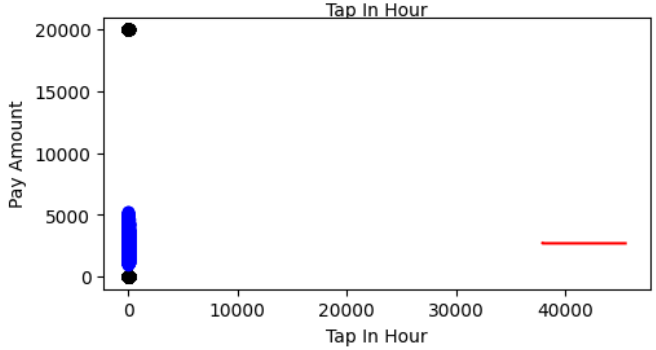
```
# Konversi kolom-kolom dengan tipe data string menjadi numerik
df['transID'] = df['transID'].astype('category').cat.codes
df['payCardBank'] = df['payCardBank'].astype('category').cat.codes
df['payCardName'] = df['payCardName'].astype('category').cat.codes
df['payCardSex'] = df['payCardSex'].astype('category').cat.codes
df['corridorID'] = df['corridorID'].astype('category').cat.codes
df['corridorName'] = df['corridorName'].astype('category').cat.codes
df['tapInStops'] = df['tapInStops'].astype('category').cat.codes
df['tapInStopsName'] = df['tapInStopsName'].astype('category').cat.codes
df['tapOutStops'] = df['tapOutStops'].astype('category').cat.codes
df['tapOutStopsName'] = df['tapOutStopsName'].astype('category').cat.codes

# Ubah format waktu menjadi datetime untuk memudahkan analisis waktu
df['tapInTime'] = pd.to_datetime(df['tapInTime'])
df['tapOutTime'] = pd.to_datetime(df['tapOutTime'])

# Ekstrak fitur waktu untuk model regresi linier
df['tapInHour'] = df['tapInTime'].dt.hour
df['tapInDay'] = df['tapInTime'].dt.day
df['tapInMonth'] = df['tapInTime'].dt.month
df['tapOutHour'] = df['tapOutTime'].dt.hour
df['tapOutDay'] = df['tapOutTime'].dt.day
df['tapOutMonth'] = df['tapOutTime'].dt.month

# Pilih fitur dan target
features = df.drop(['payAmount', 'tapInTime', 'tapOutTime'], axis=1)  #
Exclude target column and time-related columns
target = df['payAmount']

# Bagi dataset menjadi train dan test
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)

# Model Regresi Linier
reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
reg_predictions = reg_model.predict(X_test)
reg_mse = mean_squared_error(y_test, reg_predictions)
print(f'MSE Regresi Linier: {reg_mse}')

# Model Time Series (ARIMA)
df.set_index('tapInTime', inplace=True)
arima_model = ARIMA(df['payAmount'], order=(1, 1, 1))  # Sesuaikan parameter
p, d, q sesuai kebutuhan
arima_fit = arima_model.fit()
arima_predictions = arima_fit.predict(start=len(df), end=len(df) + len(X_test)
- 1)
arima_mse = mean_squared_error(y_test, arima_predictions)
print(f'MSE ARIMA: {arima_mse}')

# Visualisasi hasil
plt.figure(figsize=(12, 6))

# Plot hasil regresi
plt.subplot(2, 2, 1)
plt.scatter(X_test['tapInHour'], y_test, color='black', label='Actual')
plt.scatter(X_test['tapInHour'], reg_predictions, color='blue',
label='Regression Prediction')
plt.xlabel('Tap In Hour')
plt.ylabel('Pay Amount')
plt.title('Regression Results')
```

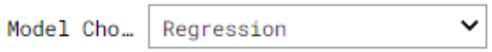
```
plt.legend()

# Plot hasil ARIMA
plt.subplot(2, 2, 2)
plt.plot(arima_predictions, color='red', label='ARIMA Prediction')
plt.plot(y_test.reset_index(drop=True), color='black', label='Actual')
plt.xlabel('Sample Index')
plt.ylabel('Pay Amount')
plt.title('ARIMA Results')
plt.legend()

# Perbandingan kedua model
plt.subplot(2, 2, 3)
plt.scatter(X_test['tapInHour'], y_test, color='black', label='Actual')
plt.scatter(X_test['tapInHour'], reg_predictions, color='blue',
label='Regression Prediction')
plt.plot(arima_predictions, color='red', label='ARIMA Prediction')
plt.xlabel('Tap In Hour')
plt.ylabel('Pay Amount')
plt

# Define a function to visualize the results
def visualize_results(tap_in_hour, model_choice):
    plt.figure(figsize=(8, 4))

    if model_choice == 'Regression':
        predictions = reg_model.predict(X_test[X_test['tapInHour'] ==
tap_in_hour])
        plt.scatter(X_test[X_test['tapInHour'] == tap_in_hour]['tapInHour'],
y_test[X_test['tapInHour'] == tap_in_hour], color='black', label='Actual')
        plt.scatter(X_test[X_test['tapInHour'] == tap_in_hour]['tapInHour'],
predictions, color='blue', label='Regression Prediction')
        plt.xlabel('Tap In Hour')
        plt.ylabel('Pay Amount')
        plt.title('Regression Results')
        plt.legend()
    elif model_choice == 'ARIMA':
        predictions = arima_fit.predict(start=len(df), end=len(df) +
len(X_test) - 1)
        plt.plot(predictions, color='red', label='ARIMA Prediction')
        plt.plot(y_test.reset_index(drop=True), color='black', label='Actual')
        plt.xlabel('Sample Index')
        plt.ylabel('Pay Amount')
        plt.title('ARIMA Results')
        plt.legend()

# Create interactive widgets
tap_in_hour_widget = widgets.IntSlider(min=df['tapInHour'].min(),
max=df['tapInHour'].max(), step=1, description='Tap In Hour')
model_choice_widget = widgets.Dropdown(options=['Regression', 'ARIMA'],
value='Regression', description='Model Choice')

# Create interactive dashboard
interact(visualize_results, tap_in_hour=tap_in_hour_widget,
model_choice=model_choice_widget)
```

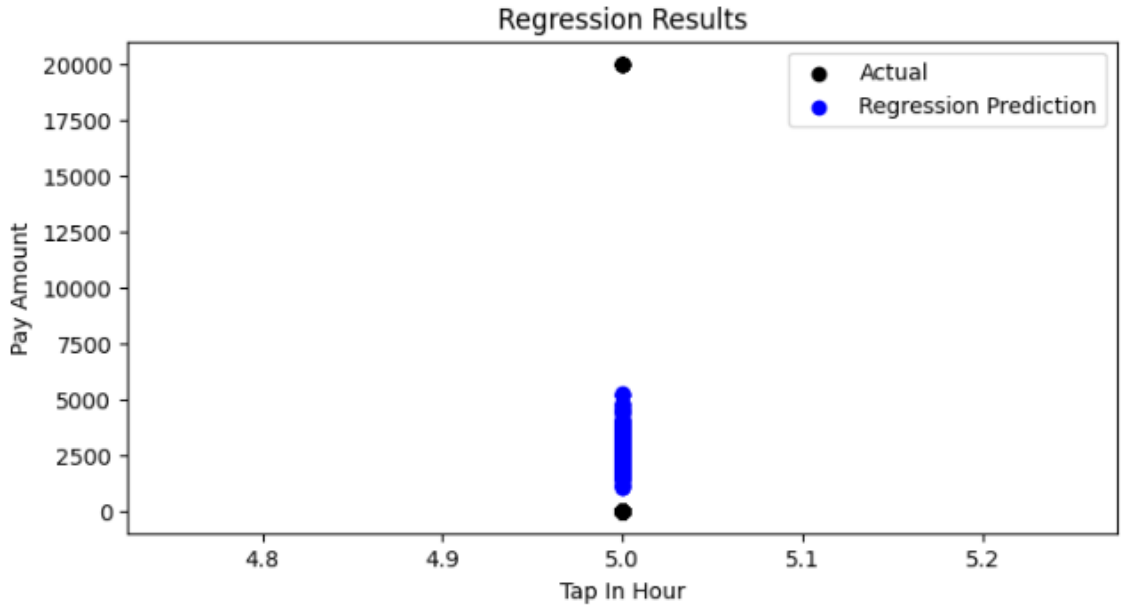
```
MSE ARIMA: 16428569.026207227
MSE Regresi Linier: 16092570.953509133
```











Prescriptive Analytics

Business Description

Continuing from the mid-term exam on analytical data and predictive analytics using tap-in and tap-out data of Transjakarta passengers in April 2023. From the analytical data mid-term exam to describe and explore the data. The description aims to determine the peak hours of Transjakarta passengers for each corridor, with peak morning hours from 05:00 to 08:00 and peak evening hours from 16:00 to 19:00. The assumption, supported by data, is that during peak hours, there is a significant number of journeys from the outskirts of Jakarta, such as East Jakarta, North Jakarta, and Tangerang, to the city center of Jakarta, specifically Jakarta Pusat. Evidence shows that corridors like 12H (Rusun Penjaringan – Penjaringan), 11 (Matraman – Pulogebang), and 13 (Ciledug - Tendean) have the highest number of passengers.

Moving on to predictive analytics to forecast the number of passengers for each corridor and predict the payAmount. This will serve as the foundation for conducting prescriptive analytics using fuzzy logic to determine the recommended bus frequency based on the number of passengers for each corridor, tapInHour, and travel_time. The system's output will indicate the need for improvement, marked with ❌ to signify that the system is not functioning properly (explained in the code). However, if the system produces a ✔ symbol, it indicates that the system is operating correctly. This will help prioritize focus areas based on the corridors under examination.

Objectives/Problems

The system's objective is to model and simulate the control of passenger density and bus waiting times based on several variables such as travel time, tap-in time, passenger count, and corridor, aligning the bus departure frequency with predefined categories (explained in the code) to serve as the foundational focus for priority improvement.

Decision Making

If the system issues the symbol ❌, then the corridor needs to be investigated regarding the frequency of buses not meeting passenger demand. An alternative course of action to address this issue may involve a strategy of increasing the fleet size or implementing headway control strategies.

If the system issues the symbol ✔, then the corridor is operating in accordance with the specified code rules, as explained. There is no need for further investigation.

Data Descrition

The data used is the same as the data used for the Data Analytics and Predictive Analytics Final Exam..

1. tapInHour

   df['tapInHour'] = df['tapInTime'].dt.hour

2. runtime

   df['runtime'] = (df['tapOutTime'] - df['tapInTime'])

3. travel_time

   df['travel_time'] = pd.to_timedelta(df['runtime']).dt.total_seconds() / 60

4. jumlah_penumpang

   jumlah_penumpang = penumpang.loc[penumpang['corridorID'] == corridorID,
   'jumlah_penumpang'].values[0]

5. penumpang

penumpang = df.groupby('corridorID')['corridorID'].count().reset_index(name='jumlah_penumpang')

for index, row in penumpang.iterrows():

    print(f"corridorID: {row['corridorID']}, JumlahPenumpang: {row['jumlah_penumpang']}")

Dashboard Designing Process

Designing a dashboard using Python code executed in a Kaggle notebook. The Kaggle notebook can be accessed on the following page: https://www.kaggle.com/code/fauzysyaputra/transjakarta-project .

Python code:

1. Installation and Library Import, as well as Loading Data: The code begins by installing and importing the required libraries: scikit-fuzzy, numpy, and pandas. Data from the dataset (dfTransjakarta.csv) is loaded into a Pandas DataFrame with the name df.

```
# Instalasi dan impor pustaka yang diperlukan
!pip install -U scikit-fuzzy
!pip install numpy
!pip install pandas

import pandas as pd
import numpy as np
import datetime
import skfuzzy as fuzz
    from skfuzzy import control as ctrl

    # Memuat dataset
    df = pd.read_csv('/kaggle/input/transjkt/dfTransjakarta.csv')
    print(df)
```

2. Data Pre-processing: The checkNA function is defined to check for null values in the DataFrame and remove them. Columns related to dates are converted into datetime objects, and additional columns (runtime and travel_time) are created.

```
def checkNA(data):
    # Memeriksa nilai null dan menghapusnya
    if data.isna().sum().any():
        print('Ada nilai null, kita akan menghapusnya...')
        data.dropna(inplace=True)
    else:
        pass

checkNA(df)

    df.isna().sum()

    df.info()

    df['tapInTime'] = pd.to_datetime(df['tapInTime'])
    df['tapInHour'] = df['tapInTime'].dt.hour
df['tapOutTime'] = pd.to_datetime(df['tapOutTime'])
    df['runtime'] = (df['tapOutTime'] - df['tapInTime'])
    df['travel_time'] =
pd.to_timedelta(df['runtime']).dt.total_seconds() / 60
    print(df["travel_time"])
    print(df["tapInHour"])
```

3. Fuzzy Logic System: Fuzzy input and output variables (travel_time, tapInHour, jumlah_penumpang, and bus_frequency) are defined using Antecedent and Consequent classes. Membership functions for each variable are defined using fuzzy sets such as 'good', 'average', 'poor', etc.

```
travel_time = ctrl.Antecedent(np.arange(0, 200, 1), 'travel_time')
#format dalam menit
tapInHour = ctrl.Antecedent(np.arange(0, 24, 1), 'tapInHour') #format 24
jam
jumlah_penumpang = ctrl.Antecedent(np.arange(0, 500, 10),
'jumlah_penumpang')

bus_frequency = ctrl.Consequent(np.arange(0, 11, 1), 'bus_frequency')

# Define membership functions (use descriptive names)
travel_time['good'] = fuzz.trapmf(travel_time.universe, [0, 0, 15, 30])
travel_time['average'] = fuzz.trimf(travel_time.universe, [15, 45, 75])
    travel_time['poor'] = fuzz.trapmf(travel_time.universe, [60, 90,
200, 201])
    travel_time.view()

    tapInHour['peak_morning'] = fuzz.trapmf(tapInHour.universe, [4, 7,
9, 11])
    tapInHour['afternoon'] = fuzz.trimf(tapInHour.universe, [10, 15,
18])
    tapInHour['peak_evening'] = fuzz.trapmf(tapInHour.universe, [16,
20, 23, 24])
    tapInHour.view()

    jumlah_penumpang['low'] = fuzz.trapmf(jumlah_penumpang.universe,
[0, 50, 100, 150])
    jumlah_penumpang['average'] =
fuzz.trimf(jumlah_penumpang.universe, [100, 250, 400])
    jumlah_penumpang['high'] = fuzz.trapmf(jumlah_penumpang.universe,
[300, 400, 500, 501])
    jumlah_penumpang.view()

```

```
bus_frequency['good'] = fuzz.trapmf(bus_frequency.universe, [0, 2, 3, 5])
bus_frequency['average'] = fuzz.trimf(bus_frequency.universe, [3, 5, 7])
bus_frequency['poor'] = fuzz.trimf(bus_frequency.universe, [6, 8, 10])
bus_frequency.view()
```

5. Fuzzy Rules: Fuzzy rules are defined to model the relationship between input and output variables based on fuzzy sets.

```
# Define fuzzy rules
rule1 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] & jumlah_penumpang['low'], bus_frequency['average'])
rule2 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] & jumlah_penumpang['average'], bus_frequency['average'])
rule3 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] & jumlah_penumpang['high'], bus_frequency['poor'])
rule4 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] & jumlah_penumpang['average'], bus_frequency['poor'])
rule5 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] & jumlah_penumpang['high'], bus_frequency['poor'])
rule6 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] & jumlah_penumpang['low'], bus_frequency['average'])
rule7 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] & jumlah_penumpang['high'], bus_frequency['poor'])
rule8 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] & jumlah_penumpang['low'], bus_frequency['good'])
rule9 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] & jumlah_penumpang['average'], bus_frequency['poor'])

rule10 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] & jumlah_penumpang['low'], bus_frequency['poor'])
rule11 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] & jumlah_penumpang['average'], bus_frequency['average'])
rule12 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] & jumlah_penumpang['high'], bus_frequency['poor'])
rule13 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] & jumlah_penumpang['average'], bus_frequency['good'])
rule14 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] & jumlah_penumpang['high'], bus_frequency['average'])
rule15 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] & jumlah_penumpang['low'], bus_frequency['average'])
rule16 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] & jumlah_penumpang['high'], bus_frequency['average'])
rule17 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] & jumlah_penumpang['low'], bus_frequency['average'])
rule18 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] & jumlah_penumpang['average'], bus_frequency['good'])

rule19 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning'] & jumlah_penumpang['low'], bus_frequency['poor'])
```

```
23.       rule20 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
    jumlah_penumpang['average'], bus_frequency['average'])
24.       rule21 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening']
    & jumlah_penumpang['high'], bus_frequency['good'])
25.       rule22 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning']
    & jumlah_penumpang['average'], bus_frequency['average'])
26.       rule23 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
    jumlah_penumpang['high'], bus_frequency['good'])
27.       rule24 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening']
    & jumlah_penumpang['low'], bus_frequency['poor'])
28.       rule25 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning']
    & jumlah_penumpang['high'], bus_frequency['good'])
29.       rule26 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
    jumlah_penumpang['low'], bus_frequency['poor'])
30. rule27 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening'] &
    jumlah_penumpang['average'], bus_frequency['average']
```

6. Sistem Kontrol dan Simulasi: Sistem kontrol dibuat menggunakan aturan-aturan yang telah didefinisikan. Sebuah simulasi diatur (bus_simulation) untuk menghitung output (bus_frequency) berdasarkan nilai input.

```
1. # Create a control system and simulate
2. bus_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
    rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule14,
    rule15,rule16,rule17,rule18,rule19,rule20,rule21,rule22,rule23,rule24,ru
    le25,rule26,rule27])
3. bus_simulation = ctrl.ControlSystemSimulation(bus_ctrl)
```

7. Input Pengguna dan Simulasi: Pengguna diminta untuk memasukkan nilai waktu perjalanan, jam tap-in, dan ID koridor untuk mensimulasikan frekuensi bus.

```
1. travel_time = input("Masukkan lama kamu berkendara dalam menit: ")
2. bus_simulation.input['travel_time'] = int(travel_time)
3.
4. tapInHour = input("Masukkan Jam Berapa: ")
5. bus_simulation.input['tapInHour'] = int(tapInHour)
6.
7. corridorID = input("Masukkan corridorID: ")
8. jumlah_penumpang = penumpang.loc[penumpang['corridorID'] == corridorID,
    'jumlah_penumpang'].values[0]
9. bus_simulation.input['jumlah_penumpang'] = jumlah_penumpang
10.
11.       bus_simulation.compute()
12.
13.       print("Bus frequency:",
    round(bus_simulation.output['bus_frequency'], 2))
14.       bus_frequency.view(sim=bus_simulation)
```

8. Simulation Loop and Output Results: A loop simulates the bus frequency for various scenarios, and the results are printed.

```
1. i = 0
2. total = 0
3. row = 37900
```

```
4.
5. while i < row:
6.     bus_frequency = bus_simulation.output['bus_frequency']
7.     bus_simulation.input['travel_time'] = int(travel_time)
8.     bus_simulation.input['tapInHour'] = int(tapInHour)
9.     corridorID = input("Masukkan corridorID: ")
10.          jumlah_penumpang = penumpang.loc[penumpang['corridorID'] ==
   corridorID, 'jumlah_penumpang'].values[0]
11.          bus_simulation.input['jumlah_penumpang'] = jumlah_penumpang
12.          bus_simulation.compute()
13.
14.          if (bus_simulation.output['bus_frequency'] < 10) and
   (jumlah_penumpang == 'average'):
15.              result = "✔"
16.              total += 1
17.          elif (bus_simulation.output['bus_frequency'] < 5) and
   (jumlah_penumpang == 'good'):
18.              result = "✔"
19.              total += 1
20.          elif (bus_simulation.output['bus_frequency'] > 10) and
   (jumlah_penumpang == 'poor'):
21.              result = "✔"
22.              total += 1
23.          else:
24.              result = "✖"
25.
26.          print(i + 1, bus_simulation.output['bus_frequency'],
   bus_frequency, result, sep="   ")
27.
28.          # Assuming 'data' is a DataFrame where you want to store
   results
29.          df.loc[i, 'Classification Result'] =
   bus_simulation.output['bus_frequency']
30.          df.to_csv("dfTransjakarta - New.csv", index=False)
31.          i += 1
32.
33.      print("")
34.      print(total, "/", row)
35. print("Accuracy = ", total / row * 100, "%")
```

WHOLE CODE

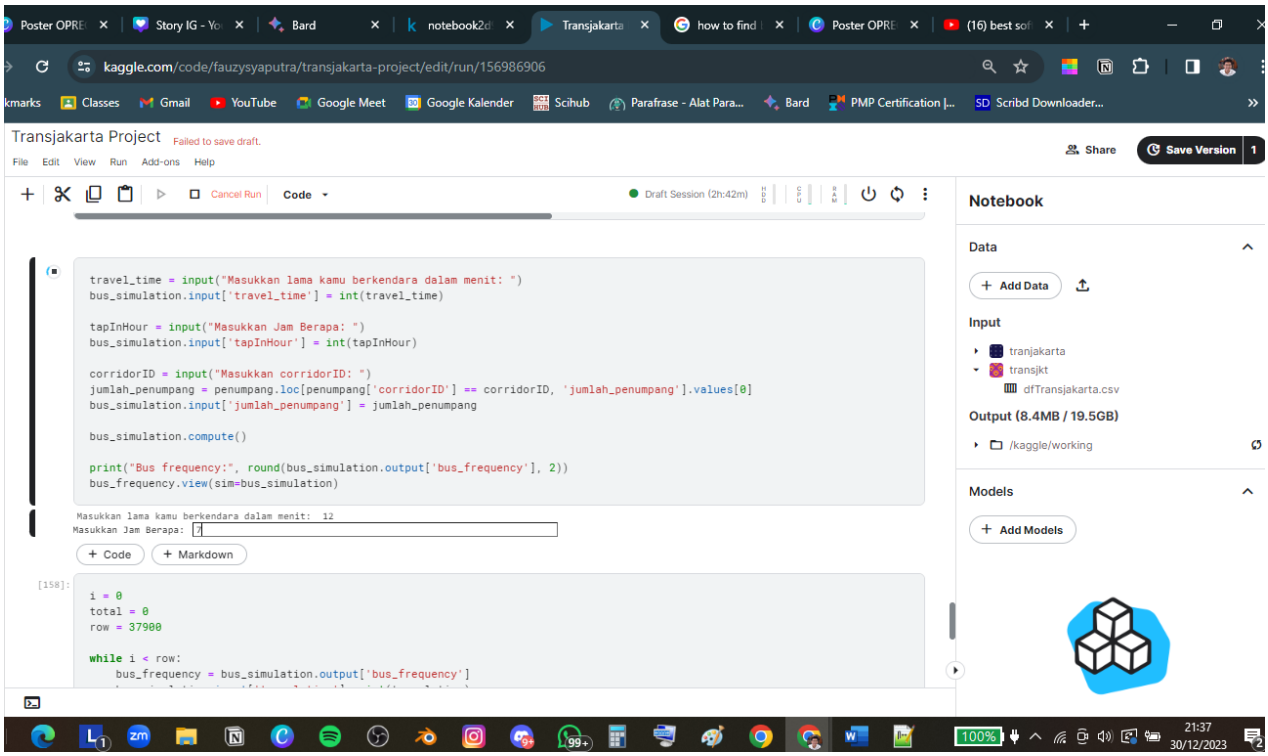
```
!pip install -U scikit-fuzzy
!pip install numpy
!pip install pandas
```

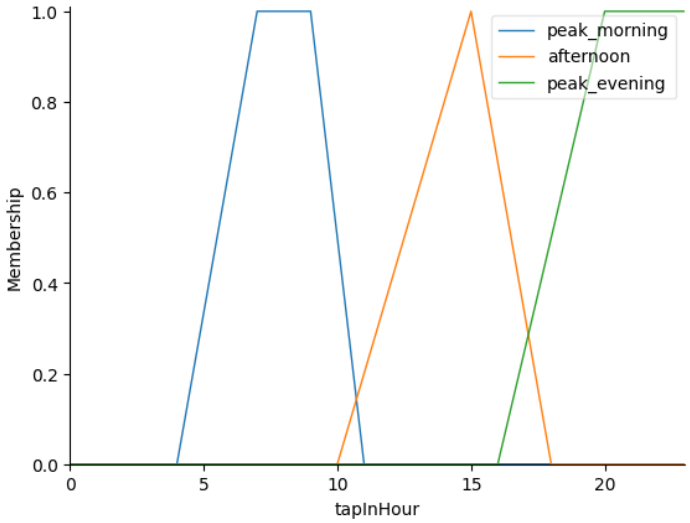
```
#Install module scikitfuzzy, numpy, dan pandas
import pandas as pd
import numpy as np
import datetime
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Load  dataset

df = pd.read_csv('/kaggle/input/transjkt/dfTransjakarta.csv')
print(df)

def checkNA(data):
    if data.isna().sum().any():
        print('There are null values, we will remove them...')
        data.dropna(inplace=True)
    else:
        pass

checkNA(df)

df.isna().sum()

df.info()

df['tapInTime'] = pd.to_datetime(df['tapInTime'])
df['tapInHour'] = df['tapInTime'].dt.hour
df['tapOutTime'] = pd.to_datetime(df['tapOutTime'])
df['runtime'] = (df['tapOutTime'] - df['tapInTime'])
#mengubah ke menit
df['travel_time'] = pd.to_timedelta(df['runtime']).dt.total_seconds() / 60
print(df["travel_time"])
print(df["tapInHour"])

penumpang =
df.groupby('corridorID')['corridorID'].count().reset_index(name='jumlah_penump
ang')
for index, row in penumpang.iterrows():
    print(f"corridorID: {row['corridorID']}, JumlahPenumpang:
{row['jumlah_penumpang']}")

print(penumpang)

corridorID = '10'  # Replace with the actual corridorID
jumlah_penumpang = penumpang.loc[penumpang['corridorID'] == corridorID,
'jumlah_penumpang'].values[0]
print("Jumlah penumpang for corridor", corridorID, ":", jumlah_penumpang)

travel_time = ctrl.Antecedent(np.arange(0, 200, 1), 'travel_time') #format
dalam menit
tapInHour = ctrl.Antecedent(np.arange(0, 24, 1), 'tapInHour') #format 24 jam
jumlah_penumpang = ctrl.Antecedent(np.arange(0, 500, 10), 'jumlah_penumpang')

bus_frequency = ctrl.Consequent(np.arange(0, 11, 1), 'bus_frequency')

# Define membership functions (use descriptive names)
travel_time['good'] = fuzz.trapmf(travel_time.universe, [0, 0, 15, 30])
travel_time['average'] = fuzz.trimf(travel_time.universe, [15, 45, 75])
travel_time['poor'] = fuzz.trapmf(travel_time.universe, [60, 90, 200, 201])
travel_time.view()
```

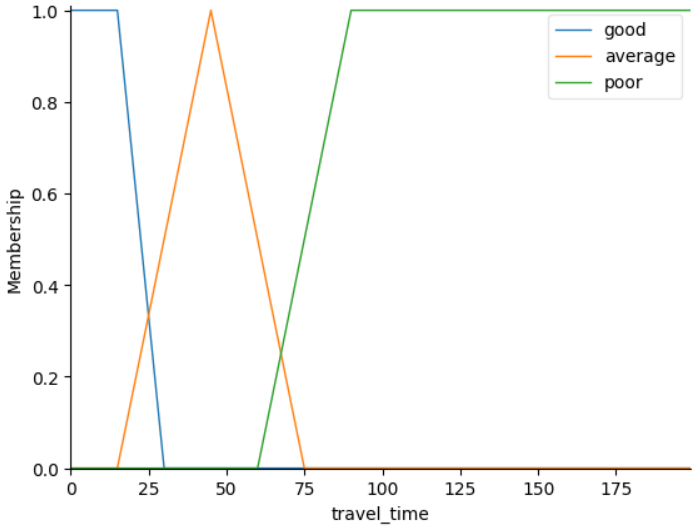
```
tapInHour['peak_morning'] = fuzz.trapmf(tapInHour.universe, [4, 7, 9, 11])
tapInHour['afternoon'] = fuzz.trimf(tapInHour.universe, [10, 15, 18])
tapInHour['peak_evening'] = fuzz.trapmf(tapInHour.universe, [16, 20, 23, 24])
tapInHour.view()

jumlah_penumpang['low'] = fuzz.trapmf(jumlah_penumpang.universe, [0, 50, 100,
150])
jumlah_penumpang['average'] = fuzz.trimf(jumlah_penumpang.universe, [100, 250,
400])
jumlah_penumpang['high'] = fuzz.trapmf(jumlah_penumpang.universe, [300, 400,
500, 501])
jumlah_penumpang.view()

bus_frequency['good'] = fuzz.trapmf(bus_frequency.universe, [0, 2, 3, 5])
bus_frequency['average'] = fuzz.trimf(bus_frequency.universe, [3, 5, 7])
bus_frequency['poor'] = fuzz.trimf(bus_frequency.universe, [6, 8, 10])
bus_frequency.view()

# Define fuzzy rules
rule1 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] &
jumlah_penumpang['low'], bus_frequency['average'])
rule2 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] &
jumlah_penumpang['average'], bus_frequency['average'])
rule3 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] &
jumlah_penumpang['high'], bus_frequency['poor'])
rule4 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] &
jumlah_penumpang['average'], bus_frequency['poor'])
rule5 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] &
jumlah_penumpang['high'], bus_frequency['poor'])
rule6 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] &
jumlah_penumpang['low'], bus_frequency['average'])
rule7 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_morning'] &
jumlah_penumpang['high'], bus_frequency['poor'])
rule8 = ctrl.Rule(travel_time['poor'] & tapInHour['afternoon'] &
jumlah_penumpang['low'], bus_frequency['good'])
rule9 = ctrl.Rule(travel_time['poor'] & tapInHour['peak_evening'] &
jumlah_penumpang['average'], bus_frequency['poor'])

rule10 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] &
jumlah_penumpang['low'], bus_frequency['poor'])
rule11 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] &
jumlah_penumpang['average'], bus_frequency['average'])
rule12 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] &
jumlah_penumpang['high'], bus_frequency['poor'])
rule13 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] &
jumlah_penumpang['average'], bus_frequency['good'])
rule14 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] &
jumlah_penumpang['high'], bus_frequency['average'])
rule15 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] &
jumlah_penumpang['low'], bus_frequency['average'])
rule16 = ctrl.Rule(travel_time['average'] & tapInHour['peak_morning'] &
jumlah_penumpang['high'], bus_frequency['average'])
rule17 = ctrl.Rule(travel_time['average'] & tapInHour['afternoon'] &
jumlah_penumpang['low'], bus_frequency['average'])
rule18 = ctrl.Rule(travel_time['average'] & tapInHour['peak_evening'] &
jumlah_penumpang['average'], bus_frequency['good'])

rule19 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning'] &
jumlah_penumpang['low'], bus_frequency['poor'])
rule20 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
jumlah_penumpang['average'], bus_frequency['average'])
```

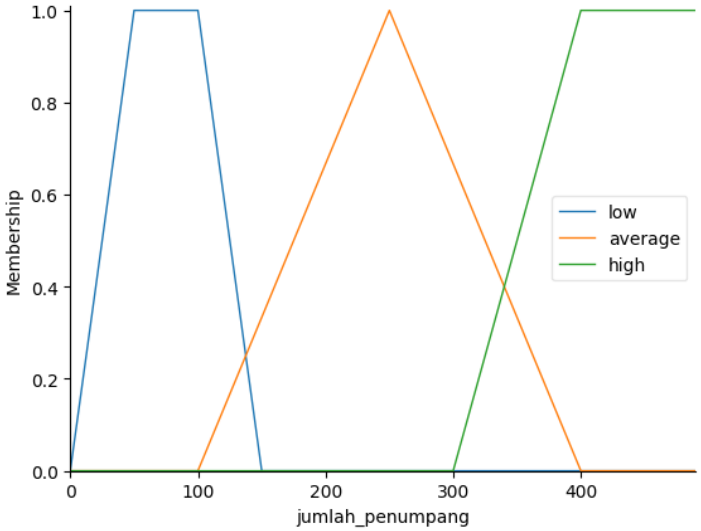
```
rule21 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening'] &
jumlah_penumpang['high'], bus_frequency['good'])
rule22 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning'] &
jumlah_penumpang['average'], bus_frequency['average'])
rule23 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
jumlah_penumpang['high'], bus_frequency['good'])
rule24 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening'] &
jumlah_penumpang['low'], bus_frequency['poor'])
rule25 = ctrl.Rule(travel_time['good'] & tapInHour['peak_morning'] &
jumlah_penumpang['high'], bus_frequency['good'])
rule26 = ctrl.Rule(travel_time['good'] & tapInHour['afternoon'] &
jumlah_penumpang['low'], bus_frequency['poor'])
rule27 = ctrl.Rule(travel_time['good'] & tapInHour['peak_evening'] &
jumlah_penumpang['average'], bus_frequency['average'])


# Create a control system and simulate
bus_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule14,
rule15,rule16,rule17,rule18,rule19,rule20,rule21,rule22,rule23,rule24,rule25,r
ule26,rule27])
bus_simulation = ctrl.ControlSystemSimulation(bus_ctrl)

travel_time = input("Masukkan lama kamu berkendara dalam menit: ")
bus_simulation.input['travel_time'] = int(travel_time)

tapInHour = input("Masukkan Jam Berapa: ")
bus_simulation.input['tapInHour'] = int(tapInHour)

corridorID = input("Masukkan corridorID: ")
jumlah_penumpang = penumpang.loc[penumpang['corridorID'] == corridorID,
'jumlah_penumpang'].values[0]
bus_simulation.input['jumlah_penumpang'] = jumlah_penumpang

bus_simulation.compute()

print("Bus frequency:", round(bus_simulation.output['bus_frequency'], 2))
bus_frequency.view(sim=bus_simulation)

i = 0
total = 0
row = 37900

while i < row:
    bus_frequency = bus_simulation.output['bus_frequency']
    bus_simulation.input['travel_time'] = int(travel_time)
    bus_simulation.input['tapInHour'] = int(tapInHour)
    corridorID = input("Masukkan corridorID: ")
    jumlah_penumpang = penumpang.loc[penumpang['corridorID'] == corridorID,
'jumlah_penumpang'].values[0]
    bus_simulation.input['jumlah_penumpang'] = jumlah_penumpang
    bus_simulation.compute()

    if (bus_simulation.output['bus_frequency'] < 10) and (jumlah_penumpang ==
'average'):
        result = "✔"
        total += 1
    elif (bus_simulation.output['bus_frequency'] < 5) and (jumlah_penumpang ==
'good'):
        result = "✔"
        total += 1
```

```
    elif (bus_simulation.output['bus_frequency'] > 10) and (jumlah_penumpang
== 'poor'):
        result = "✔"
        total += 1
    else:
        result = "✖"

    print(i + 1, bus_simulation.output['bus_frequency'], bus_frequency,
result, sep="    ")

    # Assuming 'data' is a DataFrame where you want to store results
    df.loc[i, 'Classification Result'] =
bus_simulation.output['bus_frequency']
    df.to_csv("dfTransjakarta - New.csv", index=False)
    i += 1

print("")
print(total, "/", row)
print("Accuracy = ", total / row * 100, "%")
```

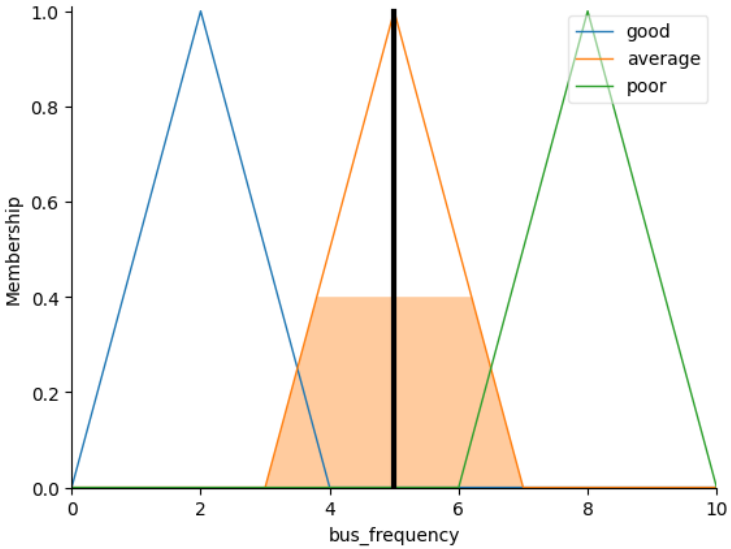








Enter the length of your drive in minutes: 12
Enter What Time: 12
Input corridorID: 1
Bus frequency: 5.0



Input corridorID: 2
1 5.0 5.0 ✗