

FINAL PROJECT

# UBER FARE PREDICTION

Presented by **The Outliers**

# OUR TEAM

Muhammad **Afif** Y.G.

**Peri** Pirdaus

**Anel** Fuad Abiyyu

**Eugenia** Jean Yovini

**Zhafira** Haura

**Brian** Adam Bhagaskara

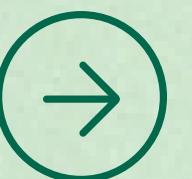
# Tujuan Penelitian

Penelitian ini bertujuan untuk **membangun model regresi** yang dapat **memperkirakan tarif perjalanan Uber (fare\_amount)** secara akurat sebelum perjalanan dimulai, menggunakan variabel-variabel historis seperti:

- **Lokasi penjemputan dan pengantaran**
- **Waktu pemesanan**
- **Jumlah penumpang**

## Tujuan Utama

- Menjawab faktor apa saja yang paling berpengaruh terhadap tarif
- Menilai tingkat akurasi model prediksi
- Memberikan dukungan untuk strategi harga dinamis, transparansi tarif, dan segmentasi pelanggan



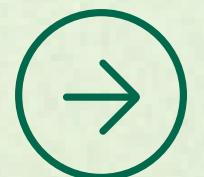
# Metodologi Penelitian

## Jenis Penelitian

Penelitian ini merupakan **penelitian kuantitatif prediktif** yang dilakukan melalui proses analisis data dan penerapan **algoritma machine learning**. Fokus utamanya adalah **membangun dan membandingkan model regresi** untuk **memprediksi tarif perjalanan Uber berdasarkan fitur lokasi, waktu, dan jumlah penumpang**.

## Sumber dan Jenis Data

- **Jenis data:** Data sekunder dalam bentuk tabular, terdiri dari variabel numerik dan kategorikal.
- **Sumber data:** Dataset publik dari Kaggle ("Uber Fare Prediction"), yang berisi data historis perjalanan Uber di New York City. (<https://www.kaggle.com/datasets/yasserh/uber-fares-dataset/data>)
- **Fitur penting:** pickup\_datetime, pickup\_longitude, pickup\_latitude, dropoff\_longitude, dropoff\_latitude, passenger\_count, dan fare\_amount.



# BUSINESS UNDERSTANDING



# Tahapan Analisis

Penelitian ini menggunakan pendekatan **Cross Industry Standard Process for Data Mining (CRISP-DM)** sebagai kerangka kerja analisis data, dari pemahaman bisnis hingga implementasi model. Pendekatan ini membuat **proses lebih sistematis dan selaras dengan kebutuhan bisnis** Uber dalam memprediksi tarif perjalanan secara akurat.

## Business Understanding

- **Konteks Bisnis:** Uber melayani jutaan perjalanan setiap hari dengan skala data yang sangat besar. Dalam sistem transportasi berbasis aplikasi, tarif perjalanan (`fare_amount`) merupakan elemen penting yang menentukan kepuasan pelanggan dan profitabilitas perusahaan.
- **Permasalahan Bisnis:** Tarif yang tidak akurat atau tidak konsisten dapat mengurangi kepercayaan pengguna, memicu persepsi harga yang tidak adil, dan menyulitkan strategi pricing dinamis.
- **Kebutuhan Bisnis:** Uber memerlukan sistem prediksi tarif otomatis yang akurat, berdasarkan variabel yang tersedia saat pemesanan seperti lokasi, waktu, jumlah penumpang, jarak tempuh, hari dan jam perjalanan, serta musim.

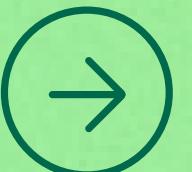


# DATA UNDERSTANDING & EDA AWAL



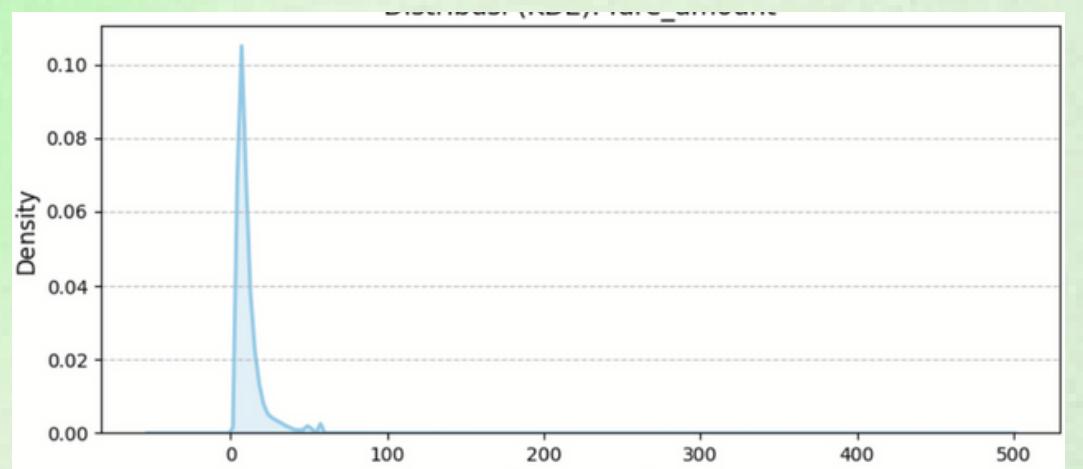
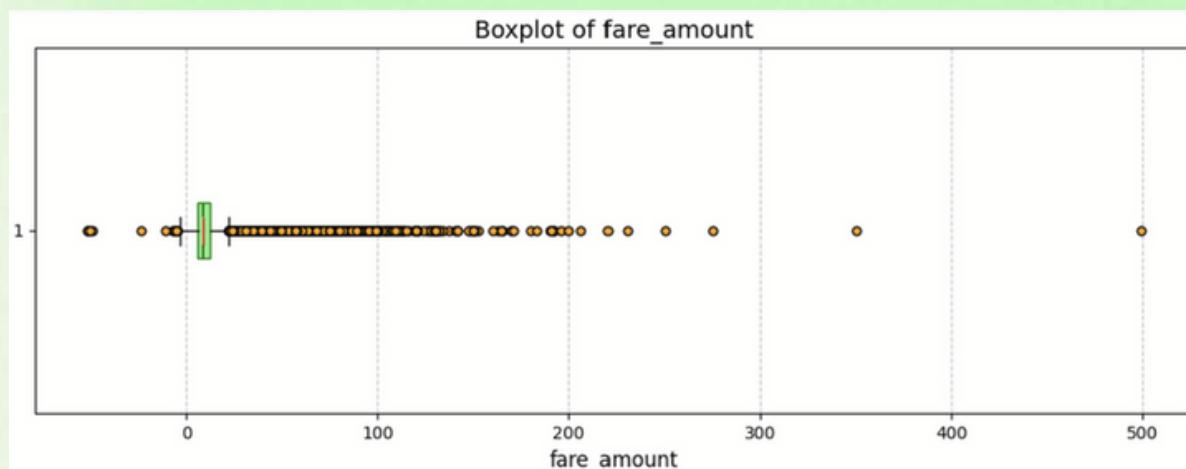
# Data Understanding & EDA Awal

Kolom	Deskripsi	Tipe Data	Dipakai?	Alasan
<b>Unnamed: 0</b>	Index dari file (ID urut hasil ekspor CSV).	Integer (ID)	✗	Hanya nomor urut, tidak ada informasi berguna.
<b>key</b>	Timestamp unik (gabungan tanggal dan ID).	String/Timestamp	✗	Duplikat dengan pickup_datetime, jarang dipakai kecuali untuk identifikasi unik.
<b>fare_amount</b>	Jumlah biaya (target yang dianalisis atau diprediksi).	Float	✓	Variabel target utama untuk analisis/modeling.
<b>pickup_datetime</b>	Waktu penjemputan.	Datetime	✓	Bisa diekstrak menjadi fitur turunan (hari, jam, bulan, dll.).
<b>pickup_longitude</b>	Koordinat longitude titik penjemputan.	Float	✓	Digunakan untuk menghitung jarak atau zona.
<b>pickup_latitude</b>	Koordinat latitude titik penjemputan.	Float	✓	Digunakan bersama longitude untuk menghitung jarak.
<b>dropoff_longitude</b>	Koordinat longitude titik tujuan.	Float	✓	Digunakan untuk menghitung jarak perjalanan.
<b>dropoff_latitude</b>	Koordinat latitude titik tujuan.	Float	✓	Digunakan untuk perhitungan jarak.
<b>passenger_count</b>	Jumlah penumpang dalam perjalanan.	Integer	✓	Bisa berpengaruh pada tarif atau jadi fitur tambahan.



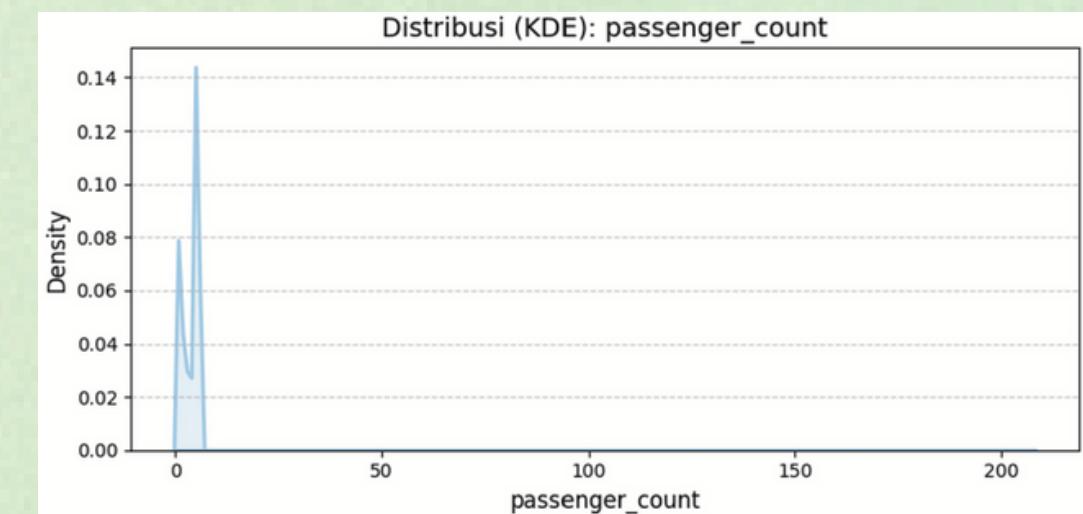
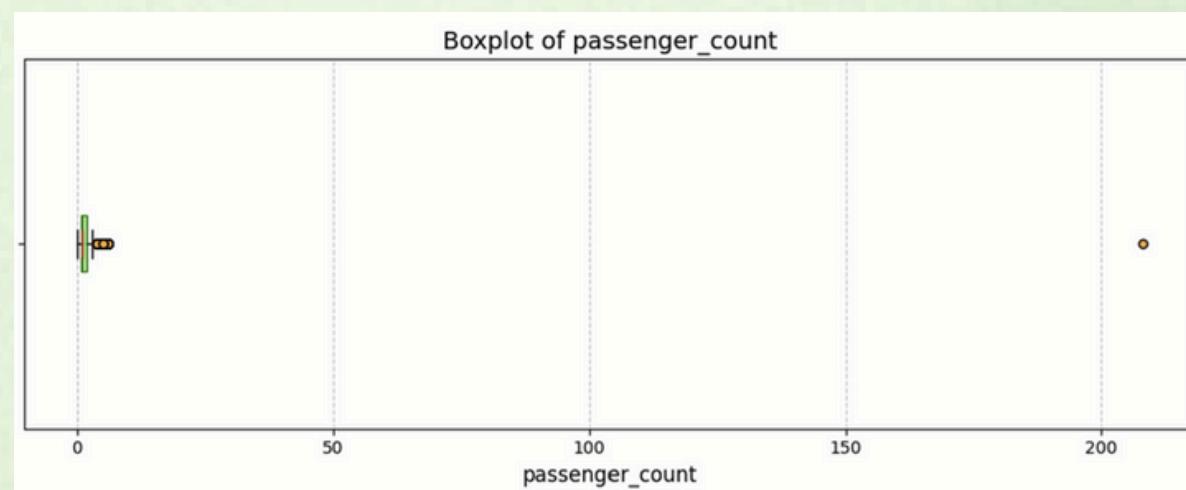
# Data Understanding & EDA Awal

- **Fare Amount**



`fare_amount` memiliki distribusi miring ke kanan dengan banyak outlier, sebagian besar nilai berada di kisaran 0–20, namun terdapat nilai ekstrem hingga sekitar 500 yang perlu dicek lebih lanjut.

- **Passenger Count**



`passenger_count` didominasi nilai 1–6, namun terdapat outlier ekstrem (208) yang kemungkinan error dan perlu dibersihkan.



# DATA PREPARATION & EDA AKHIR



# Data Preparation & EDA Akhir

## 1. Membagi berdasarkan tahun, bulan, hari, dan jam

```
[ ] #Membuat kolom tahun,bulan,hari dan jam dari pickup_datetime
df['pickup_year'] = df['pickup_datetime'].dt.year
df['pickup_month'] = df['pickup_datetime'].dt.month
df['pickup_weekday'] = df['pickup_datetime'].dt.weekday
df['pickup_hour'] = df['pickup_datetime'].dt.hour
```

pickup_year	pickup_month	pickup_weekday	pickup_hour
2015	5	3	19
2009	7	4	20
2009	8	0	21
2009	6	4	8
2014	8	3	17

Langkah ini penting karena **waktu pemesanan dapat memengaruhi harga tarif Uber**, misalnya pada jam sibuk atau akhir pekan tarif cenderung lebih tinggi. Dengan memecah pickup\_datetime, kita bisa **menangkap pola temporal tersebut secara lebih detail** untuk membantu model prediksi.

## 2. Musim berdasarkan bulan

```
[ ] #Extract Fitur Bulan Dijadikan Musim
def get_season(mnth):
    if mnth in [12,1,2]:
        return 'winter'
    elif mnth in [3,4,5]:
        return 'spring'
    elif mnth in [6,7,8]:
        return 'summer'
    elif mnth in [9,10,11]:
        return 'fall'

df['pickup_season'] = df['pickup_month'].apply(get_season)
season_map = {'winter': 0, 'spring': 1, 'summer': 2, 'fall': 3}
df['season_code'] = df['pickup_season'].map(season_map)
```

pickup_season	season_code
spring	1
summer	2

Mengubah **data bulan menjadi fitur musim** untuk menambahkan konteks temporal (musiman) ke data, yang bisa berpengaruh terhadap pola perjalanan.

**Pola perjalanan dapat bervariasi tergantung musim** (misalnya, lebih ramai saat summer atau winter holiday), sehingga fitur ini bisa meningkatkan akurasi model prediktif.



# Data Preparation & EDA Akhir

## 3. Kategori Jam

```
[ ] #Extract Fitur Waktu Dijadikan Rush Hour
def hour_category(hour):
    if 7 <= hour <= 9 or 16 <= hour <= 19:
        return 'rush_hour'
    elif 0 <= hour <= 5:
        return 'night'
    else:
        return 'off_peak'

[ ] df['pickup_hour_category'] = df['pickup_hour'].apply(hour_category)
```

```
[ ] #Menghapus Fare Amount Negatif, tidak mungkin jika negatif
df = df.drop(df[df['fare_amount'] < 0].index)
```

Mengelompokkan **jam menjadi** kategori waktu (**rush hour, night, off-peak**) untuk memahami **perilaku perjalanan berdasarkan waktu**.

Kategori waktu ini membantu **mengidentifikasi kapan permintaan transportasi meningkat** (misalnya saat rush hour), serta memperbaiki analisis tren dan perencanaan operasional.

## 4. Memvalidasi Koordinat

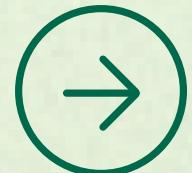
```
[ ] #Menyaring koordinat tidak valid:
#Baris dengan nilai latitude di luar -90 sampai 90 atau longitude di luar -180
#sampai 180 dihapus karena tidak sesuai dengan batas koordinat

#Filter lokasi: hanya wilayah NYC
df = df[
    (df['pickup_latitude'].between(-90, 90)) &
    (df['pickup_longitude'].between(-180, 180)) &
    (df['dropoff_latitude'].between(-90, 90)) &
    (df['dropoff_longitude'].between(-180, 180)) &
    (df['pickup_latitude'] != 0) &
    (df['pickup_longitude'] != 0) &
    (df['dropoff_latitude'] != 0) &
    (df['dropoff_longitude'] != 0)
]
```

```
[ ] #Tidak mungkin tidak membawa penumpang dan melebihi 6
df = df.drop(df[(df['passenger_count'] > 6) | (df['passenger_count'] < 1)].index)
```

```
[ ] print(df['passenger_count'].value_counts())
→ passenger_count
1    135626
2     28863
5     13738
3      8705
6      4202
4      4182
Name: count, dtype: int64
```

Data dengan lokasi tidak logis atau nol bisa mengganggu akurasi model dan perlu dibersihkan agar hanya mencakup wilayah NYC.



# Data Preparation & EDA Akhir

## 5. Menghitung Jarak (Rumus Haversine)

Untuk **menghitung jarak antara dua titik di permukaan bumi**, digunakan Rumus Haversine yang memperhitungkan kelengkungan bumi, berdasarkan koordinat lintang (latitude) dan bujur (longitude).

$$d = 2R \times \arcsin \left( \sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \times \cos(\phi_2) \times \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \right)$$

Keterangan:

- $\phi_1, \phi_2$  : lintang titik 1 dan titik 2 (dalam radian)
- $\lambda_1, \lambda_2$  : bujur titik 1 dan titik 2 (dalam radian)
- $\Delta\phi = \phi_2 - \phi_1$  : selisih lintang
- $\Delta\lambda = \lambda_2 - \lambda_1$  : selisih bujur
- $R$  : jari-jari bumi (sekitar 6371 km)
- $d$  : jarak antar dua titik (dalam kilometer)
- (d) : jarak antar dua titik (dalam kilometer)

**Konversi ke radian:**

$$\phi = \text{latitude} \times \left(\frac{\pi}{180}\right), \quad \lambda = \text{longitude} \times \left(\frac{\pi}{180}\right)$$

```
[ ] # Mengimpor modul matematika untuk operasi trigonometri dan konversi derajat ke radian
import math

# Mendefinisikan fungsi haversine untuk menghitung jarak antara dua titik di permukaan bumi
def haversine(lat1, lon1, lat2, lon2):
    R = 6371 # Jari-jari bumi dalam kilometer

    # Mengubah semua koordinat dari derajat ke radian agar bisa digunakan dalam fungsi trigonometri
    lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2])

    # Menghitung selisih lintang dan bujur
    dlat = lat2 - lat1
    dlon = lon2 - lon1

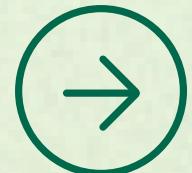
    # Menerapkan Rumus Haversine
    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
    c = 2 * math.asin(math.sqrt(a)) # Menghitung jarak dalam bentuk sudut (radian)

    return R * c # Mengalikan dengan jari-jari bumi untuk mendapatkan jarak dalam kilometer

# Menerapkan fungsi haversine ke setiap baris DataFrame untuk menghitung kolom baru 'distance'
df['dist'] = df.apply(
    lambda row: haversine(
        row['pickup_latitude'], row['pickup_longitude'], # Titik awal (penjemputan)
        row['dropoff_latitude'], row['dropoff_longitude'] # Titik akhir (tujuan)
    ),
    axis=1 # axis=1 artinya fungsi diterapkan ke setiap baris (bukan kolom)
)

[ ] df_nyc_pickup = df[
    (df['pickup_longitude'] >= -74.05) & (df['pickup_longitude'] <= -73.75) &
    (df['pickup_latitude'] >= 40.5) & (df['pickup_latitude'] <= 40.92)
]
```

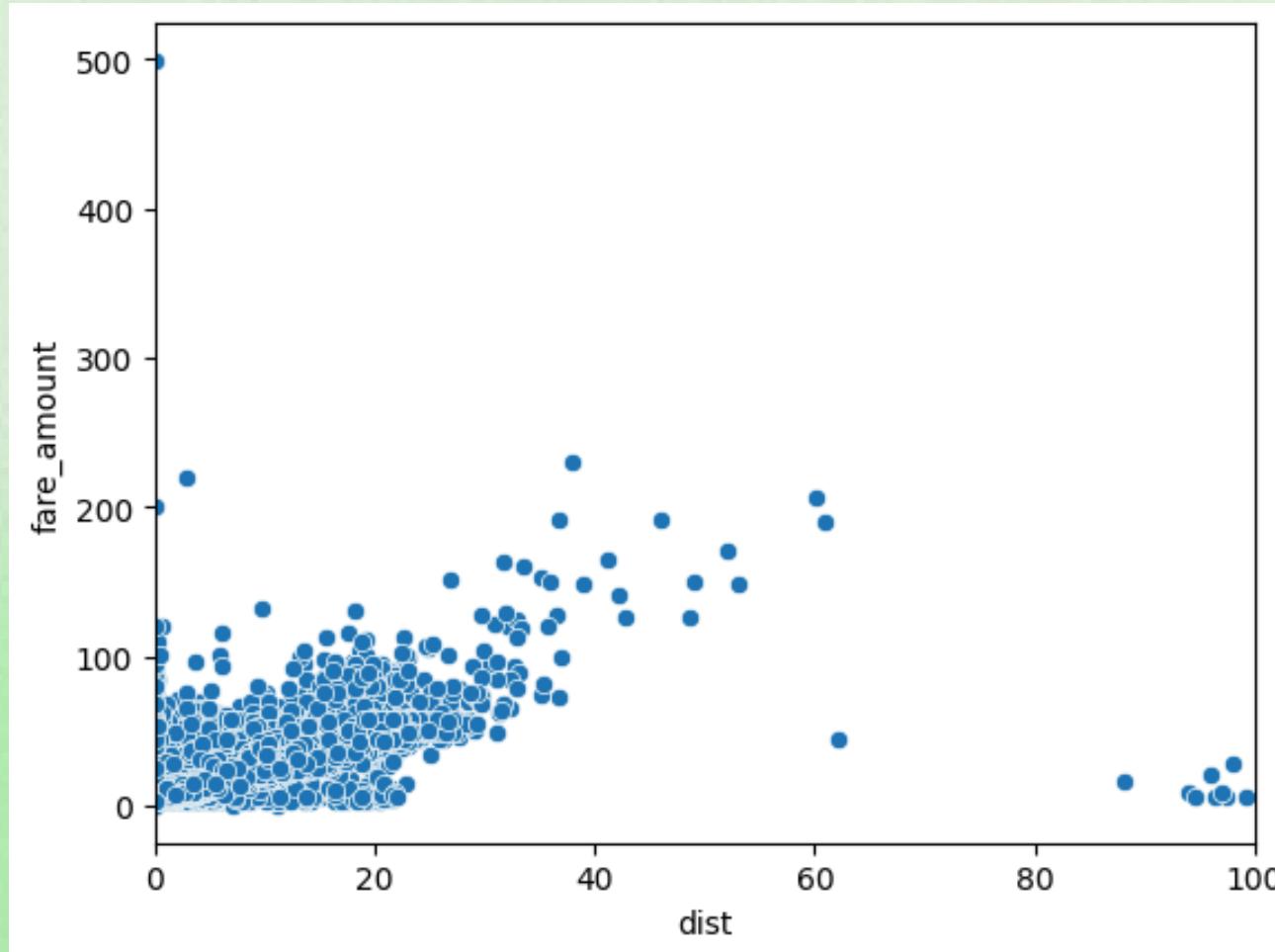
Jarak perjalanan menjadi fitur penting karena sangat memengaruhi tarif dan durasi perjalanan; semakin jauh jaraknya, biasanya tarif dan waktu tempuh semakin tinggi. Jarak membantu mendeteksi outlier.



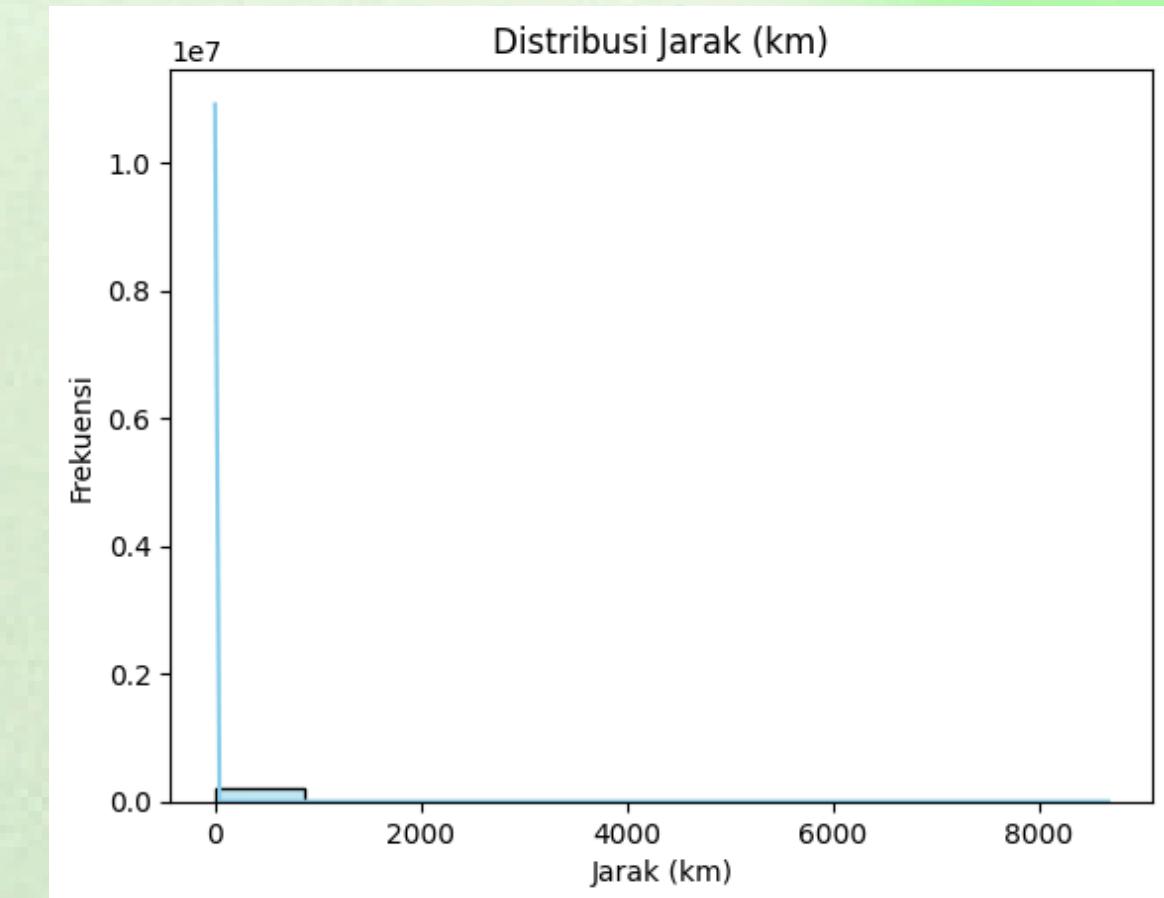
# Data Preparation & EDA Akhir

## 6. Scatterplot & Histplot

```
sns.scatterplot(x=df_nyc_pickup['dist'], y=df_nyc_pickup['fare_amount'])
plt.xlim([0, 100])
```



Scatterplot menunjukkan hubungan positif antara jarak dan tarif, semakin jauh jarak perjalanan, umumnya semakin tinggi tarifnya.



Terlalu banyak outlier. Histplot menunjukkan distribusi jarak dan tarif yang tidak merata; sebagian besar perjalanan berjarak pendek dengan tarif rendah, sementara perjalanan jauh lebih jarang terjadi.



# Data Preparation & EDA Akhir

## 7. Filtering Jarak Ekstrem

```
[ ] df_nyc_pickup[df_nyc_pickup['dist'] > 100]
```

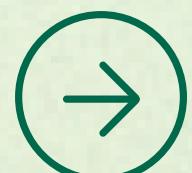
	pickup_year	pickup_month	pickup_weekday	pickup_hour	pickup_season	season_code	pickup_hour_category	dist
	2012	2	5	11	winter	0	off_peak	5651.396519
	2011	2	0	16	winter	0	rush_hour	126.654338
	2011	9	6	11	fall	3	off_peak	4445.382454
	2011	2	5	2	winter	0	night	125.275929
	2011	5	5	7	spring	1	rush_hour	176.519082

```
[ ] df_nyc_pickup[(df_nyc_pickup['dist'] > 60) & (df_nyc_pickup['dist'] < 100)]
df60_100 = df_nyc_pickup[(df_nyc_pickup['dist'] > 60) & (df_nyc_pickup['dist'] < 100)]
```

	pickup_year	pickup_month	pickup_weekday	pickup_hour	pickup_season	season_code	pickup_hour_category	dist
1	2012	7	6	8	summer	2	rush_hour	60.851156
1	2014	11	6	22	fall	3	off_peak	60.100889
1	2010	6	2	21	summer	2	off_peak	96.021362
1	2010	6	0	9	summer	2	rush_hour	97.382812
1	2009	9	0	17	fall	3	rush_hour	94.009832

Menyeleksi data dengan jarak tempuh sangat tinggi (>100 km) dan membuat subset untuk data dengan jarak antara 60–100 km.

Banyak data dengan jarak tempuh ekstrem yang kemungkinan outlier atau error input. Menyaring data ini penting agar hasil analisis lebih masuk akal dan tidak bias.



# Data Preparation & EDA Akhir

## 8. Visualisasi Peta

```
[ ] import folium

# Ambil titik awal sebagai pusat peta (tengah NYC)
nyc_center = [40.7128, -74.0060] # NYC lat-lon

# Inisialisasi peta
m = folium.Map(location=nyc_center, zoom_start=9)

# Ambil 100 sampel untuk efisiensi visualisasi
sampled_df = df60_100.sample(n=10, random_state=1)

# Tambahkan garis perjalanan dan titik pickup/dropoff
for _, row in sampled_df.iterrows():
    pickup = [row['pickup_latitude'], row['pickup_longitude']]
    dropoff = [row['dropoff_latitude'], row['dropoff_longitude']]

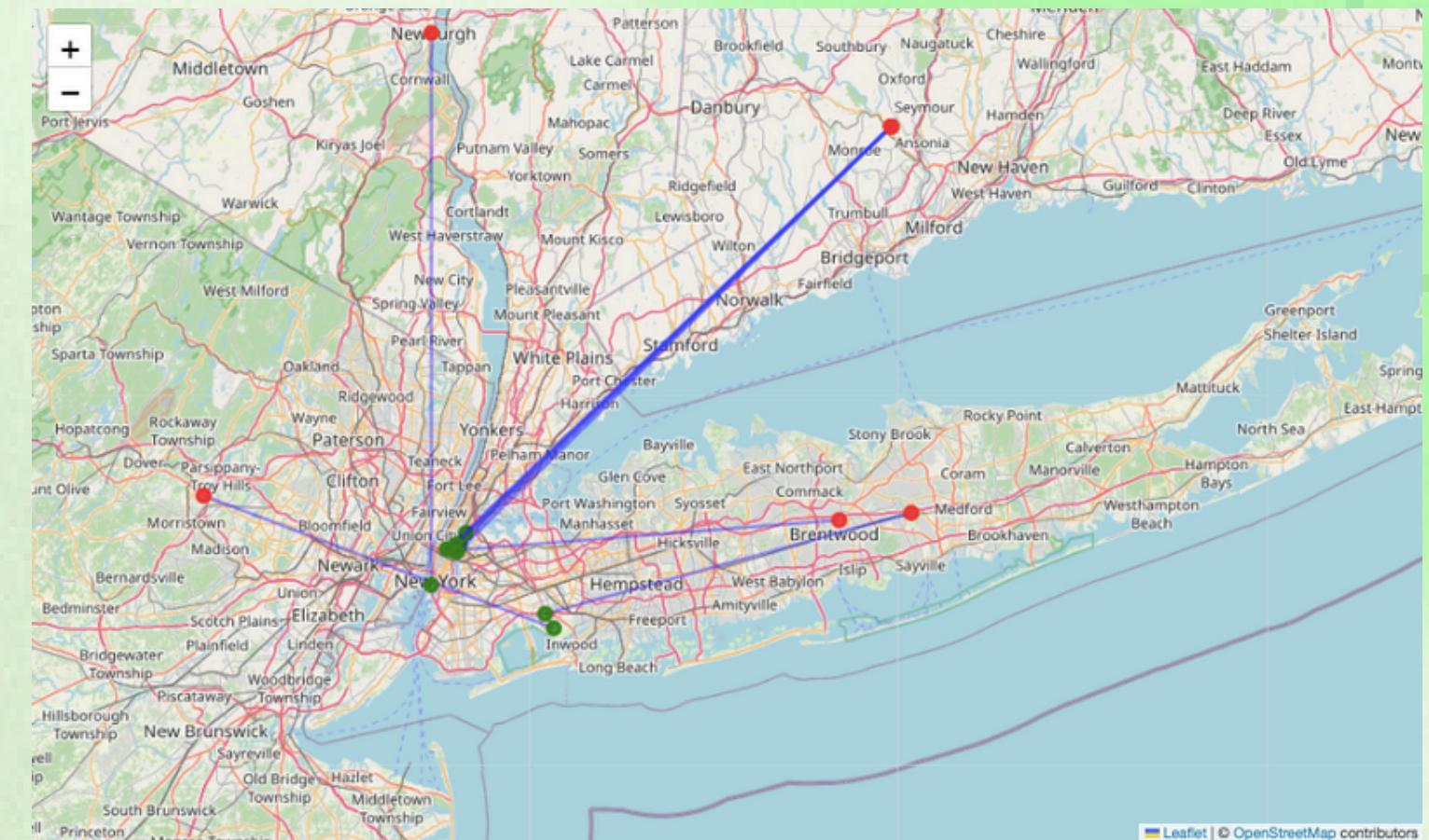
    # Garis rute
    folium.PolyLine(locations=[pickup, dropoff], color='blue', weight=2, opacity=0.5).add_to(m)

    # Titik pickup
    folium.CircleMarker(location=pickup, radius=4, color='green', fill=True, fill_opacity=0.7).add_to(m)

    # Titik dropoff
    folium.CircleMarker(location=dropoff, radius=4, color='red', fill=True, fill_opacity=0.7).add_to(m)

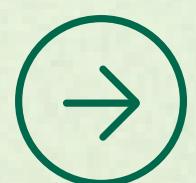
# Tampilkan peta
m
```

Terlihat tidak meyakinkan > 60km, kita cek 40km



Memetakan rute perjalanan dengan jarak lebih dari 60 km menggunakan peta interaktif (folium) untuk memverifikasi keakuratan data.

Terlihat beberapa perjalanan dengan rute yang tidak masuk akal atau tidak sesuai kenyataan geografis. Ini memunculkan keraguan atas validitas data tersebut.



# Data Preparation & EDA Akhir

## 9. Visualisasi Peta

```

df_35_61 = df_nyc_pickup[(df_nyc_pickup['dist'] > 35) & (df_nyc_pickup['dist'] < 61)]
df_35_61

[1]: import folium

# Ambil titik awal sebagai pusat peta (tengah NYC)
nyc_center = [40.7128, -74.0060] # NYC lat-lon

# Inisialisasi peta
m = folium.Map(location=nyc_center, zoom_start=9)

# Ambil 100 sampel untuk efisiensi visualisasi
sampled_df = df_35_61

# Tambahkan garis perjalanan dan titik pickup/dropoff
for _, row in sampled_df.iterrows():
    pickup = [row['pickup_latitude'], row['pickup_longitude']]
    dropoff = [row['dropoff_latitude'], row['dropoff_longitude']]

    # Garis rute
    folium.PolyLine(locations=[pickup, dropoff], color='blue', weight=2, opacity=0.5).add_to(m)

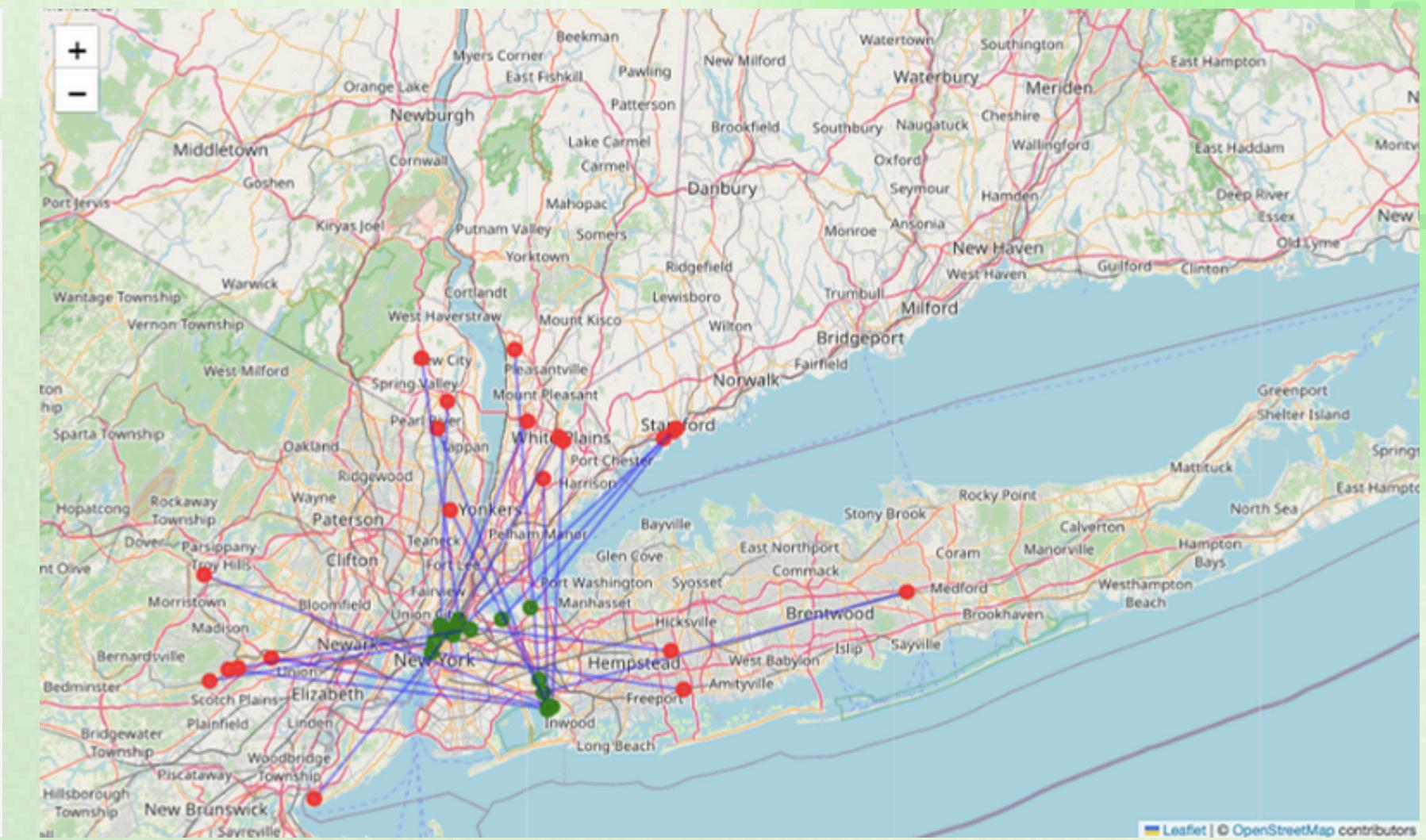
    # Titik pickup
    folium.CircleMarker(location=pickup, radius=4, color='green', fill=True, fill_opacity=0.7).add_to(m)

    # Titik dropoff
    folium.CircleMarker(location=dropoff, radius=4, color='red', fill=True, fill_opacity=0.7).add_to(m)

# Tampilkan peta
m

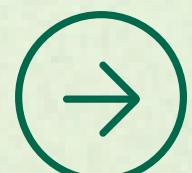
```

Terlihat tidak meyakinkan > 60km, kita cek 40km



Menurunkan threshold ke jarak 35–60 km untuk mengecek apakah perjalanan di range ini lebih wajar.

Pola perjalanan terlihat lebih realistik dalam kisaran ini, mengindikasikan bahwa pembatasan jarak bisa membantu menghilangkan data noise atau error.

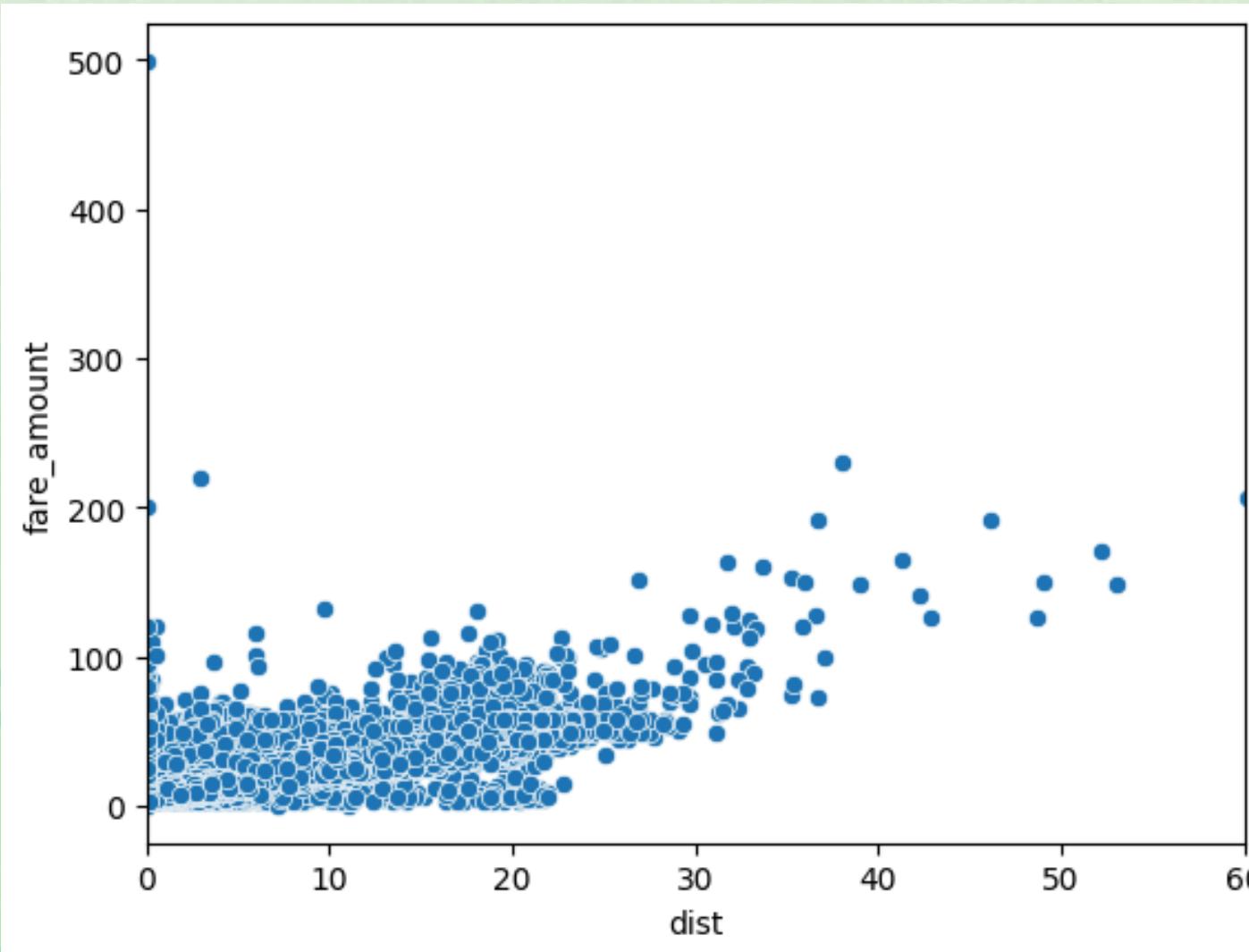


# Data Preparation & EDA Akhir

## 10. Visualisasi Peta

```
[ ] #Kita drop diatas 61km  
df_clear = df_nyc_pickup[df_nyc_pickup['dist'] < 61]
```

```
▶ sns.scatterplot(x=df_clear['dist'], y=df_clear['fare_amount'])  
plt.xlim([0, 60])
```



- Data dengan jarak lebih dari 61 km dihapus karena dianggap outlier.
- Dari scatter plot terlihat bahwa sebagian besar tarif berada di bawah 100 dengan jarak tempuh < 20 km.
- Terdapat pola linier lemah antara jarak dan tarif, namun juga ada anomali yang perlu dibersihkan.



# Data Preparation & EDA Akhir

## 11. Pembersihan Outliers (1/2)

Setelah tahapan *data preparation* & *advanced EDA* dilakukan, nilai atau pola yang tidak biasa, menyimpang, atau berbeda dari mayoritas data lainnya yang telah teridentifikasi kami klasifikasikan menjadi dua (2) kategori, yaitu **anomali** dan **outliers**.

### Anomali:

- Tidak wajar, perlu ditelusuri
- Kemungkinan besar merupakan data invalid

### Yang merupakan anomali:

Data dengan:

- Jarak 0 km (17 rows)
  - Tarif < 1 USD & Jarak Tempuh > 1 km (19 rows)
  - Tarif > 100 USD & Jarak Tempuh < 1 km | (5 rows)
  - Jumlah penumpang 208 (1 row)
- \*) dari 200K row data



**Tindak Lanjut:**  
Eliminasi Data

### Outliers:

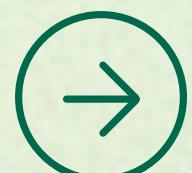
- Titik data yang berada jauh dari distribusi umum data
- Bisa jadi merupakan data valid



**Tindak Lanjut:**  
Tahapan Pre-processing

### Yang merupakan outliers:

Jumlah passengers, fare amount di luar distribusi normal



# Data Preparation & EDA Akhir

## 11. Pembersihan Outliers (2/2)

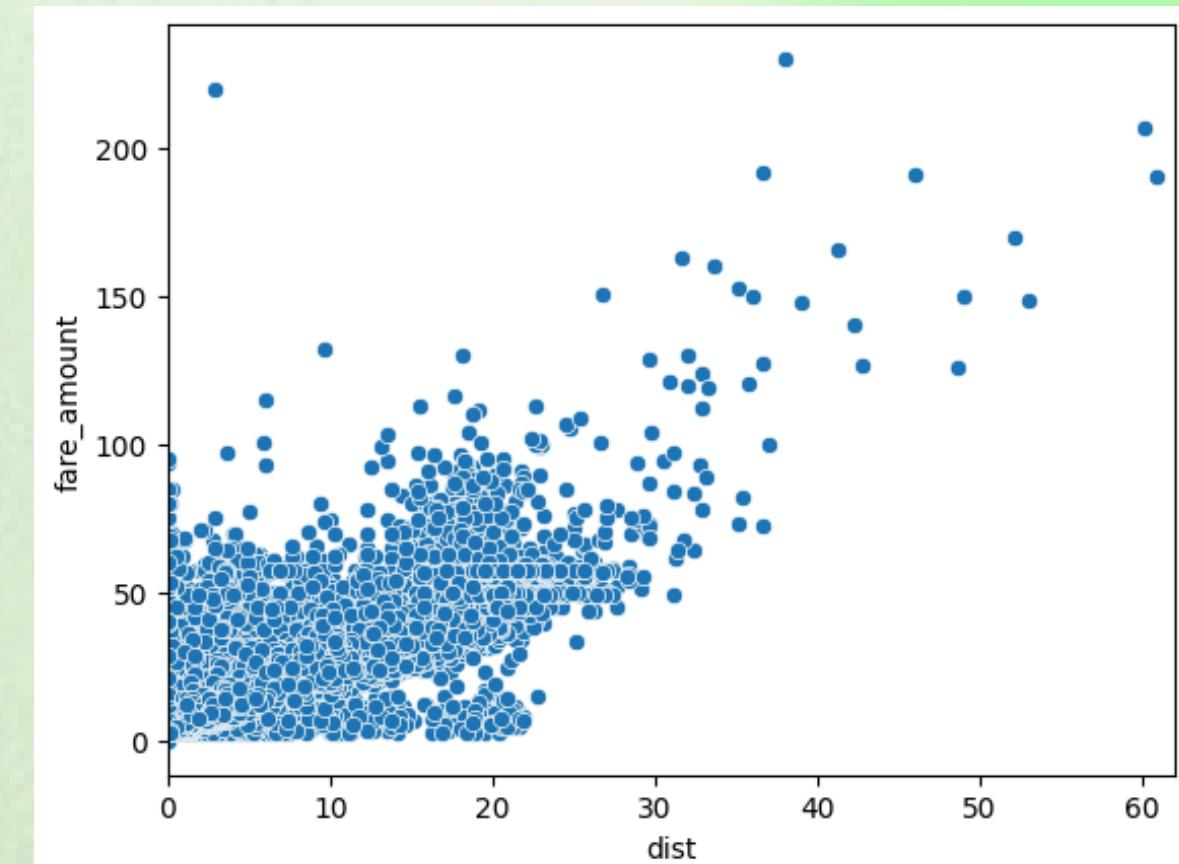
```
[ ] indices = df_clear[df_clear['dist'] == 0].index
df_clear.drop(indices, axis=0, inplace=True)
df_clear.reset_index(drop=True, inplace=True)
df_clear
```

```
[ ] indices = df_clear[(df_clear['fare_amount'] < 1) & (df_clear['dist'] > 1)].index
df_clear.drop(indices, axis=0, inplace=True)
df_clear.reset_index(drop=True, inplace=True)
df_clear
```

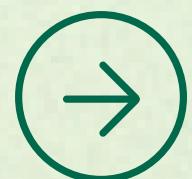
```
[ ] indices = df_clear[(df_clear['fare_amount'] > 100) & (df_clear['dist'] < 1)].index
df_clear.drop(indices, axis=0, inplace=True)
df_clear.reset_index(drop=True, inplace=True)
df_clear
```

- Dihapus data dengan jarak = 0, tarif < 1 tapi jarak > 1, dan tarif > 100 untuk jarak < 1 km.
- Pembersihan ini penting agar model tidak bias oleh nilai ekstrem atau tidak logis.

```
[ ] sns.scatterplot(x=df_clear['dist'], y=df_clear['fare_amount'])
plt.xlim([0, 62])
```



Visualisasi setelah pembersihan lebih bersih dan menunjukkan tren yang lebih realistik antara jarak dan tarif.



# Data Preparation & EDA Akhir

## 12. Hitung Fare per km

```
[ ] # Hitung fare per km
df_clear['fare_per_km'] = df_clear['fare_amount'] / df_clear['dist']

# Buang nilai tidak valid (misalnya dist = 0)
df_clear = df_clear[df_clear['fare_per_km'].notna() & df_clear['fare_per_km'] != np.inf]

# Menggunakan IQR untuk deteksi outlier pada fare_per_km
Q1 = df_clear['fare_per_km'].quantile(0.25)
Q3 = df_clear['fare_per_km'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter data yang masuk akal
df_clear = df_clear[(df_clear['fare_per_km'] >= lower_bound) & (df_clear['fare_per_km'] <= upper_bound)]

# Hapus kolom bantu jika tidak diperlukan
df_clear.drop(columns='fare_per_km', inplace=True)

# (Opsional) Tampilkan hasil
print(f"Jumlah data setelah filter: {len(df_clear)}")
```

➡ Jumlah data setelah filter: 181895

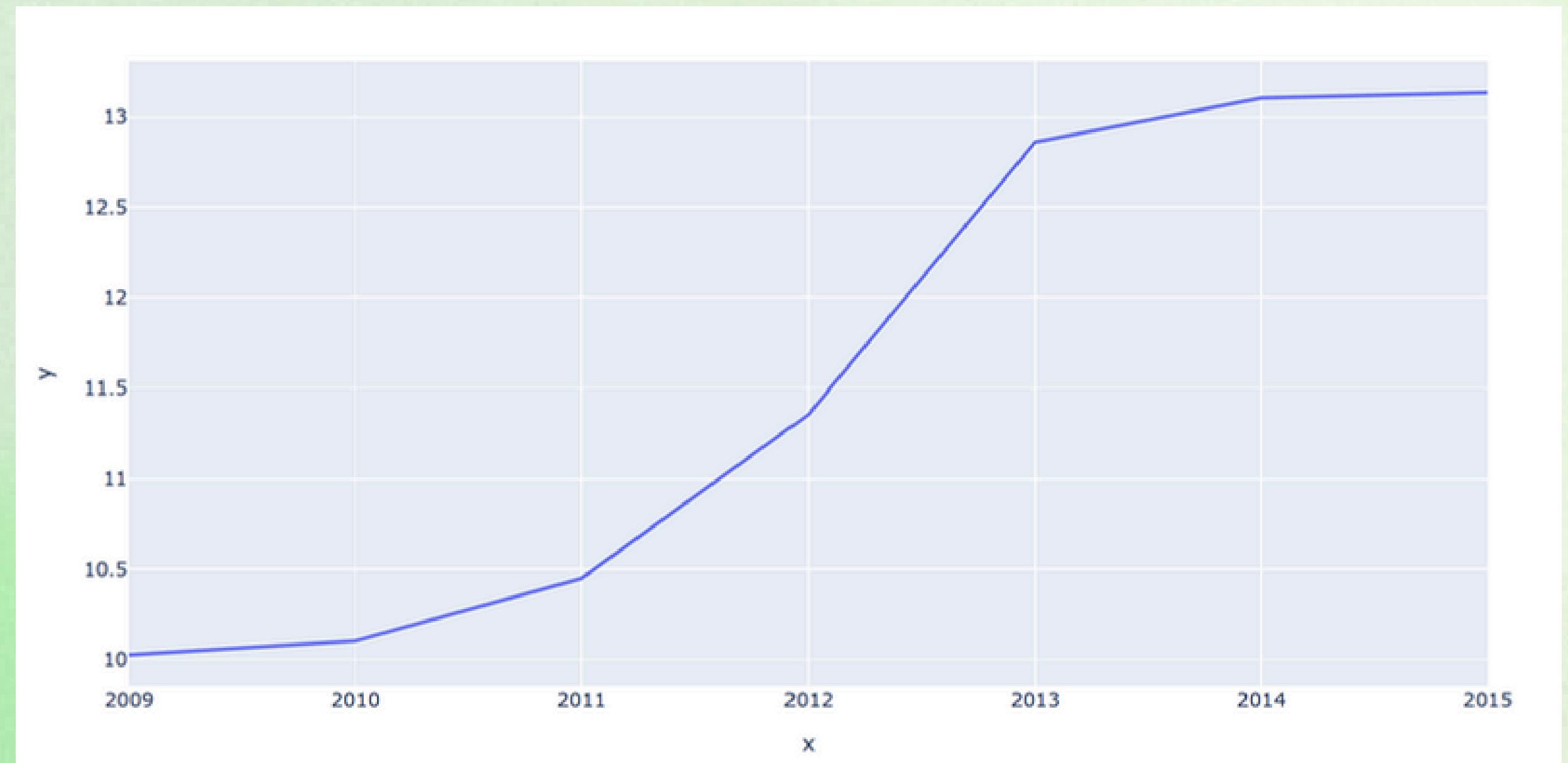
- Variabel baru fare\_per\_km dihitung untuk analisis efisiensi tarif per km.
- Menggunakan IQR untuk membuang outlier fare/km sehingga hasil lebih representatif.
- Setelah filter, diperoleh 181.895 data bersih → menunjukkan data cukup besar dan valid untuk modeling.

# EXPLORATORY DATA ANALYSIS



# Exploratory Data Analysis (EDA)

## Rata-rata Fare Amount per Tahun

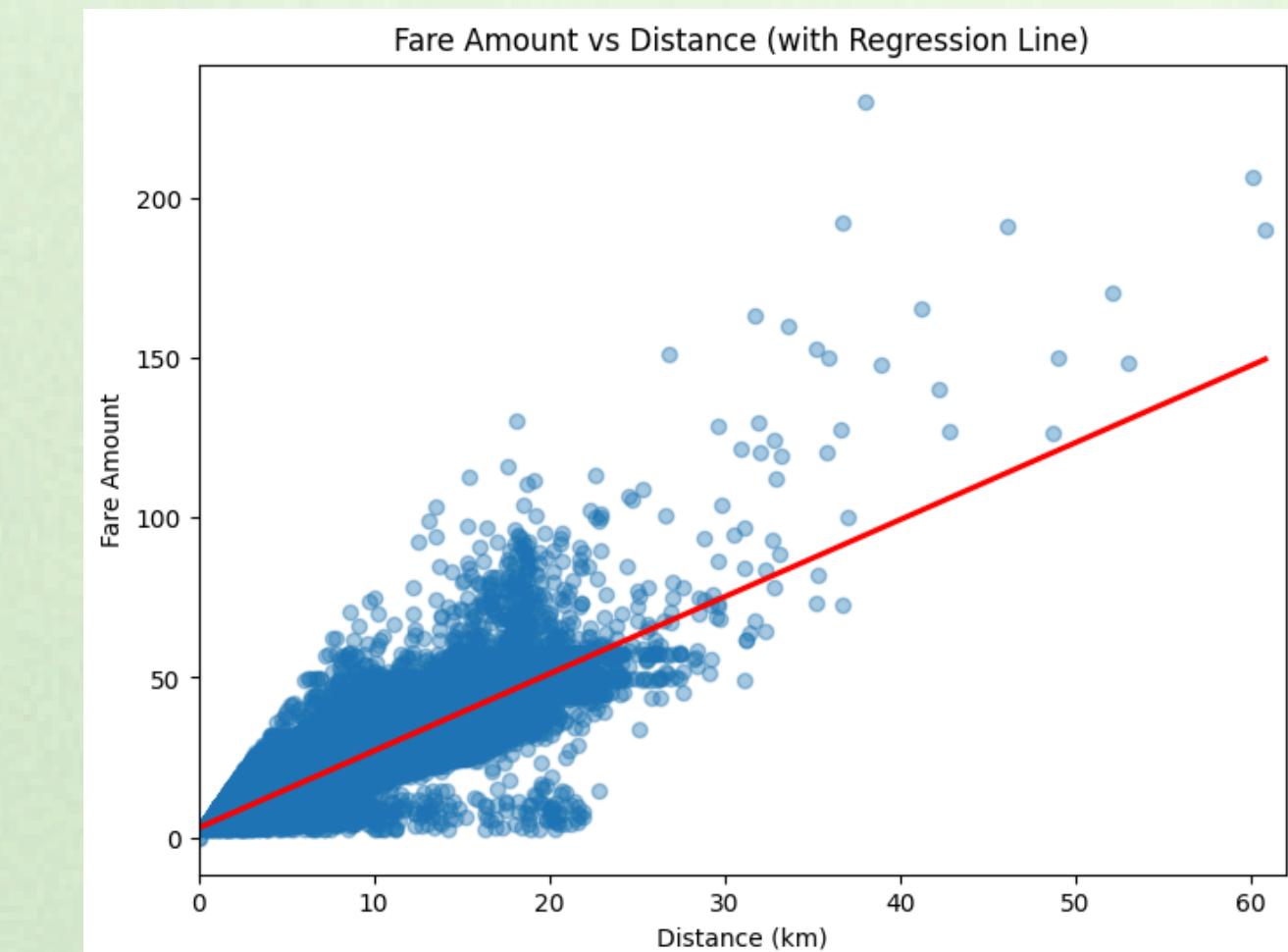
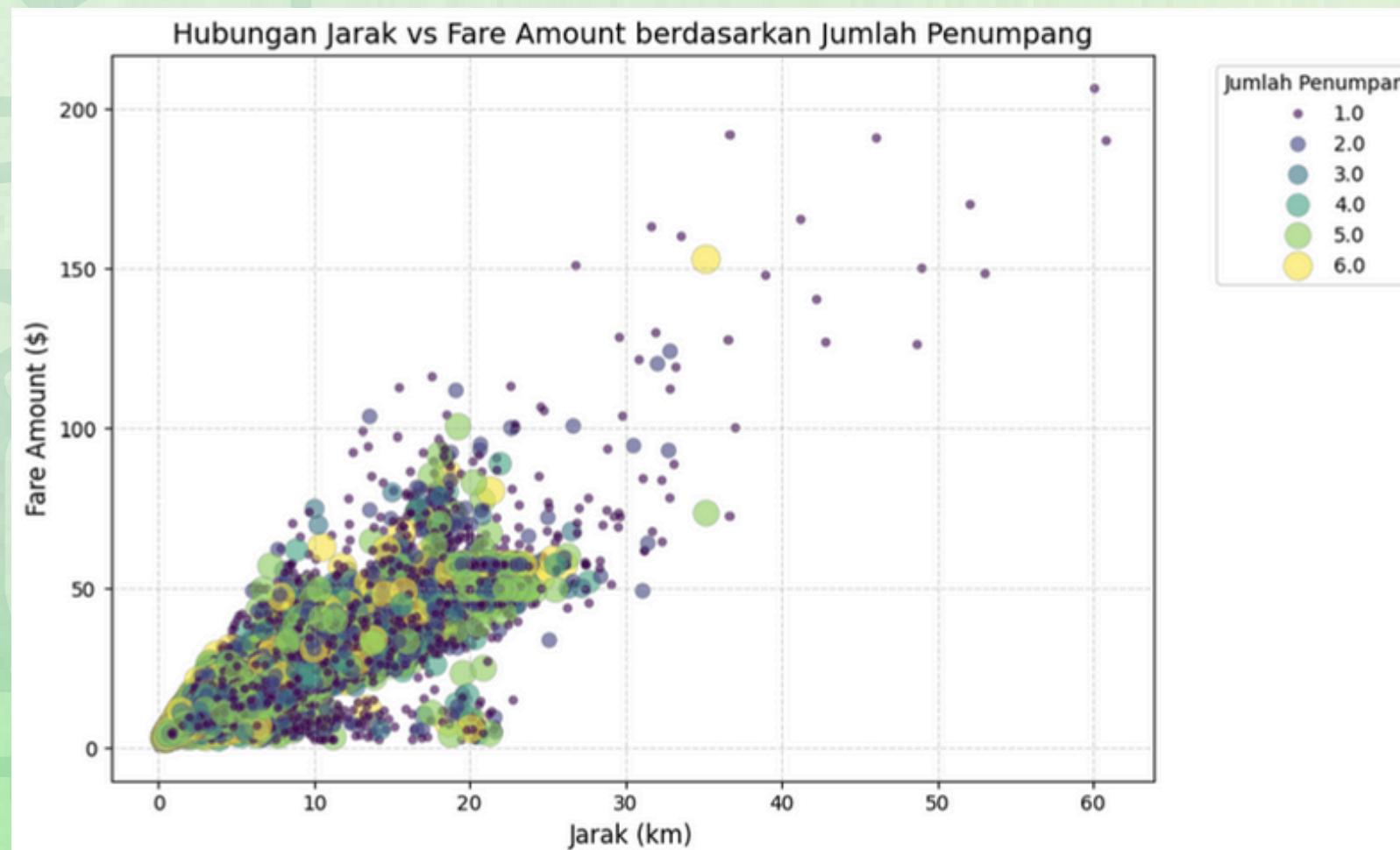


Terlihat adanya tren peningkatan rata-rata fare amount dari tahun ke tahun. Ini bisa mengindikasikan adanya kenaikan tarif atau perjalanan yang makin jauh.



# Exploratory Data Analysis (EDA)

## Hubungan Jarak vs Fare Amount berdasarkan Jumlah Penumpang

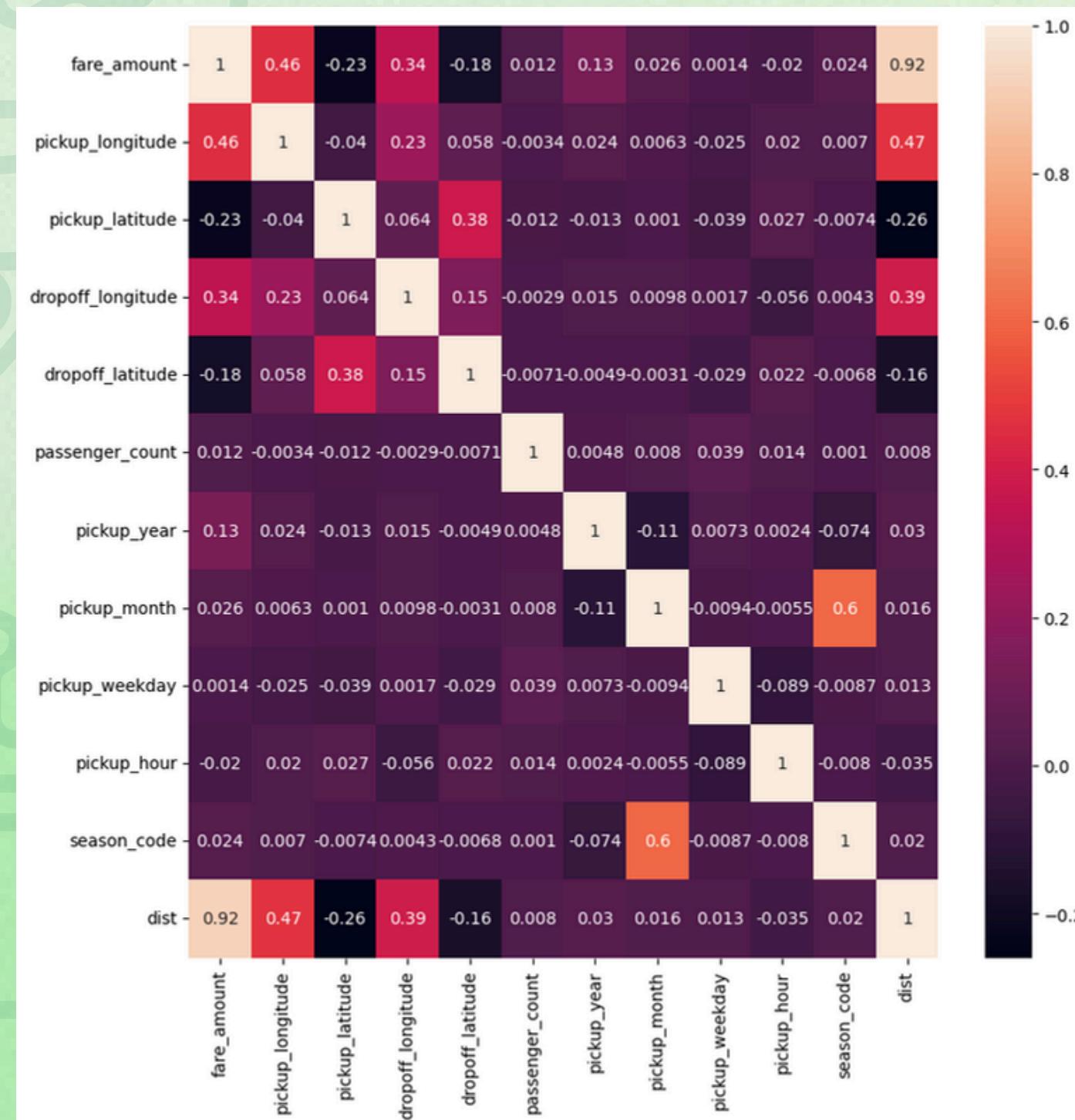


Terdapat hubungan linier positif yang jelas antara jarak perjalanan (dist) dan tarif (fare\_amount). Terlepas dari variasi jumlah penumpang, semakin jauh jarak tempuh, semakin tinggi tarifnya. Hal ini diperkuat oleh garis regresi yang menunjukkan bahwa jarak merupakan faktor paling berpengaruh dalam menentukan fare amount



# Exploratory Data Analysis (EDA)

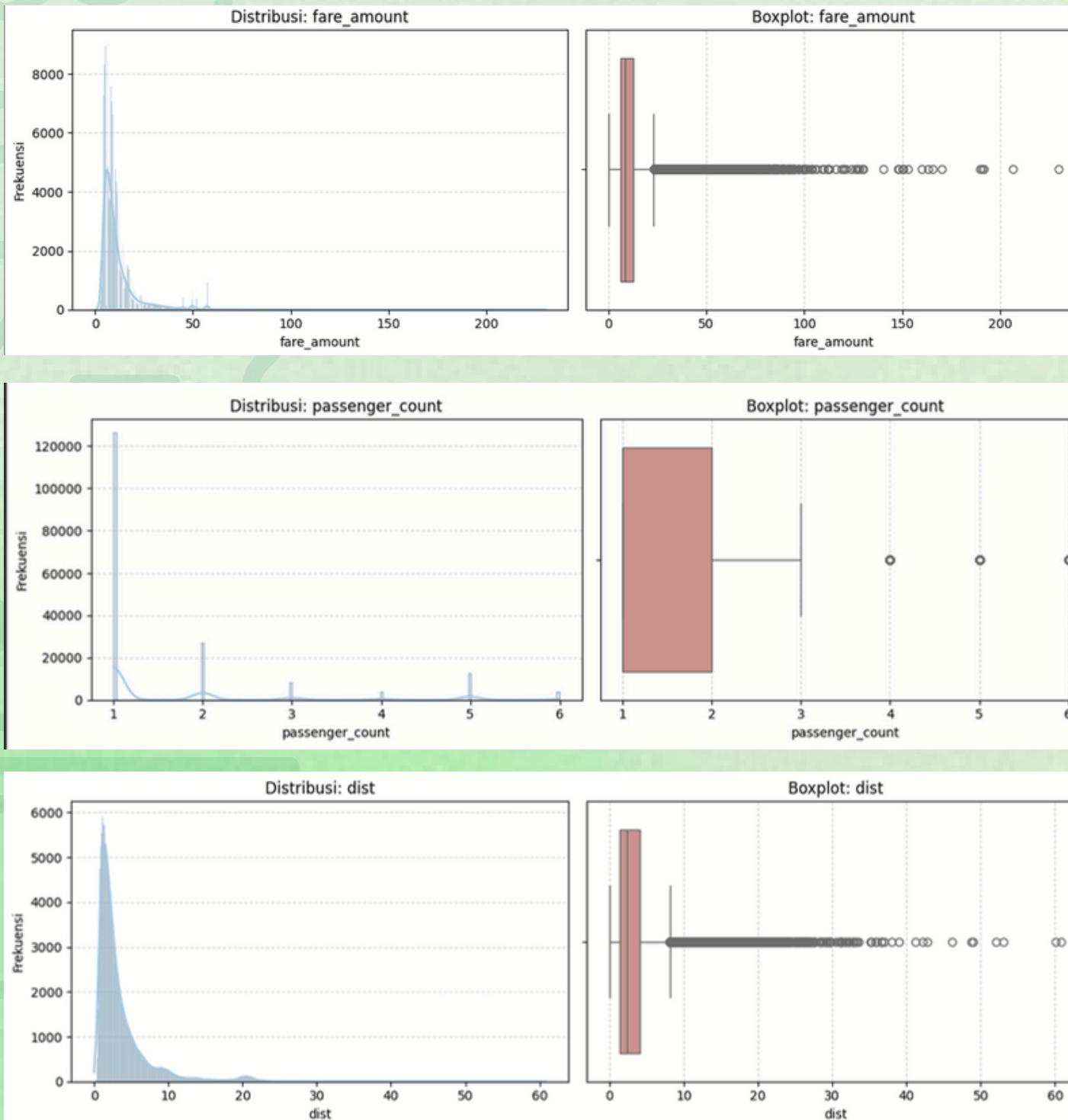
## Hubungan Jarak vs Fare Amount berdasarkan Jumlah Penumpang



- Jarak (dist) adalah fitur paling relevan untuk regresi fare amount.
- Variabel koordinat bisa jadi tambahan fitur pelengkap, tetapi tidak dominan.
- Variabel waktu dan penumpang memiliki kontribusi kecil dan bisa dikesampingkan dalam model awal.

# Exploratory Data Analysis (EDA)

## Distribusi & Boxplot



Setelah pembersihan logika (menghapus nilai yang benar-benar tidak mungkin), distribusi masih mengandung banyak outlier.

Outlier muncul di semua variabel:

- Fare\_amount: Banyak tarif tinggi yang jarang terjadi.
- Passenger\_count: Penumpang lebih dari 4 jarang.
- Distance (dist): Perjalanan panjang (20–60 mil) termasuk jarang.

Distribusi cenderung right-skewed di semua variabel numerik.



# DATA PREPROCESSING



# Data Preprocessing

## 1. Menghapus empat kolom koordinat geografis dari DataFrame

```
dfp.drop(columns = {"pickup_latitude","pickup_longitude","dropoff_latitude","dropoff_longitude"},inplace=True)
```

## 2. Menghitung dan Menangani Outlier per Kolom dengan Metode IQR dan filter

Untuk mendeteksi outlier pada setiap kolom numerik, kita dapat menggunakan metode IQR (Interquartile Range). Outlier didefinisikan sebagai data yang:

- Lebih kecil dari:

$$Lower Bound = Q1 - 1.5 \times IQR$$

- Atau lebih besar dari:

$$Upper Bound = Q3 + 1.5 \times IQR$$

Keterangan:

- $Q1$ : kuartil pertama (25%)
- $Q3$ : kuartil ketiga (75%)
- $IQR=Q3-Q1$ : rentang antar kuartil



# Data Preprocessing

## Perbandingan sebelum dan setelah penghapusan outlier

Kolom	Jumlah Outlier (Sebelum)	Percentase (Sebelum)	Jumlah Outlier (Sesudah)	Percentase (Sesudah)
fare_amount	14.771	8.12%	5.100	3.44%
passenger_count	20.461	11.25%	31.995	21.56%
dist	15.189	8.35%	6.594	4.44%

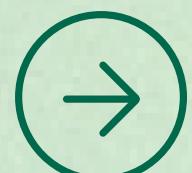
- **Passenger\_count** memiliki outlier tertinggi (11,25%)
- **Fare\_amount dan dist** mengandung >8% outlier

Outliers berpotensi menurunkan akurasi model sehingga dilakukan penanganan melalui eliminasi berdasarkan metode **IQR** dengan hasil sebagai berikut:

- Setelah eliminasi outliers fare amount dan dist, proporsi outlier berhasil ditekan secara signifikan, dimana distribusi outlier **fare amount turun** menjadi **3,44%** ( $\downarrow 4.68\%$ ) dan **dist** menjadi **4,44%** ( $\downarrow 3,91\%$ ).
- **Pembersihan ini meningkatkan kualitas data** untuk model prediksi yang lebih stabil dan akurat.

### Legend:

-  **Eliminated**
-  **Keep**



# Data Preprocessing

## 3. Split Features & Target

```
x = dfp.drop(columns='fare_amount')  
y = dfp['fare_amount']
```

Kode ini mengambil tabel data dfp dan memecahnya menjadi "data pertanyaan" (x) dan "data jawaban" (y).

## 4. Log Transform

```
#Log-transform target  
y_log = np.log1p(y)  
  
#Hanya kolom 'dist' yang ditransformasi log  
x_log = X.copy()  
x_log['dist'] = np.log1p(x_log['dist'])
```

Fungsi np.log1p(x) digunakan untuk menghitung  $\log(1+x)$ . Ini adalah cara yang lebih stabil secara numerik untuk menangani nilai nol jika ada di dalam data.

## 5. Split Train/Test

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train_log, y_test_log = train_test_split(  
    X, y_log, test_size=0.2, random_state=42)
```

membagi data menjadi data latih (train) dan data uji (test) dengan proporsi 80:20.

- **Target (y)** yang digunakan adalah versi log-transform (y\_log) agar model lebih stabil.
- Parameter **random\_state=42** memastikan hasil pembagian konsisten setiap kali dijalankan.

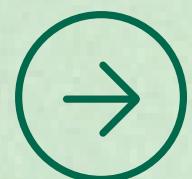


# Data Preprocessing

## 6. Pembagian Fitur Numerik

Fitur	Penanganan	Alasan / Catatan
<b>passenger_count</b>	StandardScaler (optional)	Perlu distandarisasi agar setara skalanya dengan fitur lain
<b>dist</b>	StandardScaler	Fitur utama penentu tarif, perlu diskalakan agar tidak mendominasi model
<b>pickup_year</b>	StandardScaler (opsional)	Biasanya hanya 1 nilai (misal 2009), tapi tetap disertakan agar tetap konsisten

- Fitur **passenger\_count** dan **dist** memiliki rentang nilai yang berbeda, sehingga standarisasi dilakukan untuk memastikan model tidak bias terhadap fitur berskala besar.
- **dist** sangat memengaruhi tarif, namun tanpa scaling, model cenderung terlalu mengandalkannya dan mengabaikan pola lain.
- Walau **pickup\_year** tidak variatif, menyertakannya tetap penting untuk menjaga format input data yang konsisten saat deployment.



# Data Preprocessing

## 7. Pembagian Fitur Siklikal

Fitur	Domain Nilai	Penanganan	Tujuan Encoding Siklikal
<b>pickup_month</b>	1 – 12	Siklikal (sin, cos)	Menangkap pola musiman (Januari–Desember sebagai siklus)
<b>pickup_weekday</b>	0 – 6	Siklikal (sin, cos)	Representasi siklik dari Senin–Minggu
<b>pickup_hour</b>	0 – 23	Siklikal (sin, cos)	Model memahami jam 23 ke jam 0 sebagai waktu berdekatan
<b>season_code</b>	0 – 3	Siklikal (sin, cos)	Representasi 4 musim (winter, spring, summer, fall) dalam siklus tahunan

Rumus untuk menghitung siklikal dengan sin dan cos:

$$x = \sin\left(\frac{a \times 2\pi}{\max(a)}\right) \quad y = \cos\left(\frac{a \times 2\pi}{\max(a)}\right)$$

Beberapa fitur seperti bulan, hari dalam seminggu, dan jam bersifat berulang secara periodik. Jika direpresentasikan sebagai angka biasa (misalnya 1–12 untuk bulan), model bisa keliru menganggap hubungan linier, padahal:

- **Januari** dan **Desember** seharusnya berdekatan, bukan jauh.
- **Jam 23** dan **jam 0** itu kontinu, bukan ujung-ujung dari skala numerik.



# Data Preprocessing

## 8. One-Hot Encoding

Fitur Asli	Kategori	Penanganan	Penjelasan
<code>pickup_hour_category</code>	'rush_hour', 'off_peak', 'night'	One-Hot Encoding ( <code>drop_first=True</code> )	Diubah menjadi dua kolom dummy: <code>hour_cat_off_peak</code> , <code>hour_cat_rush_hour</code> ; kategori 'night' dijadikan baseline agar menghindari multikolinearitas

### Kenapa `drop_first=True`?

Untuk mencegah dummy variable trap yang dapat menyebabkan multikolinearitas pada model linier atau ensemble seperti Gradient Boosting.

### Mengapa baseline-nya 'night'?

Karena dengan `drop_first`, pandas otomatis membuang kategori pertama (yaitu 'night'), menjadikannya referensi dalam interpretasi model. Jika baris tersebut bernilai 0 pada `hour_cat_off_peak` dan `hour_cat_rush_hour`, maka secara otomatis masuk ke kategori night.

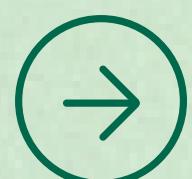


# Data Preprocessing

## 9. Scaling Fitur Numerik

Jenis Fitur	Nama Kolom	Penanganan	Penjelasan
Fitur Numerik	passenger_count, dist, pickup_year	StandardScaler	Diskalakan agar memiliki distribusi dengan rata-rata 0 dan standar deviasi 1
Fitur Siklikal	month_sin, month_cos, weekday_sin, weekday_cos, hour_sin, hour_cos, season_sin, season_cos	StandardScaler	Diskalakan agar nilai sinus/cosinus tidak bias terhadap model karena skala berbeda
Fitur Kategorikal (OH)	pickup_hour_category_off_peak, pickup_hour_category_rush_hour	Tidak di-scale	Sudah berupa dummy variabel hasil one-hot, tidak perlu di-scale

- **StandardScaler** hanya digunakan pada fitur yang bersifat numerik kontinu dan hasil encoding siklikal.
- Fitur dummy hasil **one-hot encoding tidak di-scale** karena sudah dalam bentuk 0/1.
- Proses scaling ini **hanya diterapkan untuk** model regresi linear seperti **Linear Regression, Ridge, dan Lasso**, dan tidak diperlukan pada model berbasis pohon keputusan (tree-based models).



# MODEL TRAINING





## Model Yang dimasukkan ke dalam machine learning di split Data

- X: Semua fitur kecuali fare\_amount
- y: Target yang diprediksi (fare\_amount)
- Pembagian data: 80% train, 20% test

## Processing Manual untuk standarisasi Data

- Fitur Numerik: dist, passenger\_count, pickup\_year → distandarisasi (StandardScaler)
- Fitur Ordinal: pickup\_season (urutan musim) → OrdinalEncoder
- Fitur Nominal: pickup\_month, weekday, hour → OneHotEncoder

Penggabungan "Semua fitur numerik, ordinal, dan nominal digabung menjadi X\_train\_final & X\_test\_final"

## Model yang digunakan untuk memprediksi tarif :

Model	Deskripsi
Linear Regression	Model dasar tanpa regularasi
Ridge Regression	Linear + regularisasi L2 (menekan overfitting)
Lasso Regression	Linear + regularisasi L1 (bisa seleksi fitur)
Random Forest	Ensemble berbasis pohon keputusan
Gradient Boosting	Pohon berurutan yang memperbaiki kesalahan
XGBoost	Versi optimasi dari boosting, efisien & akurat

**Enam model regresi ini digunakan untuk membandingkan akurasi prediksi tarif, mulai dari model linear sederhana hingga model ensemble yang kompleks.**

## Model-Model Regresi yang Digunakan & Alasan Pemilihannya

Model	Kenapa Digunakan?
Linear Regression	<ul style="list-style-type: none"> <li>Digunakan sebagai baseline (model paling dasar)</li> <li>Model dijelaskan, cepat dilatih</li> <li>Jika data punya hubungan linear, hasilnya bisa sangat baik</li> </ul>
Ridge Regression	<ul style="list-style-type: none"> <li>Menambahkan regularisasi L2 agar model tidak overfitting (terlalu cocok dengan data latih)</li> <li>Cocok jika ada banyak fitur yang berkorelasi</li> </ul>
Lasso Regression	<ul style="list-style-type: none"> <li>Selain mencegah overfitting, Lasso juga bisa menghapus fitur yang tidak relevan secara otomatis</li> <li>sangat berguna untuk feature selection</li> </ul>
Random Forest	<ul style="list-style-type: none"> <li>Model yang kuat untuk data non-linear</li> <li>Tidak sensitif terhadap outlier atau noise</li> <li>Bisa menangani fitur kategorik setelah preprocessing</li> </ul>
Gradient Boosting	<ul style="list-style-type: none"> <li>Membuat model yang terus belajar dari kesalahan sebelumnya</li> <li>Akurat untuk data yang kompleks</li> <li>Tapi lebih lambat dibanding Random Forest</li> </ul>
XGBoost	<ul style="list-style-type: none"> <li>Versi modern dan efisien dari boosting</li> <li>Lebih cepat bisa di-tuning lebih baik dan sangat akurat</li> <li>Cocok untuk proyek nyata atau kompetisi</li> </ul>

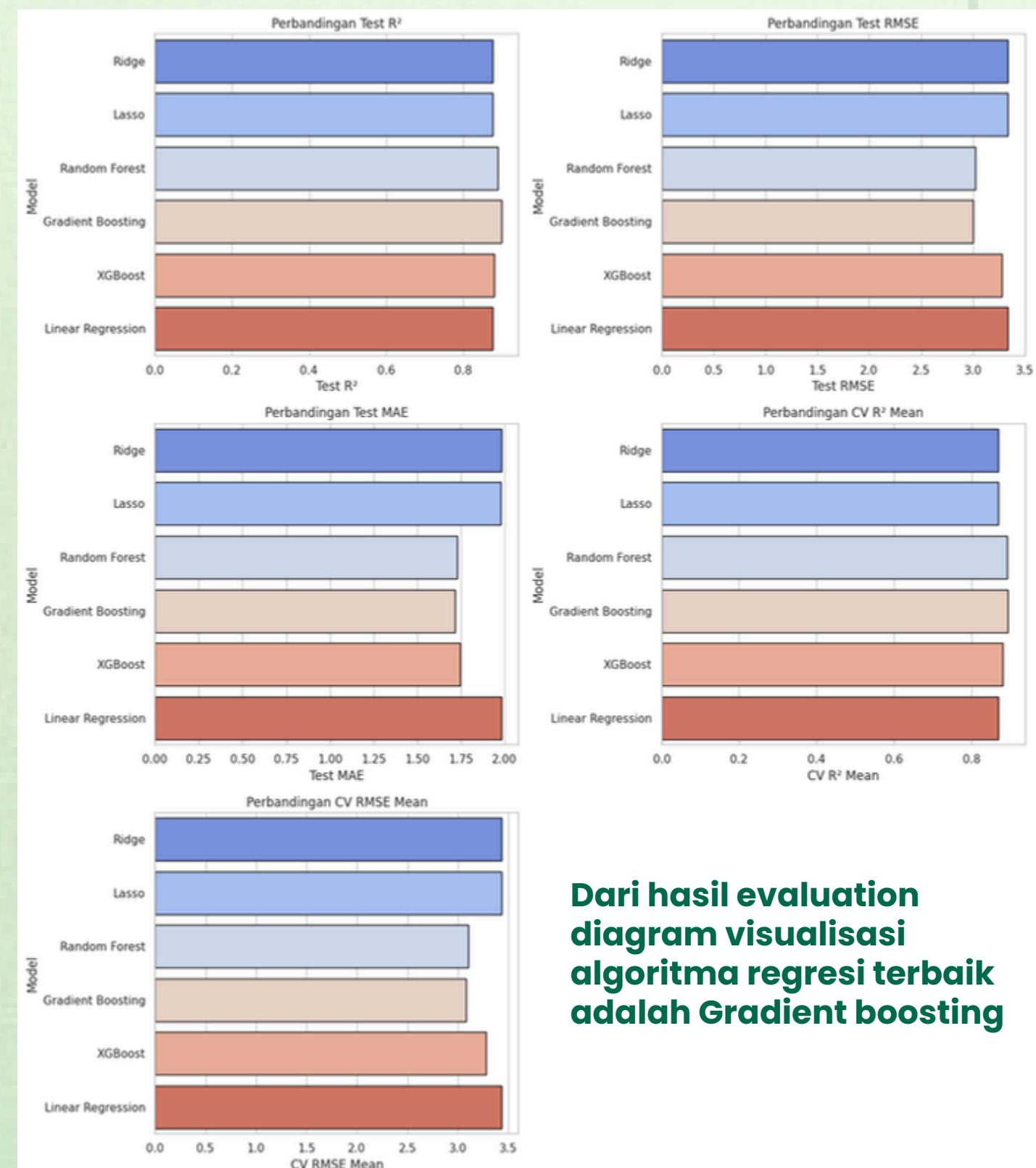


## Melatih semua model pada X\_train\_final dan y\_train

```
for name, model in models.items():
    model.fit(X_train_final, y_train)
```

### Metrik Evaluasi:

- **R2: Seberapa baik model menjelaskan variasi data target (semakin mendekati 1, semakin baik).**
- **RMSE (Root Mean Square Error): Rata-rata kesalahan prediksi (semakin kecil, semakin baik).**
- **MAE (Mean Absolute Error): Rata-rata selisih absolut prediksi vs aktual (semakin kecil, semakin baik).**
- **CV (Cross Validation): Evaluasi dengan validasi silang untuk menguji generalisasi model.**



Dari hasil evaluation diagram visualisasi algoritma regresi terbaik adalah Gradient boosting

# MODEL EVALUATION



# Pemilihan Algoritma Regresi Terbaik

## Tabel Perbandingan Awal Model Algoritma

Model	Train R <sup>2</sup>	Test R <sup>2</sup>	Train RMSE	Test RMSE	Train MAE	Test MAE	CV R <sup>2</sup> Mean	CV RMSE Mean
Ridge	0.8671	0.8768	3.4556	3.3396	1.9875	1.982	0.869	3.4329
Lasso	0.8671	0.8768	3.4562	3.3396	1.9845	1.9784	0.869	3.4334
Random Forest	0.9156	0.8901	2.7541	3.0216	1.6453	1.7275	0.8928	3.1052
Gradient Boosting	<b>0.9082</b>	<b>0.9002</b>	<b>2.8715</b>	<b>3.0054</b>	<b>1.6775</b>	<b>1.7173</b>	<b>0.8946</b>	<b>3.0791</b>
XGBoost	0.8965	0.881	3.0499	3.2813	1.7069	1.7464	0.8803	3.2819
Linear Regression	0.8671	0.8768	3.4556	3.3396	1.9875	1.982	0.869	3.4329

Dari hasil evaluation dari tabel di atas algoritma regresi terbaik adalah Gradient boosting

Metrik	Gradient Boosting	Catatan
Test R <sup>2</sup>	<b>0.9002</b>	▲ Paling tinggi → artinya model menjelaskan variansi data paling baik
Test RMSE	<b>3.0054</b>	▼ Paling rendah → kesalahan prediksi rata-rata terkecil
Test MAE	<b>1.7173</b>	▼ Paling rendah kedua setelah Random Forest
CV R <sup>2</sup> Mean	<b>0.8946</b>	▲ Tertinggi dari semua model saat cross-validation
CV RMSE Mean	<b>3.0791</b>	▼ Paling kecil dari semua model saat cross-validation

# Skenario Uji dan Hasil Evaluasi (Fokus pada Gradient Boosting)

**Tabel hasil evaluasi dari berbagai konfigurasi Gradient Boosting:**

Konfigurasi	Train R <sup>2</sup>	Test R <sup>2</sup>	Train RMSE	Test RMSE	Train MAE	Test MAE	CV R <sup>2</sup> Mean	CV RMSE Mean
Log Transform & Drop Outlier Fare + Dist	0.8681	0.8573	3.4141	3.5451	1.695	1.745	0.8363	0.2203
Log Transform & Drop Outlier Fare Only	0.8711	0.8406	3.3853	3.6979	1.7037	1.7479	0.8356	0.2208
Log Transform (Tanpa Outlier Removal)	0.8451	0.8486	3.865	3.79	1.7257	1.7534	0.8343	0.2226
<b>Drop Outlier Fare &amp; Dist</b>	<b>0.8816</b>	<b>0.8618</b>	<b>3.2347</b>	<b>3.4891</b>	<b>1.7383</b>	<b>1.8089</b>	<b>0.8599</b>	<b>3.5169</b>
Drop Outlier Fare Only	0.8838	0.851	3.2142	3.5755	1.739	1.797	0.8596	3.5209
Drop Outlier Dist Only	0.8781	0.8334	3.4422	3.846	1.7825	1.8231	0.8319	4.0037
Tanpa Drop Outlier	0.877	0.8468	3.4439	3.8123	1.7792	1.8404	0.829	4.0454
Drop Outlier Fare & Dist + filter passenger_count	0.8816	0.8618	3.2347	3.4891	1.7383	1.8089	0.8599	3.5169

**Model Terbaik: Gradient Boosting (Drop Outlier Fare & Dist, tanpa Log Transform)**

Model ini memberikan performa paling optimal secara keseluruhan:

- Test RMSE terendah (**3.4891**): Prediksi lebih akurat secara rata-rata kesalahan kuadrat.
- Test R<sup>2</sup> tertinggi (**0.8618**): Menjelaskan variasi fare\_amount paling baik di antara seluruh konfigurasi.
- Test MAE kompetitif (**1.8089**): Selisih absolut rata-rata yang tetap rendah.

**Catatan:** Walaupun model dengan log transform + drop outlier fare & dist sedikit lebih unggul dalam MAE (1.7450), namun konfigurasi tanpa log transform tetap lebih stabil dan kuat secara keseluruhan, karena memiliki R<sup>2</sup> dan RMSE terbaik.

# MODEL DEPLOYMENT



# Model Deployment

Setelah model berhasil dikembangkan, dilakukan **model deployment** yang bertujuan untuk **mengintegrasikan model yang sudah dilatih ke dalam sebuah platform yang dapat diakses**, sehingga model tersebut dapat digunakan untuk memprediksi data baru yang telah diinput secara **real-time**.

Model prediksi uber taxi fare prediction ini dapat diakses melalui link berikut:



## [Uber Taxi Fare](#)

Untuk dapat memprediksi uber taxi fare, user dapat menginput detail dari perjalanan mereka sbb:

1. **Jumlah Penumpang**
2. **Tanggal Penjemputan**
3. **Waktu Penjemputan**
4. **Latitude Penjemputan**
5. **Longitude Penjemputan**
6. **Latitude Tujuan**
7. **Longitude Tujuan**



Setelah itu, akan muncul hasil prediksi tarif sbb:



**Streamlit**

Prediksi Tarif Taksi Uber

Masukkan detail perjalanan untuk memprediksi tarif.

Jumlah Penumpang: 1

Tanggal Penjemputan: 2025/01/22

Waktu Penjemputan: 12:23

Latitude Penjemputan: 40.788934

Longitude Penjemputan: -74.006167

Latitude Tujuan: 40.723217

Longitude Tujuan: -73.999532

Prediksi Tarif

Hasil Prediksi Tarif

Prediksi Tarif Taksi Uber adalah: \$7.07



# CONCLUSION



# Conclusion

- **Tujuan Proyek:**
  - Membangun model machine learning untuk memprediksi fare\_amount (tarif perjalanan) secara akurat berdasarkan data perjalanan seperti jarak, waktu, dan jumlah penumpang.
- **Model Terbaik:**
  - Model Gradient Boosting Regressor terpilih sebagai model dengan performa terbaik untuk memprediksi tarif.
- **Proses & Penanganan Data:**
  - Data dibersihkan dan diproses terlebih dahulu (preprocessing) untuk menangani nilai yang hilang dan anomali (outliers).
  - Fitur-fitur kategorikal (seperti waktu) diubah menjadi format numerik menggunakan One-Hot Encoding dan Ordinal Encoding.
  - Fitur numerik (seperti jarak) diskalakan untuk mengoptimalkan performa model.

- Kesimpulan :

- Model Gradient Boosting mampu memprediksi tarif perjalanan dengan tingkat akurasi yang tinggi, menjadikannya alat yang andal untuk estimasi harga.
- Faktor-faktor seperti jarak tempuh (distance\_km), waktu (hour, day, month), dan lokasi penjemputan/pengantaran (yang direpresentasikan oleh pickup\_longitude, pickup\_latitude, dll.) adalah prediktor paling signifikan dalam menentukan tarif.

- Rekomendasi Bisnis:

- Implementasi Model: Integrasikan model Gradient Boosting ke dalam aplikasi Uber untuk memberikan estimasi tarif yang lebih akurat dan transparan kepada pelanggan sebelum perjalanan dimulai.
- Optimalisasi Harga Dinamis: Manfaatkan pemahaman tentang pengaruh waktu (jam sibuk, hari libur) dari model untuk menyempurnakan strategi penetapan harga dinamis (dynamic pricing).
- Peningkatan Efisiensi Operasional: Gunakan prediksi tarif untuk menganalisis rute dan waktu yang paling menguntungkan, membantu dalam alokasi sumber daya pengemudi yang lebih efisien.

# LINK ADDRESS

## Data Source

<https://www.kaggle.com/datasets/yasserh/uber-fares-dataset/data>

## Google Colab

<https://colab.research.google.com/drive/1QS4eZGUlgcU7thumN3MwKiyell-OTjWI?usp=sharing>

## Streamlit

<https://ubertaxifare.streamlit.app/>

# Pembagian Tugas & Persentase Kontribusi

## Exploratory Data Analysis (EDA)

Anel Fuad Abiyyu

17%

## Data Preprocessing

Zhafira Haura

17%

## Model Training

Peri Pirdaus

17%

## Model Evaluation

Brian Adam Bhagaskara

17%

## Model Deployment (termasuk UI)

Eugenia Jean Yovini

17%

## Conclusion

Muhammad Afif Y.G.

17 %

FINAL PROJECT

THANK YOU  
SO MUCH!

Presented by **The Outliers**