# 12111609

April 24, 2024

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     data_offers = pd.read_csv('datasets/data_offers.csv')
     data_orders = pd.read_csv('datasets/data_orders.csv')
     # pd.set_option('display.max_columns', None)
     # pd.set_option('display.max_rows', None)
```

```
[ ]: len(data_offers)
```

```
[ ]: 334363
```

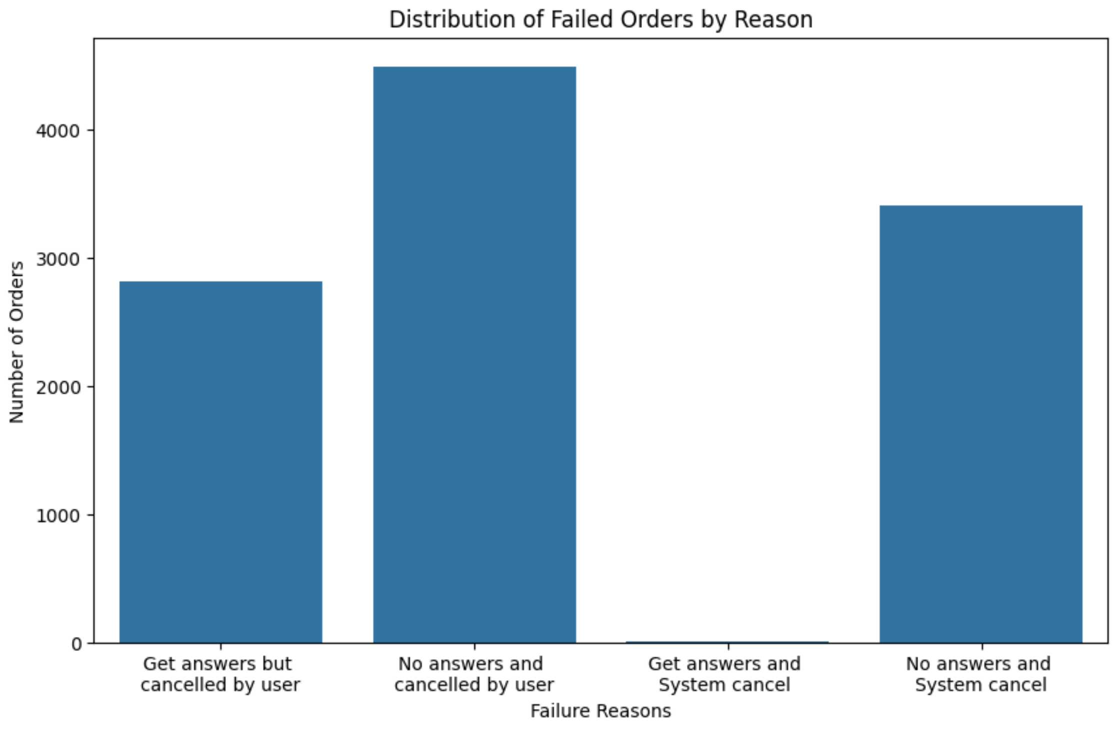
```
[ ]: data_offers['offer_count'] = data_offers.groupby('order_gk')['offer_id'].
       ↪transform('count')
     data_offers = data_offers.drop(columns=['offer_id'])
     data_offers = data_offers.drop_duplicates()
```

## 1 1

- There are several reasons:
  - `is_driver_assigned_key==1` means user canceled the order althought there are answers
  - `is_driver_assigned_key==0` && 'cancellations_time_in_seconds!=0' means that the customs cannot wait for an answer and cancel the order.
  - `is_driver_assigned_key==0` && `cancellations_time_in_seconds==0` means that the order is cancled by system

```
[ ]: merged_data = pd.merge(data_orders, data_offers, on='order_gk', how='left')
```

```
[ ]: failure_reasons = {
         'Get answers but \ncancelled by user':␣
       ↪merged_data[(merged_data['is_driver_assigned_key'] == 1) &␣
       ↪(merged_data['order_status_key'] == 4)],
         'No answers and \ncancelled by user':␣
       ↪merged_data[(merged_data['is_driver_assigned_key'] == 0) &␣
       ↪(merged_data['order_status_key'] == 4)],
    'Get answers and \nSystem cancel ':␣
↪merged_data[(merged_data['is_driver_assigned_key'] == 1) &␣
↪(merged_data['order_status_key'] == 9)],
    'No answers and \nSystem cancel':␣
↪merged_data[(merged_data['is_driver_assigned_key'] == 0) &␣
↪(merged_data['order_status_key'] == 9)],
}

failure_counts = {reason: df.shape[0] for reason, df in failure_reasons.items()}

plt.figure(figsize=(10, 6))
sns.barplot(x=list(failure_counts.keys()), y=list(failure_counts.values()))
plt.title('Distribution of Failed Orders by Reason')
plt.xlabel('Failure Reasons')
plt.ylabel('Number of Orders')
plt.show()
```



The histogram shows that:

- If there are no answers, the proportion of cancellation by users is larger.
- Group by answer or not, we can find that if there are no answers the order is more probable to be cancelled.
- Group by user or system, we can find that the order is more probable to be cancelled by users.

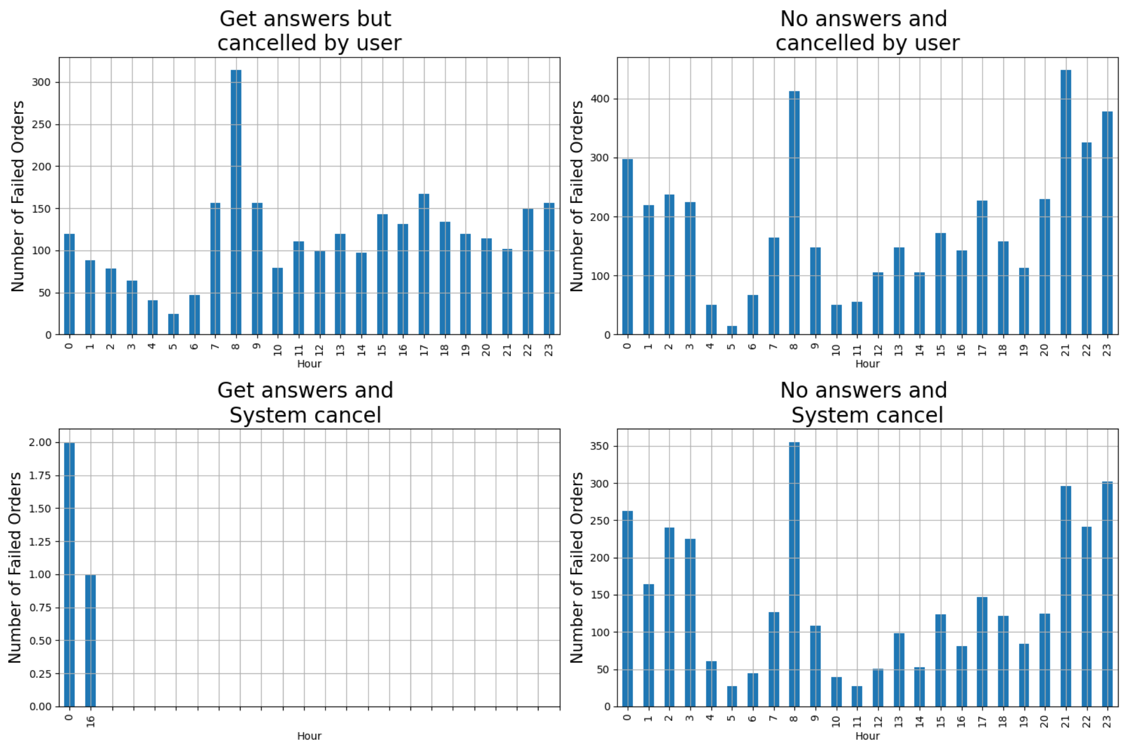
## 2 2

[ ]:
```python
merged_data['order_datetime'] = pd.to_datetime(merged_data['order_datetime'])
merged_data['hour'] = merged_data['order_datetime'].dt.hour
merged_data.head(10)
```

[ ]:
```python
import matplotlib.pyplot as plt

fig, axs = plt.subplots(2, 2, figsize=(15, 10))  #   2x2

for i, (reason, df) in enumerate(failure_reasons.items()):
    ax = axs[i // 2, i % 2]  #
    df['order_datetime'] = pd.to_datetime(df['order_datetime'])
    df['hour'] = df['order_datetime'].dt.hour
    failure_counts_by_hour = df.groupby('hour').size()
    failure_counts_by_hour.plot.bar(ax=ax)  #
    ax.set_title(f'{reason}', fontsize=20)
    ax.set_xlabel('Hour')
    ax.set_ylabel('Number of Failed Orders', fontsize=15)
    ax.set_xticks(range(24))
    ax.grid(True)

plt.tight_layout()  #
plt.show()
```

```
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:7
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['order_datetime'] = pd.to_datetime(df['order_datetime'])
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:8
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['hour'] = df['order_datetime'].dt.hour
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:7
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  df['order_datetime'] = pd.to_datetime(df['order_datetime'])
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:8
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['hour'] = df['order_datetime'].dt.hour
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:7
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['order_datetime'] = pd.to_datetime(df['order_datetime'])
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:8
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['hour'] = df['order_datetime'].dt.hour
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:7
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['order_datetime'] = pd.to_datetime(df['order_datetime'])
/var/folders/01/36vqvpjs78j_nk7k49q3t3w00000gn/T/ipykernel_59174/3199132246.py:8
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['hour'] = df['order_datetime'].dt.hour
```



Explanation for Distribution of Failed Orders by hours:

- `Get answers and system cancel` has a discrete distribution which may because of regular system error.
- The distribution of other three have apparently correlation with time interval. They all show a high distribution around 8 a.m, which must due to morning peak. Lots of people order the car for work, which leads to crowded flow. Also, few failures occurred just before morning peak.
- `Get answers but cancelled by user` distributed reletively uniformly other times of the day, even during night.
- `No answers and cancelled by user` distributed more and more until the midnight. There is a valley around 19 p.m, which may because that people tend to wait after work.
- `No answers and system cancel` distributed more after 21 p.m, except for morning peak.

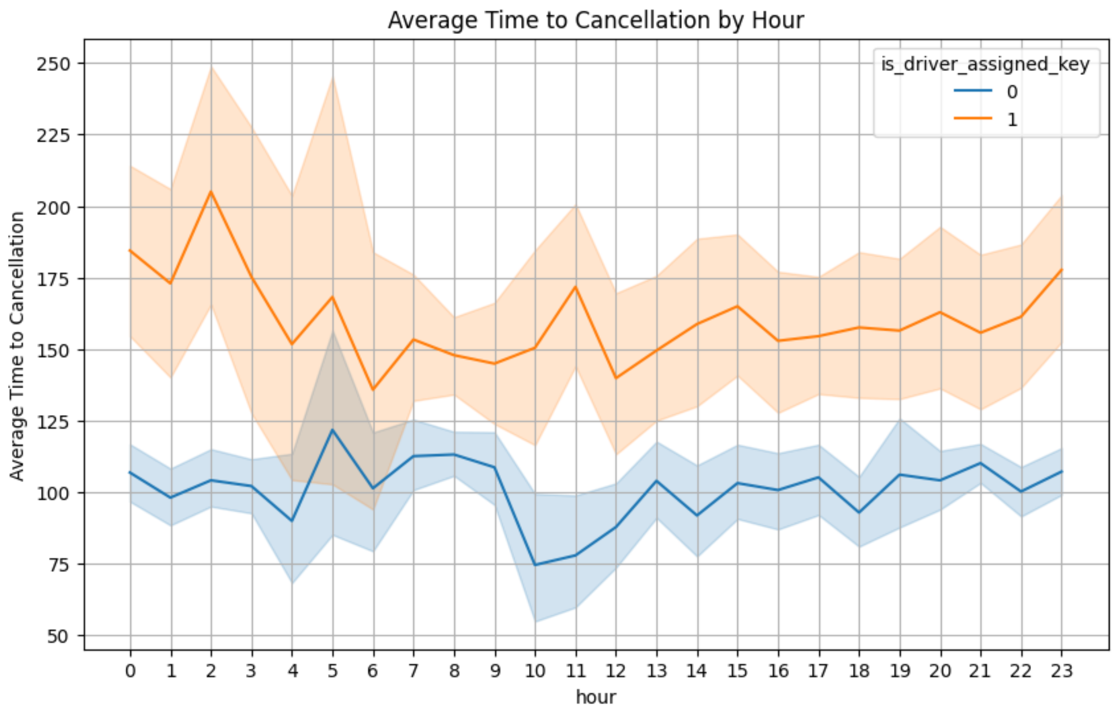
## 3 3

The chart shows that:

- In each time interval, the average time to cancellation for assigned driver is more than that if no driver assign. Which means if there are drivers assigning, there may take it longer for cancellation.

```
[ ]:
```

```
merged_data['cancellations_time_in_seconds_a'] =␣
 ↪(merged_data['cancellations_time_in_seconds'] -␣
 ↪merged_data['cancellations_time_in_seconds'].mean()) /␣
 ↪merged_data['cancellations_time_in_seconds'].std()
merged_data = merged_data[merged_data['cancellations_time_in_seconds_a'].
 ↪between(-2, 2)]

plt.figure(figsize=(10, 6))
sns.lineplot(x='hour', y='cancellations_time_in_seconds',␣
 ↪hue='is_driver_assigned_key', data=merged_data)
plt.title('Average Time to Cancellation by Hour')
# I want to show ever hour on the x-axis
plt.xticks(range(24))
# plt.xlabel('Hour')
plt.ylabel('Average Time to Cancellation')
plt.grid(True)
plt.show()
```



[ ]:
```
cancelled_orders = merged_data[merged_data['order_status_key'] == 4]
cancelled_orders_with_driver =␣
 ↪cancelled_orders[cancelled_orders['is_driver_assigned_key'] == 1]
cancelled_orders_without_driver =␣
 ↪cancelled_orders[cancelled_orders['is_driver_assigned_key'] == 0]
```

```
[ ]: avg_cancel_time_with_driver =␣
      ↪cancelled_orders_with_driver['cancellations_time_in_seconds'].mean()
     avg_cancel_time_without_driver =␣
      ↪cancelled_orders_without_driver['cancellations_time_in_seconds'].mean()
```

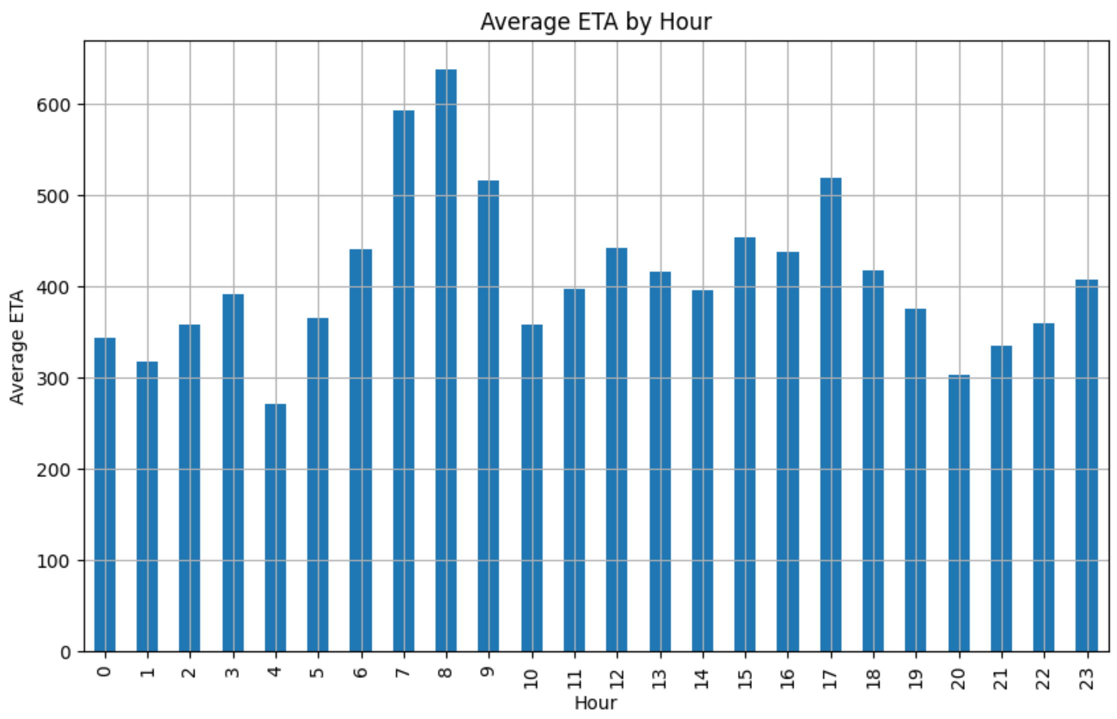
```
[ ]: print("Average cancellation time with driver:", avg_cancel_time_with_driver)
     print("Average cancellation time without driver:",␣
      ↪avg_cancel_time_without_driver)
```

```
Average cancellation time with driver: 159.16442687747036
Average cancellation time without driver: 103.89638932496075
```

## 4 4

- The result shows that there are peaks at around 8 a.m and 17 p.m which corresponse to the morning peak and evening peak.

```
[ ]: avg_eta_by_hour = merged_data.groupby('hour')['m_order_eta'].mean()

     plt.figure(figsize=(10, 6))
     avg_eta_by_hour.plot.bar()
     plt.title('Average ETA by Hour')
     plt.xlabel('Hour')
     plt.ylabel('Average ETA')
     plt.xticks(range(24))
     plt.grid(True)
     plt.show()
```



## 5 5

```
[ ]: len(merged_data)
```

```
[ ]: 6989
```

```
[ ]: import folium
     import h3

     # Calculate the hexagons
     hexagons = {}
     for index, row in merged_data.iterrows():
         lat, lon = row['origin_latitude'], row['origin_longitude']
         h3_index = h3.geo_to_h3(lat, lon, 8)
         if h3_index in hexagons:
             hexagons[h3_index] += 1
         else:
             hexagons[h3_index] = 1
     # Sort the hexagons by the number of orders
     sorted_hexagons = sorted(hexagons.items(), key=lambda x: x[1], reverse=True)
     print(sorted_hexagons)
```

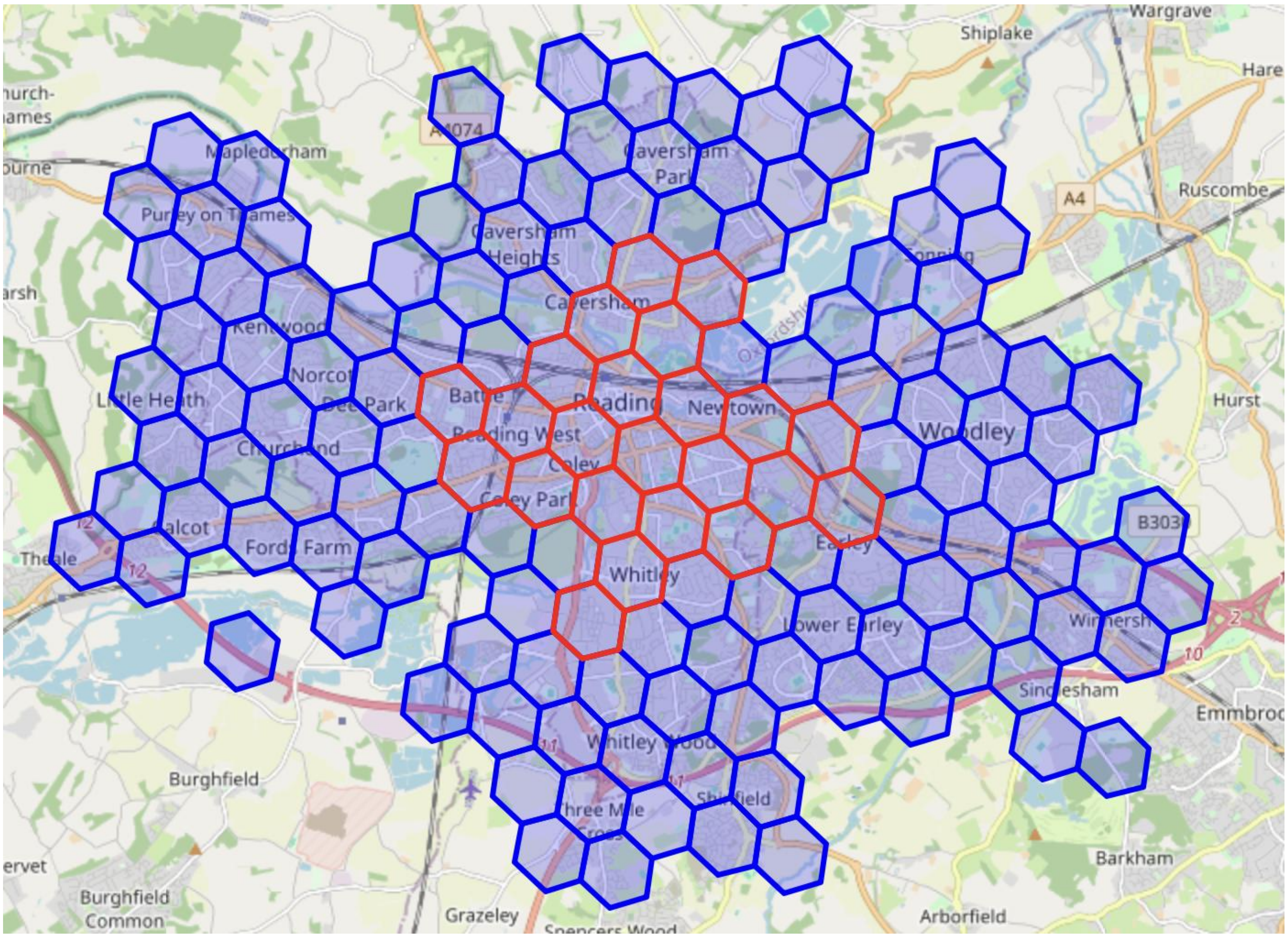
```
[('88195d2b1dfffff', 1115), ('88195d2b1bfffff', 587), ('88195d2b15fffff', 553),
('88195d2b19fffff', 463), ('88195d2b11fffff', 463), ('88195d284dfffff', 417),
('88195d2a27fffff', 247), ('88195d2b0bfffff', 246), ('88195d2b13fffff', 232),
('88195d2a25fffff', 231), ('88195d2b03fffff', 181), ('88195d2b17fffff', 125),
('88195d2861fffff', 123), ('88195d2b39fffff', 119), ('88195d2b3dfffff', 108),
('88195d2a21fffff', 92), ('88195d2b31fffff', 81), ('88195d2869fffff', 73),
('88195d2b55fffff', 65), ('88195d2b3bfffff', 61), ('88195d2845fffff', 58),
('88195d2b57fffff', 58), ('88195d2a23fffff', 56), ('88195d2b51fffff', 55),
('88195d2841fffff', 50), ('88195d2847fffff', 49), ('88195d2863fffff', 39),
('88195d2843fffff', 38), ('88195d2b09fffff', 38), ('88195d2b0dfffff', 37),
('88195d2a35fffff', 35), ('88195d2867fffff', 34), ('88195d282dfffff', 32),
('88195d2b07fffff', 32), ('88195d2b01fffff', 31), ('88195d2a2bfffff', 30),
('88195d280dfffff', 28), ('88195d2849fffff', 26), ('88195d2b37fffff', 23),
('88195d2829fffff', 23), ('88195d294bfffff', 21), ('88195d2b23fffff', 21),
('88195d2b47fffff', 21), ('88195d286bfffff', 19), ('88195d2a3dfffff', 19),
('88195d2a37fffff', 17), ('88195d2801fffff', 16), ('88195d39b7fffff', 16),
('88195d2a39fffff', 16), ('88195d2b33fffff', 15), ('88195d2b35fffff', 15),
('88195d2b43fffff', 15), ('88195d2a2dfffff', 15), ('88195d2a69fffff', 14),
('88195d2a17fffff', 13), ('88195d295bfffff', 13), ('88195d283dfffff', 13),
('88195d2b29fffff', 12), ('88195d285dfffff', 12), ('88195d2943fffff', 12),
('88195d2a15fffff', 12), ('88195d2941fffff', 12), ('88195d2b4bfffff', 11),
('88195d286dfffff', 10), ('88195d2949fffff', 10), ('88195d2b2bfffff', 10),
('88195d2b21fffff', 9), ('88195d2b5dfffff', 9), ('88195d296bfffff', 9),
('88195d74d7fffff', 8), ('88195d294dfffff', 8), ('88195d2b6bfffff', 8),
('88195d2a03fffff', 7), ('88195d2803fffff', 7), ('88195d2809fffff', 7),
('88195d2b27fffff', 7), ('88195d2a33fffff', 7), ('88195d2961fffff', 6),
('88195d2a67fffff', 6), ('88195d2947fffff', 6), ('88195d2aedfffff', 6),
('88195d7493fffff', 6), ('88195d2a13fffff', 6), ('88195d2a07fffff', 5),
('88195d2b59fffff', 5), ('88195d2a29fffff', 5), ('88195d2955fffff', 5),
('88195d284bfffff', 5), ('88195d2805fffff', 5), ('88195d2a05fffff', 4),
('88195d2967fffff', 4), ('88195d2a61fffff', 4), ('88195d2a11fffff', 4),
('88195d2b25fffff', 4), ('88195d7499fffff', 4), ('88195d2963fffff', 4),
('88195d2a0dfffff', 4), ('88195d7497fffff', 3), ('88195d280bfffff', 3),
('88195d2a01fffff', 3), ('88195d2a31fffff', 3), ('88195d2945fffff', 3),
('88195d2865fffff', 3), ('88195d749bfffff', 3), ('88195d74d1fffff', 3),
('88195d2b41fffff', 3), ('88195d2b53fffff', 3), ('88195d282bfffff', 3),
('88195d2969fffff', 2), ('88195d2a55fffff', 2), ('88195d2a0bfffff', 2),
('88195d2a63fffff', 2), ('88195d2859fffff', 2), ('88195d2823fffff', 2),
('88195d2b05fffff', 2), ('88195d2953fffff', 2), ('88195d748bfffff', 2),
('88195d39b1fffff', 2), ('88195d74d5fffff', 1), ('88195d2b63fffff', 1),
('88195d2a1bfffff', 1), ('88195d295dfffff', 1), ('88195d281dfffff', 1),
('88195d749dfffff', 1), ('88195d2a6bfffff', 1), ('88195d2a45fffff', 1),
('88195d2ae1fffff', 1), ('88195d74d3fffff', 1), ('88195d2a09fffff', 1),
('88195d2b49fffff', 1), ('88195d3993fffff', 1), ('88195d2a19fffff', 1),
('88195d39b9fffff', 1), ('88195d39b3fffff', 1), ('88195d2807fffff', 1),
('88195d2b4dfffff', 1)]
```

```
[ ]: print(merged_data.shape)
```

```
(6989, 11)
```

- The red regions are added to 80-percent of all orders, while the rest is less than 20% as the sum.

```
[ ]: # Calculate the number of hexagons that contain 80% of all orders
total_orders = merged_data.shape[0]
cumulative_orders = 0
hexagons_80_percent = []
other_hexagons = []

for h3_index, count in sorted_hexagons:
    cumulative_orders += count
    if cumulative_orders >= 0.8 * total_orders:
        other_hexagons.append(h3_index)
        continue
    hexagons_80_percent.append(h3_index)


# Create a map
m = folium.Map(location=[merged_data['origin_latitude'].mean(),␣
 ↪merged_data['origin_longitude'].mean()], zoom_start=10)

for h3_index in other_hexagons:
    boundary = h3.h3_to_geo_boundary(h3_index)
    folium.Polygon(locations=boundary, color='blue', fill=True,␣
 ↪fill_color='blue').add_to(m)


# Add the hexagons to the map
for h3_index in hexagons_80_percent:
    boundary = h3.h3_to_geo_boundary(h3_index)
    folium.Polygon(locations=boundary, color='red', fill=True,␣
 ↪fill_color='blue').add_to(m)


# Display the map
m.save('hexagons.html')
```


Wargrave
Shiplake
Hare
A4074
Mapledurham
Caversham Park
Ruscombe
A4
Purley on Thames
Caversham Heights
Sonning
Caversham
Kentwood
Norcot
Little Heath
Dee Park
Battle
Reading
Newtown
Hurst
Woodley
Reading West
Churchend
Coley
Coley Park
Calcot
B3030
Theale
Fords Farm
Earley
Whitley
Lower Earley
Winnersh
Sindlesham
Emmbrook
Whitley Wood
Burghfield
Shinfield
Three Mile Cross
Barkham
Burghfield Common
Grazeley
Spencers Wood
Arborfield