xiao

April 24, 2024

#

1 2
3 4 3 bonus h3 folium 8 80%

data_orders data_offers CSV data_orders

order_datetime - origin_longitude - origin_latitude - m_order_eta - order_gk - order_status_key - 4 - 9 - is_driver_assigned_key - cancellation_time_in_seconds - data_offers 2

order_gk - orders offer_id - ID

[48]:
```
import pandas as pd
import matplotlib.pyplot as plt

#
orders = pd.read_csv('datasets/data_orders.csv')
offers = pd.read_csv('datasets/data_offers.csv')

# change order_datetime from string to datetime format
orders['order_datetime'] = pd.to_datetime(orders['order_datetime'])
```

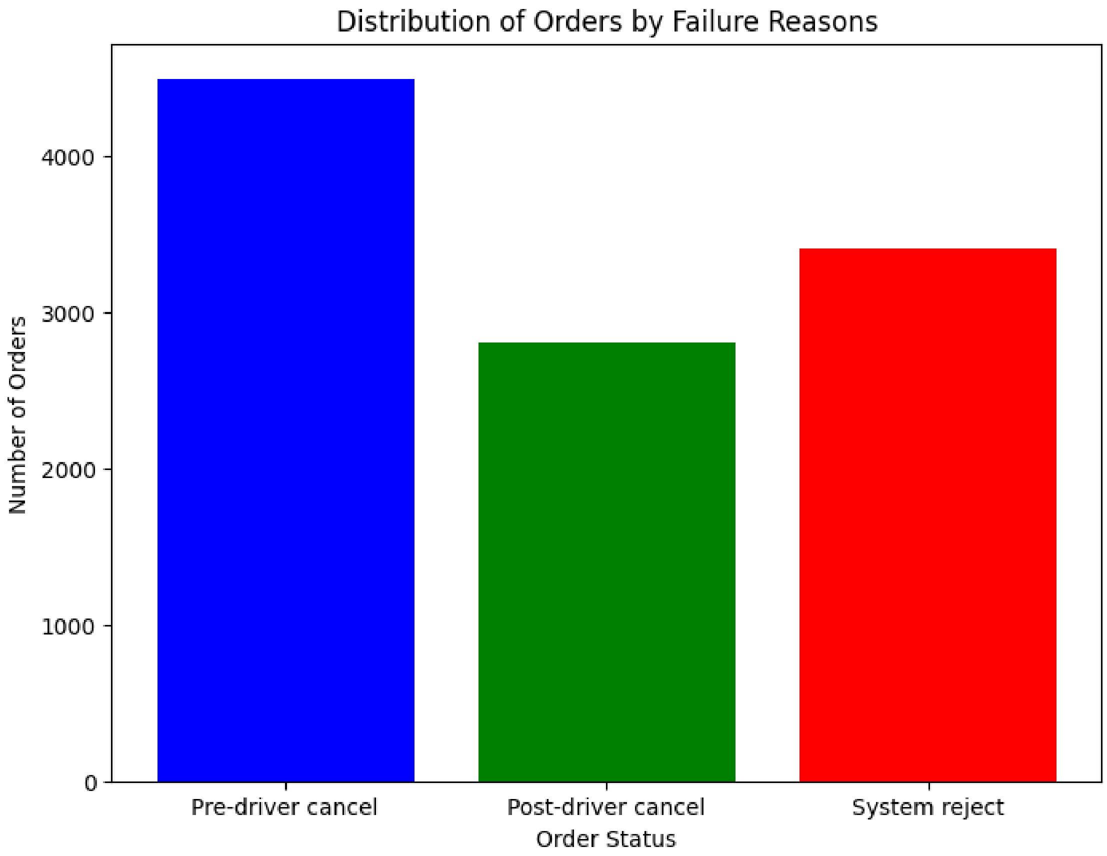
[49]:
```
# 1.
#
# -
# -
# -       9

pre_driver_cancel = orders[(orders['is_driver_assigned_key'] == 0) &␣
 ↪(orders['order_status_key'] == 4)]
post_driver_cancel = orders[(orders['is_driver_assigned_key'] == 1) &␣
 ↪(orders['order_status_key'] == 4)]
system_reject = orders[orders['order_status_key'] == 9]

#
counts = [len(pre_driver_cancel), len(post_driver_cancel), len(system_reject)]
categories = ['Pre-driver cancel', 'Post-driver cancel', 'System reject']

#
```

```
plt.figure(figsize=(8, 6))
plt.bar(categories, counts, color=['blue', 'green', 'red'])
plt.xlabel('Order Status')
plt.ylabel('Number of Orders')
plt.title('Distribution of Orders by Failure Reasons')
plt.show()
```



[50]:

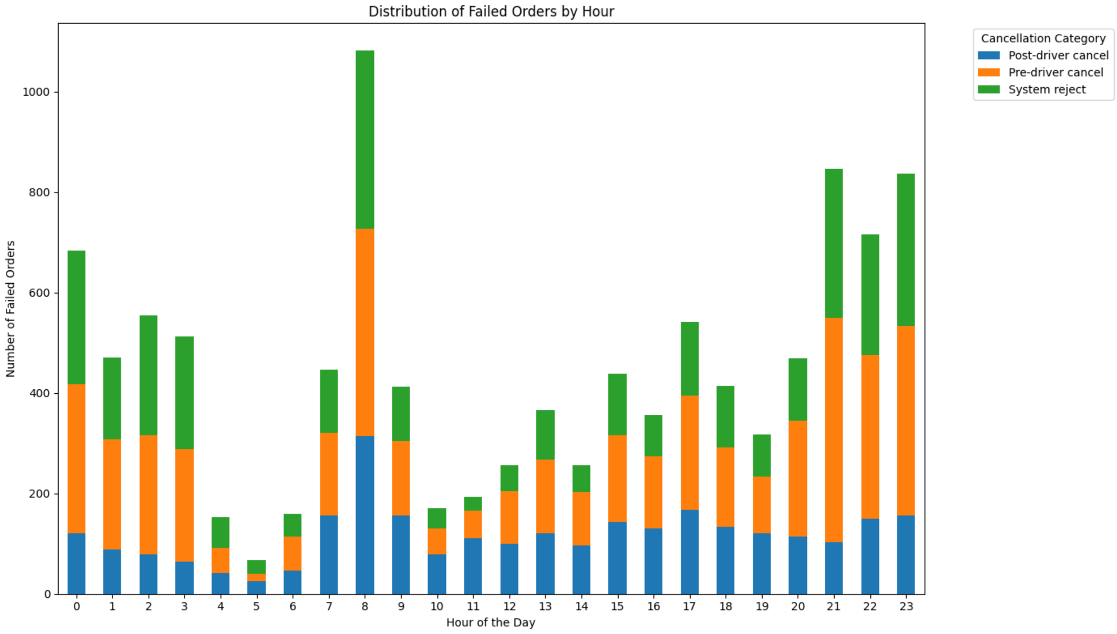
```
#  'order hour'
orders['order_hour'] = orders['order_datetime'].dt.hour

#
orders['cancel_status'] = 'Other'
orders.loc[(orders['is_driver_assigned_key'] == 0) &␣
  ↪(orders['order_status_key'] == 4), 'cancel_status'] = 'Pre-driver cancel'
orders.loc[(orders['is_driver_assigned_key'] == 1) &␣
  ↪(orders['order_status_key'] == 4), 'cancel_status'] = 'Post-driver cancel'
orders.loc[orders['order_status_key'] == 9, 'cancel_status'] = 'System reject'
```

```
#
hourly_cancellation_counts = orders.groupby(['order_hour', 'cancel_status']).
 ↪size().unstack(fill_value=0)

#
plt.figure(figsize=(14, 8))
hourly_cancellation_counts.plot(kind='bar', stacked=True, figsize=(14, 8))
plt.title('Distribution of Failed Orders by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Failed Orders')
plt.xticks(rotation=0)
plt.legend(title='Cancellation Category', bbox_to_anchor=(1.05, 1), loc='upper␣
 ↪left')
plt.tight_layout()
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```



21 23 10 11

[ ]:

```
#     –  IQR
```

[62]:

```
Q1 = orders['cancellations_time_in_seconds'].quantile(0.25)
Q3 = orders['cancellations_time_in_seconds'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

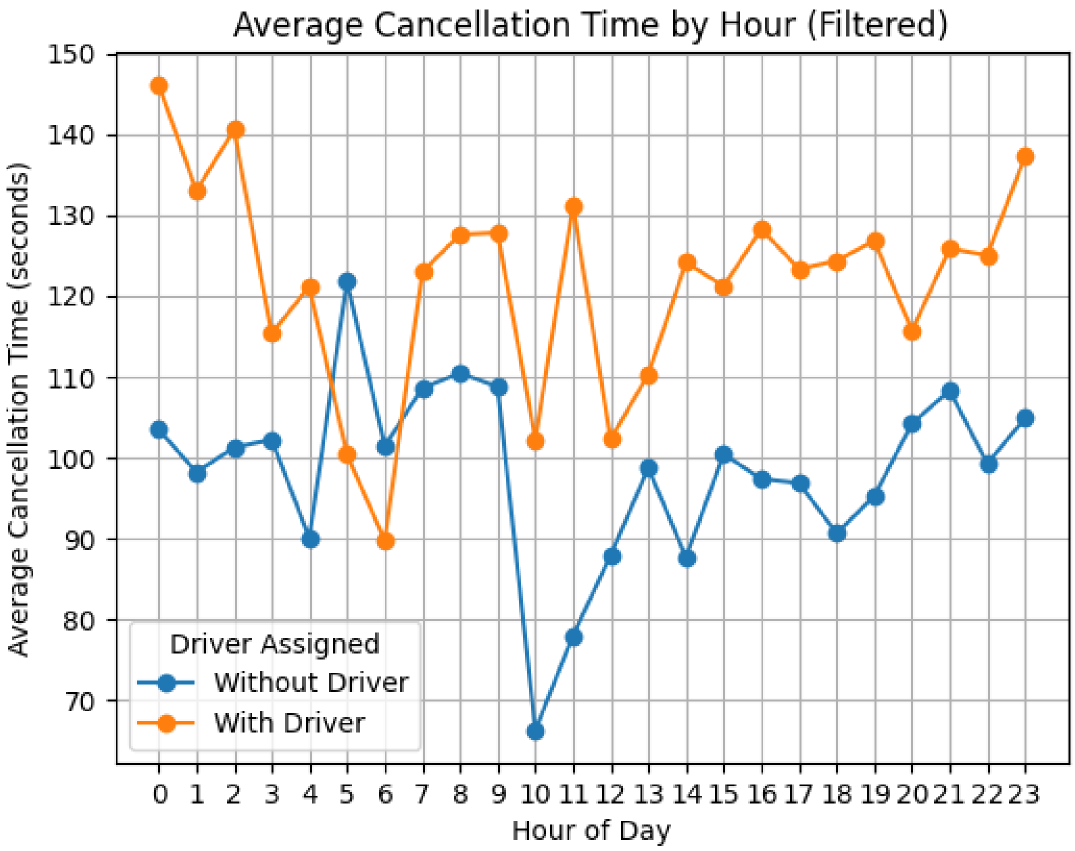
```
#
filtered_orders = orders[(orders['cancellations_time_in_seconds'] >=␣
 ↪lower_bound) &
                          (orders['cancellations_time_in_seconds'] <=␣
 ↪upper_bound)]

#print(len(orders), len(filtered_orders))

#
grouped_avg_cancellation_time = filtered_orders.
 ↪groupby(['is_driver_assigned_key',␣
 ↪'order_hour'])['cancellations_time_in_seconds'].mean().unstack(0)
grouped_avg_cancellation_time

#
plt.figure(figsize=(14, 8))
grouped_avg_cancellation_time.plot(marker='o')
plt.title('Average Cancellation Time by Hour (Filtered)')
plt.ylabel('Average Cancellation Time (seconds)')
plt.xlabel('Hour of Day')
plt.xticks(range(0, 24))
plt.legend(['Without Driver', 'With Driver'], title='Driver Assigned')
plt.grid(True)
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```

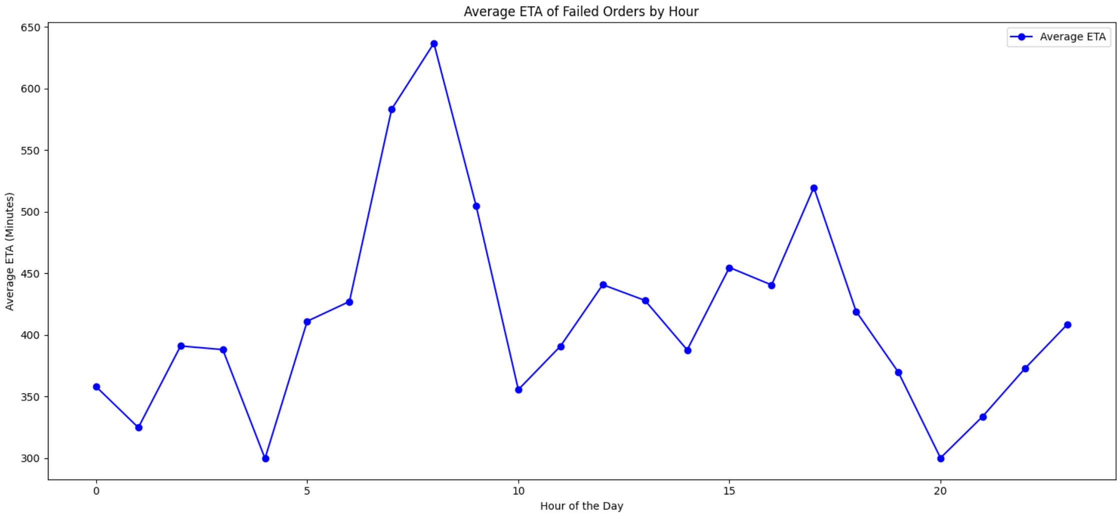


[73]:
```
from matplotlib.ticker import MaxNLocator

#      ETA
average_eta_by_hour = orders.groupby('order_hour')['m_order_eta'].mean()

#
plt.figure(figsize=(15, 7))
average_eta_by_hour.plot(kind='line', color='blue', marker='o', label='Average␣
  ↪ETA')

plt.title('Average ETA of Failed Orders by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Average ETA (Minutes)')
plt.legend()
plt.tight_layout()
plt.show()
```



[82]:

```
import pandas as pd
import h3
import folium
import branca.colormap as cm

orders['h3_code'] = orders.apply(lambda x: h3.geo_to_h3(x['origin_latitude'],␣
 ↪x['origin_longitude'], 8), axis=1)

h3_counts = orders['h3_code'].value_counts()
h3_counts

total_orders = h3_counts.sum()
cumulative_counts = h3_counts.cumsum()
threshold = total_orders * 0.8
relevant_hexes = cumulative_counts[cumulative_counts <= threshold]
relevant_hexes

map_center = [orders['origin_latitude'].mean(), orders['origin_longitude'].
 ↪mean()]
m = folium.Map(location=map_center, zoom_start=14)

max_count = h3_counts.max()
min_count = h3_counts.min()

color_scale = cm.linear.YlOrRd_09.scale(min_count, max_count)

color_dict = {hex_id: color_scale(count) for hex_id, count in h3_counts.items()}

#  relevant_hexes
```

```
for hex_id in relevant_hexes.index:
    #
    boundary = h3.h3_to_geo_boundary(hex_id, geo_json=True)
    #     folium
    boundary_coordinates = [[coord[::-1] for coord in boundary]]  #
    #
    hex_color = color_dict[hex_id]
    #
    folium.Polygon(boundary_coordinates,
                   color=hex_color,
                   weight=1,
                   fill=True,
                   fill_color=hex_color,
                   fill_opacity=0.7,
                   tooltip=f'Hex ID: {hex_id}, Count: {h3_counts[hex_id]}').
 ↪add_to(m)

m.save('hex_map_colored.html')
```

[80]:

```
import pandas as pd
import h3
import folium
from branca.colormap import linear




#   hex_id
orders['hex_id'] = orders.apply(
    lambda row: h3.geo_to_h3(row['origin_latitude'], row['origin_longitude'],␣
 ↪8), axis=1)

#    hexagon
hex_counts = data_orders['hex_id'].value_counts().reset_index()
hex_counts.columns = ['hex_id', 'count']

#
max_count = hex_counts['count'].max()
colormap = linear.YlOrRd_09.scale(0, max_count)

#
m = folium.Map(location=[orders['origin_latitude'].mean(),␣
 ↪orders['origin_longitude'].mean()], zoom_start=12)

#   hexagons
for _, row in hex_counts.iterrows():
    #   hexagon
    hex_boundary = h3.h3_to_geo_boundary(row['hex_id'], geo_json=True)
    #
    color = colormap(row['count'])
    #
    folium.Polygon(
        locations=hex_boundary,
        color=color,
        weight=2,  #
        fill_color=color,
        fill_opacity=1.0,  #
        popup=f'Orders: {row["count"]}'
    ).add_to(m)

#
colormap.add_to(m)

#
m.save('hexagon_map.html')

m
```

[80]:

```
<folium.folium.Map at 0x7fb92fb91690>
```

[ ]: