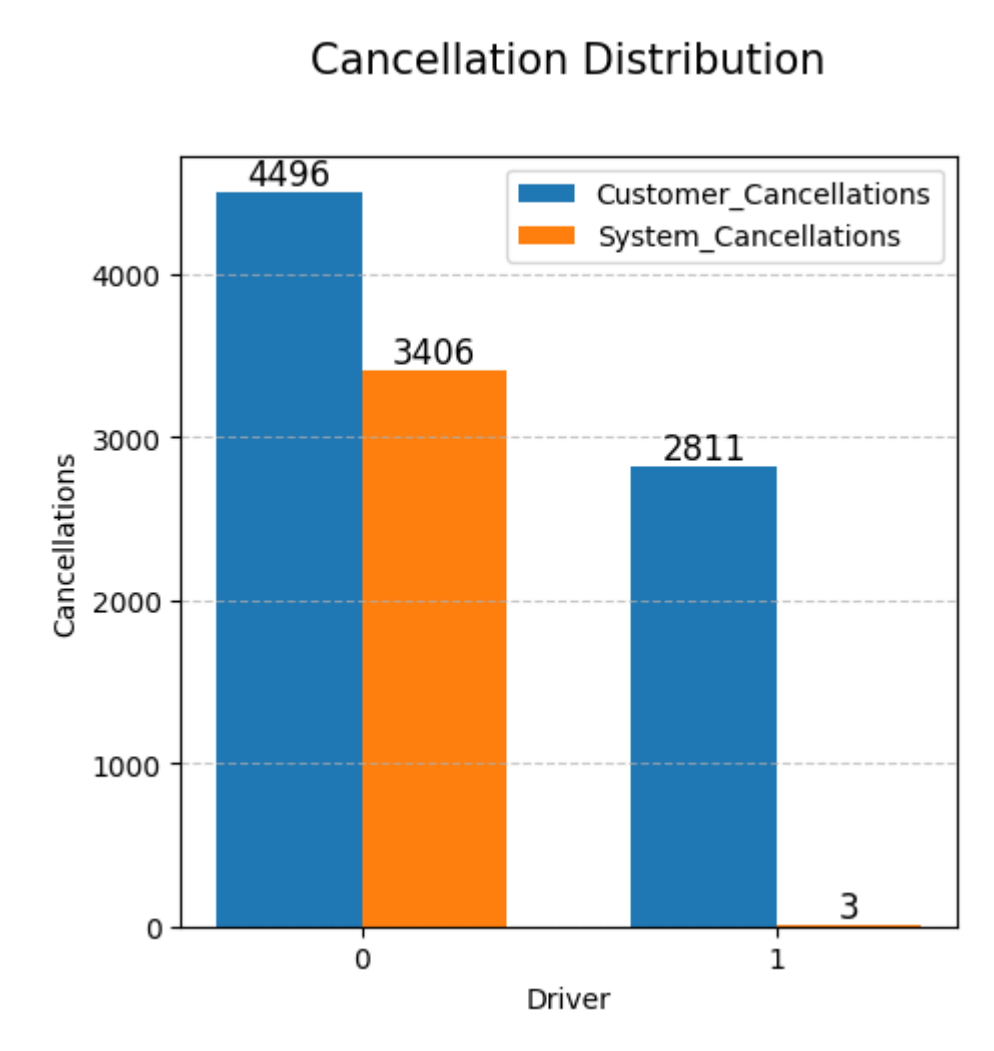
In [45]:
```python
# Project Name - Insights from Failed Orders

# Gett, previously known as GetTaxi, is an Israeli-developed technology platform solely focused on corporate Ground Transportation Management (GTM)
# They have an application where clients can order taxis, and drivers can accept their rides (offers).
# At the moment, when client clicks the Order button in the application, matching system searches for the most relevant drivers & offers them the c
# In this task, we would investigate some matching metrics for orders that didn't complete successfully;the customer didn't end up getting a car.

# Tasks Accomplished

# Task 1 - Building-up distribution of orders according to reasons for failure: cancellations before and after driver assignment, & reasons for ord
#          Analysing the resulting plot. Which category has the highest number of orders?
# Task 2 - Ploting the distribution of failed orders by hours.
#          If there's a trend that certain hours have an abnormally high proportion of one category or another.
#          Finding biggest Failed Hours & it's explaination
# Task 3 - Ploting the average time to cancellation with and without driver, by the hour.
#          Finding the outliers in the data. Drawing conclusions from the plot.
# Task 4 - Ploting the distribution of average ETA by hours & explaination.
# Task 5 - Ploting the h3 & folium packages.
#          Calculating how many sizes 8 hexes contain 80% of all orders from the original data sets and visualise the hexes.
#          Colouring them by the number of fails on the map.

# Data Description

# Table Name - Data_orders. The data_orders data set contains the following columns:

# order_datetime - time of the order
# origin_longitude - longitude of the order
# origin_latitude - latitude of the order
# m_order_eta - time before order arrival
# order_gk - order number
# order_status_key - status, an enumeration consisting of the following mapping:
#                     4 - cancelled by client,
#                     9 - cancelled by system, i.e., a reject
# is_driver_assigned_key - whether a driver has been assigned if yes then 1 else 0
# cancellation_time_in_seconds - how many seconds passed before cancellation
```

In [1]:
```python
# importing all the required libraries
import pandas as pd      # for manipulating the data
import numpy as np       # for calculations
import matplotlib as mpl   # for visual representation
import matplotlib.pyplot as plt  # (for conviniently using the short-form - plt)

df = pd.read_csv("D:\\Business Analyst Projects\Gett Taxi dataset\data_orders.csv")  # importing the data
df["Hours"] = df["order_datetime"].str.split(":").str[0].astype(int)    # deriving the hour column from the order_datetime column as it would be re
```

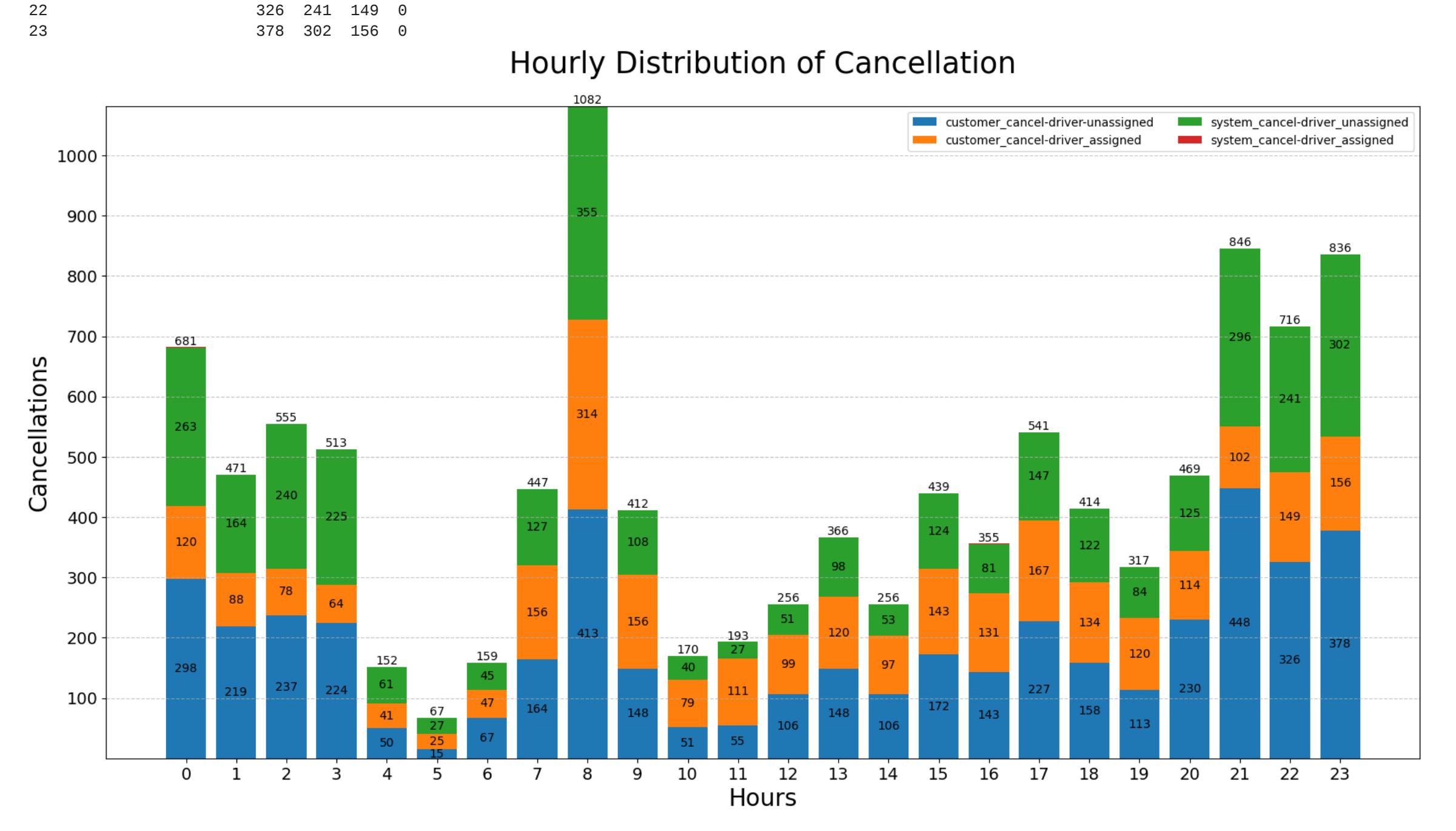
In [42]:
```python
# Task 1
# finding the distribution of order cancellations using the pivot table function
Cancellations = df.pivot_table(
    index = "is_driver_assigned_key",
    columns = "order_status_key",
    values = "order_datetime",  # this is only chosen to perform the counting
    aggfunc = "count")
print(Cancellations)  # displaying the pivot Table, 4 = customer_cancel, 9 = system_cancel, 0 - driver_unassigned, 1 - assigned
print() #just for some space
print()

Cancellations = Cancellations.reset_index()  # this is done to ease the plotting the data in the graph using column names

# Building Clustered columns Bar graph for visualizing the pivot table

categories = ['Assigned', 'Unassigned']  # these are the categories of our graph's x-axis
data_set1 = Cancellations[(4)]     # this dataset would be our y-axis 1st set of columns
data_set2 = Cancellations[(9)]     # this would be our 2nd set of columns

# Setting up positions for the clustered columns
bar_width = 0.35
index = np.arange(len(categories))
fig, ax = plt.subplots(figsize=(5,5))

# Plotting the first set of columns
bars1 = ax.bar(index - bar_width/2, data_set1, bar_width, label='Customer_Cancellations')

# Plotting the second set of columns next to the first one
bars2 = ax.bar(index + bar_width/2, data_set2, bar_width, label='System_Cancellations')

# Adding labels to the bars using bar_label
ax.bar_label(bars1, fontsize=12, label_type='edge', color='black')
ax.bar_label(bars2, fontsize=12, label_type='edge', color='black')

# Customizing the chart
ax.set_xlabel('Driver', fontsize = "10")
ax.set_ylabel('Cancellations', fontsize = "10")
ax.set_title('Cancellation Distribution', fontsize = "15", pad="30")
ax.set_xticks(np.arange(0,2,1))
plt.grid(axis="y", linestyle='--', alpha=0.7)
plt.legend()

# to display the chart
plt.show()
```

```
order_status_key           4     9
is_driver_assigned_key
0                       4496  3406
1                       2811     3
```

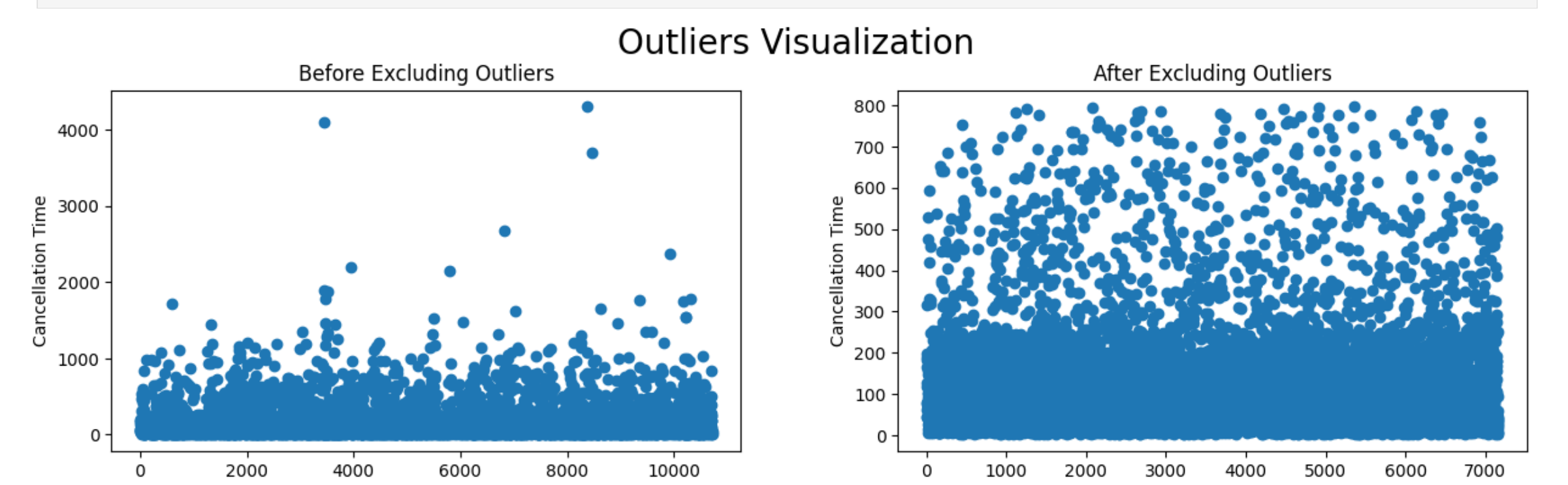


In [43]:
```python
# Task - 2 Finding the Hourly distribution of the cancelled orders

# creating a pivot table to find cancellation for every hour relating to driver-assignment & cancellation reason
Hourly_Distribution = df.pivot_table(
    index = "Hours",
    columns = ["is_driver_assigned_key","order_status_key"],
    values = "order_datetime",
    aggfunc = "count").fillna(0).astype(int)   # taking care of NaN values and setting up table with int values only

print(Hourly_Distribution)     # displaying the graph, as mentioned above, 4 is customer_cancel, 9 is system_cancel & 0-1 are driver unassigned & a
Hourly_Distribution = Hourly_Distribution.reset_index()   # this is done to ease the plotting the data in the graph using column names

# For visualizing this data we would use stacked column bar graph
fig, ax = plt.subplots(figsize=(20,10))   # using plottype - subplots & setting up the size

x = Hourly_Distribution["Hours"]      # x-axis
ax.set_xticks(np.arange(len(x)))    # setting the values on the x-axis

# These would be our 4 columns to be stack on the y-axis
y1 = Hourly_Distribution[(0,4)]    # Customer_cancel-driver_unassigned
y2 = Hourly_Distribution[(1,4)]    # Customer_cancel-driver_unassigned
y3 = Hourly_Distribution[(0,9)]    # System_cancel-driver_unassigned
y4 = Hourly_Distribution[(1,9)]    # System_cancel-driver_assigned
ax.set_yticks(np.arange(100,1101,100))

# setting up variables to be used for stacking the bar on top of each other, these would be measures for their next bar to start upon
bottom2 = np.add(y1, y2)
bottom3 = np.add(bottom2, y3)
bottom4 = np.add(bottom3, y4)

# plotting the bars
bar1 = ax.bar(x, y1, label= 'customer_cancel-driver-unassigned')
bar2 = ax.bar(x, y2, bottom = y1, label= 'customer_cancel-driver_assigned')
bar3 = ax.bar(x, y3, bottom = bottom2, label= 'system_cancel-driver_unassigned')
bar4 =  ax.bar(x, y4, bottom = bottom3, label = 'system_cancel-driver_assigned')

# Adding labels using bar_label
ax.bar_label(bar1, fontsize=10, label_type='center', color='black')
ax.bar_label(bar2, fontsize=10, label_type='center', color='black')
ax.bar_label(bar3, fontsize=10, label_type='center', color='black')
ax.bar_label(bar3, fontsize=10, label_type='edge', color='black')


# customizing the chart
ax.tick_params(axis="both", which='major', labelsize=14)
ax.set_xlabel("Hours", fontsize="20")
ax.set_ylabel("Cancellations", fontsize="20")
ax.set_title("Hourly Distribution of Cancellation", fontsize="25", pad="28")
ax.legend(loc ="upper right", ncols = 2, fontsize = 10)
plt.grid(axis="y", linestyle='--', alpha=0.7)

# to display the chart
plt.show()
```

```
is_driver_assigned_key    0         1
order_status_key          4    9    4  9
Hours
0                       298  263  120  2
1                       219  164   88  0
2                       237  240   78  0
3                       224  225   64  0
4                        50   61   41  0
5                        15   27   25  0
6                        67   45   47  0
7                       164  127  156  0
8                       413  355  314  0
9                       148  108  156  0
10                       51   40   79  0
11                       55   27  111  0
12                      106   51   99  0
13                      148   98  120  0
14                      106   53   97  0
15                      172  124  143  0
16                      143   81  131  1
17                      227  147  167  0
18                      158  122  134  0
19                      113   84  120  0
20                      230  125  114  0
21                      448  296  102  0
22                      326  241  149  0
23                      378  302  156  0
```

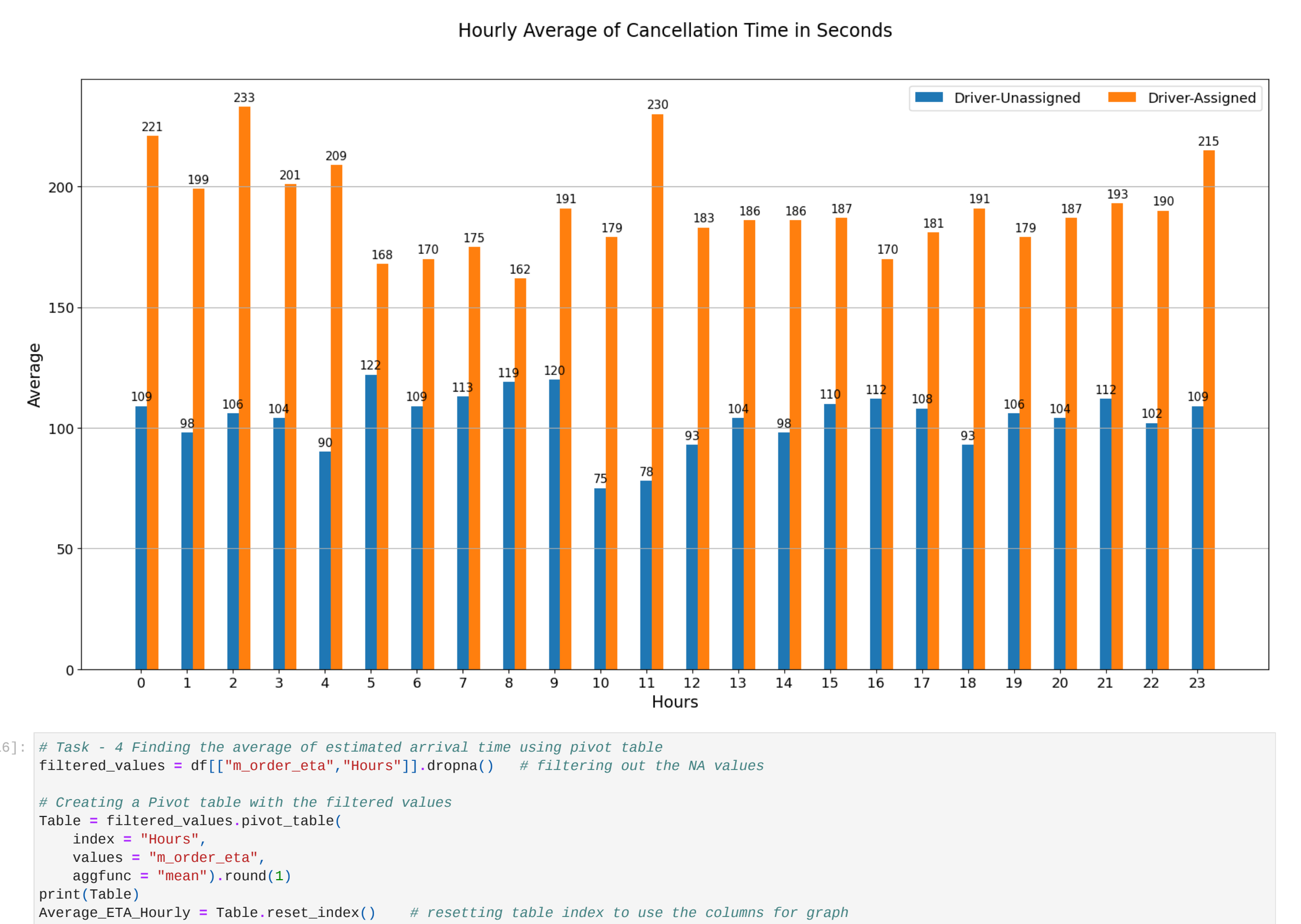


In [6]:
```python
# Task 3 - Finding the average cancellation time per hour with & without driver (excluding outliers)
# Task 3.1 - Finding the Outliers.


# Considering any value outside 3 Standard-Deviations from the Average as an Outlier.
# Our Formula becomes - Outlier >= Average + (3*Standard-Deviations)
average = df["cancellations_time_in_seconds"].dropna().mean()
standard_deviation = np.std(df["cancellations_time_in_seconds"].dropna())
outliers = round(average + (3* standard_deviation))

#Visual representation of Data before and After Outliers
plt.figure(figsize=(15, 4))  # defining the size of the subplots
plt.subplots_adjust(wspace=0.25, hspace = 0.5, top=0.85)     # setting positions of the subplots
plt.suptitle("Outliers Visualization", horizontalalignment="center", fontsize = 20)    # head title of both the subplots

# setting up plot including outliers
x = range(len(df))     # x Axis
y = df["cancellations_time_in_seconds"]    # y Axis
p1 = plt.subplot(1,2,1)     # plot 1 - 1 row, 2 columns
plt.scatter(x,y)        # scatter plot graph
# setting up labels & title
p1.set_ylabel("Cancellation Time")
p1.set_xlabel("Total dataframe values")
plt.title("Before Excluding Outliers")

# setting up plot excluding outliers
y2 = df[(df["cancellations_time_in_seconds"] < outliers)]["cancellations_time_in_seconds"]   # x Axis
x2 = np.arange(len(y2))    # y Axis
p2 = plt.subplot(1,2,2)    # plot 2 - 1 row, 2 columns

#setting up lables & titles
plt.title("After Excluding Outliers")
p2.set_ylabel("Cancellation Time")
p2.set_xlabel("Total dataframe values")
plt.scatter(x2,y2)

# to display the plots
plt.show()
```



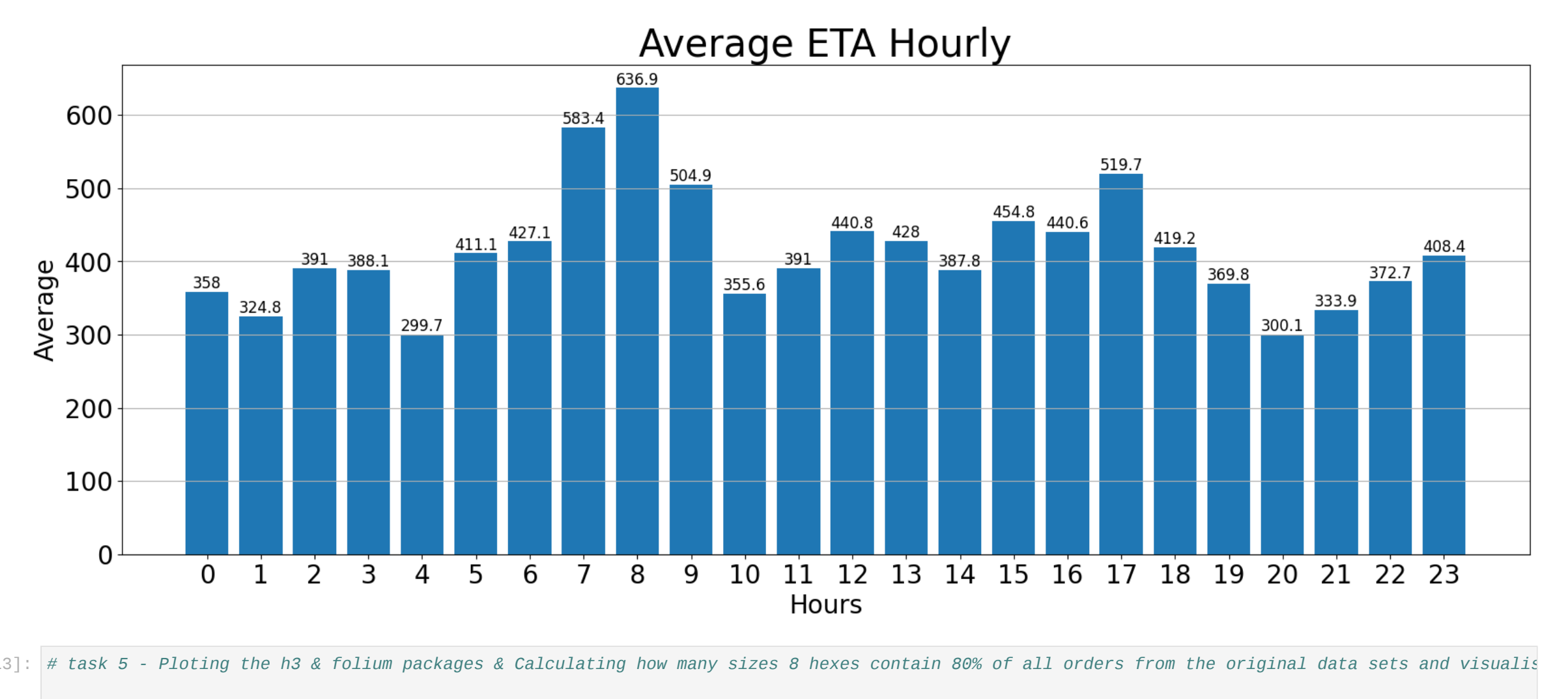
In [41]:
```python
# Task 3.2 -  Finding the Average Cancellation time per hour with & without driver assignment

#Filtering data (excluding the outliers)
filtered_data = df[(df["cancellations_time_in_seconds"] < outliers)][["cancellations_time_in_seconds","is_driver_assigned_key","Hours"]]

# Pivot Table for the Hourly Calculation of Cancellation time in seconds
Avg_Cancel_Time = filtered_data.pivot_table(
    index = "Hours",
    columns = "is_driver_assigned_key",
    values = "cancellations_time_in_seconds",
    aggfunc = "mean").round()
print(Avg_Cancel_Time)  # to display the table
print()    # just for some gap between pivot table & the graph
Avg_Cancel_Time = Avg_Cancel_Time.reset_index()        # Index reset to use the columns for the table visualization

#As we have 2 category of values for every hour, we would again use clustered bar graph
Hours = [i for i in Avg_Cancel_Time["Hours"]] # for the values in X axis
Columns = {
    'Driver-Unassigned': [j for j in Avg_Cancel_Time[(0)]],
    'Driver-Assigned': [k for k in Avg_Cancel_Time[(1)]] }  # to assign y axis values

fig, ax = plt.subplots(figsize=(20, 10)) # creating subplots to display both column values together in the same graph

x = np.arange(len(Hours))  # the label (x axis)
width = 0.25  # the width of the bars
multiplier = 0 # an increasing variable to gradually offset the bars

# contructing the rectangles (rects) of the bar graph
for key, value in Columns.items():
    offset = width * multiplier
    rects = ax.bar(x + offset, value, width, label=key)
    ax.bar_label(rects, padding=3, fontsize = 11) # for value of each bar to be displayed at it's top)
    multiplier += 1

# customizing the graph
ax.set_xticks(x) # setting up x ticks
ax.set_ylabel('Average',fontsize = 15) # y axis label
ax.set_xlabel("Hours" ,fontsize = 15) # x axis label
ax.set_title("Hourly Average of Cancellation Time in Seconds", fontsize = 17, pad="40") # chart Title
ax.tick_params(axis='both', which='major', labelsize=13)  # setting up fontsize of x & y axis ticks
ax.legend(loc='upper right', ncols=3, fontsize = 13) # legend position & fontsize
plt.grid(axis="y")    #setting up grid-lines

# to display the graph
plt.show()
```

```
is_driver_assigned_key      0      1
Hours
0                       109.0  221.0
1                        98.0  199.0
2                       106.0  233.0
3                       104.0  201.0
4                        90.0  209.0
5                       122.0  168.0
6                       109.0  170.0
7                       113.0  175.0
8                       119.0  162.0
9                       120.0  191.0
10                       75.0  179.0
11                       78.0  230.0
12                       93.0  183.0
13                      104.0  186.0
14                       98.0  186.0
15                      110.0  187.0
16                      112.0  170.0
17                      108.0  181.0
18                       93.0  191.0
19                      106.0  179.0
20                      104.0  187.0
21                      112.0  193.0
22                      102.0  190.0
23                      109.0  215.0
```

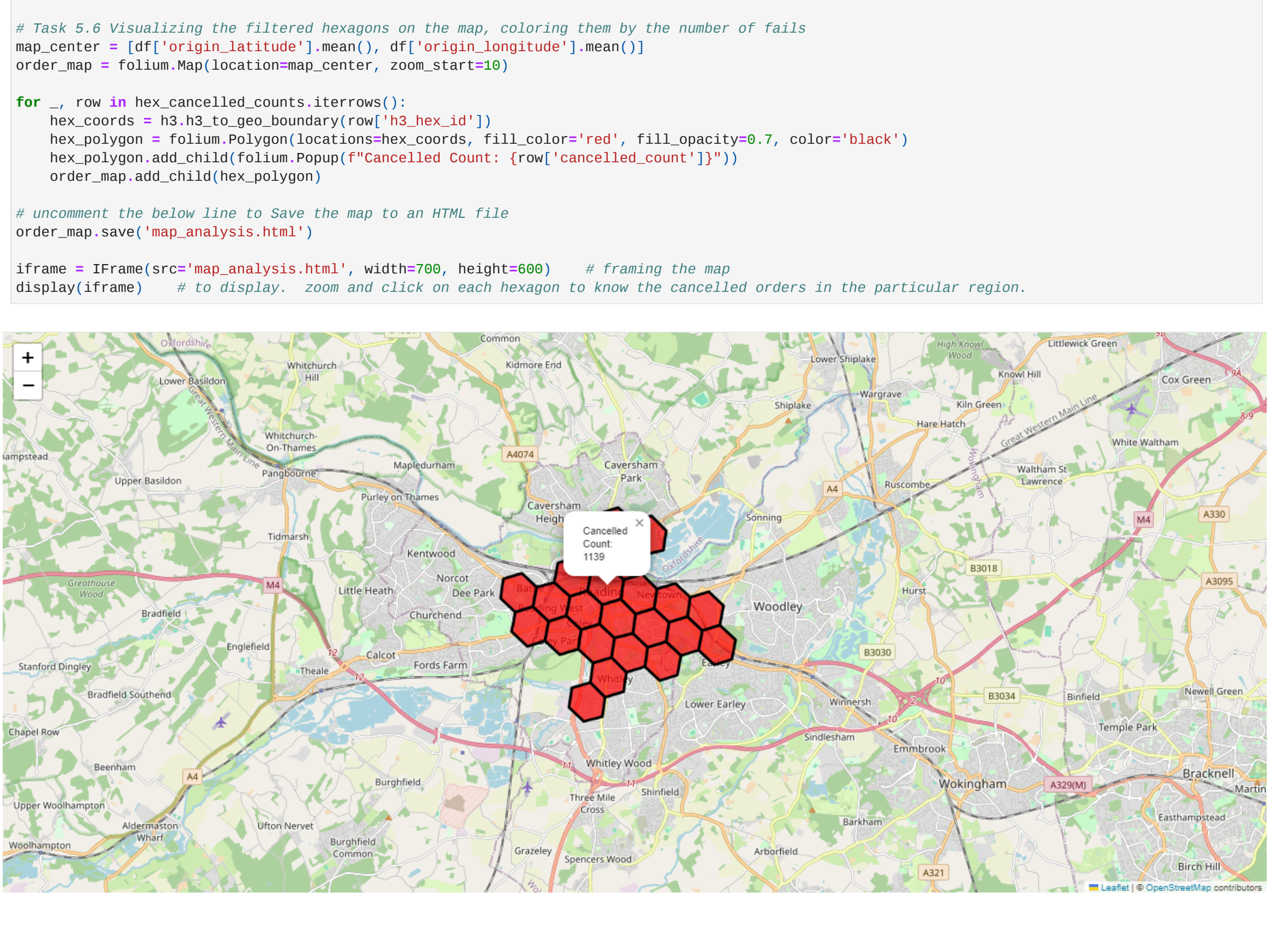


In [16]:
```python
# Task - 4 Finding the average of estimated arrival time using pivot table
filtered_values = df[["m_order_eta","Hours"]].dropna()   # filtering out the NA values

# Creating a Pivot table with the filtered values
Table = filtered_values.pivot_table(
    index = "Hours",
    values = "m_order_eta",
    aggfunc = "mean").round(1)
print(Table)
Average_ETA_Hourly = Table.reset_index()    # resetting table index to use the columns for graph
print()  # for spacing between table and graph

# Visualizing the table using matplotlib
fig, axs = plt.subplots(figsize=(20, 7))
x = Average_ETA_Hourly["Hours"]  # Keep hours at x-axis
y = Average_ETA_Hourly["m_order_eta"]  # Keep average values at y-axis
bars = axs.bar(x, y)  # Plotting both x & y axes

axs.set_xticks(x)  # To get all the Hours displayed
plt.grid(axis="y")  # Adding horizontal gridlines

# Setting labels & titles
axs.set_title("Average ETA Hourly", fontsize=30)  # Title for the graph
axs.set_ylabel("Average", fontsize=20)
axs.set_xlabel("Hours", fontsize=20)
axs.tick_params(axis='both', which='major', labelsize=20)

plt.bar_label(bars, fontsize=12, label_type='edge', color='black') # setting up bar labels to be displayed on top
plt.show()
```

```
       m_order_eta
Hours
0            358.0
1            324.8
2            391.0
3            388.1
4            299.7
5            411.1
6            427.1
7            583.4
8            636.9
9            504.9
10           355.6
11           391.0
12           440.8
13           428.0
14           387.8
15           454.8
16           440.6
17           519.7
18           419.2
19           369.8
20           300.1
21           333.9
22           372.7
23           408.4
```



In [13]:
```python
# task 5 - Ploting the h3 & folium packages & Calculating how many sizes 8 hexes contain 80% of all orders from the original data sets and visualis

import folium   # for creating map
import h3        # for hexagons
from IPython.display import IFrame, display    # to display the map


# Task 5.1 - Aggregating data by H3 hexagons & calculating the number of orders in each hexagon
df['h3_hex_id'] = df.apply(lambda row: h3.geo_to_h3(row['origin_latitude'], row['origin_longitude'], 8), axis=1)
hex_counts = df['h3_hex_id'].value_counts().reset_index()
hex_counts.columns = ['h3_hex_id', 'order_count']

# Task 5.2 -  Sorting the hexagons based on the number of orders in descending order
hex_counts = hex_counts.sort_values(by='order_count', ascending=False)

# Task 5.3 - Calculating the cumulative percentage of orders & finding the threshold for 80% of orders
hex_counts['cumulative_percentage'] = hex_counts['order_count'].cumsum() / hex_counts['order_count'].sum()
threshold_hexes = hex_counts[hex_counts['cumulative_percentage'] <= 0.8]

# Task 5.4 - Filtering out the hexagons that contribute to 80% of the orders
filtered_hexes = df[df['h3_hex_id'].isin(threshold_hexes['h3_hex_id'])]

# Task 5.5 Calculating the number of fails for each filtered hexagon
hex_cancelled_counts = filtered_hexes[filtered_hexes['order_status_key']==4].groupby('h3_hex_id')['order_status_key'].count().reset_index()
hex_cancelled_counts.columns = ['h3_hex_id', 'cancelled_count']

# Task 5.6 Visualizing the filtered hexagons on the map, coloring them by the number of fails
map_center = [df['origin_latitude'].mean(), df['origin_longitude'].mean()]
order_map = folium.Map(location=map_center, zoom_start=10)

for _, row in hex_cancelled_counts.iterrows():
    hex_coords = h3.h3_to_geo_boundary(row['h3_hex_id'])
    hex_polygon = folium.Polygon(locations=hex_coords, fill_color='red', fill_opacity=0.7, color='black')
    hex_polygon.add_child(folium.Popup(f"Cancelled Count: {row['cancelled_count']}"))
    order_map.add_child(hex_polygon)

# uncomment the below line to Save the map to an HTML file
order_map.save('map_analysis.html')

iframe = IFrame(src='map_analysis.html', width=700, height=600)    # framing the map
display(iframe)    # to display.  zoom and click on each hexagon to know the cancelled orders in the particular region.
```



In [ ]: