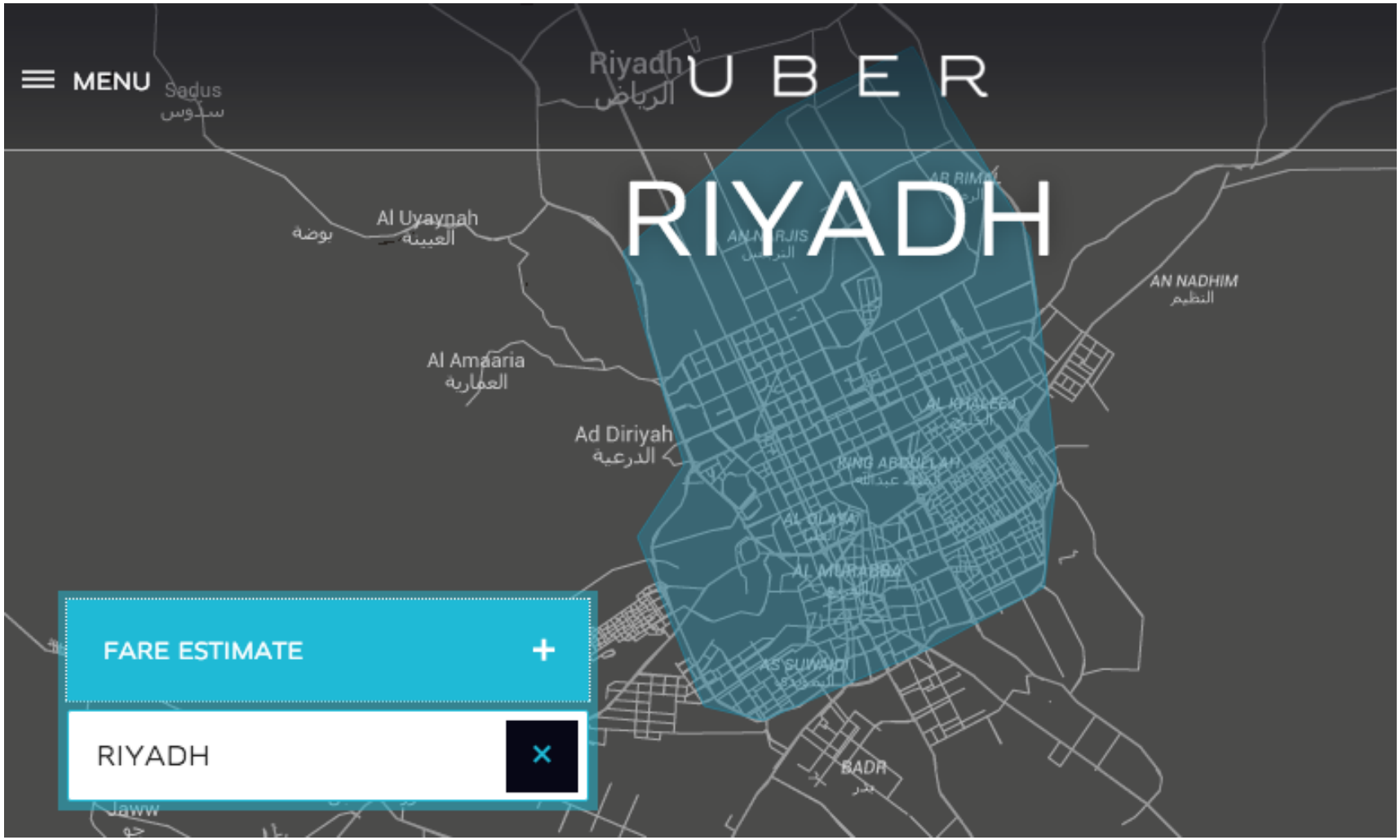
Uber - Internal Audit Analytics Exercise: 2020 strategy for Riyadh Market

In [1]:
```
from IPython.core.display import HTML
HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
The raw code for this IPython notebook is by default hidden for easier reading.
To toggle on/off the raw code, click <a href="javascript:code_toggle()">here</a>.''')
```

Out[1]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

In [2]:
```
from IPython.display import Image
Image(filename='output1.png',width=1000, height=400)
```

Out[2]:



Introduction

Uber Technologies Inc. is investing $250 million to expand in the Middle East and North Africa, which have some of the ride-sharing service's fastest-growing markets, Bloomberg reports.

Uber is already in Saudi Arabia, and the ride-sharing app is having a significant impact on the transportation economy there.

Problem Statement:

- Understanding the needs of key stakeholders and performing analysis/prototyping solutions
- In this exercise, EMEA rideshare data is to be analyzed and use it to draw a conclusion for the 2020 strategy for the Riyadh market.

Reading, Cleaning and Compiling Data from the given CSV data sample

Importing Libraries

The following libraries should be imported to run this notebook: pandas, sqlite3, numpy, matplotlib, plotly, dash, pivottablejs.

The Plotly Python library is an interactive open-source library. This can be a very helpful tool for data visualization and understanding the data simply and easily. plotly graph objects are a high-level interface to plotly which are easy to use. It can plot various types of graphs and charts like scatter plots, line charts, bar charts, box plots, histograms, pie charts, etc.

Dash is a Python framework for building analytical web applications. Dash helps in building responsive web dashboards that is good to look at and is very fast without the need to understand complex front-end frameworks or languages such as HTML, CSS, JavaScript.

PivotTable.js is a Javascript Pivot Table and Pivot Chart library with drag'n'drop interactivity, and it can now be used with Jupyter/IPython Notebook via the pivottablejs module.

In [6]:
```
import pandas as pd
import sqlite3 as sql
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Plotly graphs
from chart_studio import plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot
from plotly.subplots import make_subplots
# this helps us get the theme settings
import plotly.io as plt_io
import plotly.graph_objs as go
#import cufflinks as cf
#cf.go_offline()
#cf.set_config_file(offline=False, world_readable=True)
import plotly.express as px
#importing dash
from dash import Dash, dcc, html, Input, Output  # pip install dash (version 2.0.0 or higher)
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import dash_table
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
from IPython.core.display import HTML
HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
The raw code for this IPython notebook is by default hidden for easier reading.
To toggle on/off the raw code, click <a href="javascript:code_toggle()">here</a>.''')
```

Out[4]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Exploratory Data Analysis

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually.Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important.

Units considered to perform the analysis/output:

1. Distance: miles
2. Time: Local time in minutes
3. Fare: USD

In [7]:
```
df_riyadh=pd.read_csv('riyadh_sample.csv')
```

Displays datatype of all columns

In [8]:
```
df_riyadh.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 36 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   pickup_local_time      28750 non-null  object 
 1   pickup_utc_time        28750 non-null  object 
 2   cancel_fee_local       40000 non-null  float64
 3   cancel_fee_usd         40000 non-null  float64
 4   city_id                40000 non-null  int64  
 5   rider_app              39846 non-null  object 
 6   rider_device           40000 non-null  object 
 7   rider_trip_count       28556 non-null  float64
 8   rider_id               40000 non-null  object 
 9   partner_vehicle_count  40000 non-null  int64  
 10  driver_trip_count      28556 non-null  float64
 11  driver_id              40000 non-null  object 
 12  dropoff_local_time     28556 non-null  object 
 13  dropoff_utc_time       28556 non-null  object 
 14  esttime_to_pickup      38529 non-null  float64
 15  request_type           38537 non-null  object 
 16  entered_destination    40000 non-null  bool   
 17  paid_cash              40000 non-null  bool   
 18  completed_trip         40000 non-null  bool   
 19  surged_trip            40000 non-null  bool   
 20  trip_fare_local        40000 non-null  float64
 21  trip_fare_usd          40000 non-null  float64
 22  partner_id             40000 non-null  object 
 23  request_local_time     40000 non-null  object 
 24  request_utc_time       40000 non-null  object 
 25  distance_to_pickup     38601 non-null  float64
 26  time_to_pickup         28750 non-null  float64
 27  trip_status            40000 non-null  object 
 28  trip_distance_miles    39999 non-null  float64
 29  trip_duration_seconds  40000 non-null  int64  
 30  trip_id                40000 non-null  object 
 31  vehicle_trip_count     28556 non-null  float64
 32  vehicle_id             40000 non-null  object 
 33  vehicle_type           39693 non-null  object 
 34  pickup_geo             40000 non-null  object 
```

```
 35  dropoff_geo            40000 non-null  object
dtypes: bool(4), float64(11), int64(3), object(18)
memory usage: 9.9+ MB
```

Displays first five rows of the dataset

In [9]:
```
df_riyadh.head()
```

Out[9]:

	pickup_local_time	pickup_utc_time	cancel_fee_local	cancel_fee_usd	city_id	rider_app	rider_device	rider_trip_count	rider_id
0	2018-05-10 09:00:00	2018-05-10 06:00:00	0.0	0.0	1	3.298.10000	iphone	131.0	888bb3c7-c55b-5d41-8cd9-9a4a2554b4e4
1	NaN	NaN	0.0	0.0	1	3.298.10000	iphone	NaN	cfc5db3c-e25e-5f48-9d59-8f57fb611f86
2	2018-05-17 09:00:00	2018-05-17 06:00:00	0.0	0.0	1	3.268.10002	iphone	53.0	5296a57d-3294-54a7-a0a0-e7fa7e6d8caa
3	NaN	NaN	0.0	0.0	1	3.275.10002	iphone	NaN	0792af9c-547c-56ed-adce-e217b7eed8d2
4	2018-05-10 16:00:00	2018-05-10 13:00:00	0.0	0.0	1	3.241.2	iphone	499.0	7aebd941-f808-54e5-a7ce-ae890ed6e1e8

5 rows × 36 columns

Part 2: Analysis and Presentation

Statistical information about dataset

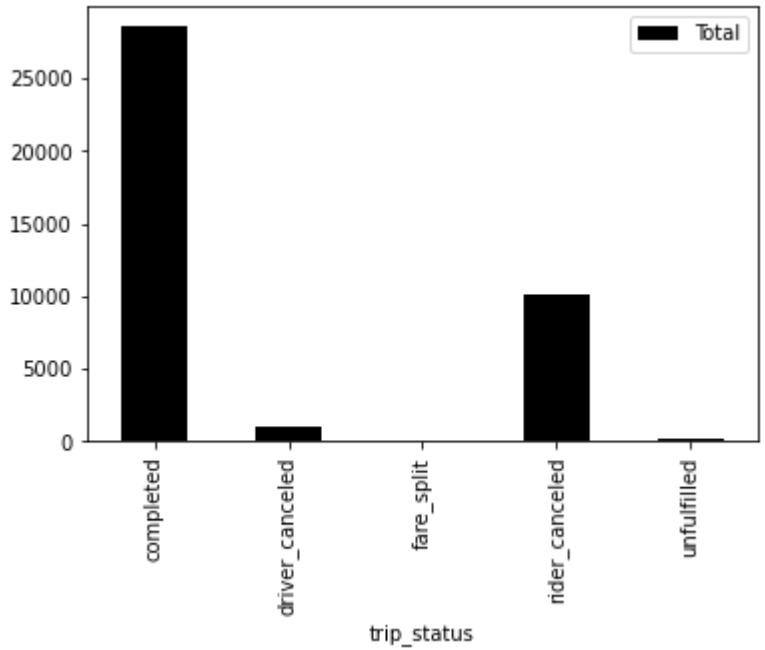
In [16]:
```
df_riyadh.describe()
```

Out[16]:

	cancel_fee_local	cancel_fee_usd	city_id	rider_trip_count	partner_vehicle_count	driver_trip_count	esttime_to_pickup	trip_fare_lo
count	40000.000000	40000.000000	40000.0	28556.000000	40000.000000	28556.000000	38529.000000	40000.0000
mean	0.209350	0.055824	1.0	170.142912	3.248325	1237.466452	282.948896	16.6363
std	1.293074	0.344802	0.0	250.283997	9.602735	1455.880407	180.717438	18.0275
min	0.000000	0.000000	1.0	1.000000	1.000000	1.000000	1.000000	0.0000
25%	0.000000	0.000000	1.0	22.000000	1.000000	251.000000	166.000000	0.0000
50%	0.000000	0.000000	1.0	77.000000	1.000000	693.000000	253.000000	12.0000
75%	0.000000	0.000000	1.0	212.000000	1.000000	1652.250000	361.000000	23.5200
max	20.000000	5.333483	1.0	3166.000000	73.000000	12735.000000	4073.000000	440.7000

In [17]:
```
df_chart1=df_riyadh.groupby('trip_status')['trip_id'].count()
df_chart1 = df_chart1.reset_index()
df_chart1=df_chart1.rename({'trip_status': 'trip_status', 'trip_id': 'Total'}, axis=1)
df_chart1.plot(kind='bar',x='trip_status',y='Total', color='black')
```

Out[17]: <AxesSubplot:xlabel='trip_status'>



In [18]:
```
df_chart1=df_riyadh.groupby('trip_status')['trip_id'].count()
df_chart1.loc['Grand Total'] = df_chart1.sum()
df_chart1.columns=['trip_status','Total']
```

In [19]:
```
df_chart1 = df_chart1.reset_index()
```

In [20]:
```
df_chart1=df_chart1.rename({'trip_status': 'trip_status', 'trip_id': 'Total'}, axis=1)
```

In [21]:
```
df_chart1['trip_status']=df_chart1['trip_status'].astype(str)
df_chart1['Total']=df_chart1['Total'].astype(int)
```

Below are some numerical findings from the data which will further help to analyze data

Completion Rate of rides

In [22]:
```
Completion_rate=(df_chart1.loc[df_chart1['trip_status'] == 'completed']['Total'])/df_riyadh['trip_id'].count()
Completion_rate=Completion_rate[0]
Completion_rate=Completion_rate*100
print('Completion_rate: '+ str(Completion_rate)+'%')
```

```
Completion_rate: 71.37%
```

Cancellation Rate of rides

In [23]:
```
Cancellation_rate=(100-Completion_rate)
print('Cancellation_rate: '+ str(round(Cancellation_rate,2))+'%')
```

```
Cancellation_rate: 28.63%
```

Total fare of rides

In [24]:
```
Total_Fare=df_riyadh['trip_fare_usd'].sum()
print('Total_Fare: ' + '$', round(Total_Fare,2))
```

```
Total_Fare: $ 177445.52
```

Percentage of Surged trips

In [25]:
```
Surged_trip_pct=(df_riyadh.loc[df_riyadh['surged_trip'] == True]['surged_trip'].count()/df_riyadh['trip_id'].cou
print('Surged_trip_pct:'+str(round(Surged_trip_pct,2))+'%')
```

```
Surged_trip_pct:25.42%
```

Average fare of completed rides

In [26]:
```
Average_Fare=Total_Fare/df_riyadh.loc[df_riyadh['trip_status']=='completed']['trip_id'].count()
print('Average_Fare: ' + '$', round(Average_Fare,2))
```

```
Average_Fare: $ 6.22
```

Trip Fare on basis of Surge

In [27]:
```
df_cancellation=df_riyadh.loc[df_riyadh['trip_status']!='completed'].groupby('trip_status')['trip_id'].count()
df_cancellation.columns=['trip_status','Total']
```

In [28]:
```
df_cancellation = df_cancellation.reset_index()
df_cancellation=df_cancellation.rename({'trip_status': 'trip_status', 'trip_id': 'Total'}, axis=1)
df_cancellation['Pct_Total']=df_cancellation['Total']/df_riyadh.loc[df_riyadh['trip_status']!='completed']['trip
```

In [29]:
```
df_surge_fare=df_riyadh.loc[df_riyadh['trip_status']=='completed'].groupby(['surged_trip'])['trip_fare_usd'].mea
```

In [30]:
```
df_surge_fare=df_surge_fare.reset_index()
```

In [31]:
```
df_surge_fare['surged_trip'] = df_surge_fare['surged_trip'].map({False: 'No Surge', True: 'Surge'})
df_surge_fare
```

Out[31]:

	surged_trip	trip_fare_usd
0	No Surge	5.531958
1	Surge	8.036503

Overview of the data:

1. The dataset provided gives information of 40k trips in the EMEA region – Riyadh city
2. Number of trips:
   - Completed: 28.5k (71.37%)
   - Trips cancelled (rider): 10.1k (25.43%)
   - Trips cancelled (driver): 1k (2.5%)
   - Trips unfulfilled: 234 (0.58%)

- Fare Split: 8 (0.02%)

3. Average fare/ride: $6(88\%paidincash), totalfare$:177k (completed rides)
4. Surged trips: 25% of the completed trips

Inference of the dataset

The below dashboard gives information on the Cancellation rate, Total fare, Average fare, Surged trip percentage. It is a dynamic tool, whenever the data changes output will also change accordingly.

To see the dynamic table, please click on the below link

In [32]:
```
data = {'Cancellation_rate' : Cancellation_rate, 'Total_Fare' :Total_Fare , 'Average_Fare' : Average_Fare, 'Surg
df=pd.DataFrame(data, index=[0])
```

In [33]:
```
app = Dash(__name__)
app.layout = dash_table.DataTable(
    id='table',
    columns=[{"name": i, "id": i} for i in df.columns],
    data=df.to_dict('records'),
)

if __name__ == '__main__':
    app.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
127.0.0.1 - - [19/Oct/2021 08:28:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:28:09] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:28:09] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:28:09] "GET /_favicon.ico?v=2.0.0 HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:28:09] "GET /_dash-component-suites/dash/dash_table/async-highlight.js HTTP/1.1"
200 -
127.0.0.1 - - [19/Oct/2021 08:28:09] "GET /_dash-component-suites/dash/dash_table/async-table.js HTTP/1.1" 200
-
```

a. Analysis based on Rider experience

The first analysis made is to check the Rider's experience based on given ETA and actual pick up time. From the table below it is observed that the time when the rider gets picked up is, in all cases, greater than the estimated pick up time.

Recommendation: Ensure that the ETA is monitored accurately so that the rider's expectation is set and the driver can try to meet it as well.

In [34]:
```
df_cr=df_riyadh[['distance_to_pickup','time_to_pickup','esttime_to_pickup','trip_status','pickup_local_time','dr
```

In [35]:
```
bins = [0, 1, 2, 3, 4, 5]
names = ['0-1', '1-2', '2-3', '3-4', '4-5','>5']
d = dict(enumerate(names, 1))

df_cr['Distance_pick_up_window'] = np.vectorize(d.get)(np.digitize(df_cr['distance_to_pickup'], bins))
```

In [36]:
```
df_cr=df_cr[['time_to_pickup','esttime_to_pickup','Distance_pick_up_window']]
df_cr=df_cr.loc[df_riyadh['trip_status']=='completed'].groupby('Distance_pick_up_window').mean()
```

In [37]:
```
df_cr['time_to_pickup']=df_cr['time_to_pickup']/60
df_cr['esttime_to_pickup']=df_cr['esttime_to_pickup']/60
```

In [38]:
```
df_cr.reset_index()
```

Out[38]:

	Distance_pick_up_window	time_to_pickup	esttime_to_pickup
0	0-1	7.967660	4.291548
1	1-2	10.879167	7.091667
2	2-3	13.107143	8.352381
3	3-4	18.472222	9.222222
4	4-5	41.666667	9.183333
5	>5	27.183333	7.003333

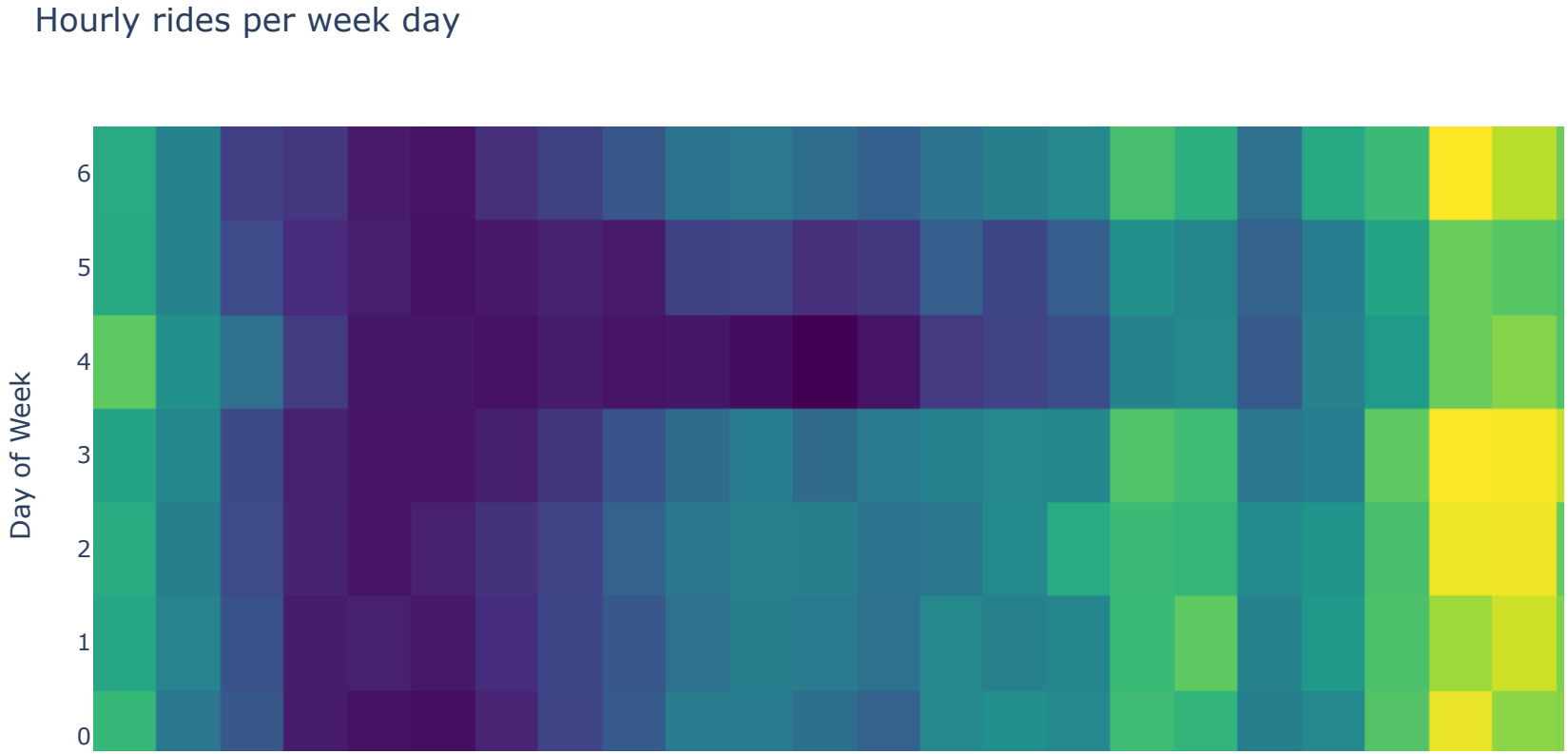
b. Analysis based on picked up rides by hours and days of the week

Here, the analysis is done based on the picked up rides by hour and days of the week. It is to be observed from the heatmap that there is a huge spike in rides between 8pm to 11pm almost everyday in the week.

In [39]:
```
df_riyadh['pickup_local_time'] = pd.to_datetime(df_riyadh['pickup_local_time'])
df_riyadh['pickup_local_hour'] = df_riyadh['pickup_local_time'].dt.hour
df_riyadh['pickup_local_day_of_week'] = df_riyadh['pickup_local_time'].dt.dayofweek
df_riyadh=df_riyadh.dropna()
```

In [40]:
```
df_hourly_rides_nt = df_riyadh[df_riyadh['trip_status']=='completed'].groupby(['pickup_local_hour', 'pickup_loca
df_hourly_rides_nt.columns = ['Hour', 'Day of Week', 'Count of Rides']
# df_hourly_rides['Day of Week']=df_hourly_rides['Day of Week'].astype(int)
```

In [41]:
```
fig = go.Figure(data=go.Heatmap(
        x=df_hourly_rides_nt['Hour'],
        y=df_hourly_rides_nt['Day of Week'],
        z=df_hourly_rides_nt['Count of Rides'],
        colorscale='Viridis'))

fig.update_layout(
    title='Hourly rides per week day',
    xaxis_nticks=36)

fig['layout']['xaxis']['title']='Hour of Day'
fig['layout']['yaxis']['title']='Day of Week'



fig.show()
```



Interactive tool - Picked up rides by hours and days of the week

To see the interactive visualization, please click on the below last link:

In [46]:
```
Hour = Dash(__name__)
```

In [47]:
```
Hour.layout = html.Div([
 html.H1("Hourly pick up (rides) with respect to Day of Week", style={'text-align': 'center'}),
 html.Div([
 html.Div([
 html.H4('Select_Day_of_Week'),
 dcc.Dropdown(
 id='DW',
 options=[{'label':'All','value': [1,2,3,4,5,6,0]},
    {'label':'Mon', 'value':1},
    {'label':'Tue','value':2},
    {'label':'Wed','value':3},
    {'label':'Thur','value':4},
    {'label':'Fri','value':5},
    {'label':'Sat','value':6},
    {'label':'Sun','value':0},
         ],
 value = [1,2,3,4,5,6,0]
 ),
 ],
 style={'width': '48%', 'display': 'inline-block'}),
 dcc.Graph(id='heatmap',
 figure = {'data': [go.Heatmap(
         z=df_hourly_rides_nt['Count of Rides'],
         x=df_hourly_rides_nt['Hour'],
         y=df_hourly_rides_nt['Day of Week'],
         name = 'first legend group',
         colorscale='Viridis')],
 'layout': go.Layout(
 title='Hourly rides per week day',
 xaxis_nticks=36,
 xaxis = dict(title = 'pickup_local_hour'),
 yaxis = dict( title = 'pickup_local_day_of_week'),
 )})
 ]),])
```

In [48]:
```
@Hour.callback(
    Output(component_id='heatmap', component_property='figure'),
    Input(component_id='DW',component_property='value')
)

def update_graph(Select_Day_of_Week):
    if Select_Day_of_Week in [1,2,3,4,5,6,0]:
        heatmap_data = df_hourly_rides_nt[(df_hourly_rides_nt['Day of Week'] == Select_Day_of_Week)][['Day of We
    else:
        heatmap_data=df_hourly_rides_nt[['Day of Week','Hour','Count of Rides']]
    return {
     'data': [go.Heatmap(
     z=heatmap_data['Count of Rides'],
     x=heatmap_data['Hour'],
     y=heatmap_data['Day of Week'],
     colorscale='Viridis')],
     'layout': go.Layout(
     title = 'Hourly Pick up by day of week',
     xaxis_nticks=36
     )
     }
```

In [49]:
```
if __name__ == '__main__':
    Hour.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
127.0.0.1 - - [19/Oct/2021 08:30:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "GET /_dash-component-suites/dash/dcc/async-dropdown.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "GET /_dash-component-suites/dash/dcc/async-graph.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:30:17] "GET /_dash-component-suites/dash/dcc/async-plotlyjs.js HTTP/1.1" 200 -
```

c. Cancellation Analysis on driver canceled rides during Surge

In [55]:
```
df_riyadh=pd.read_csv('riyadh_sample.csv')
df_driver_Ct=df_riyadh[['trip_status','surged_trip','trip_id']]
```

In [56]:
```
df_driver_Ct=df_driver_Ct.loc[df_driver_Ct['trip_status']=='driver_canceled'].groupby('surged_trip')['trip_id'].
```

In [57]:
```
df_driver_Ct=df_driver_Ct.reset_index()
```

In [58]:
```
df_driver_Ct['surged_trip'] = df_driver_Ct['surged_trip'].map({False: 'No Surge', True: 'Surge'})
df_driver_Ct
```

Out[58]:

	surged_trip	trip_id
0	No Surge	774
1	Surge	262

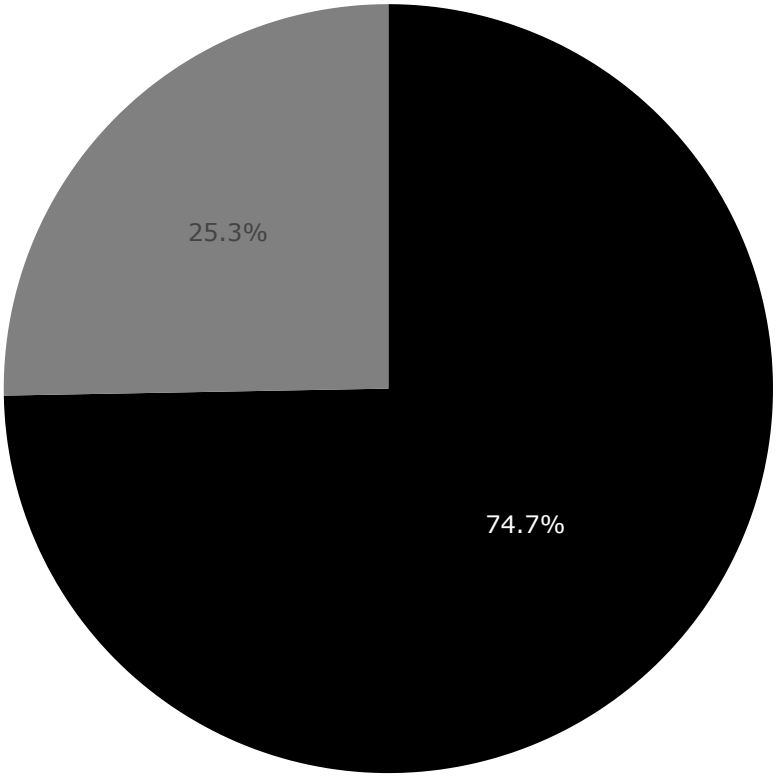
In [59]:
```
df_driver_Ct=df_driver_Ct.rename({'surged_trip':'surged_trip', 'trip_id': 'Driver_Cancelled_trip'}, axis=1)
df_driver_Ct['Percent_total_driver_cancelled_trips']=df_driver_Ct['Driver_Cancelled_trip']/df_riyadh.loc[df_riya
```

In [60]:
```
df_driver_Ct
```

Out[60]:

	surged_trip	Driver_Cancelled_trip	Percent_total_driver_cancelled_trips
0	No Surge	774	74.710425
1	Surge	262	25.289575

In [61]:
```
pct_cancelled_driver=list(df_driver_Ct['Percent_total_driver_cancelled_trips'])
names=list(df_driver_Ct['surged_trip'])
fig = px.pie(values=pct_cancelled_driver, names=names,color_discrete_sequence=["black", "gray"])
fig.show()
```



Driver cancellation during surge:

- Demand for rides increases -> Prices go up -> Riders pay more/wait

a. **Findings**: Based on the graph, 25% of the trips are getting canceled by the driver during surge

b. **Hypothesis**: Due to high surge pricing, the revenue lost for uber for cancellation of a ride during surge hours will be more than the revenue lost for cancellation of a ride during non-surge hours. If the cancellations can be further reduced to less than 25% then more revenue can be generated.

c. **Recommendation**: Advantages of surge pricing could be explained to drivers and need to ensure that the drivers don't cancel the rides during peak hours. Drivers could be incentivized if the trip is not getting canceled by a driver during surge hours.

d. Surge Analysis based on hour of the day

In [62]:
```
df_riyadh['request_local_time'] = pd.to_datetime(df_riyadh['request_local_time'])
df_riyadh['Hour_of_day']=df_riyadh['request_local_time'].dt.hour
```

In [63]:
```
df_riyadh_x=df_riyadh.groupby('Hour_of_day')['trip_id'].count()
```

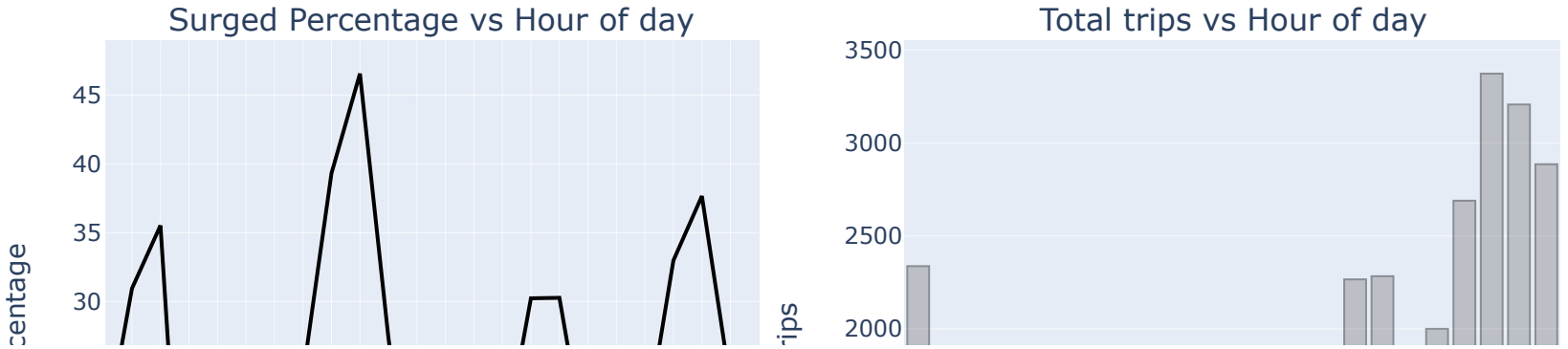
In [64]:
```
df_riyadh_x=df_riyadh_x.reset_index()
df_riyadh_x.columns=['Hour_of_day','Total_trip']
```

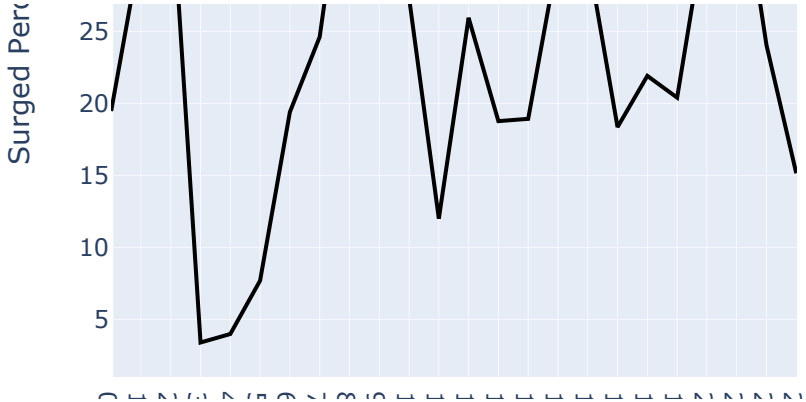
In [65]:
```
df_riyadh_y=df_riyadh.loc[df_riyadh['surged_trip'] == True].groupby('Hour_of_day')['surged_trip'].count()
df_riyadh_y=df_riyadh_y.reset_index()
df_riyadh_y.columns=['Hour_of_day','Total_surged_trip']
```

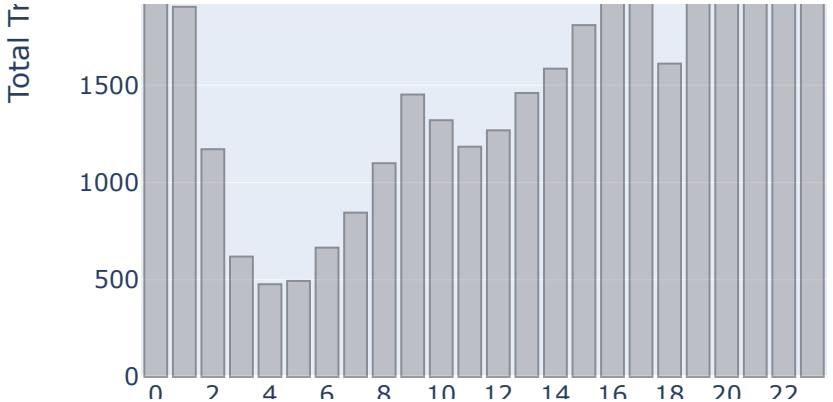
In [66]:
```
df_riyadh_result=pd.merge(df_riyadh_x, df_riyadh_y, on='Hour_of_day', how='outer')
```

In [67]:
```
df_riyadh_result['surged_pct']=(df_riyadh_result['Total_surged_trip']/df_riyadh_result['Total_trip'])*100
df_riyadh_result['Total_trip_pct']=(df_riyadh_result['Total_trip']/df_riyadh['trip_id'].count())*100
```

In [68]:
```
# set up plotly figure
fig = make_subplots(1,2,subplot_titles=('Surged Percentage vs Hour of day',  'Total trips vs Hour of day'))

# add first scatter trace at row = 1, col = 1
fig.add_trace(go.Scatter(x=df_riyadh_result['Hour_of_day'], y=df_riyadh_result['surged_pct'], line=dict(color='b
              row = 1, col = 1)

# add first bar trace at row = 1, col = 2
fig.add_trace(go.Bar(x=df_riyadh_result['Hour_of_day'], y=df_riyadh_result['Total_trip'],
                     name='Total_trip',
                     marker_color = 'gray',
                     opacity=0.4,
                     marker_line_color='black',
                    marker_line_width=1),
              row = 1, col = 2)

# edit axis labels
fig['layout']['xaxis']['title']='Hour of Day'
fig['layout']['xaxis2']['title']='Hour of Day'
fig['layout']['yaxis']['title']='Surged Percentage'
fig['layout']['yaxis2']['title']='Total Trips'
fig['layout']['xaxis_nticks']=24
fig['layout']['xaxis2_nticks']=24

fig.show()
```







How does surge works?

- Demand for rides increases -> Prices go up -> Riders pay more/wait

a. **Hypothesis**: Based on the graph, let's assume that the average surge should be at 20%

b. **Findings**: There is a huge spike in rides at 9 AM (28%) and another at 9 PM (17%) – local time considered

c. **Conclusion**: Uber should ensure that during these times, request should be fulfilled to provide a better rider experience

d. **Recommendation**: Advantages of surge pricing could be explained to drivers and need to ensure that the drivers don't cancel the rides during peak hours. Areas that are busiest during peak hours could be allocated with more drivers to ensure that requests are fulfilled

Interactive tool - Surge Analysis

To see the interactive visualization, please click on the below last link:

In [74]:
```
Surge_price = Dash(__name__)
```

In [75]:
```
# set up plotly figure
fig = make_subplots(1,1,subplot_titles='Surged Percentage vs Hour of day')

# add first scatter trace at row = 1, col = 1
fig.add_trace(go.Scatter(x=df_riyadh_result['Hour_of_day'], y=df_riyadh_result['surged_pct'], line=dict(color='b
              row = 1, col = 1)

# add first bar trace at row = 1, col = 2

fig2=make_subplots(1,1,subplot_titles='Total trips vs Hour of day')
fig2.add_trace(go.Bar(x=df_riyadh_result['Hour_of_day'], y=df_riyadh_result['Total_trip'],
                      name='Total_trip',
                      marker_color = 'gray',
                      opacity=0.4,
                      marker_line_color='black',
                     marker_line_width=1),
              row = 1, col = 1)

# edit axis labels
fig['layout']['xaxis']['title']='Hour of Day'
fig2['layout']['xaxis']['title']='Hour of Day'
fig['layout']['yaxis']['title']='Surged Percentage'
fig2['layout']['yaxis']['title']='Total Trips'
```

In [76]:
```
# ------------------------------------------------------------------------
# App Layout
Surge_price.layout = html.Div([

    html.H1("Surge Analysis by Hour", style={'text-align': 'center'}),

    dcc.Dropdown(id="slct_hour",
                 options=[
                     {"label": "All", "value": df_riyadh_result['Hour_of_day']},
                     {'label': "0" , "value":0},
                     {'label': "1" , "value":1},
                     {"label": "2", "value": 2},
                     {"label": "3", "value": 3},
                     {"label": "4", "value": 4},
                     {"label": "5", "value": 5},
                     {"label": "6", "value": 6},
                     {"label": "7", "value": 7},
                     {"label": "8", "value": 8},
                     {"label": "9", "value": 9},
                     {"label": "10", "value": 10},
                    {"label": "11", "value": 11},
                 {"label": "12", "value": 12},
                 {'label': "13" , "value":13},
                 {'label': "14" , "value":14},
                 {'label': "15" , "value":15},
                 {'label': "16" , "value":16},
                 {'label': "17" , "value":17},
                 {'label': "18" , "value":18},
                 {'label': "19" , "value":19},
                 {'label': "20" , "value":20},
                 {'label': "21" , "value":21},
                 {'label': "22" , "value":22},
                 {'label': "23" , "value":23}],
                 multi=False,
                 value=df_riyadh_result['Hour_of_day'],
                 style={'width': "30%"}
                 ),

    html.Div(id='output_container', children=[]),
    html.Br(),

    dcc.Graph(id='surge1', figure={}),
```

```
])
```

In [77]:
```
# Connect the Plotly graphs with Dash Components
@Surge_price.callback(
    [Output(component_id='output_container', component_property='children'),
     Output(component_id='surge1', component_property='figure')],
    [Input(component_id='slct_hour', component_property='value')])

def update_graph(option_hour):
    if option_hour in df_riyadh_result["Hour_of_day"].unique():
        container = "The hour chosen by user was: {}".format(option_hour)

        dff = df_riyadh_result.copy()
        dff = dff[dff["Hour_of_day"] == option_hour]


            # set up plotly figure
        fig = make_subplots(1,2,subplot_titles=('Surged Percentage vs Hour of day',  'Total trips vs Hour of day

        # add first scatter trace at row = 1, col = 1
        fig.add_trace(go.Scatter(x=dff['Hour_of_day'], y=dff['surged_pct'], line=dict(color='black'), name='Surg
                      row = 1, col = 1)

        # add first bar trace at row = 1, col = 2
        fig.add_trace(go.Bar(x=dff['Hour_of_day'], y=dff['Total_trip'],
                             name='Total_trip',
                             marker_color = 'gray',
                             opacity=0.4,
                             marker_line_color='black',
                            marker_line_width=2),
                      row = 1, col = 2)

        # edit axis labels
        fig['layout']['xaxis']['title']='Hour of Day'
        fig['layout']['xaxis2']['title']='Hour of Day'
        fig['layout']['yaxis']['title']='Surged Percentage'
        fig['layout']['yaxis2']['title']='Total Trips'
        fig['layout']['xaxis_nticks']=24
        fig['layout']['xaxis2_nticks']=24
    else:
        dff = df_riyadh_result.copy()
        dff = dff[dff["Hour_of_day"] == option_hour]
```

```
        # set up plotly figure
        fig = make_subplots(1,2,subplot_titles=('Surged Percentage vs Hour of day',  'Total trips vs Hour of day

        # add first scatter trace at row = 1, col = 1
        fig.add_trace(go.Scatter(x=dff['Hour_of_day'], y=dff['surged_pct'], line=dict(color='red'), name='Surged
                      row = 1, col = 1)

        # add first bar trace at row = 1, col = 2
        fig.add_trace(go.Bar(x=dff['Hour_of_day'], y=dff['Total_trip'],
                             name='Total_trip',
                             marker_color = 'green',
                             opacity=0.4,
                             marker_line_color='rgb(8,48,107)',
                            marker_line_width=2),
                      row = 1, col = 2)

        # edit axis labels
        fig['layout']['xaxis']['title']='Hour of Day'
        fig['layout']['xaxis2']['title']='Hour of Day'
        fig['layout']['yaxis']['title']='Surged Percentage'
        fig['layout']['yaxis2']['title']='Total Trips'
        fig['layout']['xaxis_nticks']=24
        fig['layout']['xaxis2_nticks']=24


    return container, fig
```

In [78]:
```
if __name__ == '__main__':
    Surge_price.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
127.0.0.1 - - [19/Oct/2021 08:31:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "GET /_dash-component-suites/dash/dcc/async-dropdown.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "GET /_dash-component-suites/dash/dcc/async-graph.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "GET /_dash-component-suites/dash/dcc/async-plotlyjs.js HTTP/1.1" 200 -
127.0.0.1 - - [19/Oct/2021 08:31:48] "POST /_dash-update-component HTTP/1.1" 200 -
```

e. Monthly analysis of completed and surged trips

Monthly analysis of the completed trips and surged trips is done to give a comparison of the trips based on months. The data has three months: May, June and July. However, the code has been integrated to include all the months to include the data on an ongoing basis.

In [92]:
```
Uber_Dashboard = Dash(__name__)
```

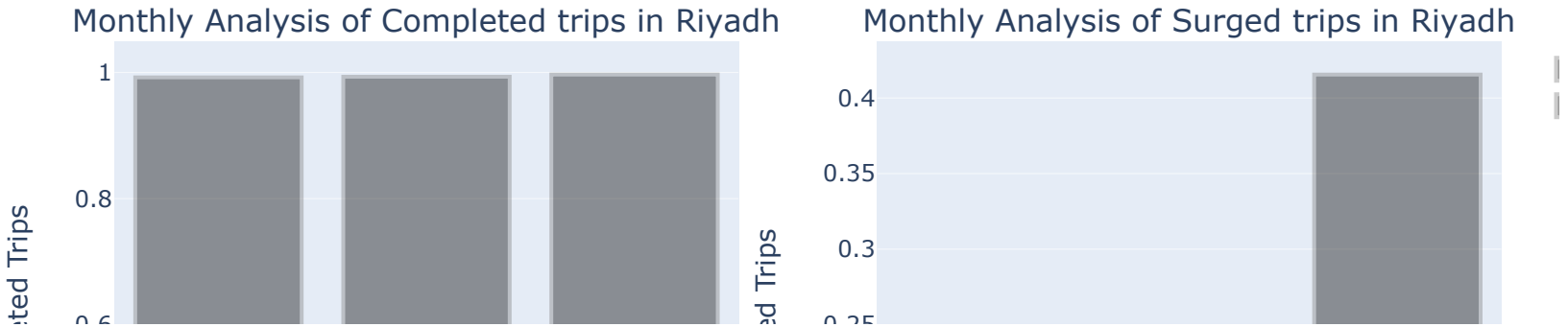
In [93]:
```
df_riyadh['pickup_local_time'] = pd.to_datetime(df_riyadh['pickup_local_time'])
df_riyadh['Month']=df_riyadh['pickup_local_time'].dt.month
df_riyadh_monthwise = df_riyadh.groupby(['Month'])[['completed_trip','surged_trip',]].mean()
df_riyadh_monthwise.reset_index(inplace=True)
df_riyadh_monthwise['Month'] = df_riyadh_monthwise['Month'].map({1:'Jan',2:'Feb',3:'Mar',4:'Apr',5: 'May',6: 'Ju
```

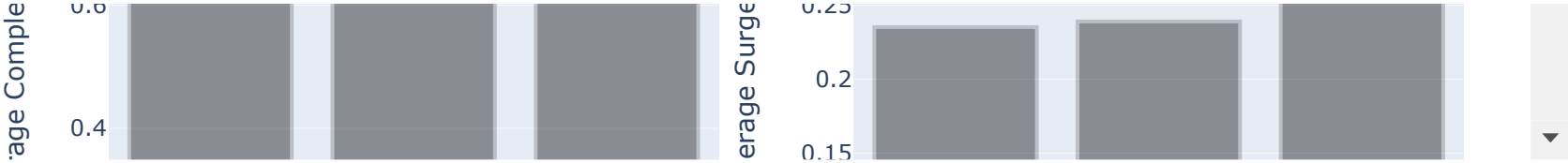
In [94]:
```
df_riyadh_monthwise
```

Out[94]:

	Month	completed_trip	surged_trip
0	May	0.992334	0.235313
1	Jun	0.993353	0.239085
2	Jul	0.996779	0.415459

In [95]:
```
# set up plotly figure
fig = make_subplots(1,2,subplot_titles=["Monthly Analysis of Completed trips in Riyadh","Monthly Analysis of Sur

# add first scatter trace at row = 1, col = 1
fig.add_trace(go.Bar(x=df_riyadh_monthwise['Month'], y=df_riyadh_monthwise['completed_trip'],
                     name='Completed_trip',
                     marker_color = 'black',
                     opacity=0.4,
                     marker_line_color='gray',
                    marker_line_width=2),
              row = 1, col = 1)


fig.add_trace(go.Bar(x=df_riyadh_monthwise['Month'], y=df_riyadh_monthwise['surged_trip'],
                     name='Surged_trip',
                     marker_color = 'black',
                     opacity=0.4,
                     marker_line_color='gray',
                    marker_line_width=2),
              row = 1, col = 2)
# edit axis labels
fig['layout']['xaxis']['title']='Months'
fig['layout']['yaxis']['title']='Average Completed Trips'
fig['layout']['xaxis2']['title']='Months'
fig['layout']['yaxis2']['title']='Average Surged Trips'
fig.show()
```





Interactive tool - Monthly analysis of completed and surged trips

To see the interactive visualization, please click on the below last link:

In [96]:
```
# ----------------------------------------------------------------------------
# App layout
Uber_Dashboard.layout = html.Div([

    html.H1("Monthwise Analysis of trips in Riyadh", style={'text-align': 'center'}),

    dcc.Dropdown(id="slct_month",
                 options=[
                     {"label": "All", "value": ['May','Jun','Jul']},
                     {"label": "Jan", "value": 'Jan'},
                     {"label": "Feb", "value": 'Feb'},
                     {"label": "Mar", "value": 'Mar'},
                     {"label": "Apr", "value": 'Apr'},
                     {"label": "May", "value": 'May'},
                     {"label": "Jun", "value": 'Jun'},
                     {"label": "Jul", "value": 'Jul'},
                     {"label": "Aug", "value": 'Aug'},
                     {"label": "Sept", "value": 'Sept'},
                     {"label": "Oct", "value": 'Oct'},
                    {"label": "Nov", "value": 'Nov'},
                 {"label": "Dec", "value": 'Dec'}],
                 multi=False,
                 value=['May','Jun','Jul'],
                 style={'width': "30%"}
                 ),

    html.Div(id='output_container', children=[]),
    html.Br(),

    dcc.Graph(id='monthwise_graph', figure={})

])
```

In [97]:
```
# ------------------------------------------------------------------------
# Connect the Plotly graphs with Dash Components
@Uber_Dashboard.callback(
    [Output(component_id='output_container', component_property='children'),
     Output(component_id='monthwise_graph', component_property='figure')],
    [Input(component_id='slct_month', component_property='value')]
)
def update_graph(option_slctd):
    print(option_slctd)
    print(type(option_slctd))

    container = "The month chosen by user was: {}".format(option_slctd)

    dff = df_riyadh_monthwise.copy()
    dff = dff[dff["Month"] == option_slctd]


        # Plotly Express
    # set up plotly figure
    fig = make_subplots(1,2,subplot_titles=["Monthwise Analysis of Completed trips in Riyadh","Monthwise Analysi

    # add first scatter trace at row = 1, col = 1
    fig.add_trace(go.Bar(x=dff['Month'], y=dff['completed_trip'],
                         name='Completed_trip',
                         marker_color = 'black',
                         opacity=0.4,
                         marker_line_color='gray'),
                  row = 1, col = 1)


    fig.add_trace(go.Bar(x=dff['Month'], y=dff['surged_trip'],
                         name='Surged_trip',
                         marker_color = 'black',
                         opacity=0.4,
                         marker_line_color='gray'),
                  row = 1, col = 2)
    # edit axis labels
    fig['layout']['xaxis']['title']='Months'
    fig['layout']['yaxis']['title']='Average Completed Trips'
    fig['layout']['xaxis2']['title']='Months'
    fig['layout']['yaxis2']['title']='Average Surged Trips'
```

```
    return container, fig
```

In [98]:
```
if __name__ == '__main__':
    Uber_Dashboard.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
127.0.0.1 - - [19/Oct/2021 08:34:10] "GET / HTTP/1.1" 200 -
```

f. Analysis based on demand and supply disparity - Vehicle type

a. **Findings**: Based on table, the gap in number of Black Car requested vs the number of Black Car being the vehicle type is more than 50%. Requested #: 8692 and Vehicle Type #: 94. The demand is not met here and instead UberX is sent for the request of Black Car. From Second table (refer the table in next analysis), it is seen that Average fare/trip of a Black Car is 3 times more than the average fare/trip of UberX

b. **Assumption**: If the rider is requesting an Uber black but getting an UberX then then the assumption is being made that rider is being charged for UberX

c. **Analysis**: The revenue can be increased by supplying Black Car, whenever requested, at least by 50% (4.5k), leading to additional revenue of $38k (22% additional revenue). Due to the disparity in vehicle type, not only Uber is bearing the losses but the rider is also dissatisfied, leading to decrease in Uber's brand value.

d. **Recommendation**: Uber could focus on increasing the number of Black Cars to make sure that whenever there is a demand for Black Car, it is met.

Library to install to run the below interactive table: pivottablejs

PivotTable.js is a Javascript Pivot Table and Pivot Chart library with drag'n'drop interactivity, and it can now be used with Jupyter/IPython Notebook via the pivottablejs module.

In [99]:
```
df_riyadh=pd.read_csv('riyadh_sample.csv')
```

In [100]:
```
df_combined=df_riyadh.groupby(['request_type','vehicle_type','trip_status']).sum()[['city_id','trip_fare_usd']]
```

In [101]:
```
!pip install pivottablejs
```

```
Requirement already satisfied: pivottablejs in c:\programdata\anaconda3\lib\site-packages (0.9.0)
```

In [102]:
```
from pivottablejs import pivot_ui
```

In [103]:
```
pivot_ui(df_combined)
```

Out[103]:

[pop out]

Table

trip_status | city_id | trip_fare_usd

Sum

city_id

vehicle_type

request_type

request_type	vehicle_type	Black Car	UberVIP	UberX	null	uberXL	Totals
					0.00		**0.00**
Black		94.00	17.00	8,501.00		80.00	**8,692.00**
UberX		577.00	241.00	27,272.00		1,514.00	**29,604.00**
	Totals	**671.00**	**258.00**	**35,773.00**	**0.00**	**1,594.00**	**38,296.00**

g. Analysis of Average Revenue generated based on Type of Vehicle

In [104]:
```
df_revenue_vt=df_riyadh.groupby('vehicle_type')['trip_fare_usd'].sum()
```

In [105]:
```
df_revenue_vt=df_revenue_vt.reset_index()
```

In [106]:
```
df_trip_vehicle=df_riyadh.loc[df_riyadh['completed_trip']==True].groupby('vehicle_type')['trip_id'].count()
```

In [107]:
```
df_trip_vehicle=df_trip_vehicle.reset_index()
```

In [108]:
```
df_trip_vehicle=df_trip_vehicle.rename({'vehicle_type': 'vehicle_type', 'trip_id': 'Total trips'}, axis=1)
```

In [109]:
```
df_rev_result=pd.merge(df_revenue_vt, df_trip_vehicle, on='vehicle_type', how='outer')
```

In [110]:
```
df_rev_result['Avg revenue per vehicle type']=df_rev_result['trip_fare_usd']/df_rev_result['Total trips']
df_rev_result
```

Out[110]:

	vehicle_type	trip_fare_usd	Total trips	Avg revenue per vehicle type
0	Black Car	4427.328621	302	14.660029
1	UberVIP	762.200990	141	5.405681
2	UberX	160754.736516	26987	5.956747
3	uberXL	10562.646521	949	11.130291

h. Analysis of cancellation rate vs type of vehicle

In [111]:
```
df_rider_Ct=df_riyadh[['trip_status','surged_trip','trip_id','vehicle_type']]
df_rider_Ct=df_rider_Ct.loc[df_rider_Ct['trip_status']!='completed'].groupby(['vehicle_type'])['trip_id'].count(
df_rider_Ct=df_rider_Ct.reset_index()
#df_rider_Ct['surged_trip'] = df_rider_Ct['surged_trip'].map({False: 'No Surge', True: 'Surge'})
df_rider_Ct=df_rider_Ct.rename({'ehicle_type':'ehicle_type', 'trip_id': 'Total_Cancelled_trip'}, axis=1)
# df_rider_Ct=df_rider_Ct.loc[df_rider_Ct['surged_trip']=='Surge']
# df_rider_Ct=df_rider_Ct[['vehicle_type','Total_Cancelled_trip']]
# # df_rider_Ct['Percent_total_rider_cancelled_trips']=df_rider_Ct['Rider_Cancelled_trip']/df_riyadh.loc[df_riya
# # # df_rider_Ct
```

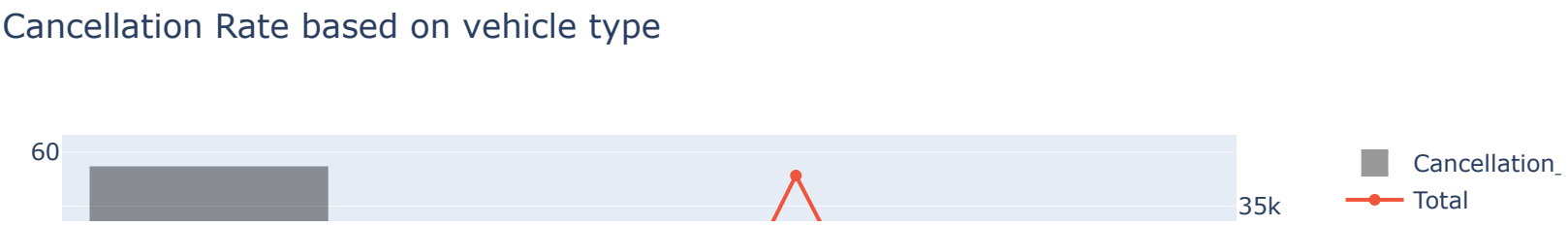
In [112]:
```
df_vehicle_total=df_riyadh.groupby('vehicle_type')['trip_id'].count()
df_vehicle_total=df_vehicle_total.reset_index()
df_vehicle_total=df_vehicle_total.rename({'vehicle_type': 'vehicle_type', 'trip_id': 'Total'}, axis=1)
```

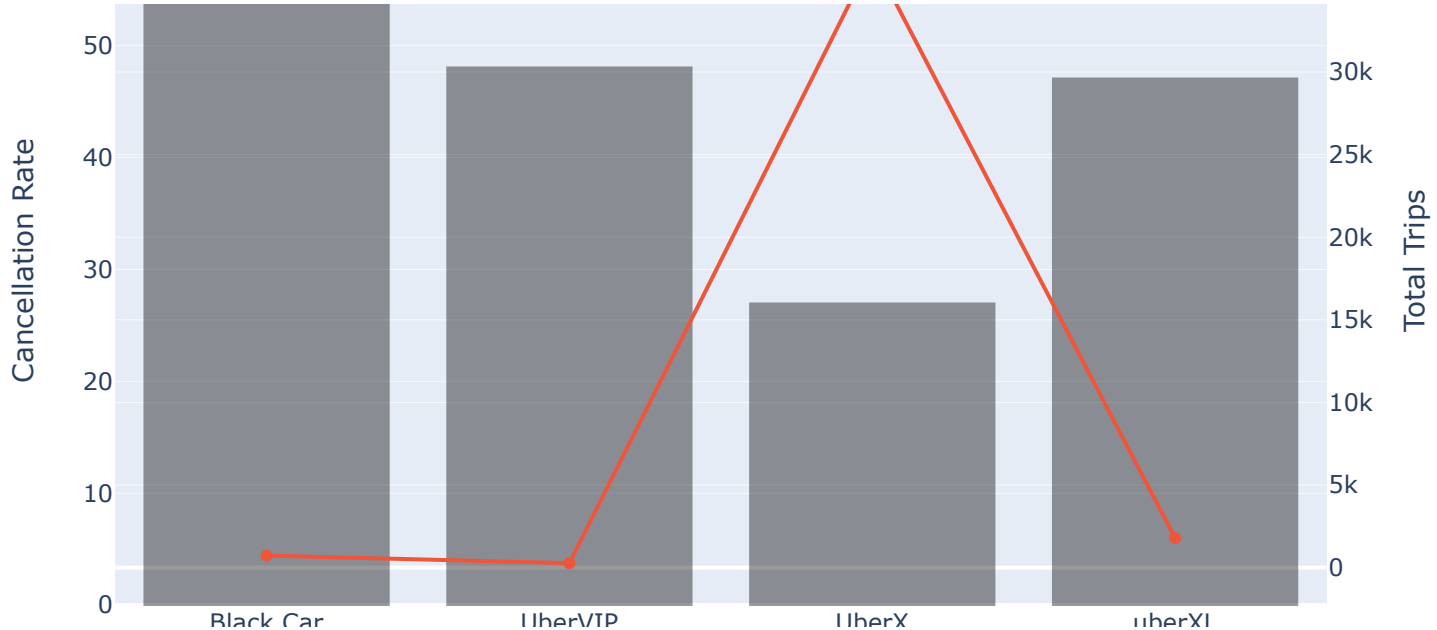
In [113]:
```
df_vehicle=pd.merge(df_rider_Ct,df_vehicle_total, on='vehicle_type', how='outer')
df_vehicle['Cancellation_rate_vehicle_type']=df_vehicle["Total_Cancelled_trip"]/df_vehicle["Total"]*100
df_vehicle
```

Out[113]:

	vehicle_type	Total_Cancelled_trip	Total	Cancellation_rate_vehicle_type
0	Black Car	426	728	58.516484
1	UberVIP	130	271	47.970480
2	UberX	9917	36904	26.872426
3	uberXL	841	1790	46.983240

In [114]:
```
# Create figure with secondary y-axis
fig = make_subplots(specs=[[{"secondary_y": True}]])

# Add traces
fig.add_trace(go.Bar(x=df_vehicle['vehicle_type'], y=df_vehicle['Cancellation_rate_vehicle_type'],
                     name='Cancellation_rate_vehicle_type',
                     marker_color = 'black',
                     opacity=0.4,
                     marker_line_color='black',
                    marker_line_width=2),
              secondary_y=False)



fig.add_trace(
    go.Scatter(x=df_vehicle['vehicle_type'], y=df_vehicle['Total'], name="Total"),
    secondary_y=True,
)

# Add figure title
fig.update_layout(
    title_text="Cancellation Rate based on vehicle type"
)

# Set x-axis title
fig.update_xaxes(title_text="Vehicle Types")

# Set y-axes titles
fig.update_yaxes(title_text=" Cancellation Rate", secondary_y=False)
fig.update_yaxes(title_text=" Total Trips", secondary_y=True)

fig.show()
```





a. **Finding**: Despite being the highest requested vehicle type, UberX is cancelled very less number of times compared to Black, VIP and XL.

b. **Hypothesis**: There could be a couple of reasons for the other vehicle types to get cancelled frequently:

1. Either the black cars are less in number, and multiple requests are going to them simultaneously, and the driver has to cancel some of the requests.
2. The black cars are not available in the near location and is taking longer wait times, and the rider itself is cancelling the ride.

c. **Recommendation**:

1. Addition to the fleet could minimize the cancellation of rides.
2. Better Allocation of other vehicle types with respect to location will lead to reduction in wait time for the rider

Secondary Analysis

Analysis of Cancellation Rate based on Rider and Driver

In [115]:
```
df_riyadh_wt=df_riyadh[['pickup_local_time','trip_status','request_local_time','driver_id','rider_id','distance_
```

In [116]:
```
df_riyadh_wt['request_local_time'] = pd.to_datetime(df_riyadh_wt['request_local_time'])
df_riyadh_wt['pickup_local_time'] = pd.to_datetime(df_riyadh_wt['pickup_local_time'])
```

In [117]:
```
df_riyadh_wt['wait_time_customer']=(df_riyadh_wt['pickup_local_time']-df_riyadh_wt['request_local_time']).dt.tot
```

In [118]:
```
bins = [0, 1, 2, 3, 4, 5]
names = ['0-1', '1-2', '2-3', '3-4', '4-5','>5']
d = dict(enumerate(names, 1))

df_riyadh_wt['Distance_pick_up_window'] = np.vectorize(d.get)(np.digitize(df_riyadh_wt['distance_to_pickup'], bi
```

In [119]:
```
df_riyadh_tt=df_riyadh_wt.loc[df_riyadh_wt['trip_status']=='driver_canceled'].groupby('Distance_pick_up_window')
```

In [120]:
```
df_riyadh_tt=df_riyadh_tt.reset_index()
df_riyadh_tt.columns=['Distance_pick_up_window','Trip_canceled_by_Driver']
```

In [121]:
```
df_riyadh_tt['driver_Cancellation_rate']=df_riyadh_tt['Trip_canceled_by_Driver']/df_riyadh_wt.loc[df_riyadh_wt['
```

In [122]:
```
df_riyadh_ct=df_riyadh[['pickup_local_time','trip_status','request_local_time','driver_id','rider_id','distance_
```

In [123]:
```
bins = [0, 1, 2, 3, 4, 5]
names = ['0-1', '1-2', '2-3', '3-4', '4-5','>5']
d = dict(enumerate(names, 1))

df_riyadh_ct['Distance_pick_up_window'] = np.vectorize(d.get)(np.digitize(df_riyadh_ct['distance_to_pickup'], bi
```

In [124]:
```
df_riyadh_ctc=df_riyadh_ct.loc[df_riyadh_ct['trip_status']=='rider_canceled'].groupby('Distance_pick_up_window')
```

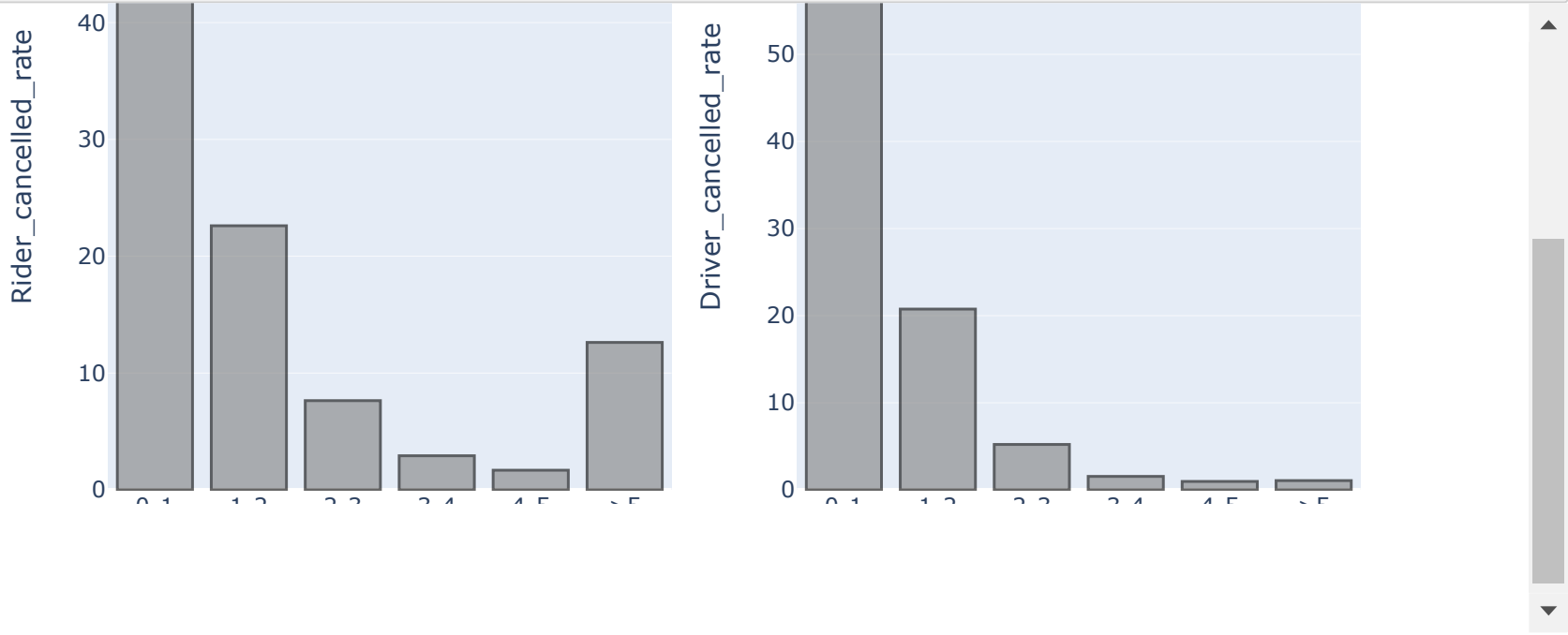
In [125]:
```
df_riyadh_ctc
```

Out[125]:
```
Distance_pick_up_window
0-1    5345
1-2    2300
2-3     777
3-4     297
4-5     171
>5     1284
Name: trip_status, dtype: int64
```

In [126]:
```
df_riyadh_ctc=df_riyadh_ctc.reset_index()
df_riyadh_ctc.columns=['Distance_pick_up_window','Trip_canceled_by_Rider']
```

In [127]:
```
df_riyadh_ctc['rider_Cancellation_rate']=df_riyadh_ctc['Trip_canceled_by_Rider']/df_riyadh_ct.loc[df_riyadh_wt['
```

In [128]:
```
fig = make_subplots(rows=1, cols=2,
                    subplot_titles=("Rides Canceled by rider","Rides Canceled by driver"))
fig.add_trace(go.Bar(x=df_riyadh_ctc['Distance_pick_up_window'],y=df_riyadh_ctc['rider_Cancellation_rate'], name
fig.update_traces(marker_color='black', marker_line_color='gray',
                  marker_line_width=1.5, opacity=0.6)

fig.add_trace(go.Bar(x=df_riyadh_tt['Distance_pick_up_window'],y=df_riyadh_tt['driver_Cancellation_rate'], name=
fig.update_traces(marker_color='gray', marker_line_color='black',
                  marker_line_width=1.5, opacity=0.6)
# edit axis labels
fig['layout']['xaxis']['title']='Distance_pickup_Window'
fig['layout']['yaxis']['title']='Rider_cancelled_rate'
fig['layout']['xaxis2']['title']='Distance_pickup_Window'
fig['layout']['yaxis2']['title']='Driver_cancelled_rate'
fig.show()
```



a. **Finding**: Here, the rides are getting cancelled within 0-1 mile distance to pickup window by both the riders and drivers.

b. **Analysis**: Various hypothesis can be built based on the time taken by the rider to cancel after a ride is being booked, if more data was provided.

c. **Recommendation** could be to provide a feedback form with some predefined options like:

1. Booked accidentally
2. Cancelled accidentally
3. Changed my mind
4. Got better deals in other ride-hailing apps/services.. etc

Facts based on pickup time vs estimate time to pickup

In [129]:
```
df_riyadh_wait_time=df_riyadh[['esttime_to_pickup','trip_status','distance_to_pickup']]
```

In [130]:
```
bins = [0, 1, 2, 3, 4, 5]
names = ['0-1', '1-2', '2-3', '3-4', '4-5','>5']
d = dict(enumerate(names, 1))

df_riyadh_wait_time['Distance_pick_up_window'] = np.vectorize(d.get)(np.digitize(df_riyadh_wait_time['distance_t
```

In [131]:
```
df_riyadh_wait_time=df_riyadh_wait_time.loc[df_riyadh['trip_status']=='rider_canceled'].groupby('Distance_pick_u
```

In [132]:
```
df_riyadh_wait_time['esttime_to_pickup']=df_riyadh_wait_time['esttime_to_pickup']/60
```

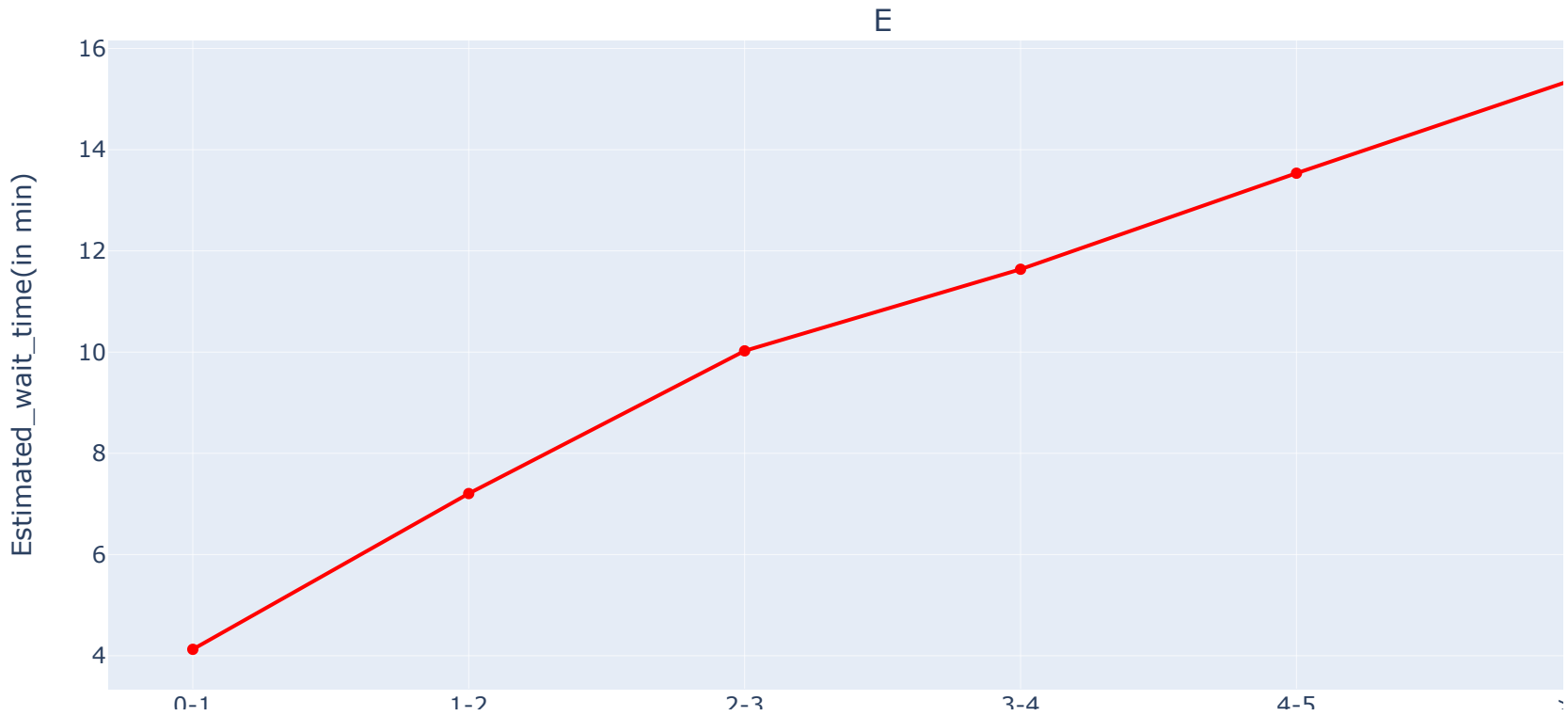
In [133]:
```
df_riyadh_wait_time=df_riyadh_wait_time.reset_index()
```

In [134]:
```
df_riyadh_wait_time=df_riyadh_wait_time[['Distance_pick_up_window','esttime_to_pickup']]
df_riyadh_wait_time
```

Out[134]:

	Distance_pick_up_window	esttime_to_pickup
0	0-1	4.126107
1	1-2	7.204435
2	2-3	10.024603
3	3-4	11.637093
4	4-5	13.535185
5	>5	15.377690

In [135]:
```
# set up plotly figure
fig = make_subplots(1,1,subplot_titles="Estimated time to pick up with distance")

# add first scatter trace at row = 1, col = 1
fig.add_trace(go.Scatter(x=df_riyadh_wait_time['Distance_pick_up_window'], y=df_riyadh_wait_time['esttime_to_pic
              row = 1, col = 1)

# edit axis labels
fig['layout']['xaxis']['title']='Distance_pickup_Window'
fig['layout']['yaxis']['title']='Estimated_wait_time(in min)'
fig.show()
```



Analysis based on Mode of payment

In [136]:
```
df_mode_payment=df_riyadh[['trip_id','paid_cash']]
```

In [137]:
```
df_mode_payment=df_mode_payment.groupby('paid_cash').count()
```

In [138]:
```
df_mode_payment=df_mode_payment.reset_index()
```

In [139]:
```
df_mode_payment['paid_cash'] = df_mode_payment['paid_cash'].map({False: 'No cash', True: 'cash'})
df_mode_payment=df_mode_payment.rename({'paid_cash': 'paid_cash', 'trip_id': 'Total'}, axis=1)
```

In [140]:
```
df_mode_payment['Total_pct']=df_mode_payment['Total']/df_riyadh['trip_id'].count()*100
```

In [141]:
```
df_mode_payment
```

Out[141]:

	paid_cash	Total	Total_pct
0	No cash	6424	16.06
1	cash	33576	83.94

Riders in Riyadh are paying in either cash or via apps, credit/ debit cards, coupons.

a. **Findings**: Based on data, 84% of the rides are paid by riders in cash

b. **Recommendation**: Uber can build better relations with Riyadh, as Saudi Arabia is steadily moving towards building a Cashless Society by 2030, by encouraging riders to pay in cash via promotions/ offers on Uber. This can be achieved by establishing relations with companies like MasterCard, Visa, PayPal, or any local financial services.