UBER_Data_Wrangling

January 10, 2020

##

Data Exploration and Wrangling For UBER ride sharing data

Description:

Here we are working with the UBER ride sharing datset of Victoria, Australia. We will load the given datasets:

dirty_data.csv : We will try to do some EDA and find different types of errors (Syntactic/Semantic)

outliers.csv : In this file we will just try to analyse and find outliers(if any) in all the dimensions

missing_value.csv : In this file we will just try to find missing values(if any).

edges.csv : This file we will use to find the journey distance between different nodes and travel time.

nodes.csv : This file we will use along with edges.csv for finding journey distance and travel time .

0.1 Step 1: Import Libraries

```
In [1]: # !pip install folium
        import pandas as pd
        import folium as fol
        import warnings
        warnings.filterwarnings('ignore')
```

0.2 Step 2: Reading Data files

```
In [2]: dirty_df = pd.read_csv("data/dirty_data.csv")
        miss_df = pd.read_csv("data/missing_value.csv")
        outlier_df = pd.read_csv("data/outliers.csv")
        edges_df = pd.read_csv("data/edges.csv")
        nodes_df = pd.read_csv("data/nodes.csv")
```

0.2.1 2.1 Understanding the dirty data and the dimensions

```
In [3]: dirty_df.head(5)
```

```
Out[3]:      Unnamed: 0  Uber Type  Origin Region  Destination Region  \
        0  ID3501840116          1              8                   1
        1  ID1393121718          0              5                   7
        2  ID1314878251          0              6                   3
```

```
3  ID1404486339          0           7                   8
4  ID5960060344          2           5                   6

   Origin Latitude  Origin Longitude  Destination Latitude  \
0       -37.815834        145.046450            -37.800304
1       -37.799614        144.932772            -37.861835
2       -37.790797        144.985865            -37.811819
3       -37.861835        144.905716            -37.815834
4       -37.809426        144.928865            -37.787442

   Destination Longitude  Journey Distance(m) Departure Date Departure Time  \
0             144.971834               7448.0     2018-02-01       01:17:26
1             144.905716              12392.0     2018-06-22       14:09:42
2             144.975182               2988.0     2018-07-06       08:23:47
3             145.046450              15150.2     2018-03-05       12:20:20
4             144.980409               6687.7     2018-04-26       12:20:43

   Travel Time(s) Arrival Time   Fare$
0         2156.82      1:53:22   23.46
1         3896.52     15:14:38   11.38
2          852.78      8:37:59    2.82
3         4911.60     13:42:11   14.80
4         1728.36     12:49:31  136.46
```

In [4]:
```
dirty_df.describe()
```

Out[4]:
```
        Uber Type  Origin Region  Destination Region  Origin Latitude  \
count  280.000000     280.000000          280.000000       280.000000
mean     0.821429       4.732143            4.964286       -36.495029
std      0.796658       2.564516            2.623059        10.039393
min      0.000000       1.000000            1.000000       -38.110916
25%      0.000000       3.000000            3.000000       -37.824031
50%      1.000000       4.500000            5.000000       -37.814843
75%      1.000000       7.000000            7.000000       -37.807044
max      3.000000       9.000000            9.000000        37.861835

       Origin Longitude  Destination Latitude  Destination Longitude  \
count        280.000000            280.000000             280.000000
mean         144.933967            -36.499018             144.932610
std            0.100735             10.038399               0.109074
min          144.654173            -38.110916             144.654173
25%          144.926658            -37.824434             144.928854
50%          144.959720            -37.814036             144.957599
75%          144.985865            -37.805628             144.992234
max          145.046450             37.816176             145.046450

       Journey Distance(m)  Travel Time(s)       Fare$
count           280.000000      280.000000  280.000000
```

```
mean          15757.016071     4107.506571   81.731607
std           16165.420364     3855.761494  167.726843
min             154.000000       38.460000    2.820000
25%            5520.750000     1426.905000   12.577500
50%            8596.500000     2555.070000   20.490000
75%           13986.000000     4293.870000   56.722500
max           51032.000000    13204.980000  857.050000
```

So, we have 280 records and there is no coverage errors.

Minimum fare is 81.73 AUD and maximum fare is 857.05 AUD (thats a really huge amount for a trip)

Journey Distance in meters is 154 meter as the minimum in records which is really a short distance for a trip.

Origin and Destination latitude maximum value is positive which is strange as latitude of all points in Victoria should be negative. We need to look into this.

Specification says Uber Type has 3 categories but we can observe its 0,1,2,3 in the data which is definitely not correct.

We have got 9 regions based on the data from where the rides were taken.

Also as per specification we need to add 2 extra dimensions which will tell us the day factor(i.e. weekday / weekend) and a time factor (i.e. morning, afternoon, night). These both dimensions can be derived from Departure Date and Departure Time respectively. So lets define a method for both, but will add these dimensions later once we validate Departure Date and Time's authenticity.

```
In [5]: # This method generates day factor based on Departure Date.
        # WEEKDAY: 0
        # WEEKEND: 1

        def day_classifier(df):
            days = ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]
            day_class = {"Monday":0,"Tuesday":0,"Wednesday":0,"Thursday":0,"Friday":0,"Saturda

            for i in range(0,len(df)):
                dt = df.loc[i,"Departure Date"].split("-")
                day = days[datetime(int(dt[0]),int(dt[1]),int(dt[2])).weekday()]
                df.loc[i,'day_factor'] = day_class[day]
```

```
In [6]: # This method generates time factor based on Departure time.
        # MORNING (i.e. 0) (6:00:00 - 11:59:59),
        # AFTERNOON (i.e. 1) (12:00:00 - 20:59:59), and
        # NIGHT (i.e. 2) (21:00 - 5:59:59)

        def time_classifier(df):
            for i in range(0, len(df)):
                if (datetime.strptime(str(df.loc[i]["Departure Time"]), FMT).hour >= 6) and (d
                    df.loc[i,'time_factor'] = 0
                elif (datetime.strptime(str(df.loc[i]["Departure Time"]), FMT).hour >= 12) and
                    df.loc[i,'time_factor'] = 1
```

```
            else:
                df.loc[i,'time_factor'] = 2
```

Now, lets try to see our different locations for the ride on the map and do other exploratory analysis to understand the data better and find out errors & inconsistencies (if any) .

0.3 Step 3: Exploratory Data Analysis on Dirty Data

Before beginning with EDA let us add an additional column "corrected" in our dataset which will keep a track if the record has been updated or corrected. It will act as a flag to keep a track which all records are been corrected while cleaning the dataset. Corrected : 1 (it has been corrected) or 0 (not corrected or no error in that whole row)

```
In [7]: dirty_df["corrected"] = 0 # by default its 0
```

0.3.1 3.1: UBER Ride points across VIC (Origin Region)

The markers representing the points are varying in color based on their region alloted to them.
Region 1 = "red"
Region 2 = "lightgray"
Region 3 = "blue"
Region 4 = "green"
Region 5 = "orange"
Region 6 = "black"
Region 7 = "darkpurple"
Region 8 = "beige"
Region 9 = "pink"

```
In [8]: # CREATING AN EMPTY MAP
        o_map = fol.Map(location=[-38, 144], tiles="Mapbox Bright", zoom_start=8)

        # METHOD DEFINES THE COLOR FOR THE MARKER BASED ON REGION
        def color_icon(region):
            if region == 1:
                color = "red"
            elif region ==2:
                color = "lightgray"
            elif region ==3:
                color = "blue"
            elif region ==4:
                color = "green"
            elif region ==5:
                color = "orange"
            elif region ==6:
                color = "black"
            elif region ==7:
                color = "darkpurple"
            elif region ==8:
                color = "beige"
```

```
        else:
            color = "pink"
        return color

    # ADDING MARKERS ON THE MAP
    for i in range(0,len(dirty_df)):
        o_map.add_child(fol.Marker([dirty_df.loc[i]['Origin Latitude'],dirty_df.loc[i]['Or

    o_map
```

```
Out[8]: <folium.folium.Map at 0x16c330cfc18>
```

0.3.2 3.2: UBER Ride points across VIC (Destination Region)

```
In [9]: # CREATING AN EMPTY MAP
        d_map = fol.Map(location=[-38, 144], tiles="Mapbox Bright", zoom_start=8)

        # ADDING MARKERS ON THE MAP
        for i in range(0,len(dirty_df)):
            d_map.add_child(fol.Marker([dirty_df.loc[i]['Destination Latitude'],dirty_df.loc[i

        d_map
```

```
Out[9]: <folium.folium.Map at 0x16c330cf8d0>
```

As we can see, based on the origin and destination points of rides on the map shows few coordinates are outside VIC infact outside Australia. So we need to fix the latitude as we also saw that there are positive latitudes which should not be the case if points belongs in VIC.

```
<h3><u> Error 1, 2 Outliers for origin/ destination latitude (Semantic Error) : </u></h3>
<p> As we know the points are only based in VIC, so any point outside that is an outlier to the
<h3><u> Fix: </u></h3>
<p> The best possible fix would be changing those negative latitudes to positive then they wil
```

```
In [10]: # correcting the incorrect lat long for origin and destination
         # setting the corrected flag for the rows which are being corrected
         dirty_df.loc[dirty_df["Origin Latitude"] > 0, "corrected"] = 1
         dirty_df.loc[dirty_df["Destination Latitude"] > 0, "corrected"] = 1

         dirty_df.loc[dirty_df["Origin Latitude"] > 0, "Origin Latitude"] = -dirty_df.loc[dirty
         dirty_df.loc[dirty_df["Destination Latitude"] > 0, "Destination Latitude"] = -dirty_di
```

0.3.3 3.3 Check Date & Time format

Let's check for the format of date and time, so that it should be consistent when we do some analysis.

```
In [11]: # IMPORTING DATETIME LIBRARIES
         from datetime import datetime
```

```
from datetime import timedelta

id_list = []
date_list =[]

# CHECKING THE FORMAT OF DATE
for i in range(0,len(dirty_df)):
    try:
        datetime.strptime(dirty_df.loc[i]['Departure Date'], '%Y-%m-%d')
    except ValueError:
        # print('ID: ' + dirty_df.loc[i]['Unnamed: 0'] + ' Invalid date: {}'.format(d
        id_list.append(dirty_df.loc[i]['Unnamed: 0'])
        date_list.append(dirty_df.loc[i]['Departure Date'])

d ={}
for i in range(0,len(id_list)):
    d[id_list[i]] = date_list[i]

invalid_date = pd.Series(d).to_frame().reset_index()
invalid_date.columns = ["id","date"]
invalid_date.head(5)
```

Out[11]:

	id	date
0	ID1959463567	2018-18-05
1	ID3198565330	2018-17-03
2	ID1328595966	2018-16-05
3	ID3130522574	2018-23-06
4	ID3230556880	2018-26-02

As we can see there are few dates not in the right format either their month and day are swapped or date is exceeding the range of the month.

Let's check time format as well.

In [12]:
```
# IMPORTING re TO CHECK THE CONSITENCIES OF TIME
import re

id2_list = []
time_list =[]


for i in range(0,len(dirty_df)):
    dept_flag = re.match("^(0+[0-9]|1[0-9]|2[0-3])[\:]([0-5][0-9])[\:]([0-5][0-9])$",d
    arr_flag = re.match("^(0+[0-9]|1[0-9]|2[0-3])[\:]([0-5][0-9])[\:]([0-5][0-9])$",di
    if not dept_flag:
        id2_list.append(dirty_df.loc[i]['Unnamed: 0'])
        time_list.append([dirty_df.loc[i]['Departure Time'],"Departure Time"])
        #print('ID: ' + dirty_df.loc[i]['Unnamed: 0'] + ' Invalid date: {}'.format(di
```

```
            if not arr_flag:
                id2_list.append(dirty_df.loc[i]['Unnamed: 0'])
                time_list.append([dirty_df.loc[i]['Arrival Time'],"Arrival Time"])
                #print('ID: ' + dirty_df.loc[i]['Unnamed: 0'] + ' Invalid date: {}'.format(di

        time = {}

        for i in range(0,len(id2_list)):
            time[id2_list[i]] = time_list[i]

        invalid_time = pd.DataFrame.from_dict(time).transpose()
        invalid_time.reset_index(inplace=True)
        invalid_time.columns = ["id","time","type"]
        invalid_time.head(5)
```

Out[12]:

	id	time	type
0	ID3501840116	1:53:22	Arrival Time
1	ID1314878251	8:37:59	Arrival Time
2	ID1804119838	0:40:21	Arrival Time
3	ID1447619795	7:37:30	Arrival Time
4	ID3116900903	4:46:32	Arrival Time

So, there are some time values which are inconsistent and not following the format like 00:00:00. We need to fix these inconsitencies as well before we move ahead.

Error 3, 4 Departure Date - month and date swapped, date exceeding the month (Syntactic Error) :

```
<p> The date does not follow the consistent format. At some places the month and date are swapp
<h3><u> Fix: </u></h3>
<ul>
    <li> <b> For month and date swapped: Swap them </b> </li>
    <li> <b> For date exceeding month: Reduce the day to the last day of that month. </b> </li
</ul>
```

In [13]:
```
        # METHOD TO FIX THE DATE WHICH EXCEEDS THE MONTH
        def fix_date(year, month, day):
            # Calculate the last date of the given month
            nextmonth = datetime(year, month, 1) + timedelta(days=35)
            lastday = nextmonth.replace(day=1) - timedelta(days=1)

            if len(str(month)) == 1:
                return (str(year) + "-0" + str(month) + "-" + str(min(day, lastday.day)))
            else:
                return (str(year) + "-" + str(month) + "-" + str(min(day, lastday.day)))


        # CORRECTING THE MONTH AND DAY SWAPPED (IN invalid_date)
        invalid_date["corr_date"] = invalid_date["date"]
```

```
for i in range(0, len(invalid_date)):
    dt = invalid_date.loc[i]["corr_date"].split("-")

    if int(dt[1]) > 12:
        tmp = dt[2]
        dt[2] = dt[1]
        dt[1] = tmp
        invalid_date.loc[i]["corr_date"] = dt[0]+"-" + dt[1] + "-" + dt[2]
    else:
        str_dt = fix_date(datetime(int(dt[0]),1,1).year, datetime(2000,int(dt[1]),1).r
        invalid_date.loc[i]["corr_date"] = str_dt

# FIXING THE dirty_df FOR DATES
for i in range(0,len(invalid_date)):
    dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_date.loc[i]["id"],"Departure Date"]
    dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_date.loc[i]["id"],"corrected"] = 1
```

Since now the Departure date has been fixed for the format we can add the day factor in our dataset which is like 0 for weekdays and 1 for weekends.

```
In [14]: day_classifier(dirty_df)
```

<h3><u> Inconsistency 1, 2 Inconsistent Arrival / Departure Time : </u></h3>
<p> The time does not follow the consistent format.</p>
<h3><u> Fix: </u></h3>
<p> Just need to correct the format to make it consistent. We will just change its type to date

```
In [15]: dirty_df["Arrival Time"] = pd.to_datetime(dirty_df['Arrival Time'],format= '%H:%M:%S'
         dirty_df["Departure Time"] = pd.to_datetime(dirty_df['Departure Time'],format= '%H:%M
```

0.3.4 3.4 Correlation between dimensions

Let's see some correlation between dimensions to understand the linear relationship

```
In [16]: import matplotlib.pyplot as plt

         numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
         newdf = dirty_df.select_dtypes(include=numerics)

         plt.matshow(newdf.corr())
         plt.xticks(range(len(newdf.columns)), newdf.columns, rotation="vertical")
         plt.yticks(range(len(newdf.columns)), newdf.columns)
         plt.colorbar()
         fig_size = plt.rcParams["figure.figsize"]
         fig_size[0] = 25
         fig_size[1] = 25
         plt.rcParams["figure.figsize"] = fig_size
         plt.show()
```

```
&lt;Figure size 480x480 with 2 Axes&gt;
```

From the correlation plot above we can see that:
Fare$ has a strong relation with Uber type, Journey Distance(m) and Travel Time(s).
Origin / Destination Region has a strong relation with Journey Distance & Travel Time(s).

0.3.5 3.5: Understanding different Uber Type and Fare relation

Let's see different UBER types and their counts in our dataset. We already know that there is one extra category based on the description we saw. So we need to find that odd one out.

```
In [17]: dirty_df.groupby(["Uber Type"])["Uber Type"].count()
         dirty_df.loc[(dirty_df["Uber Type"] == 3) | (dirty_df["Uber Type"] == 2)]
```

```
Out[17]:     Unnamed: 0  Uber Type  Origin Region  Destination Region  \
         4    ID5960060344          2              5                   6
         7    ID5261176554          2              1                   7
         8    ID5670774661          2              8                   1
         20   ID5851533409          2              2                   2
         22   ID5886275296          2              3                   3
         27   ID5541874337          2              1                   3
         28   ID5897208891          2              5                   9
         39   ID5938668944          2              8                   4
         40   ID5479788637          2              9                   2
         48   ID5111529081          2              4                   8
         49   ID5745366815          2              7                   4
         54   ID5180492014          2              7                   4
         58   ID5261917491          2              7                   6
         60   ID5605142514          2              5                   1
         75   ID5822299977          2              7                   1
         76   ID5557361655          2              1                   9
         78   ID5830875694          2              3                   1
         79   ID5245688619          2              6                   9
         84   ID5538431278          2              6                   3
         85   ID5695954470          2              1                   3
         86   ID5492998419          2              3                   9
         89   ID5180514055          2              6                   2
         93   ID5463823113          2              4                   7
         99   ID5806917604          2              2                   7
         104  ID5246671181          2              1                   3
         106  ID5138814094          2              5                   1
         107  ID5754133251          2              7                   3
         110  ID5124328169          2              6                   7
         112  ID5927785853          2              3                   8
         114  ID5209135749          2              8                   7
         ..            ...        ...            ...                 ...
         126  ID3362055374          3              4                   7
         129  ID1208828616          3              9                   2
```

132	ID1854884366	2	2	7
133	ID3988603151	2	3	2
136	ID1248152001	2	4	6
137	ID5863570328	2	2	6
153	ID5451784328	2	8	7
158	ID5275404451	2	9	8
162	ID5307940421	2	2	9
165	ID5780145614	2	2	7
183	ID5575311507	2	6	9
184	ID5180410566	2	3	9
195	ID5733258800	2	1	4
197	ID5593813987	2	3	9
200	ID5738135982	2	2	6
210	ID5797809501	2	9	6
217	ID5694183123	2	5	3
224	ID5795734336	2	3	2
238	ID5442284009	2	3	8
244	ID5931451825	2	9	1
247	ID5917894894	2	1	2
248	ID5362479694	2	9	2
249	ID5286942123	2	6	8
252	ID5983501769	2	5	8
253	ID5446369758	2	7	1
257	ID5574936734	2	8	3
266	ID5771069863	2	1	3
268	ID5720773996	2	7	5
273	ID5806301908	2	4	3
276	ID5514404100	2	3	6

	Origin Latitude	Origin Longitude	Destination Latitude	\
4	-37.809426	144.928865	-37.787442	
7	-37.811664	144.965229	-37.861835	
8	-37.815834	145.046450	-37.813893	
20	-37.815834	145.046450	-37.812841	
22	-37.815834	145.046450	-37.816977	
27	-37.811416	144.973073	-37.817346	
28	-37.809278	144.929146	-38.110916	
39	-37.807202	145.026637	-37.805676	
40	-38.110916	144.654173	-37.804583	
48	-37.802871	145.016633	-37.815834	
49	-37.861835	144.905716	-37.798948	
54	-37.861835	144.905716	-37.811774	
58	-37.861835	144.905716	-37.787442	
60	-37.811002	144.926609	-37.813983	
75	-37.861835	144.905716	-37.814232	
76	-37.818891	144.953370	-38.110916	
78	-37.816101	144.972610	-37.799097	
79	-37.773845	144.983689	-38.110916	

84	-37.790797	144.985865	-37.813887
85	-37.811515	144.964682	-37.820839
86	-37.812439	144.973491	-38.110916
89	-37.773803	144.983647	-37.817511
93	-37.809748	144.993152	-37.861835
99	-37.813313	144.939603	-37.861835
104	-37.812388	144.973841	-37.816808
106	-37.816662	144.927391	-37.808225
107	-37.861835	144.905716	-37.813069
110	-37.787442	144.980409	-37.861835
112	-37.820126	144.980368	-37.815834
114	-37.815834	145.046450	-37.861835
..	...	...	...
126	-37.808194	144.999863	-37.861835
129	-38.110916	144.654173	-37.818684
132	-37.815892	144.938251	-37.861835
133	-37.810004	144.995305	-37.811331
136	-37.810995	145.000582	-37.773803
137	-37.817791	144.932188	-37.773803
153	-37.807202	145.026637	-37.861835
158	-38.110916	144.654173	-37.815834
162	-37.814017	144.939616	-38.110916
165	-37.814805	144.944860	-37.861835
183	-37.790797	144.985865	-38.110916
184	-37.818560	144.999172	-38.110916
195	-37.820846	144.957268	-37.815372
197	-37.821107	145.001579	-38.110916
200	-37.814332	144.935923	-37.787433
210	-38.110916	144.654173	-37.790797
217	-37.813801	144.926999	-37.824905
224	-37.822148	144.979055	-37.823620
238	-37.822236	144.967743	-37.807202
244	-38.110916	144.654173	-37.818826
247	-37.818629	144.962994	-37.812528
248	-38.110916	144.654173	-37.812288
249	-37.773845	144.983689	-37.815834
252	-37.812669	144.931570	-37.815834
253	-37.861835	144.905716	-37.818945
257	-37.807202	145.026637	-37.823664
266	-37.800807	144.973561	-37.811033
268	-37.861835	144.905716	-37.809116
273	-37.811512	144.996490	-37.824432
276	-37.810602	145.000827	-37.790797

	Destination Longitude	Journey Distance(m)	Departure Date	Departure Time	\
4	144.980409	6687.7	2018-04-26	12:20:43	
7	144.905716	8478.0	2018-07-15	19:26:16	
8	144.959240	8062.0	2018-04-23	16:43:20	

20	144.939897	9719.0	2018-07-10	20:39:32
22	144.976883	6881.0	2018-01-06	21:22:45
27	145.005497	3351.0	2018-01-04	05:22:25
28	144.654173	43421.0	2018-07-24	11:37:16
39	144.931867	9131.0	2018-02-04	09:23:28
40	144.935393	43482.0	2018-02-23	00:39:26
48	145.046450	5851.0	2018-03-16	19:15:14
49	145.001284	12513.0	2018-04-18	18:19:08
54	145.002000	11108.0	2018-05-09	00:01:46
58	144.980409	11630.0	2018-06-07	10:54:14
60	144.954683	2995.0	2018-05-22	23:38:41
75	144.972816	8523.0	2018-04-06	05:48:32
76	144.654173	42451.0	2018-05-24	03:49:00
78	144.943050	3717.0	2018-06-25	04:05:23
79	144.654173	48197.0	2018-03-05	09:20:02
84	144.971597	3365.0	2018-03-02	06:15:57
85	144.969401	1206.0	2018-07-14	15:34:53
86	144.654173	44350.0	2018-01-19	21:41:43
89	144.948759	9403.0	2018-07-28	01:57:00
93	144.905716	10450.0	2018-04-13	20:20:03
99	144.905716	10199.0	2018-02-01	02:07:13
104	145.007359	3305.0	2018-05-13	23:45:12
106	144.971402	5163.0	2018-07-09	00:39:40
107	144.986482	9612.0	2018-06-28	18:04:22
110	144.905716	11630.0	2018-04-01	20:44:21
112	145.046450	7153.0	2018-04-05	07:07:26
114	144.905716	15151.0	2018-05-19	09:57:55
..	...	...	...	...
126	144.905716	11309.0	2018-06-26	03:18:26
129	144.942782	41229.0	2018-04-09	16:12:56
132	144.905716	10337.0	2018-02-15	22:42:54
133	144.929407	6890.0	2018-01-07	22:57:08
136	144.983647	9538.0	2018-06-08	13:30:25
137	144.983647	9607.0	2018-04-26	12:33:30
153	144.905716	13986.0	2018-03-27	20:49:16
158	145.046450	51032.0	2018-03-16	07:05:15
162	144.654173	42416.0	2018-07-01	11:47:02
165	144.905716	9590.0	2018-07-13	13:23:03
183	144.654173	47193.0	2018-07-06	12:44:09
184	144.654173	46702.0	2018-07-13	20:41:27
195	145.009057	5144.0	2018-04-13	15:03:12
197	144.654173	46999.0	2018-07-24	01:55:45
200	144.980377	6064.0	2018-02-28	15:50:34
210	144.985865	47193.0	2018-01-16	17:17:28
217	144.991917	7431.0	2018-05-17	02:52:44
224	144.942962	3710.0	2018-03-20	02:48:41
238	145.026637	6837.0	2018-05-15	19:34:49
244	144.953157	42432.0	2018-07-24	10:17:02

247	144.936209	3519.0	2018-05-28	05:29:37
248	144.937264	42927.0	2018-07-09	17:24:23
249	145.046450	10698.0	2018-02-11	21:53:48
252	145.046450	11187.0	2018-03-02	16:22:33
253	144.953212	8525.0	2018-02-27	15:06:42
257	144.967548	7002.0	2018-02-09	17:19:14
266	144.990060	2490.0	2018-01-16	07:55:40
268	144.931933	10914.0	2018-07-20	12:16:08
273	144.975799	3030.0	2018-03-15	04:26:30
276	144.985865	3444.0	2018-03-09	20:08:40

	Travel Time(s)	Arrival Time	Fare$	corrected	day_factor
4	1728.36	12:49:31	136.46	0	0.0
7	2926.74	20:15:02	224.60	0	1.0
8	2325.78	17:22:05	176.02	0	0.0
20	2757.00	21:25:29	204.54	0	0.0
22	2006.64	21:56:11	169.99	0	1.0
27	821.16	05:36:06	84.90	0	0.0
28	10400.70	14:30:36	705.08	0	0.0
39	2353.98	10:02:41	177.18	0	1.0
40	10415.64	03:33:01	723.94	0	0.0
48	1772.16	19:44:46	139.15	0	0.0
49	3955.14	19:25:03	285.92	1	0.0
54	3599.04	01:01:45	269.91	0	0.0
58	3678.54	11:55:32	257.44	0	0.0
60	765.30	23:51:26	80.49	0	0.0
75	2901.66	06:36:53	223.70	1	0.0
76	10217.04	06:39:17	711.49	0	0.0
78	968.58	04:21:31	94.18	0	0.0
79	11519.40	12:32:01	779.80	0	0.0
84	958.08	06:31:55	77.12	0	0.0
85	344.76	15:40:34	50.55	0	1.0
86	10690.16	00:39:51	742.60	0	0.0
89	2890.84	02:44:30	227.73	0	1.0
93	3465.90	21:17:48	250.36	0	0.0
99	3357.60	03:03:10	254.14	0	0.0
104	848.10	23:59:20	93.93	0	1.0
106	1315.44	03:29:15	116.30	0	0.0
107	3256.14	22:31:40	240.01	0	0.0
110	3697.02	21:45:58	274.03	0	1.0
112	2052.36	11:24:14	150.19	0	0.0
114	4917.48	10:35:46	348.42	0	1.0
..	...	...	...	...	...
126	3653.52	04:19:19	28.40	0	0.0
129	9894.96	18:57:50	20.50	0	0.0
132	3387.24	23:39:21	14.98	0	0.0
133	1778.16	23:26:46	24.62	0	1.0
136	2899.38	14:18:44	11.52	0	0.0

```
137         2754.54     13:19:24  204.37           0          0.0
153         4298.28     19:37:38  308.66           0          0.0
158        12681.06     10:36:36  857.05           0          0.0
162        10167.60     08:57:35  696.94           0          1.0
165         3189.60     14:16:12  233.88           0          0.0
183        11530.62     15:56:19  790.23           0          0.0
184        11336.82     23:50:23  778.55           0          0.0
195         1290.12     15:24:42  107.38           0          0.0
197        11282.94     05:03:47  782.69           0          0.0
200         1524.72     16:15:58  124.13           1          0.0
210        11535.78     20:29:43  790.36           0          0.0
217         1924.02     03:24:48  158.30           0          0.0
224         1010.34     03:05:31   97.77           0          0.0
238         1761.96     20:04:10  138.54           0          0.0
244        10211.94     13:07:13  693.56           0          0.0
247          881.22     05:44:18   89.23           0          0.0
248        10288.50     20:15:51  707.29           0          0.0
249         2756.46     22:39:44  220.85           0          1.0
252         3137.76     17:14:50  231.44           0          0.0
253         2926.14     15:55:28  216.62           0          0.0
257         1816.50     17:49:30  142.60           0          0.0
266          610.20     08:05:50   52.72           0          0.0
268         3539.46     13:15:07  257.10           0          0.0
273          699.96     04:38:09   75.68           0          0.0
276          970.38     20:24:50   83.75           0          0.0

[64 rows x 16 columns]
```

In [18]:
```python
dirty_df.groupby(["Uber Type"])["Fare$"].mean()
```

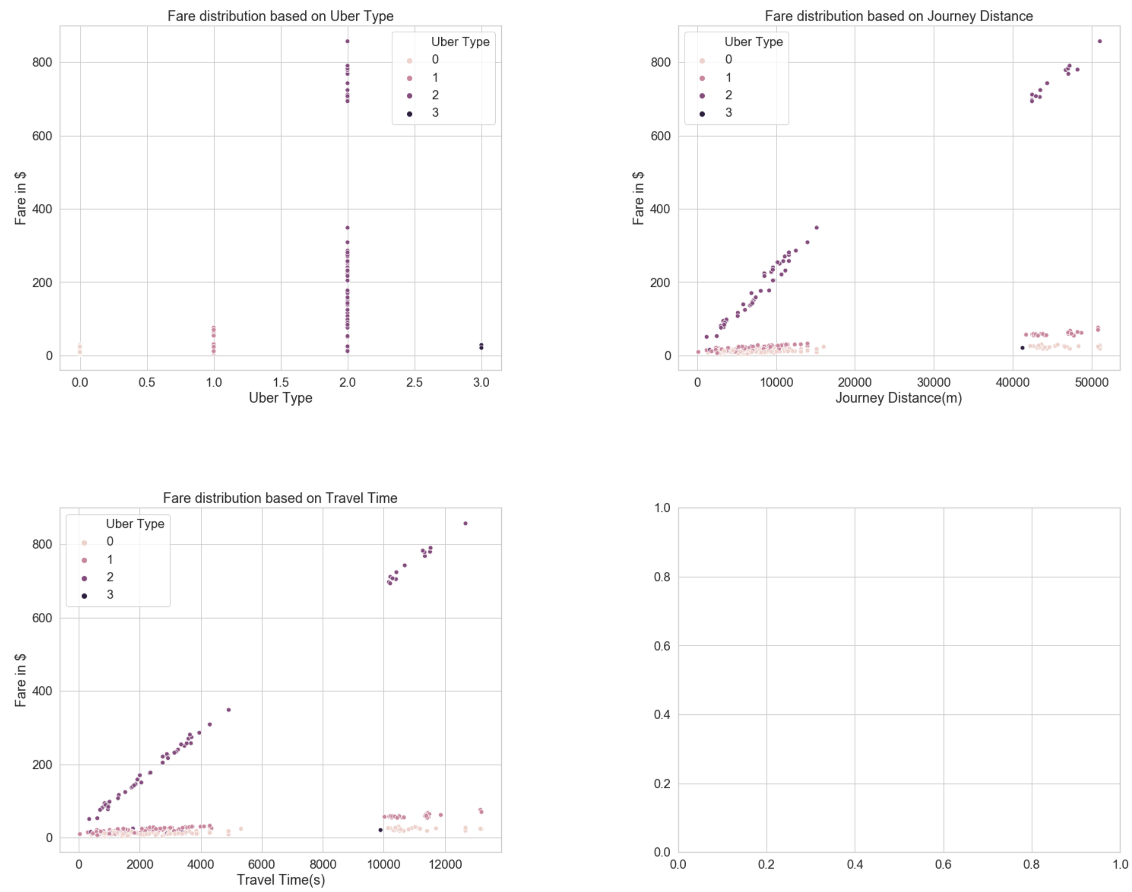
Out[18]:
```
Uber Type
0     14.365948
1     27.985400
2    296.305806
3     24.450000
Name: Fare$, dtype: float64
```

In [19]:
```python
dirty_df.groupby(["Uber Type"])["Fare$"].describe()
```

Out[19]:
```
           count        mean         std    min       25%      50%      75%  \
Uber Type
0          116.0   14.365948    6.111474   2.82    9.8275   12.850   17.995
1          100.0   27.985400   17.570685   6.43   14.7475   22.665   29.685
2           62.0  296.305806  260.439693  11.52  109.6100  218.735  302.975
3            2.0   24.450000    5.586144  20.50   22.4750   24.450   26.425

              max
Uber Type
0           29.81
```

```
1          75.55
2         857.05
3          28.40
```

In [20]:
```
import seaborn as sb
# SETTING STYLE FOR seaborn PLOT
sb.set(font_scale=1.5)
sb.set_style('whitegrid')

# CONFIGURING seaborn (sb) WITH THE DATA AND AXES VALUES
f, ax = plt.subplots(figsize=(25,20),ncols=2, nrows=2)
f.subplots_adjust(wspace=0.4, hspace=0.4)

a=sb.scatterplot(ax = ax[0][0], x= dirty_df['Uber Type'], y='Fare$', hue=dirty_df['Ube
b=sb.scatterplot(ax = ax[0][1], x= dirty_df['Journey Distance(m)'], y='Fare$', hue=di
c=sb.scatterplot(ax = ax[1][0], x= dirty_df['Travel Time(s)'], y='Fare$', hue=dirty_d

# PLOTTING THE FIGURE
a.set_ylabel('Fare in $')
b.set_ylabel('Fare in $')
c.set_ylabel('Fare in $')

ax[0][0].set_title("Fare distribution based on Uber Type")
ax[0][1].set_title("Fare distribution based on Journey Distance")
ax[1][0].set_title("Fare distribution based on Travel Time")

plt.show()
plt.clf()
plt.close()
```



By viewing these scatter plots, we can see the two records of UBER type 3 are coinciding with UBER type 0. This tells us that it could be a wrong entry for those two records and ideally they should be 0. But let's try to verify it with the help of transaction ID (Unnamed: 0) as it follows a pattern and its third character is same for all the same UBER Type.

In [21]:
```
# TRYING TO IDENTIFY THE PATTERN OF ID AND UBER TYPE
dirty_df.loc[dirty_df["Uber Type"] == 0]["Unnamed: 0"]
dirty_df.loc[dirty_df["Uber Type"] == 1]["Unnamed: 0"]
dirty_df.loc[dirty_df["Uber Type"] == 2]["Unnamed: 0"]
dirty_df.loc[dirty_df["Uber Type"] == 3]["Unnamed: 0"]
```

Out[21]:
```
126    ID3362055374
129    ID1208828616
Name: Unnamed: 0, dtype: object
```

Uber Type 0: 1
Uber Type 1: 3,
Uber Type 2: 5,
Uber Type 3: 1,3 (Only 2 records and also we know that only 3 categories of UBER type exists in the data. So this must be changed)

Let's try to observe the pattern visually and more carefully.

```
In [22]: # SETTING STYLE FOR seaborn PLOT
         sb.set(font_scale=1.5)
         sb.set_style('whitegrid')

         # CONFIGURING seaborn (sb) WITH THE DATA AND AXES VALUES
         plt.subplots(figsize=(10,5))

         a=sb.scatterplot(x=dirty_df['Uber Type'], y=dirty_df["Unnamed: 0"].str.extract(r'^ID(

         # PLOTTING THE FIGURE
         a.set_ylabel('IDs Char (ID3...) or (ID1..) ')

         plt.title("ID classification on Uber Type")

         plt.show()
         plt.clf()
         plt.close()
```

As we observed that, there is a pattern in 'Unnamed: 0' or ID column for different UBER types, like for type 0: ID1… and for type 1: ID3… and so on. Also in the 'ID classification on UBER type' scatter plot we get to see that for type 1 and type 2 there are few IDs with slight verification.

Uber Type 0: '1'

Uber Type 1: '3', '1' ('1' is hardly for two or three records, which are not following the trend which means this is an error wherever there is '1' for this type)

Uber Type 2: '5', '3', '1' ('3' and '1' are hardly for two or three records, which are not following the trend which means this is an error wherever there is '3' and '1' for this type)

Uber Type 3: 1,3 (Only 2 records and also we know that only 3 categories of UBER type exists in the data. So this must be changed)

In [23]:
```
uber_type = {'0':[],'1':[],'2':[]}

for i in range(len(dirty_df)):
    if (dirty_df['Uber Type'][i]==0) & (dirty_df['Unnamed: 0'][i][2]!='1'):
        uber_type['0'].append(dirty_df.loc[i]["Unnamed: 0"])
        #print(dirty_df.loc[i]["Unnamed: 0"], dirty_df.loc[i]["Uber Type"])
    elif (dirty_df['Uber Type'][i]==1) & (dirty_df['Unnamed: 0'][i][2]!='3'):
        uber_type['1'].append(dirty_df.loc[i]["Unnamed: 0"])
        #print(dirty_df.loc[i])
    elif (dirty_df['Uber Type'][i]==2) & (dirty_df['Unnamed: 0'][i][2]!='5'):
        uber_type['2'].append(dirty_df.loc[i]["Unnamed: 0"])
        #print(dirty_df.loc[i])
uber_type
```

Out[23]:
```
{'0': [],
 '1': ['ID1885469348', 'ID1574949198'],
 '2': ['ID1701214714', 'ID1854884366', 'ID3988603151', 'ID1248152001']}
```

For UBER type 1: ‘ID1885469348’, ‘ID1574949198’ does not follow the trend.

For UBER type 2: ‘ID1701214714’, ‘ID1854884366’, ‘ID3988603151’, ‘ID1248152001’ does not follow the trend.

```
<h3><u> Error 5 --> Additional UBER Type (i.e. type 3) (Syntactic Error) : </u></h3>
<p> There are only 3 UBER types as per specification in the dataset which are 0,1,2. As type 3
<h3><u> Fix: </u></h3>
<p> Based on the pattern observed with the fare and IDs, we have 2 records in type 3 where one
```

In [24]:
```
utype = dirty_df.loc[dirty_df["Uber Type"] == 3]
for i in range (0, len(utype)):
    if utype.iloc[i]["Unnamed: 0"][2] == '1' :   # type 0
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"Uber Type
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"corrected

    elif utype.iloc[i]["Unnamed: 0"][2] == '3' : # type 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"Uber Type
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"corrected

    else:                                        # type 2
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"Uber Type
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == utype.iloc[i]["Unnamed: 0"],"corrected

dirty_df.loc[dirty_df["Uber Type"] == 3]
```

Out[24]:
```
Empty DataFrame
Columns: [Unnamed: 0, Uber Type, Origin Region, Destination Region, Origin Latitude, (
Index: []
```

```
<h3><u> Error 6 --> UBER Type 1,2 (Semantic Error) : </u></h3>
<p> The type 1 and type 2 has got two to four IDs which does not follow the pattern of their I
<h3><u> Fix: </u></h3>
<p> The best possible fix we can do, is to change the UBER type for these 2 to 4 records to the
```

In [25]:
```python
for i in range(0,len(uber_type['1'])):
    if uber_type['1'][i][2] == '1':
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"Uber Type"] = 0
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"corrected"] = 1

    elif uber_type['1'][i][2] == '3':
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"Uber Type"] = 1
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"corrected"] = 1

    elif uber_type['1'][i][2] == '5':
        print(dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"Uber Type"])
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"Uber Type"] = 2
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['1'][i],"corrected"] = 1

for i in range(0,len(uber_type['2'])):
    if uber_type['2'][i][2] == '1':
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"Uber Type"] = 0
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"corrected"] = 1

    elif uber_type['2'][i][2] == '3':
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"Uber Type"] = 1
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"corrected"] = 1

    elif uber_type['2'][i][2] == '5':
        print(dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"Uber Type"])
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"Uber Type"] = 2
        # marking the corrected flag as 1
        dirty_df.loc[dirty_df["Unnamed: 0"] == uber_type['2'][i],"corrected"] = 1
```

0.3.6 3.6 Validating Regions alloted

As we have seen while plotting a map for origin points and destination points on the map, quite a few points were falling in another region's cluster and marked with some other region. So we need to fix those regions or rather try to predict the closest possible region for the point which seems like an outlier on the map.

We will use our outlier.csv and missing_value.csv to train our model for prediction and then dirty data will be used for testing and getting the final predictions for regions.

In [26]:
```python
# FINDING THE EUCLIDEAN DISTANCE BETWEEN ANY 2 POINTS
import math

def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination

    radius = 6378 # km

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = radius * c
    return d
```

In [27]:
```python
miss1 = pd.DataFrame([miss_df['Origin Region'],miss_df['Origin Latitude'],miss_df['Ori
miss1.columns = ['Region','Latitude','Longitude']
miss2 = pd.DataFrame([miss_df['Destination Region'],miss_df['Destination Latitude'],mi
miss2.columns = ['Region','Latitude','Longitude']

out1 = pd.DataFrame([outlier_df['Origin Region'],outlier_df['Origin Latitude'],outlie
out1.columns = ['Region','Latitude','Longitude']
out2 = pd.DataFrame([outlier_df['Destination Region'],outlier_df['Destination Latitude
out2.columns = ['Region','Latitude','Longitude']

dirty1 = pd.DataFrame([dirty_df['Origin Region'],dirty_df['Origin Latitude'],dirty_df
dirty1.columns = ['Region','Latitude','Longitude']
dirty2 = pd.DataFrame([dirty_df['Destination Region'],dirty_df['Destination Latitude']
dirty2.columns = ['Region','Latitude','Longitude']

alldata_reg = miss1.append([miss2,out1,out2,dirty1,dirty2])
alldata_reg.drop_duplicates(inplace=True)
alldata_reg.reset_index(inplace=True, drop = True)
alldata_reg["dist_center(km)"] = 0
alldata_reg["predicted_region"] = -1

region_center = alldata_reg.groupby('Region').agg('mean')
region_center.reset_index(inplace=True)
#region_center
```

In [28]:
```python
for i in range(0,len(alldata_reg)):
    origin = [alldata_reg.loc[i,"Latitude"], alldata_reg.loc[i,"Longitude"]]
    destination = [region_center.loc[region_center["Region"] == alldata_reg.loc[i]["Re
    alldata_reg.loc[i, "dist_center(km)"] = distance(origin,destination)

    predict_dict = {}
```

```python
        if alldata_reg["dist_center(km)"][i] > 2:
            # check distance with other centers
            for k in range(0, len(region_center)):
                dist = distance(origin, [region_center.loc[k,"Latitude"], region_center.lo
                predict_dict[region_center.loc[k,"Region"]] = dist

            mini = min(val for val in predict_dict.values())
            alldata_reg.loc[i,"predicted_region"] = float(str([key for key,v in predict_di
        else:
            alldata_reg.loc[i,"predicted_region"] = alldata_reg["Region"][i]
```

In [29]:
```python
invalid_loc = alldata_reg.loc[alldata_reg["Region"] != alldata_reg["predicted_region"]
invalid_loc.reset_index(inplace=True)
invalid_loc
```

Out[29]:

	index	Region	Latitude	Longitude	dist_center(km)	predicted_region
0	29	3.0	-37.819703	145.010013	2.093026	4.0
1	170	1.0	-37.803153	144.936730	2.011528	5.0
2	228	2.0	-37.815834	145.046450	9.267136	8.0
3	229	7.0	-37.773803	144.983647	7.613746	6.0
4	230	3.0	-37.815834	145.046450	5.293421	8.0
5	238	6.0	-37.820082	144.968573	3.827233	1.0
6	400	3.0	-37.800768	144.970004	2.391001	1.0
7	402	4.0	-38.110916	144.654173	45.244006	9.0
8	403	4.0	-37.805676	144.931867	5.986392	5.0

<h3><u> Error 7 --> Wrong Region Allocation for few points (Coverage Error) : </u></h3>
<p> The region has been alloted wrong for some points though visually they seem to belong in so
<h3><u> Fix: </u></h3>
<p> The best possible fix we can do, is to take mean of lat and long for each region which wil

In [30]:
```python
for i in range(0, len(invalid_loc)):
    dirty_df.loc[(dirty_df['Origin Region'] != invalid_loc.loc[i, 'predicted_region']]
    dirty_df.loc[(dirty_df['Destination Region'] != invalid_loc.loc[i, 'predicted_regi

    dirty_df.loc[(dirty_df['Origin Region'] != invalid_loc.loc[i, 'predicted_region']]
    dirty_df.loc[(dirty_df['Destination Region'] != invalid_loc.loc[i, 'predicted_regi
```

In [31]:
```python
# Make an empty map
m = fol.Map(location=[-38, 145], tiles="Mapbox Bright", zoom_start=8)

# I can add marker one by one on the map
for i in range(0,len(dirty_df)):
    m.add_child(fol.Marker([dirty_df.loc[i]['Origin Latitude'],dirty_df.loc[i]['Origin

m
```

Out[31]: <folium.folium.Map at 0x16c36155128>

0.3.7 3.7 Validating Journey Distance(m) (Is it really the shortest path?)

Here we will try to validate whether the journey distance given in the dataset is actually the shortest distance for any ride or not, as it could be wrong due to some data entry issues. So it would be better if we verify before moving ahead. We will use Dijkstra algorithm to find the shortest journey distance between the source point and the destination point and then compare it with the given journey distance in the dataset.

In [32]:
```
# importing networkx package for applying Dijkstra algorithm
import networkx as netx

# creating graph using edges and nodes in the edges.csv and nodes.csv respectively
graph = netx.from_pandas_edgelist(edges_df,'u','v',['distance(m)'])

id3_list = []
dist_list = []

for i in range(0,len(dirty_df)):
    # computing shortest distance for all nodes in dirty data
    distance, path = netx.single_source_dijkstra(graph, source = int(nodes_df['Unnamed

    # checking if distance from dirty data and shortest distance from dijkstra are no
    if distance != round(dirty_df.iloc[i]['Journey Distance(m)']):
        id3_list.append(dirty_df.iloc[i]["Unnamed: 0"])
        dist_list.append([round(dirty_df.iloc[i]["Journey Distance(m)"]), distance])

dist = {}

for i in range(0,len(id3_list)):
    dist[id3_list[i]] = dist_list[i]

invalid_dist = pd.DataFrame.from_dict(dist).transpose()
invalid_dist.reset_index(inplace=True)
invalid_dist.columns = ["id","given_distance","computed_distance"]
invalid_dist
```

Out[32]:

	id	given_distance	computed_distance
0	ID1404486339	15150.0	15151.0
1	ID5960060344	6688.0	6689.0
2	ID3428249873	43592.0	43586.0
3	ID3780031863	9079.0	9080.0
4	ID1804119838	9174.0	9165.0

```
<h3><u> Error 8 --> Wrong Journey Distance(m) (Semantic Error) : </u></h3>
<p> The distance is not the shortest distance in few cases and in some cases its completely wro
<h3><u> Fix: </u></h3>
<p> The best possible fix we can do, is to replace those wrong distance values by the computed
```

In [33]:
```
# fixing the wrong distance with the correct value
for i in range(0, len(invalid_dist)):
```

```
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_dist.loc[i,"id"], "Journey Distance
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_dist.loc[i,"id"], "corrected"] = 1
```

0.3.8 3.8 Validating Travel time based on the shortest distance

```
In [34]: # calculates the travel time based on the edges for a path
         def traveltime(path, edges):
             tt = 0
             for i in range(len(path)-1):
                 # print(edges[((edges.u==path[i]) & (edges.v== path[i+1])) | ((edges.v==path[
                 tt = tt + (edges[((edges.u==path[i]) & (edges.v== path[i+1])) | ((edges.v==pat

             return tt

         def time_diff(at,dt):
             FMT = '%H:%M:%S'
             at = str(at)
             dt = str(dt)
             tdelta = datetime.strptime(at, FMT) - datetime.strptime(dt, FMT)
             if tdelta.days < 0:
                 tdelta = timedelta(days=0,seconds=tdelta.seconds, microseconds=tdelta.microsec

             return tdelta.total_seconds()

In [35]: invalid_tt = []
         tt_dict={}

         for i in range (0,len(dirty_df)):
             flag=0
             if dirty_df.loc[i]["corrected"] == 1:
                 continue
             else:
                 p = netx.all_shortest_paths(graph, source = int(nodes_df['Unnamed: 0'][(nodes_
                 correct_tt_list = []

                 for each in p:
                     flag=0

                     correct_tt = traveltime(each,edges_df)
                     calculated_tt = time_diff(dirty_df.loc[i]['Arrival Time'],dirty_df.loc[i]
                     correct_tt_list.append(correct_tt)

                     if int(correct_tt) == int(dirty_df.loc[i]['Travel Time(s)']):
                         flag=1
                         break

                 if flag == 0:
                     tt_dict[dirty_df.loc[i]['Unnamed: 0']] = {"Dept_Time":str(dirty_df.loc[i]
```

```
invalid_tt = pd.DataFrame.from_dict(tt_dict).transpose()
invalid_tt.reset_index(inplace=True)
invalid_tt
```

```
Out[35]:           index  Arr_Time Dept_Time Diff_Time          Given_Time  \
         0  ID1412535745  04:46:15  04:06:25    2390.0  2427.1600000000008
         1  ID3253404384  09:19:05  08:56:51    1334.0             1414.88
         2  ID5695954470  15:40:34  15:34:53     341.0              344.76
         3  ID5492998419  00:39:51  21:41:43   10688.0            10690.16
         4  ID3299532134  15:08:51  12:18:34   10217.0  10263.159999999996
         5  ID5180514055  02:44:30  01:57:00    2850.0   2890.839999999999
         6  ID3477926128  19:16:29  18:31:20    2709.0              2755.6
         7  ID1984431955  17:30:40  16:32:05    3515.0   3504.100000000001
         8  ID3434059484  10:46:12  09:34:34    4298.0   4353.280000000002
         9  ID1855502491  09:12:00  08:46:11    1549.0  1608.9800000000007

                                                  Path_Time
         0  [2390.1600000000003, 2408.3400000000006, 2402...
         1                                        [1334.88]
         2  [341.76, 331.56, 349.02, 349.37999999999994, 3...
         3              [10688.16, 10689.0, 10688.16, 10689.0]
         4                             [10217.159999999996]
         5  [2850.839999999999, 2852.639999999999, 2852.87...
         6                                 [2709.6, 2709.6]
         7  [3515.1000000000004, 3523.32, 3514.32, 3522.54...
         8  [4298.280000000001, 4316.460000000001, 4310.52...
         9                             [1549.9800000000007]
```

<h3><u> Error 9 --> Wrong Travel Time(s) (Semantic Error) : </u></h3>
<p> The given travel time does not matches with the computed time using edges and nodes to finc
<h3><u> Fix: </u></h3>
<p> We observed that amongst multiple travel time for different paths atleast 1 path's travel t

```
In [36]: for i in range(0,len(invalid_tt)):
             dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_tt.loc[i]["index"], "Travel Time(s)
             dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_tt.loc[i]["index"], "corrected"] =
```

0.3.9 3.9 Validating Arrival Time (Except for the corrected rows)

We need to verify whether the arrival time is correct or not assuming that Departure time and Travel Time is correct. Since we have already corrected Travel time for few rows, therefore we will exclude those rows as we cannot use the corrected Travel Time to correct Arrival Time.

```
In [37]: arr_dict = {}
         for i in range (0,len(dirty_df)):
             if dirty_df.loc[i]["corrected"] == 1:
                 continue
             else:
```

```
        arr_time = str(dirty_df.iloc[i]['Arrival Time']).split(":")
        dept_time = str(dirty_df.iloc[i]['Departure Time']).split(":")
        travel_time = dirty_df.iloc[i]['Travel Time(s)']

        if arr_time[0] < dept_time[0]:
            arr_time_secs = (24*3600)+ (int(arr_time[0])*3600) + (int(arr_time[1])*60
        else:
            arr_time_secs = (int(arr_time[0])*3600) + (int(arr_time[1])*60) + int(arr_

        dept_tt_secs = (int(dept_time[0])*3600) + (int(dept_time[1])*60) + int(dept_t

        if arr_time_secs != dept_tt_secs:
            arr_dict[dirty_df.iloc[i]['Unnamed: 0']] = {"Travel Time": dirty_df.iloc[i

invalid_arr = pd.DataFrame.from_dict(arr_dict).transpose()
invalid_arr.reset_index(inplace=True)
invalid_arr
```

Out[37]:

```
            index Arrival Time Departure Time Travel Time arr_time(secs)  \
0    ID3825994991     07:48:48       04:07:56     13173.4          28128
1    ID5138814094     03:29:15       00:39:40     1315.44          12555
2    ID5754133251     22:31:40       18:04:22     3256.14          81100
3    ID1131521142     07:13:23       02:43:10      527.52          26003
4    ID1735859616     20:12:25       19:25:11     2816.16          72745
5    ID1330698608     17:48:02       14:28:58       10334          64082
6    ID5927785853     11:24:14       07:07:26     2052.36          41054
7    ID3504958325     00:27:45       18:28:21     10027.2          88065
8    ID5209135749     10:35:46       09:57:55     4917.48          38146
9    ID3922149472     11:56:32       07:39:40       38.46          42992
10   ID3745871299     20:08:24       20:43:44     2120.64          72504
11   ID3545976158     14:22:07       14:51:12     1745.04          51727
12   ID1767870478     00:00:09       01:28:49     5320.68          86409
13   ID3747187176     05:55:02       06:04:58      596.34         107702
14   ID5451784328     19:37:38       20:49:16     4298.28         157058
15   ID5307940421     08:57:35       11:47:02     10167.6         118655
16   ID1631693085     14:55:44       15:24:41     1737.18         140144
17   ID3193841203     11:46:19       12:39:49     3210.66         128779
18   ID1807212196     00:33:50       00:45:04      674.16           2030
19   ID1546431038     00:11:10       00:24:08      778.32            670

   dept_tt(secs)
0          28049
1           3695
2          68318
3          10317
4          72727
5          62471
6          27698
```

```
7          76528
8          40792
9          27618
10         76744
11         55217
12         10649
13         22494
14         79254
15         52589
16         57218
17         48799
18          3378
19          2226
```

```
<h3><u> Error 10 --> Wrong Arrival Time (Semantic Error) : </u></h3>
<p> The given arrival time does not matches with the computed time using Departure Time + Trave
<p>
    <ul>
        <li>Case-1: Arrival time is less than Departure time that is why their difference does
        <li>Case-2: Departure Time is less than Arrival Time which is correct but their differe
    </ul>
</p>
<h3><u> Fix: </u></h3>
<p> We can simply add Departure Time and Travel time(s) and use that to fix the Arrival Time fo
```

In [38]:

```python
from datetime import timedelta
FMT = '%H:%M:%S'

for i in range(0,len(invalid_arr)):
    dept_time = invalid_arr.loc[i]["Departure Time"]
    arr_time = invalid_arr.loc[i]["Arrival Time"]
    travel_time =  invalid_arr.loc[i]["Travel Time"]

    tdelta = datetime.strptime(str(dept_time), FMT) - datetime.strptime(str(arr_time)

    if (datetime.strptime(str(arr_time), FMT) < datetime.strptime(str(dept_time), FMT
        # Need To swap Arrival time and Departure time
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_arr.iloc[i]["index"],"Arrival T
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_arr.iloc[i]["index"],"Departure
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_arr.iloc[i]["index"],"correcte
    else:
        # Add Departure Time + Travel time
        corr_arr_time = datetime.strptime(str(dept_time), FMT) + timedelta(seconds=tra
        corr_tt = datetime.strptime(str(corr_arr_time.hour)+":"+str(corr_arr_time.minu

        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_arr.iloc[i]["index"],"Arrival T
        dirty_df.loc[dirty_df["Unnamed: 0"] == invalid_arr.iloc[i]["index"],"correcte
```

Since now Arrival Time is fixed and wherever there was a need of swap with Departure Time that is also done. so we are ensured about the authenticity of our data now and we can add time factor as another dimension in our data.

In [39]:
```
time_classifier(dirty_df)
```

We have checked every dimension during our EDA process, understood the relations between the dimensions and wrangled the data as well for some data quality checks like synatctic, semantic errors. So we can now move ahead to fix missing values and do outlier detection based on our cleansed 'dirty data'.

0.4 Step 4: Handling Missing Values

In [40]:
```
miss_df.head(5)
```

Out[40]:
```
     Unnamed: 0  Uber Type  Origin Region  Destination Region  \
0  ID3133406343        1.0              3                   2
1  ID1253659061        NaN              6                   7
2  ID1691043777        0.0              8                   9
3  ID3252951283        NaN              2                   8
4  ID1471443835        NaN              6                   5

   Origin Latitude  Origin Longitude  Destination Latitude  \
0       -37.824062        144.984864            -37.825952
1       -37.787442        144.980409            -37.861835
2       -37.815834        145.046450            -38.110916
3       -37.813579        144.937065            -37.815834
4       -37.773803        144.983647            -37.804542

   Destination Longitude  Journey Distance(m) Departure Date Departure Time  \
0             144.952635               3416.0     2018-04-06       11:27:06
1             144.905716              11630.0     2018-05-28       11:51:33
2             144.654173              51032.0     2018-02-21       03:44:52
3             145.046450              10062.0     2018-01-24       08:16:58
4             144.919403              11487.0     2018-07-18       10:51:55

   Travel Time(s) Arrival Time  Fare$
0          863.28     11:41:29   7.32
1         3697.02     12:53:10   8.53
2        12681.06      7:16:13  29.93
3         2833.62      9:04:11  17.40
4         3148.20     11:44:23   7.41
```

In [41]:
```
miss_df.describe()
```

Out[41]:
```
       Uber Type  Origin Region  Destination Region  Origin Latitude  \
count  59.000000      80.000000           80.000000        80.000000
mean    0.762712       4.775000            5.037500       -37.839842
std     0.750609       2.648023            2.887966         0.086233
```

```
min      0.000000        1.000000               1.000000        -38.110916
25%      0.000000        2.000000               2.000000        -37.819819
50%      1.000000        5.000000               5.000000        -37.815449
75%      1.000000        7.000000               8.000000        -37.807202
max      2.000000        9.000000               9.000000        -37.773803

       Origin Longitude  Destination Latitude  Destination Longitude  \
count         80.000000             80.000000              80.000000
mean         144.940476            -37.854735             144.934162
std            0.097708              0.104607               0.119151
min          144.654173            -38.110916             144.654173
25%          144.932489            -37.823210             144.933225
50%          144.951550            -37.815834             144.971999
75%          144.985865            -37.807202             145.001500
max          145.046450            -37.773803             145.046450

       Journey Distance(m)  Travel Time(s)        Fare$
count            80.000000        80.00000    61.000000
mean          16142.337500      4171.56600    91.805902
std           16821.558393      4083.26798   222.322015
min            2096.000000       490.68000     4.860000
25%            5251.500000      1343.50500     9.130000
50%            8023.000000      2286.90000    16.890000
75%           15151.000000      4911.60000    62.610000
max           51061.000000     13204.98000  1087.920000
```

As we can see Uber Type and Fare$ has got few missing values (blank, NaN, or NA in this case). So we need to figure out a way to handle these missing values, may be by imputing them with their closest possible values.

Let's add day factor and time factor in this dataset as they might prove useful in imputing the missing values for Fare$

In [42]:
```
# adding day factors and time factors based on Departure date and time
day_classifier(miss_df)
time_classifier(miss_df)
```

Since we have identified 2 dimensions which have missing values in it, let's try to impute them with the help of the relations they have with other dimensions in the data which we figure out in our preious step while doing EDA.

0.4.1 4.1 Imputing Uber Type (Can be derived from Unnamed: 0 or ID)

We already know that Uber type is not actually missing and we can clearly derive it from the transaction ID or 'Unnamed: 0'. The 3rd character of the column 'Unnamed: 0' is fixed for specific Uber Types.

In [43]:
```
mdf = miss_df["Uber Type"].isna()

for i in range (0, len(miss_df)):
```

```
        if mdf[i]:
            if miss_df.iloc[i]["Unnamed: 0"][2] == '1' :   # type 0
                miss_df.loc[i,"Uber Type"] = 0
            elif miss_df.iloc[i]["Unnamed: 0"][2] == '3' : # type 1
                miss_df.loc[i,"Uber Type"] = 1
            else:                                          # type 2
                miss_df.loc[i,"Uber Type"] = 2
```

0.4.2 4.2 Imputing Fare$ (MAR)

We have seen from the correlation plot above in previous step.3 (EDA) that Fare$ depends on dimension like Departure Date factor(Weekday, Weekend), Time Factor(Morning, Afternoon, Night), Journey Distance(m), Travel Time(s) and Uber Type (but we won't use Uber Type as we already imputed that value.) We will use Uber Type to build our different linear regression models like each model dedicated to each Uber Type.

4.2.1 Data (train/test) preparation (for model building):

```
In [44]: # DATA FOR UBER TYPE 0
         dd_ut0 = dirty_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor',
         miss_ut0 = miss_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor'

         model_data_ut0 = pd.concat([miss_ut0, dd_ut0])
         model_data_ut0.drop_duplicates(inplace=True)
         model_data_ut0.head(5)
```

Out[44]:

	Journey Distance(m)	Travel Time(s)	day_factor	time_factor	Fare$
1	11630.0	3697.02	0.0	0.0	8.53
2	51032.0	12681.1	0.0	2.0	29.93
4	11487.0	3148.2	0.0	0.0	7.41
5	4902.0	1197.48	0.0	1.0	7.93
9	2610.0	808.68	0.0	2.0	13.02

```
In [45]: # DATA FOR UBER TYPE 1
         dd_ut1 = dirty_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor',
         miss_ut1 = miss_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor'

         model_data_ut1 = pd.concat([miss_ut1, dd_ut1])
         model_data_ut1.drop_duplicates(inplace=True)
         model_data_ut1.head(5)
```

Out[45]:

	Journey Distance(m)	Travel Time(s)	day_factor	time_factor	Fare$
0	3416.0	863.28	0.0	0.0	7.32
3	10062.0	2833.62	0.0	0.0	17.40
6	51032.0	12681.1	0.0	2.0	73.45
7	42799.0	10253.7	0.0	2.0	62.61
8	10360.0	3395.46	0.0	1.0	23.46

In [46]:
```
# DATA FOR UBER TYPE 2
dd_ut2 = dirty_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor',
miss_ut2 = miss_df[['Journey Distance(m)','Travel Time(s)','day_factor','time_factor'

model_data_ut2 = pd.concat([miss_ut2, dd_ut2])
model_data_ut2.drop_duplicates(inplace=True)
model_data_ut2.head(5)
```

Out[46]:

	Journey Distance(m)	Travel Time(s)	day_factor	time_factor	Fare$
10	42211.0	10117.6	0.0	0.0	913.83
14	2096.0	490.68	0.0	1.0	64.90
24	7623.0	2222.58	0.0	1.0	219.41
28	8056.0	2059.56	0.0	1.0	206.01
32	7051.0	1802.76	0.0	1.0	180.82

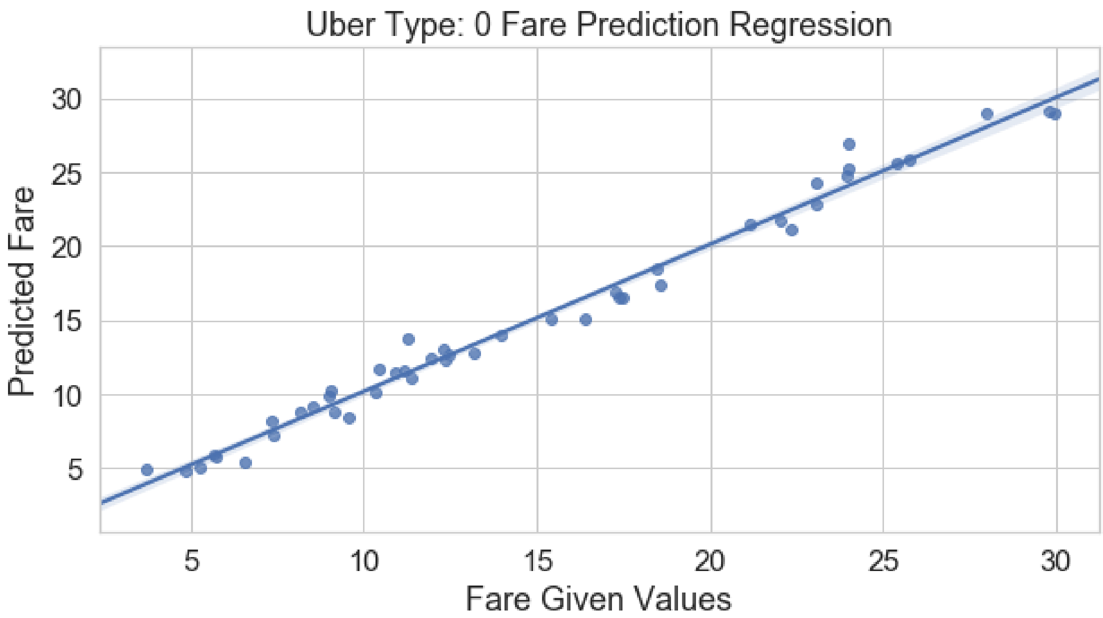
In [47]:
```
# PACKAGE IMPORT REQUIRED FOR LINEAR REGRESSION MODEL
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import numpy as np
```

4.2.2 Building Linear Regression Models (for Uber Type 0, 1, 2):

In [48]:
```
# MODEL FOR UBER TYPE: 0 ------------------------------------------------
x = model_data_ut0.iloc[:,:-1] #creating x variable for training
y = model_data_ut0['Fare$'] #creating y variable for training

#import random

# for i in range(0,100):
# a = random.randint(0,102)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state

lm_ut0 = LinearRegression()
reg = lm_ut0.fit(x_train,y_train)
predictions = lm_ut0.predict(x_test)

plt.subplots(figsize=(10,5))
f.subplots_adjust(wspace=0.4, hspace=0.4)

# plt.scatter(y_test,predictions,color="red")
a = sb.regplot(x=y_test, y=predictions)

a.set_title("Uber Type: 0 Fare Prediction Regression")
a.set_xlabel("Fare Given Values")
a.set_ylabel("Predicted Fare")
b.set_xlabel("Residual")
b.set_ylabel("Given Fare")
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicti
print('R Square:', metrics.r2_score(y_test, predictions))
```

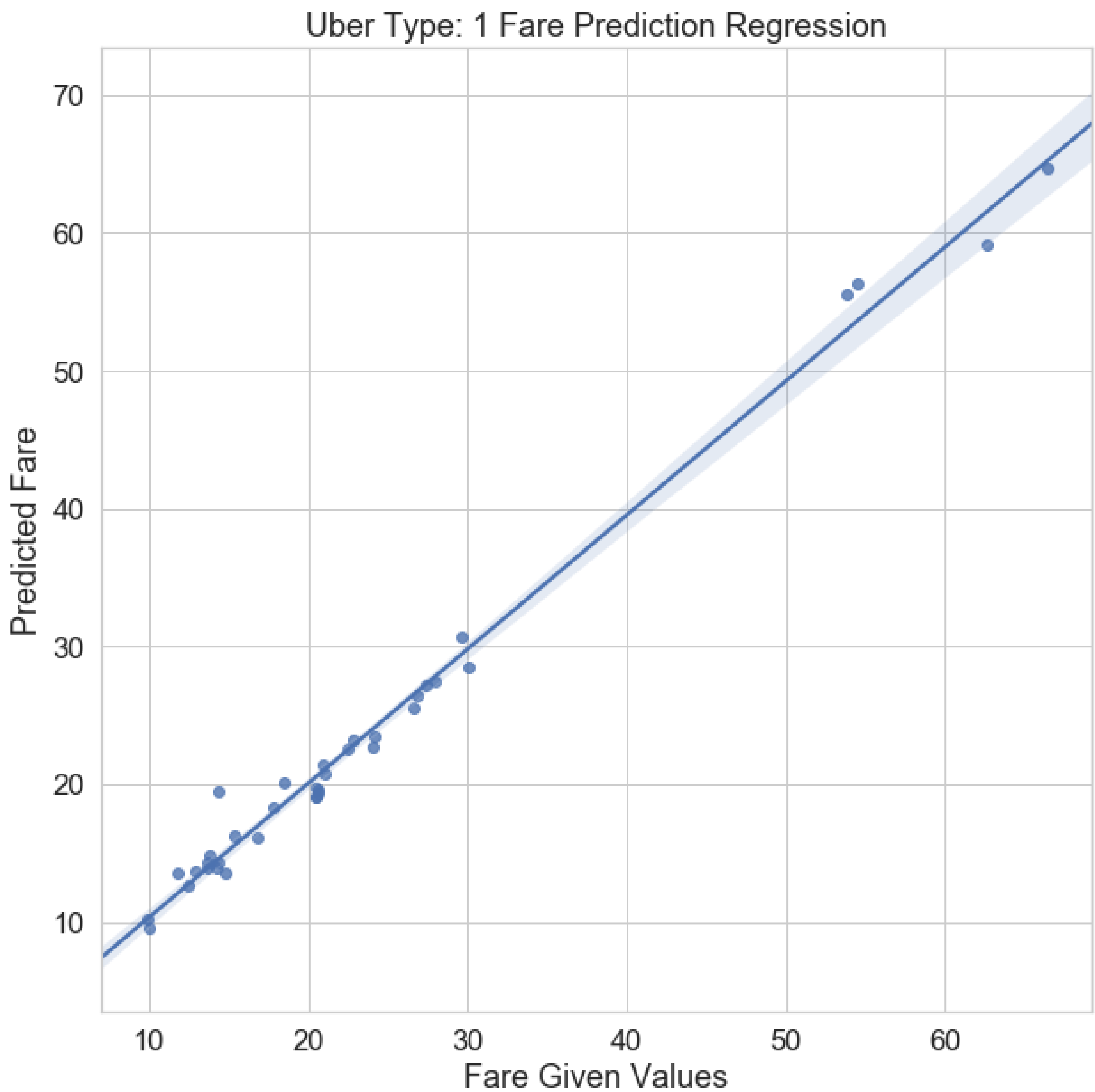
```
Mean Absolute Error: 0.7009423410846926
Mean Squared Error: 0.8478767295441546
Root Mean Squared Error: 0.9208022206446695
R Square: 0.9839741850249923
```




```
In [49]: # MODEL FOR UBER TYPE: 1 -----------------------------------------------
         x = model_data_ut1.iloc[:,:-1] #creating x variable for training
         y = model_data_ut1['Fare$'] #creating y variable for training


         # import random

         # for i in range(0,100):
         # a = random.randint(0,102)
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_stat

         lm_ut1 = LinearRegression()
         reg = lm_ut1.fit(x_train,y_train)
         predictions = lm_ut1.predict(x_test)
```

```
plt.subplots(figsize=(10,10))

# plt.scatter(y_test,predictions,color="red")
a = sb.regplot(x=y_test, y=predictions)

a.set_title("Uber Type: 1 Fare Prediction Regression")
a.set_xlabel("Fare Given Values")
a.set_ylabel("Predicted Fare")
b.set_xlabel("Predicted Fare")
b.set_ylabel("Given Fare")

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicti
print('R Square:', metrics.r2_score(y_test, predictions))
#print("------------------" + str(a) + "--------------------")
```

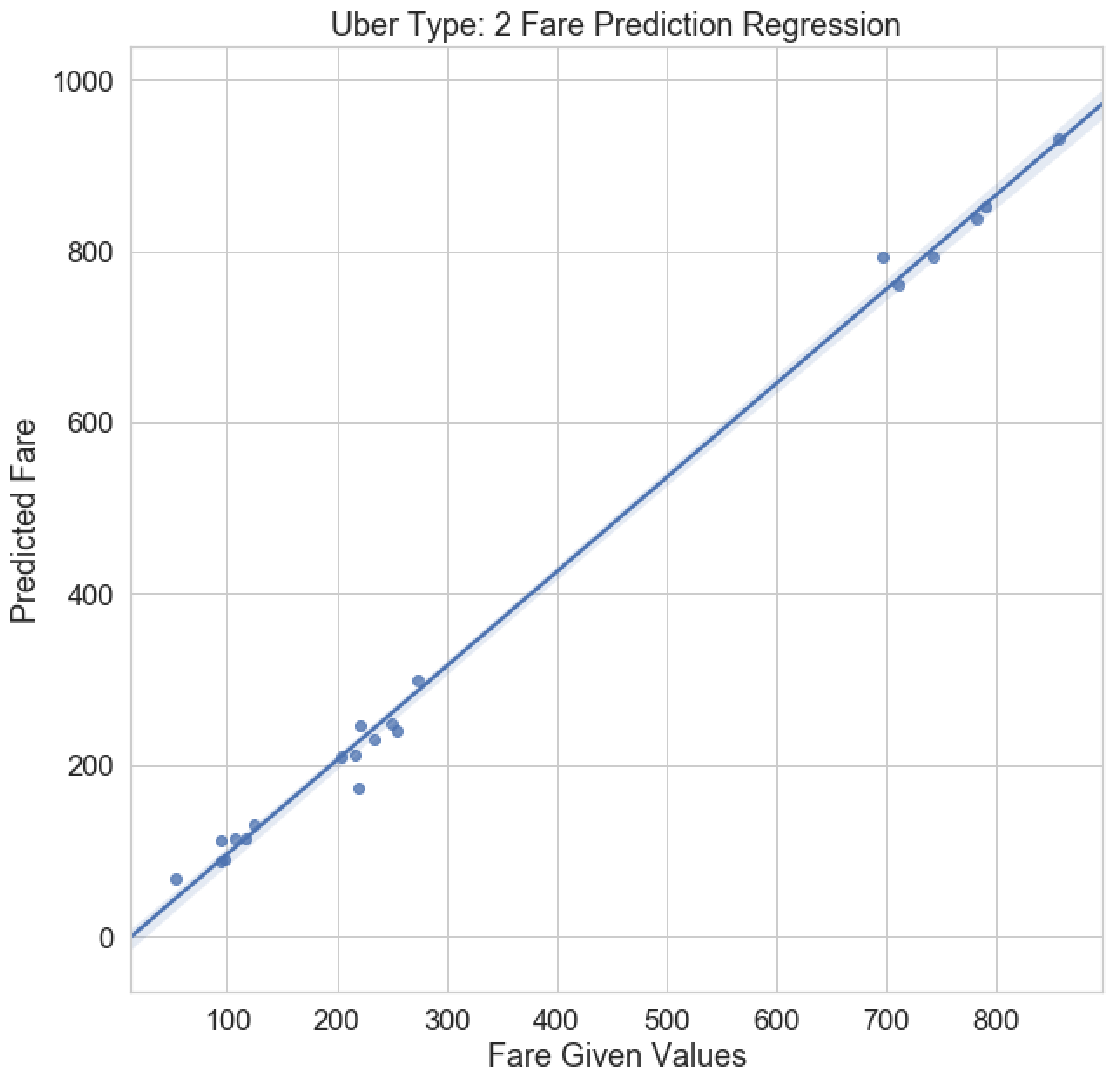
```
Mean Absolute Error: 1.0478266848550952
Mean Squared Error: 1.9947125811336175
Root Mean Squared Error: 1.4123429403419048
R Square: 0.9894120925916262
```




```
In [50]: # MODEL FOR UBER TYPE: 2 ----------------------------------------------
         x = model_data_ut2.iloc[:,:-1] #creating x variable for training
         y = model_data_ut2['Fare$'] #creating y variable for training

         #import random

         #for i in range(0,100):
         #a = random.randint(0,102)
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state

         lm_ut2 = LinearRegression()
         reg = lm_ut2.fit(x_train,y_train)
         predictions = lm_ut2.predict(x_test)
```

```python
plt.subplots(figsize=(10,10))
# plt.scatter(y_test,predictions,color="red")
sb.regplot(x=y_test, y=predictions)

plt.title("Uber Type: 2 Fare Prediction Regression")
plt.xlabel("Fare Given Values")
plt.ylabel("Predicted Fare")

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicti
print('R Square:', metrics.r2_score(y_test, predictions))
#print("------------------" + str(a) + "--------------------")
```

```
Mean Absolute Error: 27.410901166249502
Mean Squared Error: 1476.8268613421596
Root Mean Squared Error: 38.42950508843641
R Square: 0.9806146378732044
```



4.2.3 Predicting & Imputing for missing 'Fare$' values using the models build:

```
In [51]: miss_ut0 = miss_df.loc[(miss_df['Uber Type'] == 0) & (miss_df['Fare$'].isna()),:]

         x = miss_ut0.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_facto

         predictions = lm_ut0.predict(x)

         miss_ut0.loc[:,'Fare$'] = predictions
         miss_ut0.reset_index(inplace=True)

         for i in range(0,len(miss_ut0)):
             miss_df.loc[miss_df["Unnamed: 0"] == miss_ut0.iloc[i]["Unnamed: 0"],"Fare$"] = mis
```

In [52]:
```
miss_ut1 = miss_df.loc[(miss_df['Uber Type'] == 1) & (miss_df['Fare$'].isna()),:]

x = miss_ut1.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_facto

predictions = lm_ut1.predict(x)

miss_ut1.loc[:,'Fare$'] = predictions
miss_ut1.reset_index(inplace=True)

for i in range(0,len(miss_ut1)):
    miss_df.loc[miss_df["Unnamed: 0"] == miss_ut1.iloc[i]["Unnamed: 0"],"Fare$"] = mis
```

In [53]:
```
miss_ut2 = miss_df.loc[(miss_df['Uber Type'] == 2) & (miss_df['Fare$'].isna()),:]

x = miss_ut2.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_facto

predictions = lm_ut2.predict(x)

miss_ut2.loc[:,'Fare$'] = predictions
miss_ut2.reset_index(inplace=True)

for i in range(0,len(miss_ut2)):
    miss_df.loc[miss_df["Unnamed: 0"] == miss_ut2.loc[i, "Unnamed: 0"],["Fare$"]] = mi
```

0.5 Step 5: Finding Outliers (if any)

In [54]:
```
outlier_df.head(5)
```

Out[54]:
```
   Unnamed: 0  Unnamed: 0.1  Uber Type  Origin Region  Destination Region  \
0           0  ID1717434181          0              3                   1
1           1  ID1773039045          0              1                   2
2           2  ID1239989104          0              6                   9
3           3  ID5826371523          2              1                   8
4           4  ID1551372482          0              6                   1

   Origin Latitude  Origin Longitude  Destination Latitude  \
0       -37.818138        144.969210            -37.801461
1       -37.813124        144.940103            -37.818970
2       -37.787433        144.980377            -38.110916
3       -37.817975        144.951207            -37.815834
4       -37.790797        144.985865            -37.814852

   Destination Longitude  Journey Distance(m) Departure Date Departure Time  \
0             144.958161               2688.0     2018-02-22       08:04:10
1             144.946595               1085.0     2018-01-02       23:21:27
2             144.654173              47033.0     2018-06-10       05:35:50
3             145.046450               8909.0     2018-02-14       07:33:23
4             144.945010               5321.0     2018-02-16       17:39:40
```

```
   Travel Time(s) Arrival Time   Fare$
0          668.40      8:15:18    5.44
1          268.56     23:25:55   10.92
2        11350.50      8:45:00   34.64
3         2464.98      8:14:27  160.91
4         1505.82     18:04:45    9.47
```

In [55]:
```
outlier_df.describe()
```

Out[55]:
```
       Unnamed: 0   Uber Type  Origin Region  Destination Region  \
count  110.000000  110.000000     110.000000          110.000000
mean    54.500000    0.609091       5.400000            4.990909
std     31.898276    0.691965       2.552926            2.600266
min      0.000000    0.000000       1.000000            1.000000
25%     27.250000    0.000000       3.250000            3.000000
50%     54.500000    0.000000       5.000000            5.000000
75%     81.750000    1.000000       8.000000            7.000000
max    109.000000    2.000000       9.000000            9.000000

       Origin Latitude  Origin Longitude  Destination Latitude  \
count       110.000000        110.000000            110.000000
mean        -37.857197        144.922784            -37.854792
std           0.110472          0.121296              0.100414
min         -38.110916        144.654173            -38.110916
25%         -37.822244        144.927584            -37.861835
50%         -37.813130        144.952488            -37.815834
75%         -37.803085        144.990006            -37.806549
max         -37.773803        145.046450            -37.773803

       Destination Longitude  Journey Distance(m)  Travel Time(s)       Fare$
count             110.000000           110.000000      110.000000  110.000000
mean              144.927376         18316.890909     4699.745455   55.364273
std                 0.111762         18094.827817     4343.303921  138.250853
min               144.654173          1085.000000      268.560000    2.490000
25%               144.905716          5535.750000     1415.280000   10.335000
50%               144.951796          8691.500000     2451.600000   16.355000
75%               144.991124         42923.500000    10325.190000   31.287500
max               145.046450         51061.000000    13204.980000  796.910000
```

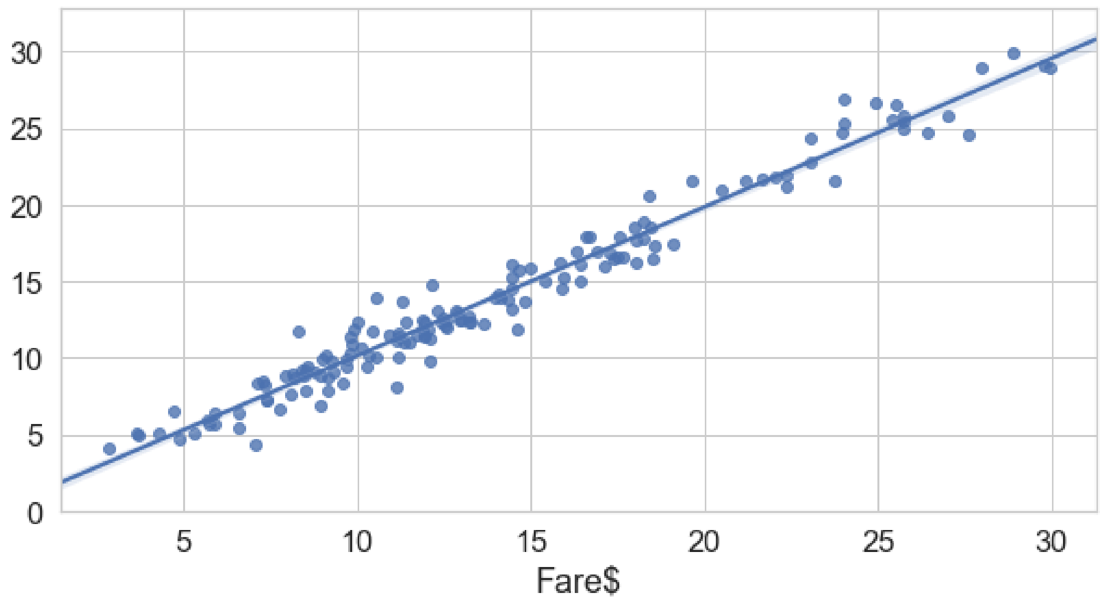
In [56]:
```
# adding day factors and time factors based on Departure date and time
day_classifier(outlier_df)
time_classifier(outlier_df)
```

For Uber Type 0 (finding outliers):

In [57]:
```
import statistics
x = model_data_ut0.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time
y = model_data_ut0["Fare$"]
```

```
# predicting fare for the whole dirty data (cleansed) + missing data (with non imputed
predictions = lm_ut0.predict(x)
model_data_ut0["predicted Fare$"] = predictions

# plotting the given fare vs. predicted fare
plt.subplots(figsize=(10,5))
sb.regplot(model_data_ut0["Fare$"],predictions)

# calculating the residual between the actual value and fitted value
model_data_ut0["Residual"] = abs(model_data_ut0["Fare$"] - model_data_ut0["predicted F

# calculating mean and std for residual
mean0 = statistics.mean(model_data_ut0["Residual"])
std0 = statistics.pstdev(model_data_ut0["Residual"])

print("Residual Mean: "+ str(mean0))
print("Residual std: "+ str(std0))
```

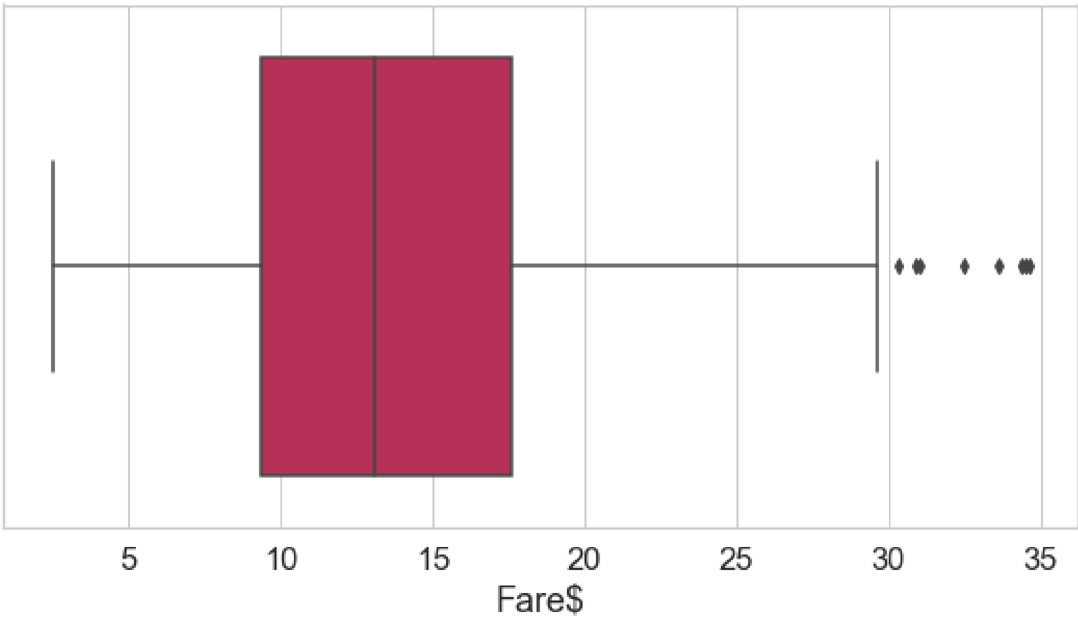
```
Residual Mean: 0.8915010182565926
Residual std: 0.7530309137853527
```




```
In [58]: # taking outlier data for uber type 0
out_ut0 = outlier_df.loc[outlier_df['Uber Type'] == 0,:]
x = out_ut0.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_facto
y = out_ut0["Fare$"]
```

```python
# predicting fare for the outlier dataset using our model
predictions = lm_ut0.predict(x)
out_ut0['predicted Fare$'] = predictions
#out_ut0.reset_index(inplace=True)

# plotting the given fare vs. predicted fare
plt.subplots(figsize=(10,5))
#sb.regplot(out_ut0["Fare$"],out_ut0['predicted Fare$'])
sb.boxplot('Fare$', data=out_ut0, palette="rocket")

# calculating the residual between the actual value and fitted value
out_ut0["Residual"] = abs(out_ut0["Fare$"] - out_ut0["predicted Fare$"])

# finding & dropping the outliers based on 3 sigma rejection rule (each residual is s
for i in out_ut0.index:
    if abs(out_ut0.loc[i]["Residual"] - mean0) > (3*std0):
        print(out_ut0.loc[i]["Fare$"], out_ut0.loc[i]['predicted Fare$'],  out_ut0.loc
        out_ut0.drop(i, 0, inplace=True)
        outlier_df.drop(i, 0, inplace=True)
```

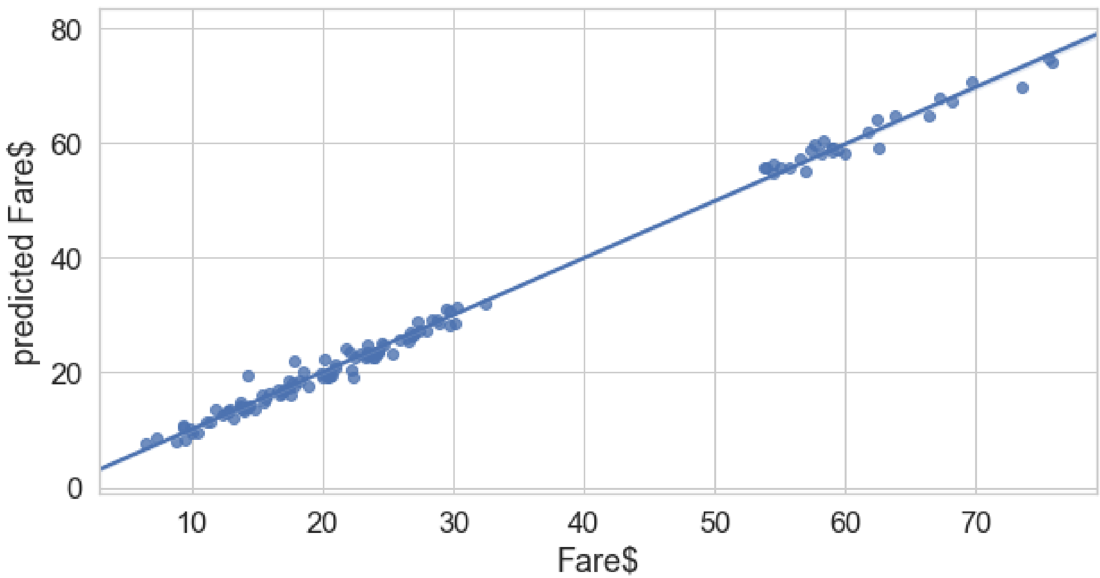
```
34.64 30.311952983494404 4.328047016505597
15.81 26.87649787789685 11.066497877896849
7.225 11.768156788516631 4.543156788516631
32.49 27.4616572263414 5.028342773658601
10.94 18.41986676740406 7.47986676740406
25.34 21.668131488204235 3.671868511795765
30.9 26.08165681827889 4.818343181721108
22.89 18.6112127187979 4.278787281202099
9.805 15.260892912758964 5.455892912758964
12.66 7.741652120994496 4.918347879005504
17.045 27.88752860327454 10.842528603274538
8.14 14.858365618388962 6.718365618388962
13.06 20.075021282767832 7.015021282767831
9.025 16.543976917832115 7.518976917832115
33.61 27.65967106670565 5.950328933294351
34.52 28.922483480235663 5.597516519764341
31.01 25.287472329230777 5.7225276707692245
14.265 23.345175911779265 9.080175911779264
27.96 23.187555685987263 4.772444314012738
30.33 26.389444669385867 3.9405553306141314
29.62 23.03053430937956 6.58946569062044
34.39 29.378429186260494 5.011570813739507
15.18 11.483351661628236 3.696648338371764
26.01 21.72727851872945 4.282721481270553
```



For Uber Type 1 (finding outliers):

```
In [59]: x = model_data_ut1.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time
         y = model_data_ut1["Fare$"]

         predictions = lm_ut1.predict(x)

         model_data_ut1["predicted Fare$"] = predictions
         model_data_ut1["Residual"] = abs(model_data_ut1["Fare$"] - model_data_ut1["predicted F
         model_data_ut1.head(5)

         # plotting the given fare vs. predicted fare
         plt.subplots(figsize=(10,5))
         sb.regplot(model_data_ut1["Fare$"],model_data_ut1["predicted Fare$"])

         # calculating mean and std for residual
         mean1 = statistics.mean(model_data_ut1["Residual"])
         std1 = statistics.pstdev(model_data_ut1["Residual"])

         print("Residual Mean: "+ str(mean1))
         print("Residual std: "+ str(std1))
```

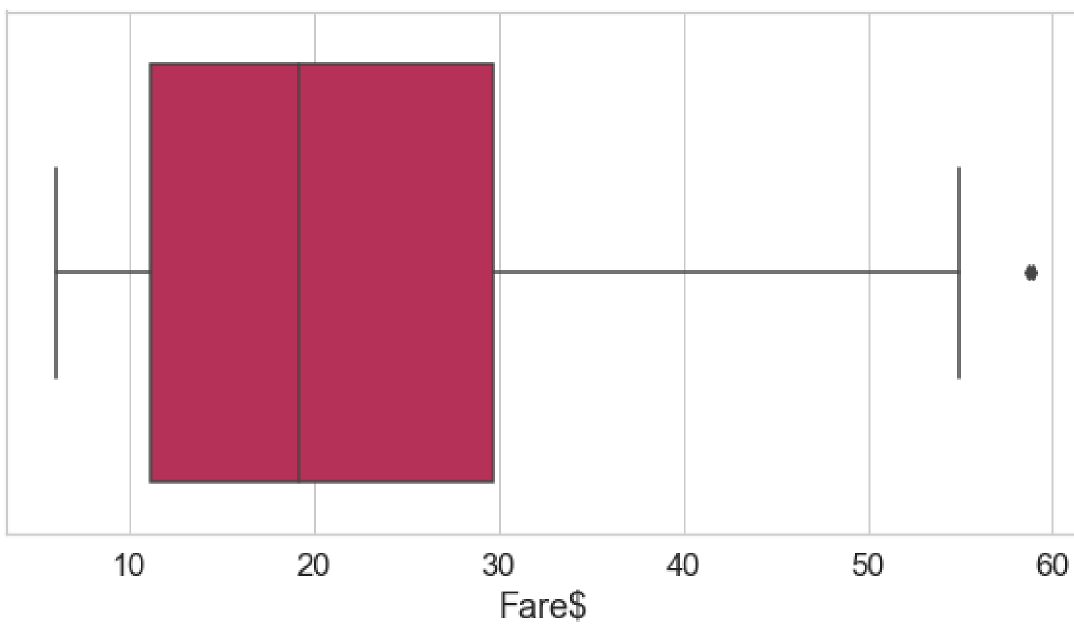
```
Residual Mean: 0.9683723817220853
Residual std: 0.8551542406195689
```




```
In [60]: out_ut1 = outlier_df.loc[outlier_df['Uber Type'] == 1,:]

         x = out_ut1.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_factor
         y = out_ut1["Fare$"]

         predictions = lm_ut1.predict(x)

         out_ut1['predicted Fare$'] = predictions
         #out_ut1.reset_index(inplace=True)

         out_ut1["Residual"] = abs(out_ut1["Fare$"] - out_ut1["predicted Fare$"])

         # plotting the given fare vs. predicted fare
         plt.subplots(figsize=(10,5))
         #sb.regplot(out_ut1["Fare$"],out_ut1["predicted Fare$"])
         sb.boxplot('Fare$', data=out_ut1, palette="rocket")

         for i in out_ut1.index:
             if abs(out_ut1.loc[i]["Residual"] - mean1) > (3*std1):
                 print(out_ut1.loc[i]["Fare$"], out_ut1.loc[i]['predicted Fare$'],  out_ut1.loc
                 out_ut1.drop(i, 0, inplace=True)
                 outlier_df.drop(i, 0, inplace=True)
```

```
24.71 59.941615621359816 35.231615621359815
49.88 60.46176072988695 10.581760729886945
29.69 34.544428177058194 4.8544281770581925
31.38 35.35343829673971 3.9734382967397117
```

```
54.95 67.1840899082007 12.234089908200701
53.81 62.061939841446744 8.251939841446742
43.97 54.052970925890094 10.082970925890095
9.4 20.725492795302436 11.325492795302436
52.46 62.699743001436346 10.239743001436345
6.965 16.92427380127309 9.959273801273088
58.89 68.01466076429216 9.124660764292159
11.355 25.54959529940397 14.194595299403968
26.95 31.49292312396113 4.542923123961131
46.44 56.59000726351887 10.150007263518873
58.72 71.55622822736086 12.836228227360863
49.13 59.49888606901527 10.368886069015268
21.93 25.8220394673005 3.892039467300499
8.49 20.564830542098537 12.074830542098537
8.795 20.44568222779455 11.65068222779455
```

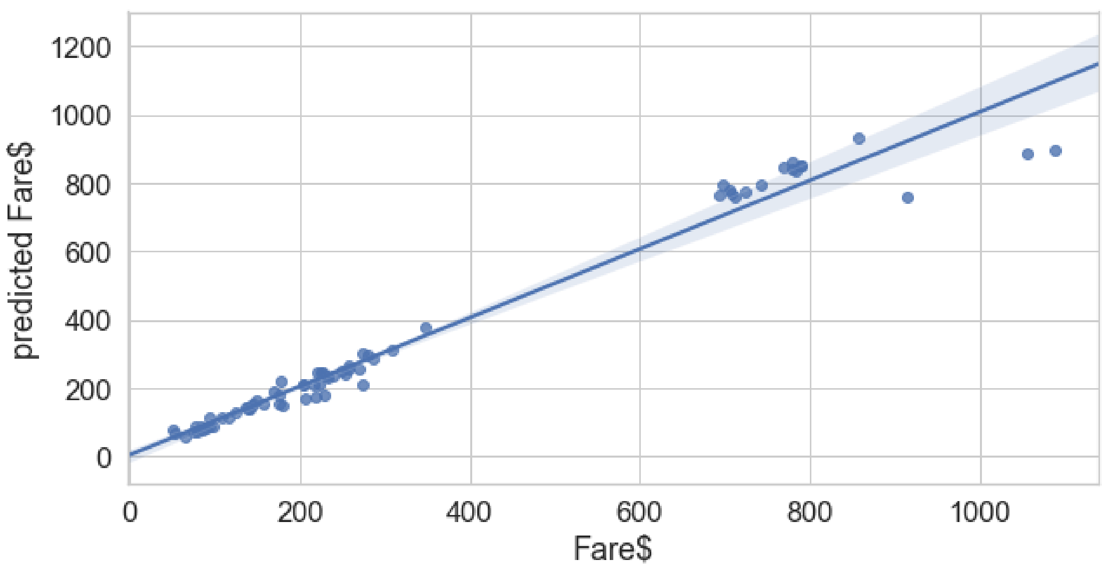


For Uber Type 2 (finding outliers):

```
In [61]: x = model_data_ut2.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time
         y = model_data_ut2["Fare$"]

         predictions = lm_ut2.predict(x)

         model_data_ut2["predicted Fare$"] = predictions
         model_data_ut2["Residual"] = abs(model_data_ut2["Fare$"] - model_data_ut2["predicted F
```

```
model_data_ut2.head(5)

# plotting the given fare vs. predicted fare
plt.subplots(figsize=(10,5))
sb.regplot(model_data_ut2["Fare$"],model_data_ut2["predicted Fare$"])

mean2 = statistics.mean(model_data_ut2["Residual"])
std2 = statistics.pstdev(model_data_ut2["Residual"])

print("Residual Mean: "+ str(mean2))
print("Residual std: "+ str(std2))
```

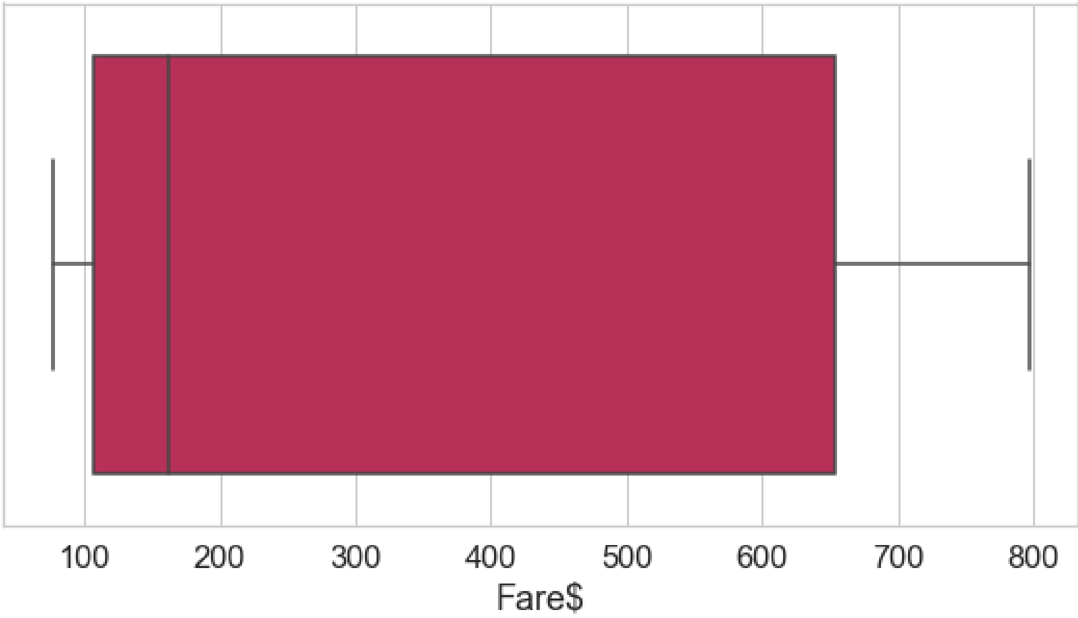
```
Residual Mean: 31.415780245421207
Residual std: 39.033182295629295
```



In [62]:
```
out_ut2 = outlier_df.loc[outlier_df['Uber Type'] == 2,:]

x = out_ut2.loc[:,['Journey Distance(m)', 'Travel Time(s)', 'day_factor', 'time_facto
y = out_ut2["Fare$"]

predictions = lm_ut2.predict(x)

out_ut2['predicted Fare$'] = predictions
out_ut2["Residual"] = abs(out_ut2["Fare$"] - out_ut2["predicted Fare$"])

plt.subplots(figsize=(10,5))
sb.boxplot('Fare$', data=out_ut2, palette="rocket")
```

```
for i in out_ut2.index:
    if abs(out_ut2.loc[i]["Residual"] - mean2) > (3*std2):
        print(out_ut2.loc[i]["Fare$"], out_ut2.loc[i]['predicted Fare$'],  out_ut2.loc
        out_ut2.drop(i, 0, inplace=True)
        outlier_df.drop(i, 0, inplace=True)
```

```
724.47 888.8372694460911 164.3672694460911
```



0.5.1 Step 6: Generating final output files

29857082_dirty_data_solution.csv
29857082_missing_value_solution.csv
29857082_outliers_solution.csv

6.1 Changing the dtypes back to original dtypes

```
In [63]: dirty_df["Arrival Time"] = dirty_df["Arrival Time"].astype("str")
         dirty_df["Departure Time"] = dirty_df["Departure Time"].astype("str")
```

6.2 Deleting additional column added in the datasets

```
In [64]: del dirty_df["day_factor"]
         del dirty_df["time_factor"]
         del dirty_df["corrected"]
```

```
In [65]: del miss_df["day_factor"]
         del miss_df["time_factor"]

         del outlier_df["day_factor"]
         del outlier_df["time_factor"]
```

6.3 Generating the solution files for dirty_data, missing_values and outliers

```
In [67]: dirty_df.to_csv("dirty_data_solution.csv", sep=',')
         miss_df.to_csv("missing_value_solution.csv", sep=',')
         outlier_df.to_csv("outliers_solution.csv", sep=',')
```

0.5.2 Step 7: Summary

After doing the EDA we found certain errros and inconsistencies the dirty data which we fixed (Approx. in 77 records). The errors which we figured out are as follows: 1. Invalid origin latitude 2. Invalid destination latitude 3. Departure Date (month and date swapped) 4. Departure Date (exceeding the dates of month) 5. Additional Uber Type 6. Uber Type 1, 2 (some ids not following the pattern) 7. Wrong region allocated 8. Wrong journey distance 9. Wrong travel time 10. Wrong arrival time

Then we tried to impute the missing values for Uber Type and Fare$ based on the linear model we generated as Fare depends on 4 other dimensions like (Journey distance, Travel time, Day Factor, Time Factor).

Then we tried to detect the outliers based on different types of Uber based on 3 sigma rejection rule and removed the outliers.