Business Case: Delhivery

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021. They aim to build the operating system for commerce, through a combination of world-class infrastructure, logistics operations of the highest quality, and cutting-edge engineering and technology capabilities.

Problem: The company wants to understand and process the data coming out of data engineering pipelines:

Our Approach:

• Clean, sanitize and manipulate data to get useful features out of raw fields

• Make sense out of the raw data and help the data science team to build forecasting models on it

1. Approach 1: Basic data cleaning and exploration:
    - Handle missing values in the data.
    - Analyze the structure of the data.
    - Try merging the rows.
2. Appoach 2: Build some features to prepare the data for actual analysis. Extract features from the below fields:
    - Destination Name: Split and extract features out of destination. City-place-code (State)
    - Source Name: Split and extract features out of destination. City-place-code (State)
    - Trip_creation_time: Extract features like month, year and day etc
3. Approach 3: In-depth analysis and feature engineering:
    - Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required
    - Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.
    - Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value.
    - Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value.
    - Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value.
    - Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value.
    - Find outliers in the numerical variables (you might find outliers in almost all the variables), and check it using visual analysis
    - Handle the outliers using the IQR method.
    - Do one-hot encoding of categorical variables (like route_type)
    - Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.

Approach 1:

Basic data cleaning and exploration:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm,levene,shapiro,mannwhitneyu,ttest_ind,ttest_rel
import statsmodels.api as sm
import pylab as py
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler,StandardScaler
```

```
df=pd.read_csv("del.csv")
df.info()
```

```
<ipython-input-4-2c837621b3fd>:1: DtypeWarning: Columns (12) have mixed types. Specify dtype option on import or set low_memor
  df=pd.read_csv("del.csv")
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57345 entries, 0 to 57344
Data columns (total 24 columns):
```

```
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            57345 non-null  object
 1   trip_creation_time              57345 non-null  object
 2   route_schedule_uuid             57345 non-null  object
 3   route_type                      57344 non-null  object
 4   trip_uuid                       57344 non-null  object
 5   source_center                   57344 non-null  object
 6   source_name                     57211 non-null  object
 7   destination_center              57344 non-null  object
 8   destination_name                57248 non-null  object
 9   od_start_time                   57344 non-null  object
 10  od_end_time                     57344 non-null  object
 11  start_scan_to_end_scan          57344 non-null  float64
 12  is_cutoff                       57344 non-null  object
 13  cutoff_factor                   57344 non-null  float64
 14  cutoff_timestamp                57344 non-null  object
 15  actual_distance_to_destination  57344 non-null  float64
 16  actual_time                     57344 non-null  float64
 17  osrm_time                       57344 non-null  float64
 18  osrm_distance                   57344 non-null  float64
 19  factor                          57344 non-null  float64
 20  segment_actual_time             57344 non-null  float64
 21  segment_osrm_time               57344 non-null  float64
 22  segment_osrm_distance           57344 non-null  float64
 23  segment_factor                  57344 non-null  float64
dtypes: float64(11), object(13)
memory usage: 10.5+ MB
```

Obeservation

From the above table, we can see that we have:

1. Total 144866 RAWS including the raws having NULL values.
2. Columns 1 t0 10 are of the "object" data type since they include IDs or a mix of numeric and alphabetic characters.
3. Columns 15 to 23, we have "float" data type as they contains data such as: Time, Date, Distance, Etc
4. We don't know what's in columns 12 to 14 and 19 & 23. Therefore, we will eliminate these columns from our dataframe.

```
t=df.drop(columns=["is_cutoff","cutoff_factor","cutoff_timestamp","factor","segment_factor"])
t.dropna(inplace=True)
t.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57115 entries, 0 to 57343
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   data                            57115 non-null  object
 1   trip_creation_time              57115 non-null  object
 2   route_schedule_uuid             57115 non-null  object
 3   route_type                      57115 non-null  object
 4   trip_uuid                       57115 non-null  object
 5   source_center                   57115 non-null  object
 6   source_name                     57115 non-null  object
 7   destination_center              57115 non-null  object
 8   destination_name                57115 non-null  object
 9   od_start_time                   57115 non-null  object
 10  od_end_time                     57115 non-null  object
 11  start_scan_to_end_scan          57115 non-null  float64
 12  actual_distance_to_destination  57115 non-null  float64
 13  actual_time                     57115 non-null  float64
 14  osrm_time                       57115 non-null  float64
 15  osrm_distance                   57115 non-null  float64
 16  segment_actual_time             57115 non-null  float64
 17  segment_osrm_time               57115 non-null  float64
 18  segment_osrm_distance           57115 non-null  float64
dtypes: float64(8), object(11)
memory usage: 8.7+ MB
```

The Columns now reduced to 18 and Raws are reduced to 144316 after removing Unknown Columns and Raws containing Null values

1.(c)

Let's merge raws based on columns

```
t.groupby("trip_uuid")["trip_uuid"].nunique().sum()
```

```
5848
```

```
t.groupby("source_center")["source_center"].nunique().sum()
```

```
1345
```

```
t.groupby("destination_center")["destination_center"].nunique().sum()
```

```
1308
```

Obeservation

1. We have total 14,787 different trips
2. Total 1,496 Unique Source location
3. And 1,466 different destinations in total

Approach 2

Build some features to prepare the data for actual analysis. Extract features from the fields

2.(a). Destination Name: Split and extract features out of destination. City-place-code (State)

```
y1=t["destination_name"].astype("string")
y1=y1.str.split("_",expand=True)
y1=y1.drop([3],axis=1)
y1=y1.rename(columns={0:"D_city",1:"D_place",2:"D_code (State)"})
y1
```

	D_city	D_place	D_code (State)
0	Khambhat	MotvdDPP	D (Gujarat)
1	Khambhat	MotvdDPP	D (Gujarat)
2	Khambhat	MotvdDPP	D (Gujarat)
3	Khambhat	MotvdDPP	D (Gujarat)
4	Khambhat	MotvdDPP	D (Gujarat)
...	...	...	...
57339	Bangalore	Nelmngla	H (Karnataka)
57340	Wankaner	JivanDPP	D (Gujarat)
57341	Wankaner	JivanDPP	D (Gujarat)
57342	Wankaner	JivanDPP	D (Gujarat)
57343	Sambhal	Khwssrai	D (Uttar Pradesh)

57115 rows × 3 columns

We got a DataFrame consisting of "City", "Place", "Code (State)" from the column "destination_name"

-----Now we can merge this "y1" DataFrame with our "t" DataFrame, And named as x1 -----

```
x1=pd.concat([t,y1],axis=1)
x1.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364764
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364764
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364764
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	15374109364764
		2018-09-20	thanos::sroute:eb7bfc78-		

.

5 rows × 22 columns

Now let's see how many different cities are there in Destination

```
x1.groupby("D_city")["D_city"].nunique().cumsum()
```

```
D_city
AMD             1
Achrol          2
Addanki         3
Adoor           4
Agartala        5
             ...
Yavatmal     1138
Yellandu     1139
Yellareddy   1140
Zahirabad    1141
Zirakpur     1142
Name: D_city, Length: 1142, dtype: int64
```

There are total 1,256 different cities in Destination

2(b). Source Name: Split and extract features out of Source. City-place-code (State)

```
y2=t["source_name"].astype("string")
y2=y2.str.split("_",expand=True)
y2=y2.drop([3],axis=1)
y2=y2.rename(columns={0:"S_city",1:"S_place",2:"S_code (State)"})
y2
```

	S_city	S_place	S_code (State)
0	Anand	VUNagar	DC (Gujarat)
1	Anand	VUNagar	DC (Gujarat)
2	Anand	VUNagar	DC (Gujarat)
3	Anand	VUNagar	DC (Gujarat)
4	Anand	VUNagar	DC (Gujarat)
...	...	...	...
57339	Surat	HUB (Gujarat)	<NA>
57340	Morbi	DC (Gujarat)	<NA>
57341	Morbi	DC (Gujarat)	<NA>
57342	Morbi	DC (Gujarat)	<NA>
57343	Rampur	RoshnBgh	I (Uttar Pradesh)

57115 rows × 3 columns

We got a DataFrame consisting of "City", "Place", "Code (State)" from the column "source_name"

——Now we can merge this "y2" DataFrame with our "x1" DataFrame, And named as x ——

```
x=pd.concat([x1,y2],axis=1)
x.head(5)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093647649
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093647649
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093647649
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093647649
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093647649

5 rows × 25 columns

.

Now let's see how many different cities are there in Source

```
x.groupby("S_city")["S_city"].nunique().cumsum()
```

```
S_city
AMD               1
Achrol            2
Addanki           3
Adoor             4
Agra              5
               ...
Weir           1147
YamunaNagar    1148
Yellandu       1149
Yellareddy     1150
Zahirabad      1151
Name: S_city, Length: 1151, dtype: int64
```

There are total 1,260 different cities in Source

2(c).Trip_creation_time: Extract features like month, year and day etc

```
x["trip_creation_time"]=pd.to_datetime(x["trip_creation_time"])
x["month"]=x["trip_creation_time"].dt.month
x["year"]=x["trip_creation_time"].dt.year
x["day"]=x["trip_creation_time"].dt.day
x
```

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	153741093
...	...	...	...	...	
57339	training	2018-09-18 21:36:33.830812	thanos::sroute:c094f55b-3c48-4ce4-b649-b99a4af...	FTL	153730659
57340	training	2018-09-25	thanos::sroute:7fcb59da-b2ba-4425-98dd-	Carting	
		22:10:52.086977	9ea8d66		1537913452
57342	training		b2ba-4425-98dd-	Carting	

- First we have **converted** the column "trip_creaton_time" from **Object** to **Datetime** datatype
- Then we extract **month, year, day** seperately from the above column and named them seperately

Approach 3:

In-depth analysis and feature engineering:

57115 rows × 28 columns

3(a). Calculate the time taken between od_start_time and od_end_time and keep it as a feature.

1. Let's change the data-type of columns "od_start_time" and "od_end_time" to Datetime format.

```
x["od_start_time"]=pd.to_datetime(x["od_start_time"])
x["od_end_time"]=pd.to_datetime(x["od_end_time"])
```

2. Now we can calculate the time difference between end and start time from these columns and name this feature as "od_time_taken".

```
x["od_time_taken"]=(x["od_end_time"]-x["od_start_time"])
```

3. Let's covert this time taken into hours and minutes.

```
x["time_hour"]=x["od_time_taken"].dt.total_seconds() / 3600
x["time_min"]=round(x["od_time_taken"].dt.total_seconds() / 60 ,1)
```

```
x
```

	data	trip_creation_time	route_schedule_uuid	route_type	t
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	1537410936
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	1537410936
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	1537410936
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	1537410936
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	1537410936
...	...	...	...	...	
57339	training	2018-09-18 21:36:33.830812	thanos::sroute:c094f55b-3c48-4ce4-b649-b99a4af...	FTL	1537306593
57340	training	2018-09-25 22:10:52.086977	thanos::sroute:7fcb59da-b2ba-4425-98dd-9ea8d66...	Carting	1537913452
57341	training	2018-09-25 22:10:52.086977	thanos::sroute:7fcb59da-b2ba-4425-98dd-9ea8d66...	Carting	1537913452
		2018-09-25	thanos::sroute:7fcb59da-		

We got two new features as "time-hour" and "time_min".

thanos::sroute:d1ee5ba3-

3(b).Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ Visual analysis to check.

As we have two Samples that is "time_min" and "start_scan_to_end_scan" to analysis, here we will use **"Two-sample-t-test"** but before applying 2-sample-ttest we have to check whether these samples fullfill the requirements for 2-sample-ttest or not. If they fail then we'll go with **"Mann-Whitney U rank test"**

For that we have to check whether samples are normally distributed or not. And Variance of both the samples are equal or not.

Let's get 1000 Samples randomly from both the columns.

```
x1=np.random.choice(x["time_min"],size=1000)
x2=np.random.choice(x["start_scan_to_end_scan"],size=1000)
```

Let's check the samples are **Normally distributed** or not.

We will set Hypothesis for Normal distribution.

- Null Hypothesis(H0)- Sample is normally distributed
- Alternative Hypothesis(Ha)- Sample is not normally distributed

Alfa= 0.5

For Sample x1

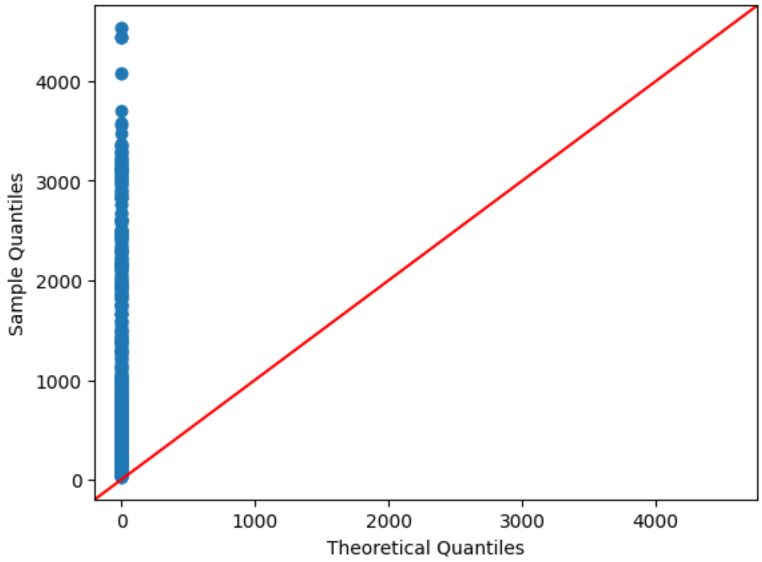
- Shapiro test

```
shapiro(x1)
```

```
ShapiroResult(statistic=0.7976281642913818, pvalue=1.601871682133463e-33)
```

- QQplot test

```
sm.qqplot(x1, line ='45')
py.show()
```



- As from the above Shapiro test **pvalue** is < then **alfa** i.e. < 0.5, So we will **REJECT THE NULL HYPOTHESIS**. Sample "x1" is not normally distributed.
- And QQplot also shows the same.

For Sample x2

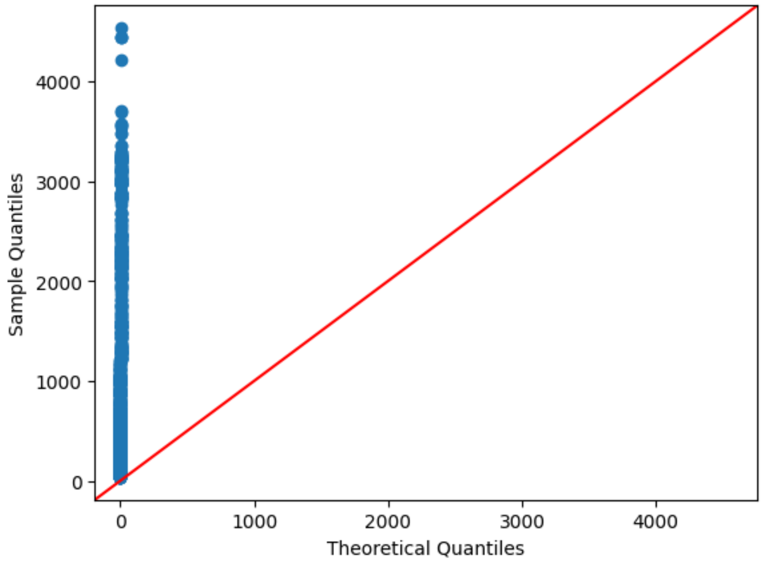
- Shapiro test

```
shapiro(x2)
```

```
ShapiroResult(statistic=0.8000094294548035, pvalue=2.2822246698443713e-33)
```

- QQplot test

```
sm.qqplot(x2, line ='45')
py.show()
```



- As from the above Shapiro test for both samples pvalue is < then alfa i.e. < 0.5, So we will REJECT THE NULL HYPOTHESIS. Sample "x2" is also not normally distributed.

- And QQplot also shows the same.

Let's check or **Variance equal Levene test**

Hypotheses test and the two hypotheses are as follows:

- Ho(Accepted): Samples have equal variance.(Po>0.05)
- Ha(Rejected): Samples have not equal variance.
- Alfa = 0.05

```
levene(x1,x2)
```

```
LeveneResult(statistic=0.0588448767873813, pvalue=0.8083559610378489)
```

Pvalue > 0.05, Hence, Fail to reject the Null Hypothesis. Samples have equal variance.

As our 2 samples didn't fulfill the requirements for 2-sample ttest as it fails for one test among the two different tests, We will go with "Mann whitney U test"

- Mann whitney U test

Hypotheses test and the two hypotheses are as follows:

- Ho(Accepted): Samples have no difference.(Po>0.05)
- Ha(Rejected): Samples have difference.
- Alfa = 0.05

```
mannwhitneyu(x1, x2)
```

```
MannwhitneyuResult(statistic=504827.0, pvalue=0.7085785337652821)
```

Pvalue > Alfa(0.05) FAIL TO REJECT NULL HYPOTHESIS,

- **HENCE, BOTH THE COLUMNS "TIME_MIN" ANS "START_SCAN_TO_END_SCAN" HAVE NO DIFFERENCE**
- **WE CAN USE TIME_MIN COLUMN FOR START_SCAN_ENS_SCAN COLUMN AS WELL**

3(c) Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value

- Let's first merg the rows of "actual_time" and "OSRM_time" on the basis of trip_uuid

```
t1=pd.Series(x.groupby("trip_uuid")["actual_time"].sum())
t2=pd.Series(x.groupby("trip_uuid")["osrm_time"].mean())
```

Let's start the HYPOTHESIS TESTING on these two samples

We will set Hypothesis for ttest.

- Null Hypothesis(H0)- Samples have no difference
- Alternative Hypothesis(Ha)- Sample have difference
- Alfa= 0.5

We will use ttest_rel. As ttest_rel is for related samples and stats

```
ttest_rel(t1,t2)
```

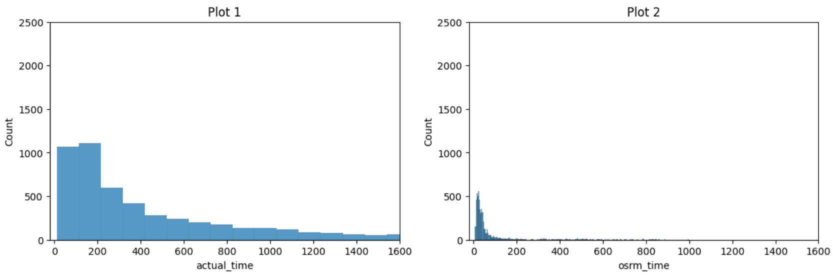
```
TtestResult(statistic=20.399089447346352, pvalue=2.0740407494378327e-89, df=5847)
```

- **As Pvalue < Alfa(0.05), We will Reject the Null Hypothesis.**
- **Hence Samples have difference.**

Let's see it on the plot

```
plt.figure(figsize=(14,4))
plt.subplot(1,2,1)
sns.histplot(t1)
plt.xlim(-20,1600)
plt.ylim(0,2500)
plt.title("Plot 1")

plt.subplot(1,2,2)
sns.histplot(t2)
plt.xlim(-20,1600)
plt.ylim(0,2500)
plt.title("Plot 2")
plt.show()
```



As we can see that they have major differences.

- Hence, both the variables are significantly different from each other
- We cannot predict the time of delivery by OSRM_TIME as ACTUAL_TIME is quite different from OSRM.
- Hence, OSRM_TIME cannot be used for actual time.

3(d).Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value

Let's get first aggregated columns

```
a1=pd.Series(x.groupby("trip_uuid")["actual_time"].sum())
a2=pd.Series(x.groupby("trip_uuid")["segment_actual_time"].mean())
```

Lets start testing the hypothesis on these two Variables

We will set Hypothesis

- Null Hypothesis(H0)- Variables has no difference
- Alternative Hypothesis(Ha)- Variables have difference
- Alfa= 0.5

** We will use ttest_rel. As ttest_rel is for related samples and stats**

```
ttest_rel(a1,a2)
```

```
TtestResult(statistic=20.40719645117746, pvalue=1.776624451390718e-89, df=5847)
```

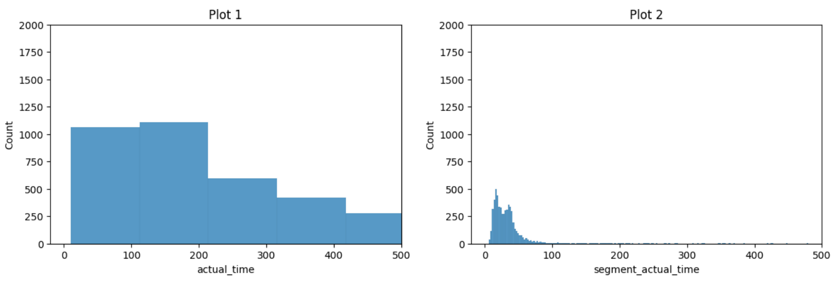
As **Pvalue < Alfa (0.05), we will Reject our NULL HYPOTHESIS**

- That shows that the columns "actual_time" and "segment_time" has differences.
- That means time taken to complete the delivery and time taken by the subset of the package delivery are not same.

Let's see using plot.

```
plt.figure(figsize=(14,4))
plt.subplot(1,2,1)
sns.histplot(a1)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 1")

plt.subplot(1,2,2)
sns.histplot(a2)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 2")
plt.show()
```



As we can see that they have major differences.

- Hence, both the variables are significantly different from each other
- We cannot predict the time of delivery by SEGMENT_ACTUAL_TIME as ACTUAL_TIME is quite different from Segment time.
- Hence, actual time cannot be used for segment_actual time.

3(e).Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value

Let's create the aggregated samples from the "osrm_distance" and "segment_osrm" columns

```
dt1=pd.Series(x.groupby("trip_uuid")["osrm_distance"].sum())
dt2=pd.Series(x.groupby("trip_uuid")["segment_osrm_distance"].mean())
```

Lets start testing the hypothesis on these two samples

We will set Hypothesis

- Null Hypothesis(H0)- Samples have no difference
- Alternative Hypothesis(Ha)- Samples have differences
- Alfa= 0.5

```
ttest_rel(dt1,dt2)
```

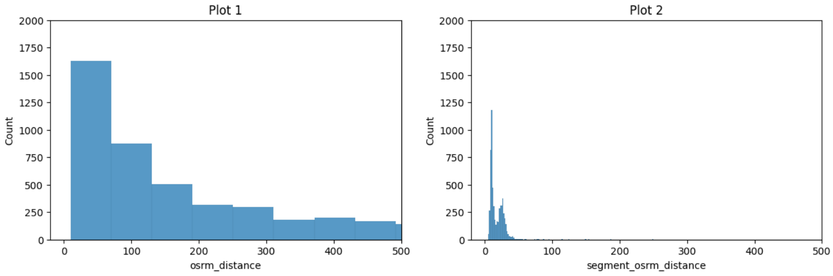
```
TtestResult(statistic=19.8516808870668, pvalue=6.34361262265443e-85, df=5847)
```

As Pvalue < Alfa (0.05), we will Reject our NULL HYPOTHESIS

- That shows that the columns "osrm_distance" and "osrm_segment_distance" has differences.
- That means Distance covered by subset of the package delivery and the shortest path between points in a given map has significant difference.

```
plt.figure(figsize=(14,4))
plt.subplot(1,2,1)
sns.histplot(dt1)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 1")

plt.subplot(1,2,2)
sns.histplot(dt2)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 2")
plt.show()
```



As we can see that they have major differences.

- Hence, both the variables are significantly different from each other
- We cannot predict the distance of SEGMENT distance as Osrm distance is quite different from Segment distance.
- Hence, osrm distance cannot be used for segment distance.

3(f). Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value

Let's create the aggregated samples from the "osrm_time" and "segment_osrm_time" columns

```
gt1=pd.Series(x.groupby("trip_uuid")["osrm_time"].sum())
gt2=pd.Series(x.groupby("trip_uuid")["segment_osrm_time"].mean())
```

Lets start testing the hypothesis on these two samples

We will set Hypothesis

- Null Hypothesis(H0)- Samples have no difference
- Alternative Hypothesis(Ha)- Samples have differences
- Alfa= 0.5

```
ttest_rel(gt1,gt2)
```

```
TtestResult(statistic=20.143966996595896, pvalue=2.6330166160349984e-87, df=5847)
```

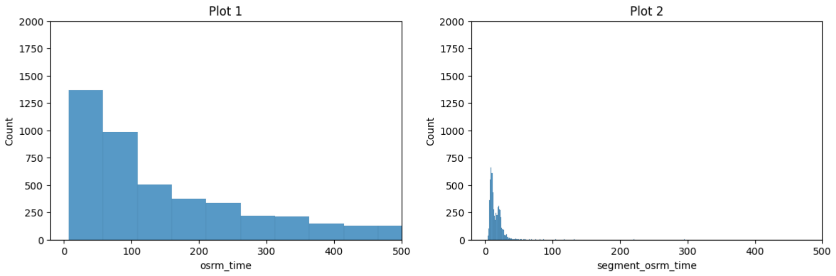
- As Pvalue < Alfa (0.05), we will Reject our NULL HYPOTHESIS

*That shows that the columns "osrm_time" and "segment_osrm_time" has differences.

- That means the shortest path between points in a given map and Time taken by the subset of the package delivery are significantly differ.

```
plt.figure(figsize=(14,4))
plt.subplot(1,2,1)
sns.histplot(gt1)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 1")

plt.subplot(1,2,2)
sns.histplot(gt2)
plt.xlim(-20,500)
plt.ylim(0,2000)
plt.title("Plot 2")
plt.show()
```



Above plot supports the same.

3(g). Find outliers in the numerical variables, and check it using visual analysis

Let's First create a new Dataset having some numerical variables in it.

```
df=x.iloc[:,11:18]
df.describe()
```

	start_scan_to_end_scan	actual_distance_to_destination	actual_time	
count	57115.000000	57115.000000	57115.000000	57
mean	949.626525	230.602326	411.701532	2
std	1024.957065	338.709300	588.802140	3
min	20.000000	9.000267	9.000000	
25%	163.000000	23.396577	52.000000	
50%	454.000000	66.192597	132.000000	
75%	1534.000000	286.511667	509.000000	2
max	4535.000000	1722.009755	4532.000000	16

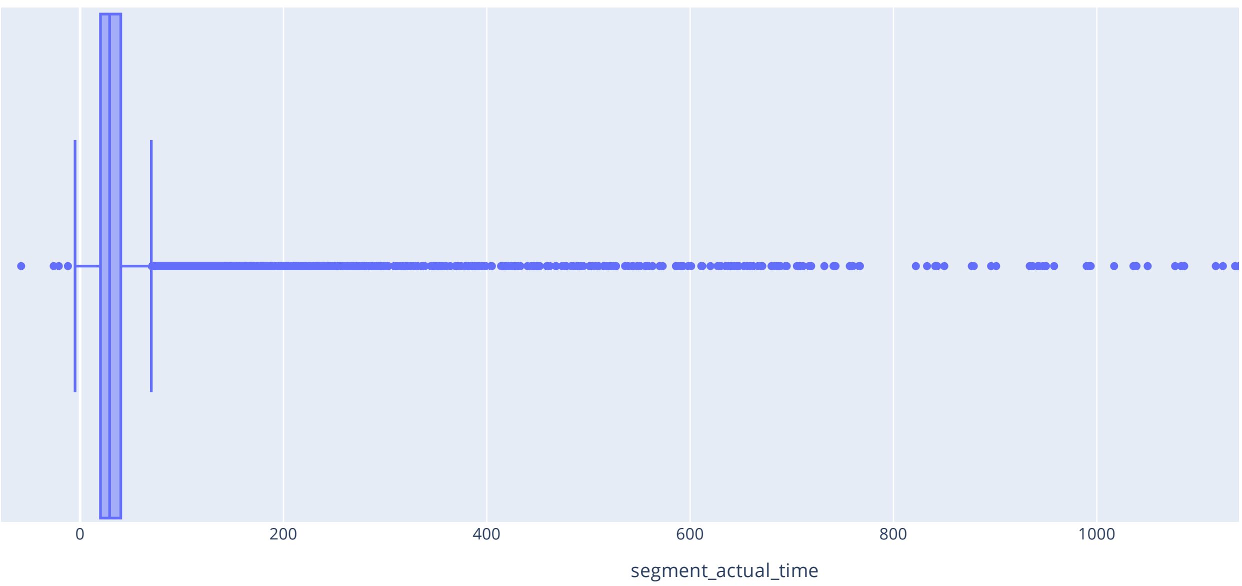
As we can see that MAX of every variable and MEAN of every variable has a huge difference between and that points out that these variables has OUTLIERS

- The mean is sensitive to outliers, but the fact the mean is so small compared to the max value indicates the max value is an outlier.

We will use data visualization techniques to inspect the data's distribution and verify the presence of outliers.

Let's use "segment_actual_time" variable first.

```
fig = px.box(df, x="segment_actual_time")
fig.show()
```

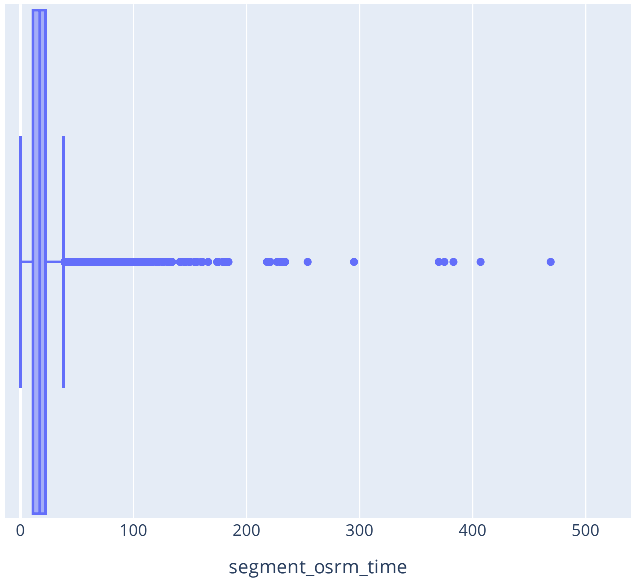


As we can see, there are a lot of outliers. That thick line near 0 is the box part of our box plot. Besides the box and side fence are some points showing outliers. Since the chart is interactive, we can zoom to get a better view of the box and points, and we can hover the mouse on the box to view of the box plot values

Finding outliers in data using a box plot on different variable

```
fig = px.box(df, x="segment_osrm_time")

fig.show()
```



Similarly we can find outliers in all the other variables using a box plot

Let's handle these outliers using the IQR method

We will first handle outliers of one variable and using same method every variable's outliers can handle.

```
def find_outliers_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   return outliers

outliers = find_outliers_IQR(df["segment_actual_time"])

print("number of outliers: "+ str(len(outliers)))

print("max outlier value: "+ str(outliers.max()))

print("min outlier value: "+ str(outliers.min()))

outliers
```

```
number of outliers: 3721
max outlier value: 2351.0
min outlier value: -211.0
21        93.0
34        94.0
72        75.0
```

```
73        78.0
106       79.0
         ...
57256     91.0
57260    148.0
57308    101.0
57333     78.0
57339    204.0
Name: segment_actual_time, Length: 3721, dtype: float64
```

Using the IQR method, we find 3,721 segment_actual_time outliers in the dataset. I printed the min and max values to verify they match the statistics we saw when using the pandas describe() function, which helps confirm we calculated the outliers correctly.

We can also pass multiple variables through the function to get back a dataframe of all rows instead of just the outliers. If the value is not an outlier, it will display as NaN (not a number):

```
outliers = find_outliers_IQR(df[["segment_actual_time","segment_osrm_time"]])

outliers
```

	segment_actual_time	segment_osrm_time
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
...	...	...
57339	204.0	NaN
57340	NaN	NaN
57341	NaN	NaN
57342	NaN	NaN
57343	NaN	NaN

57115 rows × 2 columns

After identifying the outliers, we need to decide what to do with them

There are three techniques we can use to handle outliers:

- Drop the outliers
- Cap the outliers
- Replace outliers using imputation as if they were missing values

We'll go with Droping the outliers

```
def drop_outliers_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   not_outliers = df[~((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   outliers_dropped = not_outliers.dropna().reset_index()

   return outliers_dropped

ddf=drop_outliers_IQR(df)
ddf
```

	index	start_scan_to_end_scan	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_tim
0	0	86.0	10.435660	14.0	11.0	11.9653	14.
1	1	86.0	18.936842	24.0	20.0	21.7243	10.
2	2	86.0	27.637279	40.0	28.0	32.5395	16.
3	3	86.0	36.118028	62.0	40.0	45.5620	21.
4	4	86.0	39.386040	68.0	44.0	54.2181	6.
...	...	...	...	...	...	...	.
45052	57322	1854.0	661.019643	956.0	576.0	763.5108	38.
45053	57340	63.0	9.491364	9.0	16.0	15.2694	9.
45054	57341	63.0	19.494009	23.0	28.0	28.1190	14.
45055	57342	63.0	25.562921	39.0	33.0	33.9582	16.
45056	57343	269.0	23.720797	33.0	26.0	24.5264	33.

Notice the dataframe is only 45057 rows once all the outliers have been dropped from Dataset

---

3(i).Do one-hot encoding of categorical variables (like route_type)

```
df_one_hot_encoded = pd.get_dummies(data = x[["data","trip_uuid","route_type"]], columns = ["route_type"], prefix = "
```

```
print(df_one_hot_encoded )
```

```
           data              trip_uuid  is_Carting  is_FTL
0      training  trip-153741093647649320           1       0
1      training  trip-153741093647649320           1       0
2      training  trip-153741093647649320           1       0
3      training  trip-153741093647649320           1       0
4      training  trip-153741093647649320           1       0
...         ...                      ...         ...     ...
57339  training  trip-153730659383050884           0       1
57340  training  trip-153791345208672550           1       0
57341  training  trip-153791345208672550           1       0
57342  training  trip-153791345208672550           1       0
57343      test  trip-153825759671059666           0       1

[57115 rows x 4 columns]
```

- We have taken a new dataframe having three columns, and we did ONE_HOT_ENCODING on column "route_type".
- We get two FEATURES named "is_Carting" and "is_FTL"

3(j).Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler

Let's create a dataset having numerical variables.

```
tf=x.iloc[:,11:18]
```

Performing MINMAXSCALER

```
trans = MinMaxScaler()
data = trans.fit_transform(tf)
```

```
# convert the array back to a dataframe
dataset = pd.DataFrame(data)
# summarize
print(dataset.describe())
```

```
                  0             1             2             3             4  \
count  57115.000000  57115.000000  57115.000000  57115.000000  57115.000000
mean       0.205897      0.129364      0.089034      0.127751      0.124619
std        0.227012      0.197728      0.130180      0.188797      0.189925
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.031672      0.008404      0.009507      0.013084      0.009587
50%        0.096124      0.033387      0.027194      0.036760      0.032095
75%        0.335327      0.162002      0.110546      0.155140      0.151983
max        1.000000      1.000000      1.000000      1.000000      1.000000

                  5             6
count  57115.000000  57115.000000
mean       0.096444      0.018552
std        0.019843      0.013887
min        0.000000      0.000000
25%        0.090164      0.011033
50%        0.093677      0.017051
75%        0.097970      0.022066
max        1.000000      1.000000
```

We can see that the distributions have been adjusted and that the minimum and maximum values for each variable are now a crisp 0.0 and 1.0 respectively.

Performing STANDARDSCALER

```
trans = StandardScaler()
data = trans.fit_transform(tf)
# convert the array back to a dataframe
dataset = pd.DataFrame(data)
# summarize
print(dataset.describe())
```

```
                  0             1             2             3             4  \
count  5.711500e+04  5.711500e+04  5.711500e+04  5.711500e+04  5.711500e+04
mean   1.990490e-18 -4.976225e-18 -9.952450e-18 -1.990490e-17  7.862436e-17
std    1.000009e+00  1.000009e+00  1.000009e+00  1.000009e+00  1.000009e+00
min   -9.069986e-01 -6.542602e-01 -6.839395e-01 -6.766632e-01 -6.561538e-01
25%   -7.674794e-01 -6.117563e-01 -6.109093e-01 -6.073602e-01 -6.056730e-01
50%   -4.835626e-01 -4.854049e-01 -4.750390e-01 -4.819546e-01 -4.871662e-01
75%    5.701493e-01  1.650673e-01  1.652496e-01  1.450731e-01  1.440778e-01
max    3.498102e+00  4.403246e+00  6.997825e+00  4.620071e+00  4.609134e+00

                  5             6
count  5.711500e+04  5.711500e+04
mean  -4.229791e-17 -1.005197e-16
std    1.000009e+00  1.000009e+00
min   -4.860506e+00 -1.335933e+00
25%   -3.165084e-01 -5.414384e-01
50%   -1.394696e-01 -1.080778e-01
75%    7.691126e-02  2.530561e-01
max    4.553656e+01  7.067416e+01
```

We can see that the distributions have been adjusted and that the mean is a very small number close to zero and the standard deviation is very close to 1.0 for each variable.

Business Insights

Let's see the most preferable Route type for delivery.

```
x.groupby("route_type")["route_type"].count()
```

```
route_type
Carting    17796
FTL        39319
Name: route_type, dtype: int64
```

- As "FLT" is most prefered route type
- Main reason could be - FTL shipments get to the destination sooner
- because - the truck is making no other pickups or drop-offs along the way

Let's Check from where most orders are coming or placed.

```
tp=x.groupby(["D_city","D_place"])["D_city"].count()
tp
```

```
D_city      D_place
AMD         Memnagar (Gujarat)     26
            Satellite (Gujarat)    33
Achrol      BgwriDPP                4
Addanki     Oilmilrd               22
Adoor       Town                    9
                                   ..
Yavatmal    JajuDPP                21
Yellandu    Sudimala               10
Yellareddy  JKRoad                 16
Zahirabad   Mohim                  16
Zirakpur    DC (Punjab)            33
Name: D_city, Length: 1253, dtype: int64
```

```
tp.sort_values(ascending=False)
```

```
D_city      D_place
Gurgaon     Bilaspur    5864
Bangalore   Nelmngla    3808
Bhiwandi    Mankoli     2393
Hyderabad   Shamshbd    2259
Kolkata     Dankuni     1931
                        ...
Luxettipet  ShivaDPP       1
Bellmpalli  BasthDPP       1
Kushinagar  KasyaDPP       1
Kumta       Central        1
Keshiary    MdnprDPP       1
Name: D_city, Length: 1253, dtype: int64
```

- We can see that city- "GURGAON" has the most orders followed by- "Bangalore" and "Bhiwandi"

```
tpp=x.groupby(["S_city","S_place"])["S_city"].count()
tpp.sort_values(ascending=False)
```

```
S_city        S_place
Gurgaon       Bilaspur    9455
Bangalore     Nelmngla    3947
Bhiwandi      Mankoli     3684
Hyderabad     Shamshbd    1616
Pune          Tathawde    1544
                          ...
Bengaluru     Sarjapur       1
Sonepur       Sabalpur       1
Sumerpur      BazarDPP       1
Thirthahalli  NadthiCx       1
Chikhli       KKndrDPP       1
Name: S_city, Length: 1287, dtype: int64
```

- As the higest source of the orders from any places are alse "Gurgaon" followed by "Bangalore" and "Bhiwandi"

Busiest corridor, avg distance between them, avg time taken

Busiest corridors with Average distance and Average time

```
Bu=x.groupby(["source_name","destination_name"]).agg(
    {"trip_uuid":"nunique","actual_distance_to_destination":"mean","start_scan_to_end_scan":"mean"}
    ).sort_values(by="trip_uuid",ascending=False)
bu=Bu.rename(columns={"trip_uuid":"Unique_trip","actual_distance_to_destination":"Avg_distance","start_scan_to_end_sc
bu
```

source_name	destination_name	Unique_trip	Avg_distance	Avg_time
Bangalore_Nelmngla_H (Karnataka)	Bengaluru_Bomsndra_HB (Karnataka)	61	27.452345	274.278008
	Bengaluru_KGAirprt_HB (Karnataka)	54	20.989753	184.158140
Pune_Tathawde_H (Maharashtra)	Bhiwandi_Mankoli_HB (Maharashtra)	53	66.186592	308.241379
Bhiwandi_Mankoli_HB (Maharashtra)	Mumbai Hub (Maharashtra)	48	17.400260	157.637681
Bengaluru_Bomsndra_HB (Karnataka)	Bengaluru_KGAirprt_HB (Karnataka)	46	26.923761	189.956522
...	...	...	...	...
Jind_DC (Haryana)	Gohana_DvlalDPP_D (Haryana)	1	33.357166	88.000000
Ranaghat_ArickDPP_D (West Bengal)	Kolkata_Dankuni_HB (West Bengal)	1	57.667663	926.000000
Joda_Central_D_1 (Orissa)	Karanjia_Sarubali_D (Orissa)	1	45.958988	195.000000
Jorhat_RicMilRd_D (Assam)	Mokokchung_Central_D_1 (Nagaland)	1	36.591362	429.000000
Kolkata_Dankuni_HB (West Bengal)	Bardhaman_Bankura_D (West Bengal)	1	51.272025	346.000000

2215 rows × 3 columns

- **Busiest corridore is "Bangalore_Nelmngla_H (Karnataka)" to "Gurgaon_Bilaspur_HB (Haryana)" with "61" Unique trip IDs having Average Distance "27.45 kms" and with Average time "274.2"**

CONCLUSION & RECOMENDATIONS

- AS "FTL" ROUTE IS MOST PREFERABLE FOR TRANSPORTATION, WE SHOULD FOCUS ON MAKING IT MORE SMOOTH AND INCREASE RELIABILITY ON IT AS IT IS THE FASTEST AS WELL AS NON-STOP ROUTE TYPE.
- OUR FUTURE PLAN AND STRATIGIES SHOULD CONSIDER "FTL" TYPE AS PRIORITY FOR ROUTE TYPE
- THE CITYS WHICH HAS MOST ORDERS TO DELIVER ARE "GURGAON" FOLLOWED BY "Bangalore" AND "Bhiwandi"
- WE SHOULD TRY TO IMPROVE FTL SERVICES IN THESE CITIES AS THEY HAVE THE BUSIEST ROUTES
- SMOOTH FTL SERVICES WILL IMPACT POSITIVE RESULTS IN DELIVERIES WHICH ANYHOW INCRESE SALES AND REPUTATION OF COMPANY
- BUT WE CANNOT TOTALLY DEPEND OUR TRANSPORTATION ON SINGLE ROUTE TYPE, WE SHOULD KEEP LOOKING FOR BETTER OPTION OR ALTERNATIVE OPTIONS IN CASE OF EMERGENCIES
- AS TRIPS BETWEEN BUSIEST ROUTE CITIES ARE MORE WE CAN TRY TO INCREASE THE SIZE OR CAPACITY OF TRUCK TO DECREASE THE DIFFERENT TRIPS PER DAY WHICH DELIVERS THE LOGISTICS MORE AT A SINGLE TIME DUE TO MORE DELIVERING CAPACITY WHICH CUT-OFF OUR EXPENCES PER TRIP AND GIVES US MORE PROFIT.
- ACTUAL_TIME CANNOT BE PREDICT BASED ON OSRM_TIME AS THEY BOTH HAVE DIFFERENCES
- SAME WITH ACTUAL_DISTANCE AND OSRM_DISTANCE AS THEY HAVE DIFFERENCES TOO.

0s completed at 6:42 PM