Problem Statment :

Logistics and supply chain company which 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.

The Data team builds intelligence and capabilities using this data that helps them to widen the gap between the quality, efficiency, and profitability of their business versus their competitors.

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

• 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

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df =pd.read_csv('delhivery_data.csv')
```

In [ ]:
```
# Let us check the data first.
df.head()
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320

5 rows × 24 columns

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

EDA

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool   
 13  cutoff_factor                   144867 non-null  int64  
 14  cutoff_timestamp                144867 non-null  object 
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

In [ ]:

```
#We can see the numnber of rows and columns
print(f'Rows :{df.shape[0]}')
print(f'Columns :{df.shape[1]}')
```

```
Rows :144867
Columns :24
```

In [3]:

```
# Droping the unwanted and Unknown field
df.drop(['is_cutoff','cutoff_factor','cutoff_timestamp','factor','segment_factor
```

In [ ]:

```
#Let us check now if the cloumns are droped
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

In [ ]:

```
# Finding the null values
df.isnull().sum()
# We can see their are null vaulus in coulumn source_name and destination_name
```

Out[ ]:

	0
data	0
trip_creation_time	0
route_schedule_uuid	0
route_type	0
trip_uuid	0
source_center	0
source_name	293
destination_center	0
destination_name	261
od_start_time	0
od_end_time	0
start_scan_to_end_scan	0
actual_distance_to_destination	0
actual_time	0
osrm_time	0
osrm_distance	0
segment_actual_time	0
segment_osrm_time	0
segment_osrm_distance	0

dtype: int64

In [ ]:

```
df[df['source_name'].isnull()]['source_center'].value_counts()
```

Out[ ]:

source_center	count
IND282002AAD	128
IND342902A1B	90
IND126116AAA	20
IND509103AAC	17
IND577116AAA	16
IND465333A1B	6
IND841301AAC	5
IND505326AAB	5
IND331022A1B	3
IND852118A1B	3

dtype: int64

In [ ]:
```
df[df['destination_name'].isnull()].head(3)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uui
110	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756?
111	training	2018-09-25 08:53:04.377810	thanos::sroute:4460a38d-ab9b-484e-bd4e-f4201d0...	FTL	trip-153786558437756?
982	test	2018-10-01 20:56:18.155260	thanos::sroute:d0ebdacd-e09b-47d3-be77-c9c4a05...	FTL	trip-153842737815495?

- The above source centers do not have the name . Which can be difficult to track or of the dirvers to know which place they would go .So it is better to maintain the name
- But for our analysis we can fill the null values with source centers

In [ ]:
```
df[df['destination_name'].isnull()]['destination_center'].value_counts()
```

Out[ ]:

destination_center	count
IND282002AAD	151
IND342902A1B	16
IND577116AAA	16
IND852118A1B	15
IND505326AAB	11
IND126116AAA	10
IND841301AAC	9
IND250002AAC	9
IND509103AAC	9
IND122015AAC	8
IND465333A1B	3
IND331001A1C	3
IND221005A1A	1

dtype: int64

In [ ]:

```
destination_centers = [
    'IND282002AAD', 'IND342902A1B', 'IND577116AAA', 'IND852118A1B',
    'IND505326AAB', 'IND126116AAA', 'IND841301AAC', 'IND250002AAC',
    'IND509103AAC', 'IND122015AAC', 'IND465333A1B', 'IND331001A1C',
    'IND221005A1A'
]

for center in destination_centers:
    print(f"\nDestination Center: {center}")
    print(df[df['destination_center'] == center]['destination_name'].value_count
# we can see evrery destination_name is null
```

```
Destination Center: IND282002AAD
Series([], Name: count, dtype: int64)

Destination Center: IND342902A1B
Series([], Name: count, dtype: int64)

Destination Center: IND577116AAA
Series([], Name: count, dtype: int64)

Destination Center: IND852118A1B
Series([], Name: count, dtype: int64)

Destination Center: IND505326AAB
Series([], Name: count, dtype: int64)

Destination Center: IND126116AAA
Series([], Name: count, dtype: int64)

Destination Center: IND841301AAC
Series([], Name: count, dtype: int64)

Destination Center: IND250002AAC
Series([], Name: count, dtype: int64)

Destination Center: IND509103AAC
Series([], Name: count, dtype: int64)

Destination Center: IND122015AAC
Series([], Name: count, dtype: int64)

Destination Center: IND465333A1B
Series([], Name: count, dtype: int64)

Destination Center: IND331001A1C
Series([], Name: count, dtype: int64)

Destination Center: IND221005A1A
Series([], Name: count, dtype: int64)
```

- Same for above Destination Center they all have null Destination name .
- We can fill null as Destination Center in Destination name column.

In [4]:
```
# have fill the null values in the columns as their source center and Destinatio
#in their respective name columns

df['source_name']=df['source_name'].fillna(df['source_center'])
```

In [5]:
```
df['destination_name']=df['destination_name'].fillna(df['destination_center'])
```

In [ ]:
```
#Now let us see the data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 19 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144867 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144867 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  segment_actual_time             144867 non-null  float64
 17  segment_osrm_time               144867 non-null  float64
 18  segment_osrm_distance           144867 non-null  float64
dtypes: float64(8), object(11)
memory usage: 21.0+ MB
```

- we can see the columns which has time are objet data type so we can change to datetime data type so that we can easyly get the data from the column.

In [6]:
```
df['trip_creation_time']=pd.to_datetime(df['trip_creation_time'],format='%Y-%m-%
```

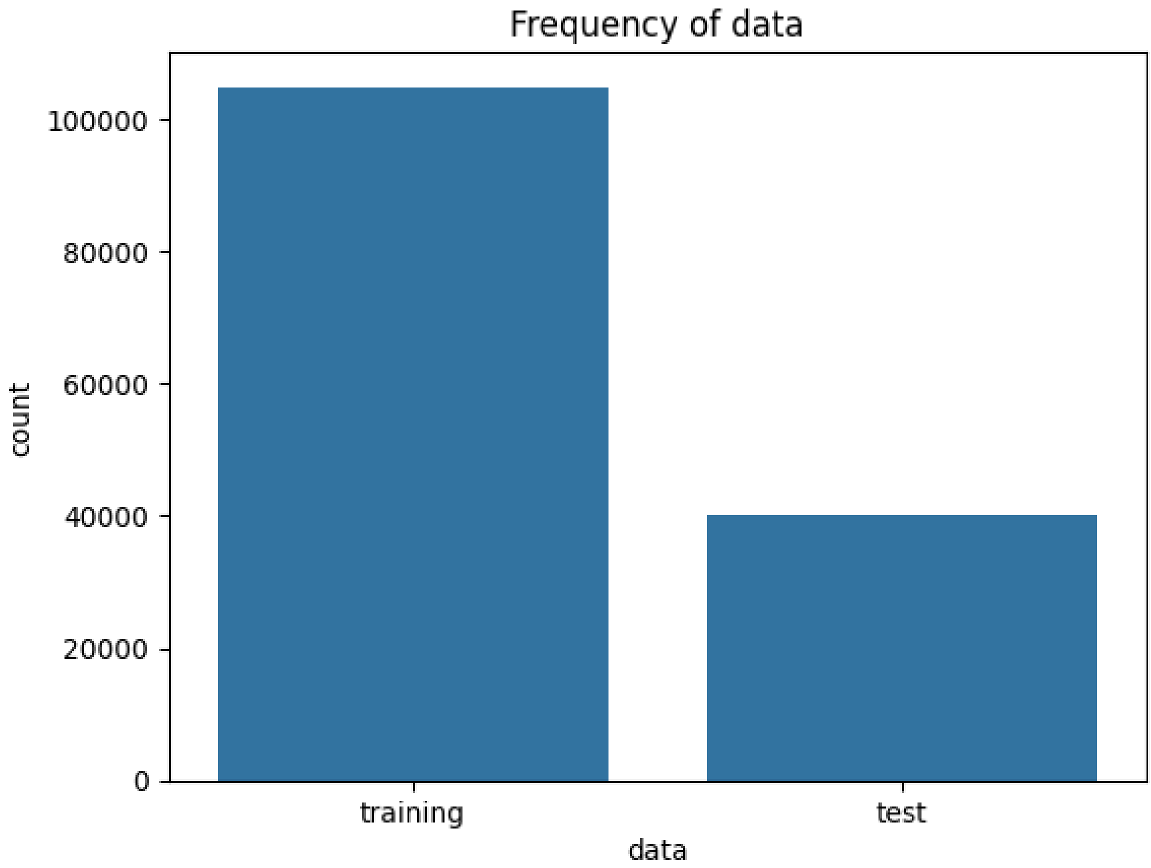
In [7]:
```
df['od_start_time']=pd.to_datetime(df['od_start_time'],format='%Y-%m-%d %H:%M:%S
df['od_end_time']=pd.to_datetime(df['od_end_time'],format='%Y-%m-%d %H:%M:%S.%f'
```

In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
np.int64(0)
```

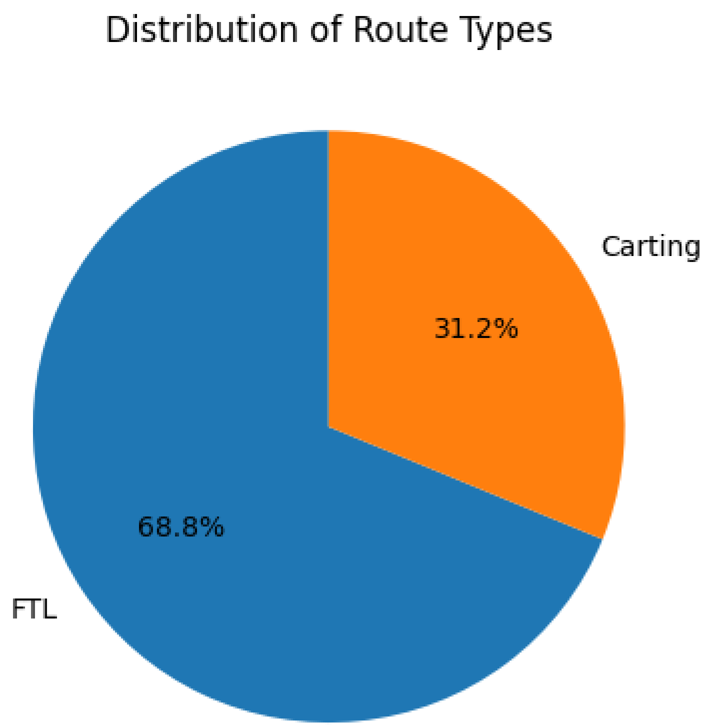
- I want to vizualise how many traing and how many testing are their
- I also want to know about route_type

In [ ]:
```
sns.countplot(x=df['data'])
plt.title('Frequency of data')
plt.show()
```



- As we can see their is more training data compare to test data .

In [ ]:
```
#route_type

route_counts = df['route_type'].value_counts()
plt.pie(route_counts, labels=route_counts.index, autopct='%1.1f%%', startangle=9
plt.title('Distribution of Route Types')
plt.show()
```



In [ ]:
```
df.head(1)
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320

In [9]:
```
df['Source_state']=df['source_name'].str.split('(').str[1].str[:-1]
df['destination_state']=df['destination_name'].str.split('(').str[1].str[:-1]
```

Feature Engineering and Hypotisis testing

In [ ]:
```
df.groupby('trip_uuid')['trip_uuid'].value_counts()
```

Out[ ]:

trip_uuid	count
trip-153671041653548748	39
trip-153671042288605164	9
trip-153671043369099517	89
trip-153671046011330457	2
trip-153671052974046625	7
...	...
trip-153861095625827784	7
trip-153861104386292051	2
trip-153861106442901555	6
trip-153861115439069069	17
trip-153861118270144424	4

14817 rows × 1 columns

dtype: int64

In [ ]:
```
df[df['trip_uuid']=='trip-153671052974046625']
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_u
78217	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78218	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78219	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78220	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78221	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78222	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046
78223	training	2018-09-12 00:02:09.740725	thanos::sroute:d9f07b12-65e0-4f3b-bec8-df06134...	FTL	t 153671052974046

In [ ]:

```
df[df['trip_uuid']=='trip-153861118270144424']
```

Out[ ]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuic
11570	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424
11571	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424
11572	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424
11573	test	2018-10-03 23:59:42.701692	thanos::sroute:412fea14-6d1f-4222-8a5f-a517042...	FTL	trip-153861118270144424

In [ ]:

```
# I can also drop segment wise time because we do not know the segment
#df.drop(['segment_actual_time','segment_osrm_time','segment_osrm_distance'],axi
```

In [54]:

```
# Now goruping the the common trip_uuid and performing the agg function on it

data_centers=df.groupby(['trip_uuid','source_center','destination_center','data'
```

```
                                                                    'actual_dis
                                                                    'actual_tim
                                                                    'osrm_time'
                                                                    'osrm_dista
                                                                    }).reset_in
```

In [55]:
```
data_centers.groupby(['source_center', 'destination_center'])['trip_uuid'].count
```

Out[55]:

source_center	destination_center	trip_uuid
IND562132AAA	**IND560300AAA**	151
	IND560099AAB	127
IND560099AAB	**IND560300AAA**	121
IND560300AAA	**IND562132AAA**	108
IND411033AAA	**IND421302AAG**	107

dtype: int64

- We can see the above most frequently used corridors
  1. IND562132AAA->IND560300AAA which is 151
  2. IND562132AAA ->IND560099AAB which is 127

In [10]:
```
data=df.groupby(['trip_uuid','Source_state','destination_state','data']).agg({'c
                                                                    'actual_dis
                                                                    'actual_tim
                                                                    'osrm_time'
                                                                    'osrm_dista
                                                                   'segment_act
                                                                   'segment_osr
                                                                   'segment_osr
                                                                    }).reset_in
```

In [58]:
```
data.describe()
```

Out[58]:

	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_
count	16657	16657	16657.000000	16
mean	2018-09-22 15:09:29.658076672	2018-09-22 21:32:38.367513344	401.975086	
min	2018-09-12 00:00:16.535741	2018-09-12 00:50:10.814399	23.000000	
25%	2018-09-17 06:08:40.561678080	2018-09-17 12:54:28.577978112	127.000000	
50%	2018-09-22 06:06:01.573842944	2018-09-22 13:33:44.977102080	219.000000	
75%	2018-09-27 19:31:17.301129984	2018-09-28 02:15:40.360954880	445.000000	
max	2018-10-06 04:27:23.392375	2018-10-08 03:00:24.353479	7898.000000	
std	NaN	NaN	522.558609	

In [61]:

```
col=['actual_time','osrm_time','osrm_distance','actual_distance_to_destination']
for i in col:
  a=data[i].mean()
  print(f'Average of {i} :{a}')
```

```
Average of actual_time :273.54385543615297
Average of osrm_time :121.69952572492045
Average of osrm_distance :87.92251465658266
Average of actual_distance_to_destination :71.02438237948297
```

Average OSRM Distance: 87.92

Average OSRM Time: 121.69

Average Actual Time: 273.54385543615297

In [ ]:

```
data.shape
```

Out[ ]: (16657, 14)

- I have merged the rows with commn trip_uuid source_center,destination_center and took the numarical coloumn .
- Here we can observe for one trip is around trip like you can see below.

In [ ]:

```
data.head()
```

Out[ ]:

	trip_uuid	Source_state	destination_state	data	od_start_time	od_e
0	trip-153671041653548748	Madhya Pradesh	Uttar Pradesh	training	2018-09-12 00:00:16.535741	20 16:39:4
1	trip-153671041653548748	Uttar Pradesh	Haryana	training	2018-09-12 16:39:46.858469	20 13:40:2
2	trip-153671042288605164	Karnataka	Karnataka	training	2018-09-12 02:03:09.655591	20 03:01:5
3	trip-153671043369099517	Haryana	Punjab	training	2018-09-14 03:40:17.106733	20 17:34:5
4	trip-153671043369099517	Karnataka	Haryana	training	2018-09-12 00:00:33.691250	20 03:40:1

- Need to check the corellation

In [16]:

```
data['Source_state'].value_counts()
# maximum soure_state is maharashtra
#Lower is Tripura
#We can take some action were other lower service used states to use this logist
```

Out[16]:

Source_state	count
Maharashtra	2797
Karnataka	2416
Haryana	1839
Tamil Nadu	1150
Uttar Pradesh	978
Telangana	838
Gujarat	819
Delhi	790
West Bengal	682
Punjab	663
Rajasthan	564
Andhra Pradesh	551
Madhya Pradesh	455
Bihar	398
Kerala	320
Assam	290
Jharkhand	189
Uttarakhand	182
Orissa	178
Chandigarh	160
Himachal Pradesh	143
Goa	74
Arunachal Pradesh	46
Chhattisgarh	43
Jammu & Kashmir	34
Pondicherry	19
Dadra and Nagar Haveli	15
Meghalaya	13
Mizoram	5
Nagaland	5
Tripura	1

dtype: int64

In [ ]:
```
data['destination_state'].value_counts()
#Maximum destination state is Maharashtra
#lower destination state is Daman & Diu
```

Out[ ]:

destination_state	count
Maharashtra	2705
Karnataka	2468
Haryana	1815
Tamil Nadu	1127
Uttar Pradesh	988
Telangana	902
Gujarat	819
Punjab	743
West Bengal	713
Delhi	674
Rajasthan	602
Andhra Pradesh	551
Madhya Pradesh	454
Bihar	403
Kerala	304
Assam	272
Jharkhand	197
Orissa	190
Uttarakhand	180
Himachal Pradesh	161
Chandigarh	123
Goa	74
Arunachal Pradesh	52
Chhattisgarh	43
Jammu & Kashmir	32
Pondicherry	24
Dadra and Nagar Haveli	17
Meghalaya	13
Mizoram	7
Tripura	2
Nagaland	1
Daman & Diu	1

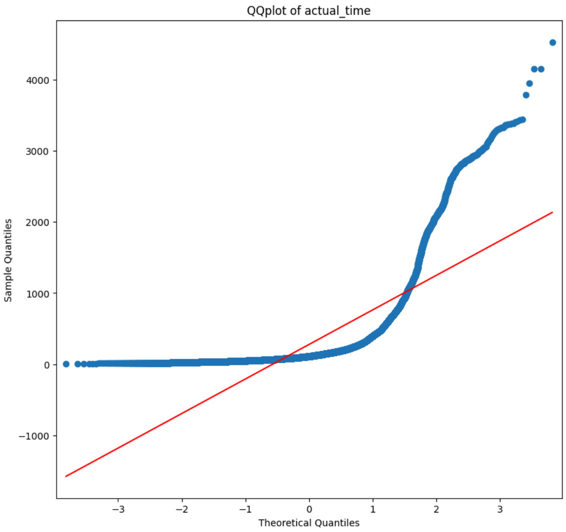
dtype: int64

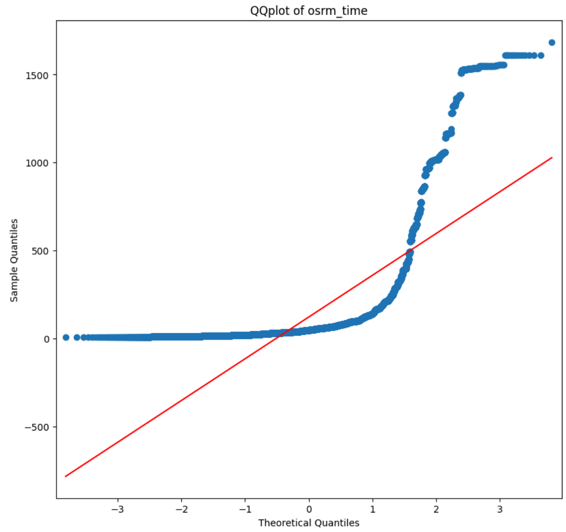
In [17]:
```
data_trip_uuid=data.groupby(['trip_uuid']).agg({'od_start_time':'max','od_end_ti
                                                                     'actual_dis
                                                                     'actual_tim
                                                                     'osrm_time'
                                                                     'osrm_dista
                                                                    'segment_act
                                                                    'segment_osr
                                                                    'segment_osr
                                                                     }).reset_in
```

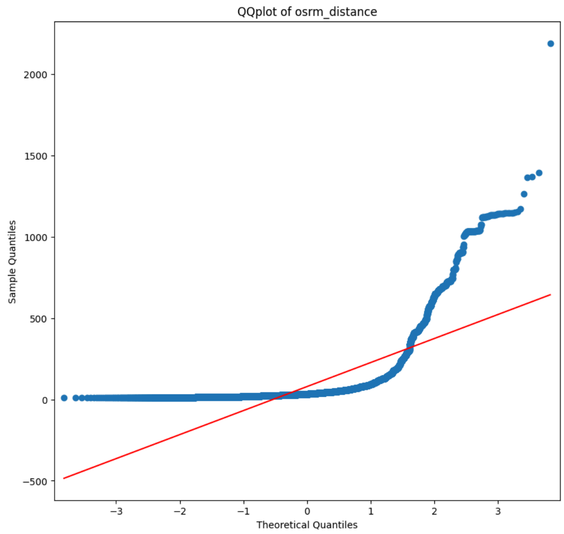
In [18]:
```
#let us do a normaity test using QQplot
import scipy.stats as stats
```

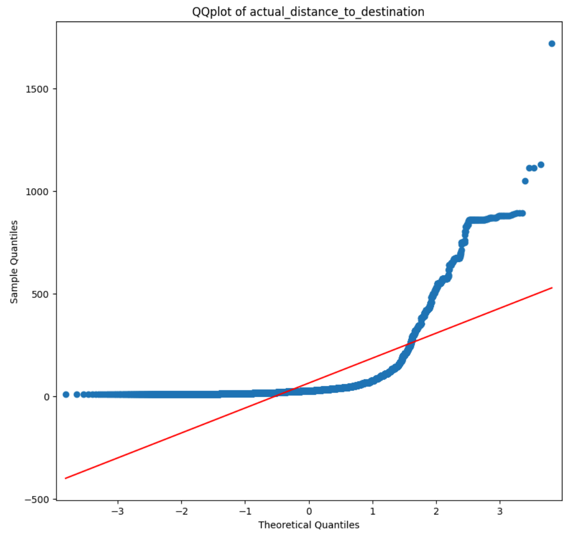
In [20]:
```
import statsmodels.api as sm
```

In [39]:
```
fig, axis = plt.subplots(2, 2,figsize=(21, 20))

sm.qqplot(data_trip_uuid['actual_time'],line='s',ax=axis[0,0])
axis[0,0].set_title('QQplot of actual_time')

sm.qqplot(data_trip_uuid['osrm_time'],line='s',ax=axis[0,1])
axis[0,1].set_title('QQplot of osrm_time')

sm.qqplot(data_trip_uuid['osrm_distance'],line='s',ax=axis[1,0])
axis[1,0].set_title('QQplot of osrm_distance')

sm.qqplot(data_trip_uuid['actual_distance_to_destination'],line='s',ax=axis[1,1]
axis[1,1].set_title('QQplot of actual_distance_to_destination')

plt.show()
```









- We can see that the data is not normaly distributed so we can use Kruskal Wallis test for hypothisis testing

In [34]:
```
from  scipy import stats
```

In [40]:
```
#segment_actual_time VS osrm_time
stat, p = stats.kruskal(data_trip_uuid['actual_time'],data_trip_uuid['osrm_time'


print(f"Kruskal-Wallis H-statistic: {stat}")
print(f"P-value: {p}")


if p < 0.05:
    print("Significant difference between actual_time and osrm_time .")
else:
    print("No significant difference between actual_time and osrm_time.")
```

```
Kruskal-Wallis H-statistic: 4881.379995727001
P-value: 0.0
Significant difference between actual_time and osrm_time .
```

- By this we can tell that we need more accurate prediction of time throught OSRM

In [41]:
```
#actual_time VS segment_actual_time
stat, p = stats.kruskal(data_trip_uuid['actual_time'],data_trip_uuid['segment_ac


print(f"Kruskal-Wallis H-statistic: {stat}")
print(f"P-value: {p}")


if p < 0.05:
    print("Significant difference between actual_time and segment_actual_time ."
else:
    print("No significant difference between actual_time and segment_actual_time
```

```
Kruskal-Wallis H-statistic: 276.9587099963026
P-value: 3.454288758342234e-62
Significant difference between actual_time and segment_actual_time .
```

- Here also we can see their is a significat diffrence

In [42]:
```
stat, p = stats.kruskal(data_trip_uuid['actual_distance_to_destination'],data_tr


print(f"Kruskal-Wallis H-statistic: {stat}")
print(f"P-value: {p}")


if p < 0.05:
    print("Significant difference between actual_distance_to_destination and osr
else:
    print("No significant difference between actual_distance_to_destination and
```

```
Kruskal-Wallis H-statistic: 774.5728435215805
P-value: 1.821772536232272e-170
Significant difference between actual_distance_to_destination and osrm_distance .
```

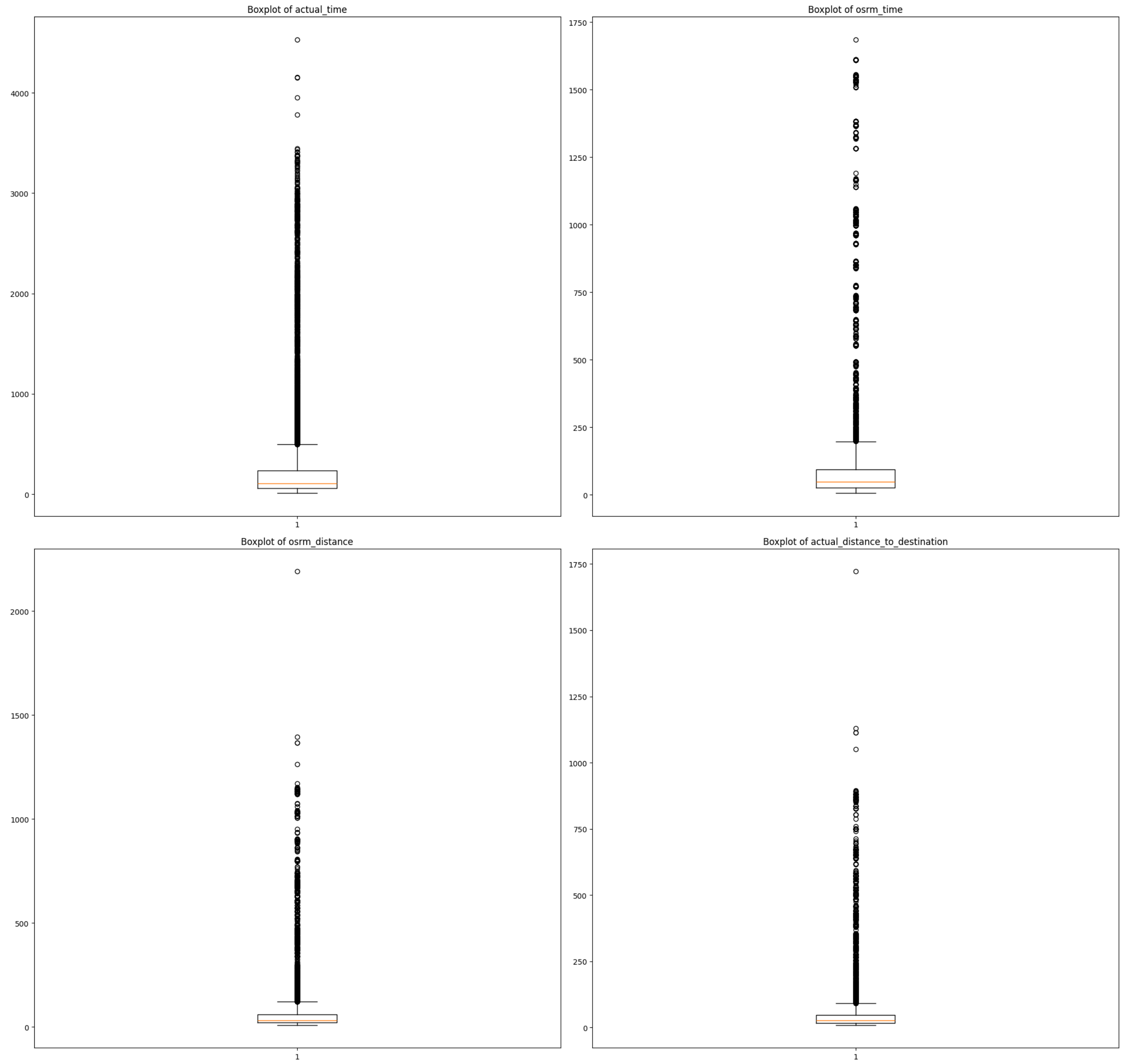
- I have tested three groups for comparing ORSM so the prection need to be acurate .

Outliers

- Need to check if their are any outlier's

In [50]:
```
import matplotlib.pyplot as plt

fig, axis = plt.subplots(2, 2, figsize=(21, 20))

# Boxplot for actual_time
axis[0, 0].boxplot(data_trip_uuid['actual_time'].dropna())
axis[0, 0].set_title('Boxplot of actual_time')

# Boxplot for osrm_time
axis[0, 1].boxplot(data_trip_uuid['osrm_time'].dropna())
axis[0, 1].set_title('Boxplot of osrm_time')

# Boxplot for osrm_distance
axis[1, 0].boxplot(data_trip_uuid['osrm_distance'].dropna())
```

```
axis[1, 0].set_title('Boxplot of osrm_distance')

# Boxplot for actual_distance_to_destination
axis[1, 1].boxplot(data_trip_uuid['actual_distance_to_destination'].dropna())
axis[1, 1].set_title('Boxplot of actual_distance_to_destination')

plt.tight_layout()
plt.show()
```



In [52]:

```
columns = ['actual_time', 'osrm_time', 'osrm_distance', 'actual_distance_to_dest

for col in columns:
    Q1 = data_trip_uuid[col].quantile(0.25)
    Q3 = data_trip_uuid[col].quantile(0.75)
    IQR = Q3 - Q1
    print(f"IQR for {col}: {IQR}")
```

```
IQR for actual_time: 174.0
IQR for osrm_time: 69.0
IQR for osrm_distance: 39.914138095238094
IQR for actual_distance_to_destination: 30.552448111226653
```

- Their are outliers present in the data what we can do :

1. Remove Outliers

2. Cap or Winsorize Outliers
3. Log or Power Transform
4. Impute (if due to error or missing logic)
5. Label them

Business Insights

- Maharashtra has the highest number of orders both as a source and destination state.Tripura (as source) and Daman & Diu (as destination) have very low activity.

- most frequently used corridors

    1. IND562132AAA->IND560300AAA which is 151
    2. IND562132AAA ->IND560099AAB which is 127 These represent high-volume operational lanes.

-   1. Average OSRM Distance: 87.92
    2. Average OSRM Time: 121.69
    3. Average Actual Time: 273.54385543615297

- Tested if differences exist between groups (e.g., time prediction across routes).Found statistically significant differences → model needs to handle variability better.

- Found outliers in time and distance features. Suggested handling methods:

1. Remove Outliers
2. Cap or Winsorize Outliers
3. Log or Power Transform
4. Impute (if due to error or missing logic)
5. Label them

Recommendations

- Focus marketing and expansion strategies on low-demand states to increase penetration.
- Optimize most frequently used corridors using real-time tracking, fleet management, and exception handling.
- Improve OSRM predictions using additional variables (traffic, weather, time of day).
- Also use robust ML models that don't assume normality.
- For outliers Use method that aligns with business logic.