Objective:

To clean, sanitize and manipulate data to get useful features out of raw data for the data science team to build forecasting models on it.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind,pearsonr
```

In [2]:
```
df=pd.read_csv(r"D:\Rahul\Scaler\Case Study\Delhivery\delhivery_data.csv")
df
```

Out[2]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	de
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	
...	...	...	...	...	...	...	...	
144862	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	
144863	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	
144864	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	
144865	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	
144866	training	2018-09-20 16:24:28.436231	thanos::sroute:f0569d2f-4e20-4c31-8542-67b86d5...	Carting	trip-153746066843555182	IND131028AAB	Sonipat_Kundli_H (Haryana)	

144867 rows × 24 columns

Column Profiling:

- data - tells whether the data is testing or training data
- trip_creation_time – Timestamp of trip creation
- route_schedule_uuid – Unique Id for a particular route schedule
- route_type – Transportation type
    - FTL – Full Truck Load: FTL shipments get to the destination sooner, as the truck is making no other pickups or drop-offs along the way
    - Carting: Handling system consisting of small vehicles (carts)
- trip_uuid - Unique ID given to a particular trip (A trip may include different source and destination centers)
- source_center - Source ID of trip origin
- source_name - Source Name of trip origin
- destination_cente – Destination ID
- destination_name – Destination Name
- od_start_time – Trip start time

- od_end_time – Trip end time
- start_scan_to_end_scan – Time taken to deliver from source to destination
- is_cutoff – Unknown field
- cutoff_factor – Unknown field
- cutoff_timestamp – Unknown field
- actual_distance_to_destination – Distance in Kms between source and destination warehouse
- actual_time – Actual time taken to complete the delivery (Cumulative)
- osrm_time – An open-source routing engine time calculator which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) and gives the time (Cumulative)
- osrm_distance – An open-source routing engine which computes the shortest path between points in a given map (Includes usual traffic, distance through major and minor roads) (Cumulative)
- factor – Unknown field
- segment_actual_time – This is a segment time. Time taken by the subset of the package delivery
- segment_osrm_time – This is the OSRM segment time. Time taken by the subset of the package delivery
- segment_osrm_distance – This is the OSRM distance. Distance covered by subset of the package delivery
- segment_factor – Unknown field

In [3]:
```
df.shape
```

Out[3]:
```
(144867, 24)
```

The raw dataset consists of **144867 Rows and 24 Columns**

In [4]:
```
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 [5]:
```
# Converting respective float columns to datetime
time_cols=['trip_creation_time','od_start_time','od_end_time','cutoff_timestamp']
for col in time_cols:
    df[col]=pd.to_datetime(df[col])
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
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
is_cutoff                          0
cutoff_factor                      0
cutoff_timestamp                   0
actual_distance_to_destination     0
actual_time                        0
osrm_time                          0
osrm_distance                      0
factor                             0
segment_actual_time                0
segment_osrm_time                  0
segment_osrm_distance              0
segment_factor                     0
dtype: int64
```

Here we have few null values for source name and destination name

In [7]:
```
df[df['source_name'].isna()==True]['source_center'].unique()
```

Out[7]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [8]:
```
df[df['destination_name'].isna()==True]['destination_center'].unique()
```

Out[8]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

Since no direct imputation can be applied to fill the names of source and destination we'll simply fill them with 'NA' for further analysis

In [9]:
```
df.fillna('NA',inplace=True)
```

In [10]:
```
df.describe()
```

Out[10]:

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	
count	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	144867.000000	14486
mean	961.262986	232.926567	234.073372	416.927527	213.868272	284.771297	
std	1037.012769	344.755577	344.990009	598.103621	308.011085	421.119294	
min	20.000000	9.000000	9.000045	9.000000	6.000000	9.008200	
25%	161.000000	22.000000	23.355874	51.000000	27.000000	29.914700	
50%	449.000000	66.000000	66.126571	132.000000	64.000000	78.525800	
75%	1634.000000	286.000000	286.708875	513.000000	257.000000	343.193250	
max	7898.000000	1927.000000	1927.447705	4532.000000	1686.000000	2326.199100	7

Insights:

- Since maximum values of some columns is very high it shows there are many outliers present there
- For minimum value in segment actual time it is negative which can not be possible

In [11]:
```
print(f"Dataset is available from {df['trip_creation_time'].dt.date.min()} to {df['trip_creation_time'].d
```

```
Dataset is available from 2018-09-12 to 2018-10-03
```

In [12]:
```
# Merging dataset on basis of trip id, source center and destination center
merged_df=df.groupby(['trip_uuid',
                      'source_center',
                      'destination_center']).agg({"trip_creation_time":"first",
                                                   "route_type":"first",
                                                   "source_name":"first",
                                                   "destination_name":"first",
                                                   "od_start_time":"first",
                                                   "od_end_time":"first",
                                                   "start_scan_to_end_scan":"first",
                                                   "actual_distance_to_destination":"last",
                                                   "osrm_distance":"last",
                                                   "actual_time":"last",
                                                   "osrm_time":"last",
                                                   "factor":"last",
                                                   "segment_actual_time":"sum",
                                                   "segment_osrm_time":"sum",
                                                   "segment_osrm_distance":"sum",
                                                   "cutoff_timestamp":"last"}).sort_values(by=['trip_uuid'
                                                                                               'cutoff_tim
```

In [13]:
```
# Calculating total time taken for the trip in minutes
merged_df['od_time_taken']=(merged_df['od_end_time']-merged_df['od_start_time']).dt.total_seconds().div(6
```

In [14]:
```
# Dropping columns which are of no use now
merged_df.drop(columns=['od_start_time','od_end_time'],inplace=True)
```

In [15]:
```
# Finally merging dataset on basis of trip id for further analysis
final_df=merged_df.groupby(['trip_uuid']).agg({"trip_creation_time":"first",
                                               "source_center":"first",
                                               "destination_center":"last",
                                               "route_type":"first",
                                              "source_name":"first",
                                              "destination_name":"last",
                                               "od_time_taken":"sum",
                                              "start_scan_to_end_scan":"sum",
                                              "actual_distance_to_destination":"sum",
                                              "osrm_distance":"sum",
                                              "actual_time":"sum",
                                              "osrm_time":"sum",
                                              "segment_actual_time":"sum",
                                              "segment_osrm_time":"sum",
                                              "segment_osrm_distance":"sum"}).reset_index()
```

In [16]:
```
final_df.shape
```

Out[16]:
```
(14817, 16)
```

The final dataset consists of **14817 Unique trip ids with 16 columns**

Extracting Features

In [17]:
```
# Functions for extracting state, city and place name from destination and source names

def state_name(x):
    if x=='NA':
        return None
    state=x[x.find("(")+1:-1]
    return state

def city_name(x):
    if x=='NA':
        return None
    city=x.split("_")[0].split()[0]
    return city

def place_name(x):
    if x=='NA':
        return None
    l=x.split('(')[0].split('_')
    if len(l)>1:
        return l[1]
    return None
```

In [18]:
```
# Extracting geogprahical info

final_df['source_state']=final_df['source_name'].apply(lambda x:state_name(x))
final_df['destination_state']=final_df['destination_name'].apply(lambda x:state_name(x))
final_df['source_city']=final_df['source_name'].apply(lambda x:city_name(x))
final_df['destination_city']=final_df['destination_name'].apply(lambda x:city_name(x))
final_df['source_place']=final_df['source_name'].apply(lambda x:place_name(x))
final_df['destination_place']=final_df['destination_name'].apply(lambda x:place_name(x))
```

In [19]:
```
# Extracting different time units from trip creation time

final_df['year']=final_df['trip_creation_time'].dt.year
final_df['month']=final_df['trip_creation_time'].dt.month
final_df['month_name']=final_df['trip_creation_time'].dt.month_name()
final_df['day_name']=final_df['trip_creation_time'].dt.day_name()
final_df['hour']=final_df['trip_creation_time'].dt.hour
```

In [20]:
```
# Removing unwanted columns
final_df.drop(columns=['trip_creation_time','source_name','destination_name'],inplace=True)
```

In [21]:
```
final_df.head()
```

Out[21]:

	trip_uuid	source_center	destination_center	route_type	od_time_taken	start_scan_to_end_scan	actual_distance_to_
0	trip-153671041653548748	IND462022AAA	IND000000ACB	FTL	2259	2259.0	
1	trip-153671042288605164	IND572101AAA	IND562101AAA	Carting	180	180.0	
2	trip-153671043369099517	IND562132AAA	IND160002AAC	FTL	3933	3933.0	1
3	trip-153671046011330457	IND400072AAB	IND401104AAA	Carting	100	100.0	
4	trip-153671052974046625	IND583101AAA	IND583101AAA	FTL	717	717.0	

5 rows × 24 columns

In [22]:
```
final_df.columns
```

Out[22]:
```
Index(['trip_uuid', 'source_center', 'destination_center', 'route_type',
       'od_time_taken', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'osrm_distance', 'actual_time',
       'osrm_time', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'source_state', 'destination_state',
       'source_city', 'destination_city', 'source_place', 'destination_place',
       'year', 'month', 'month_name', 'day_name', 'hour'],
      dtype='object')
```

In [23]:
```
final_df.nunique()
```

Out[23]:
```
trip_uuid                         14817
source_center                       868
destination_center                  956
route_type                            2
od_time_taken                      2208
start_scan_to_end_scan             2208
actual_distance_to_destination    14801
osrm_distance                     14734
actual_time                        1853
osrm_time                           817
segment_actual_time                1890
segment_osrm_time                  1242
segment_osrm_distance             14754
source_state                         29
destination_state                    32
source_city                         664
destination_city                    758
source_place                        642
destination_place                   721
year                                  1
month                                 2
month_name                            2
day_name                              7
hour                                 24
dtype: int64
```

Insights:

- We have 14817 unique trip ids
- We have 29 States as source place whereas 31 States for delivery which might be consisting of Union Territories too
- Data of only 1 year out of which 2 months only

In [24]:
```
final_df['route_type'].value_counts()
```

Out[24]:
```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

Around **60%** of deliveries were made by **carting** and rest around **40%** were **Full Truck Load** type

In [25]:
```
len(final_df[final_df['source_center']==final_df['destination_center']])/len(final_df)*100
```

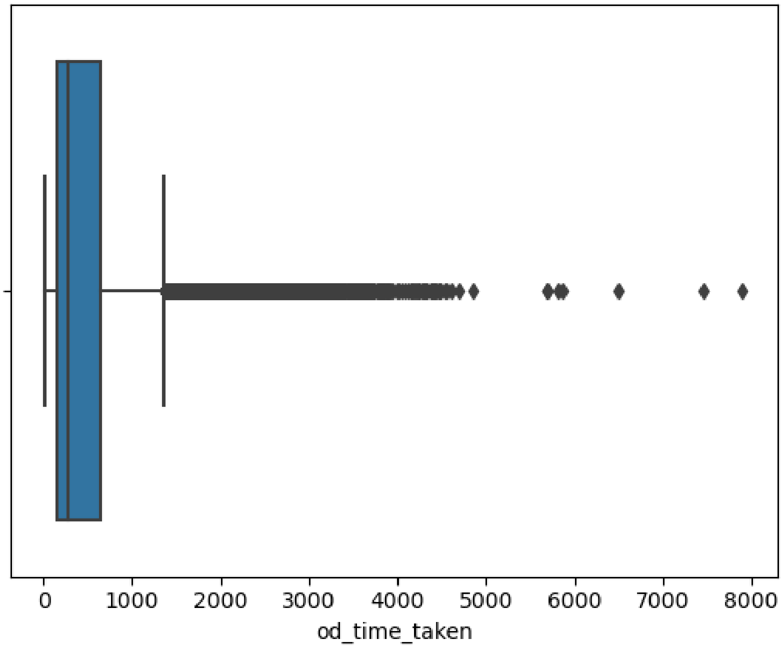
Out[25]:
```
20.037794425322264
```

There are around 20% of trips having same source and destination center

Treating Outliers

In [26]:
```
# Plotting total trip time taken for detecting outliers
sns.boxplot(x=final_df['od_time_taken'])
```

Out[26]: <AxesSubplot:xlabel='od_time_taken'>



In [27]:
```
# Treating Outliers using IQR Method
Q1,Q3=np.percentile(final_df['od_time_taken'],[25,75])
IQR= Q3 - Q1
UL= Q3 + 1.5 * IQR
LL= Q1 - 1.5 * IQR
len( final_df [(final_df['od_time_taken']<=LL) | (final_df['od_time_taken']>=UL) ] ) / len(final_df) * 10
```

Out[27]: 8.550988729162448

Around 8% of the data are outliers

In [28]:
```
# Dropping rows having outlier values of trip time taken
final_df.drop(labels= ( final_df [(final_df['od_time_taken']<=LL) | (final_df['od_time_taken']>=UL) ].ind
```

In [29]:
```
final_df.shape
```

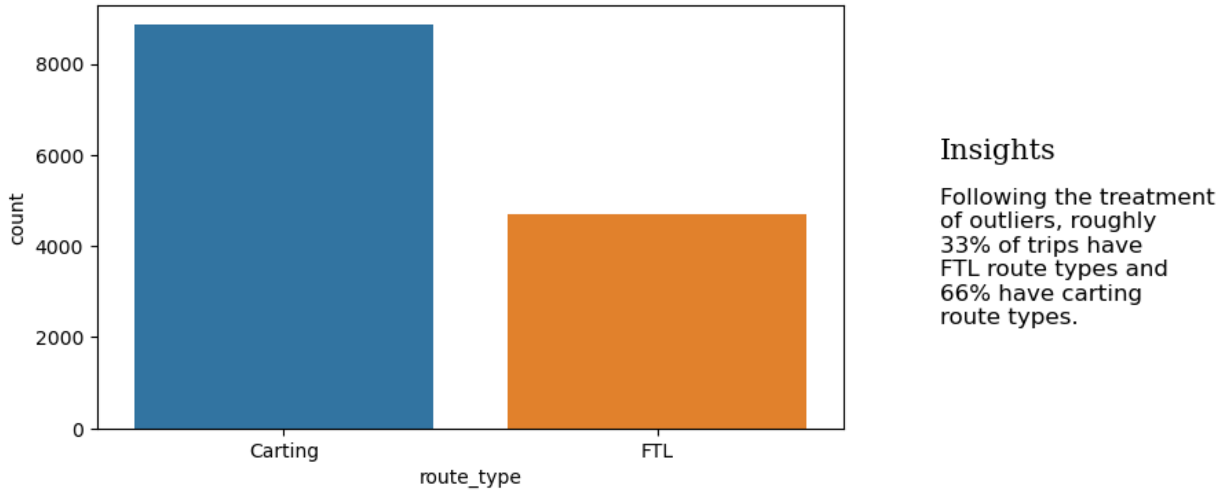
Out[29]: (13550, 24)

We are now left with **13550 rows**

GRAPHICAL ANALYSIS

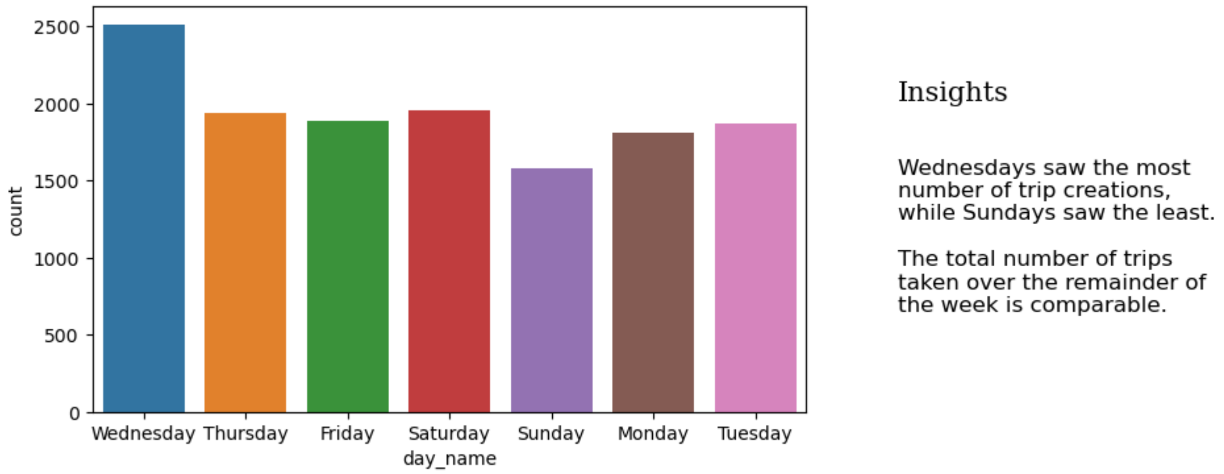
Count of Carting and FTL route types

In [30]:
```
fig=plt.figure(figsize=(7,4))
sns.countplot(x=final_df['route_type'])
fig.text(1,.6,"Insights",fontsize=15,fontfamily='serif')
fig.text(1,.3,'''Following the treatment
of outliers, roughly
33% of trips have
FTL route types and
66% have carting
route types.''',fontsize=12)
plt.show()
```



Count of trips for each day of week

In [31]:
```
fig=plt.figure(figsize=(7,4))
sns.countplot(x=final_df['day_name'])
fig.text(1,.7,"Insights",fontsize=15,fontfamily='serif')
fig.text(1,.3,'''Wednesdays saw the most
number of trip creations,
while Sundays saw the least.
\nThe total number of trips
taken over the remainder of
the week is comparable.''',fontsize=12)
plt.show()
```



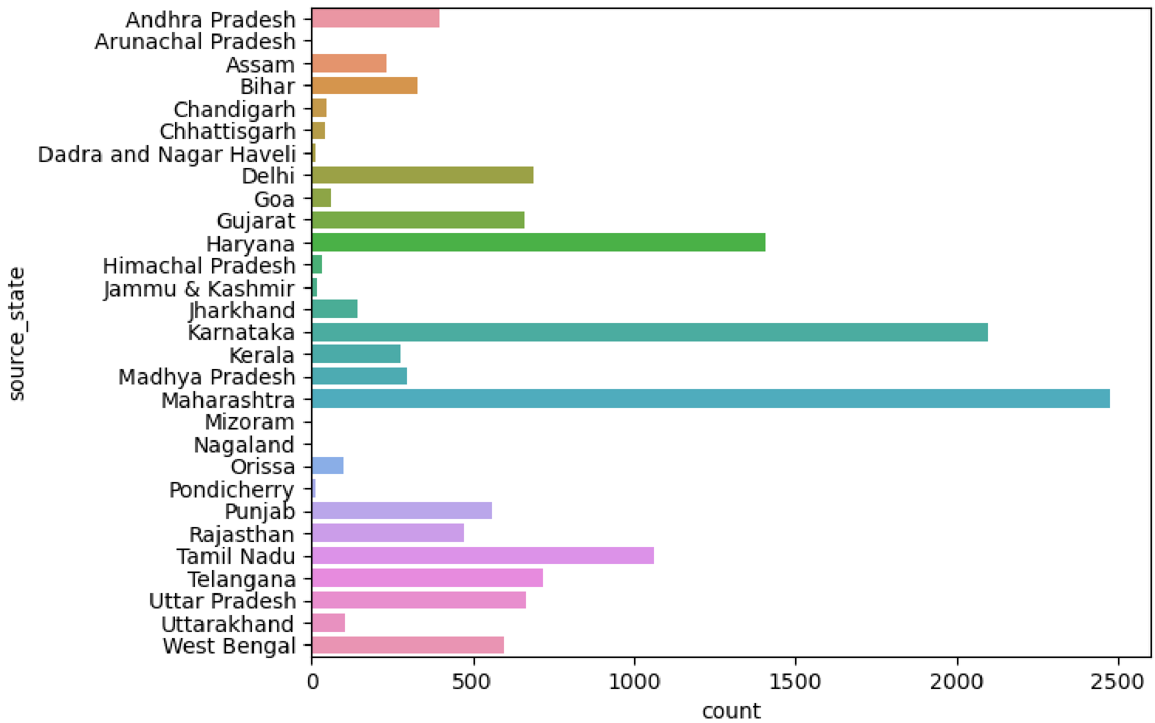
Count of orders from and to different states of India

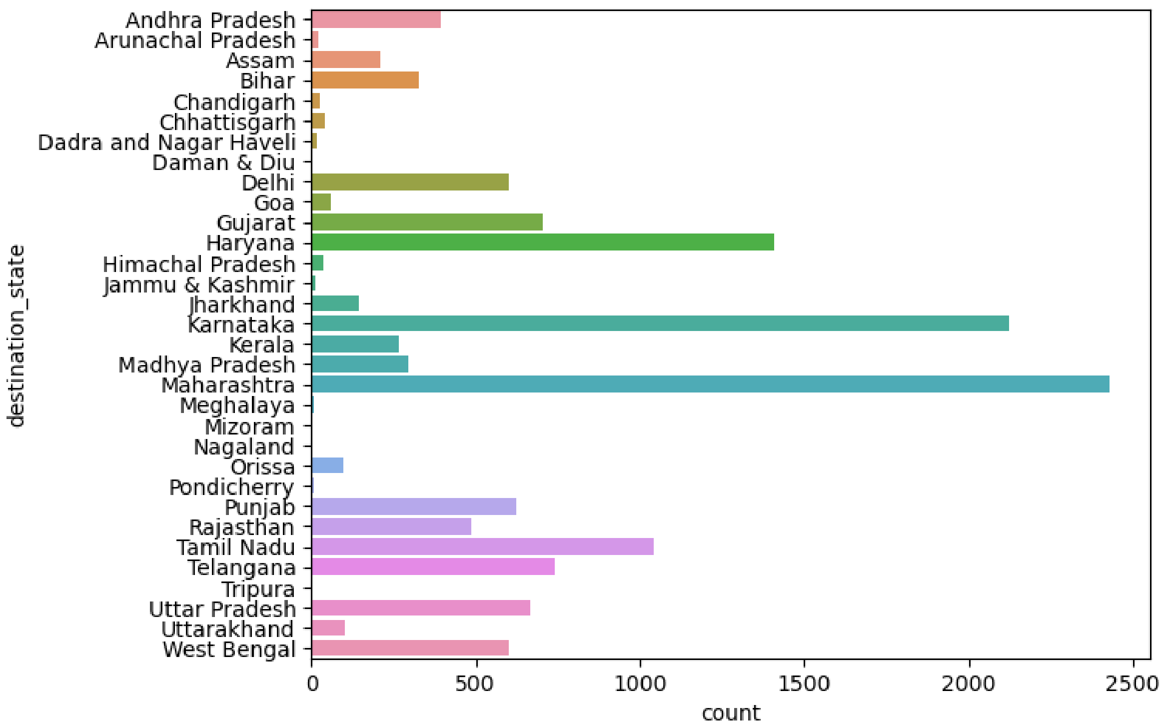
In [32]:
```
fig=plt.figure(figsize=(7,12))

plt.subplot(2,1,1)
sns.countplot(y=final_df['source_state'].sort_values())

plt.subplot(2,1,2)
sns.countplot(y=final_df['destination_state'].sort_values())

plt.show()
```





Insights:

- The majority of orders originate in and are sent from Maharashtra.
- Every state has around the same quantity of orders as a destination state or source state.

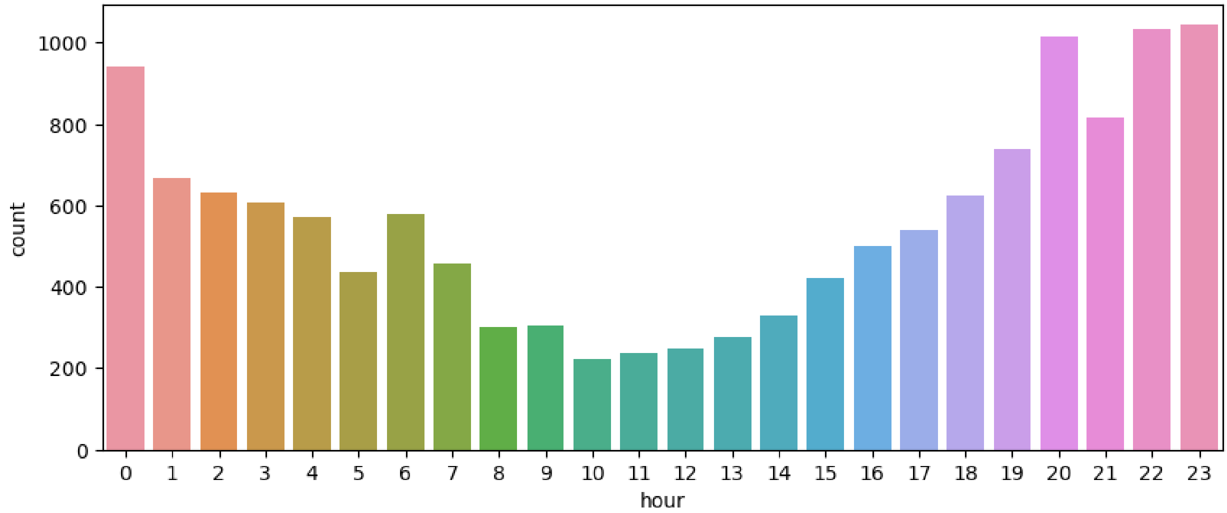
- Manipur and Sikkim do not have any orders from their state, either because Delhivery is not available there or there are no orders within this time window.

Recommendation:

- Delhivery should begin providing services in Manipur and Sikkim as well, if they aren't already, in order to serve every Indian

Count of trip creation in 24 hours of day

In [33]:
```
fig=plt.figure(figsize=(10,4))
sns.countplot(x=final_df['hour'])
plt.show()
```



Insights:

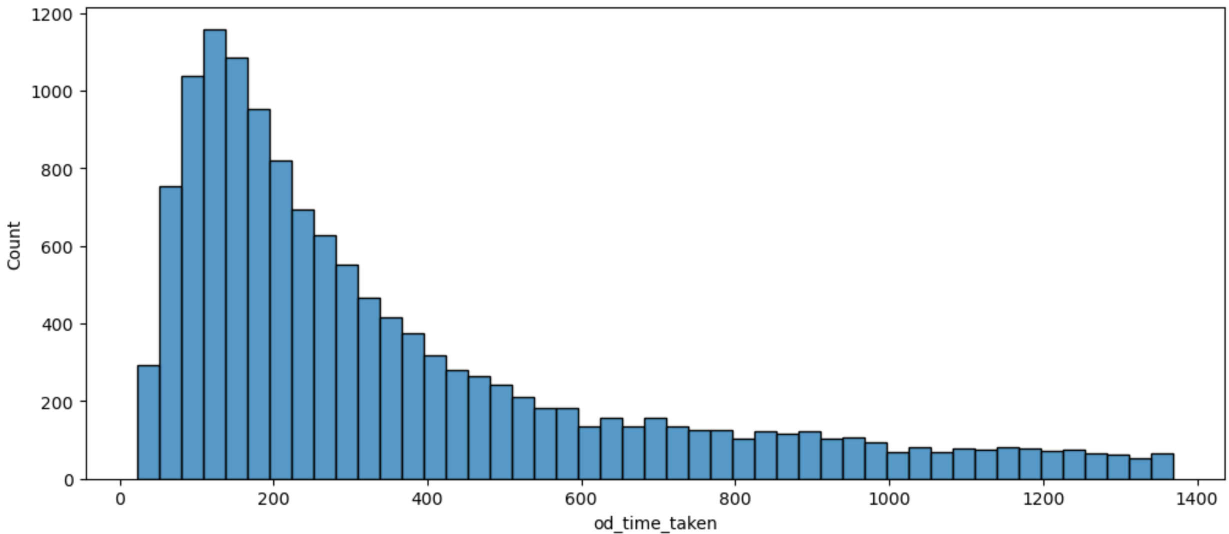
- Most trips begin in the late afternoon or early evening. This may be because large vehicles are only permitted at night in most big cities.
- Few trips are initiated between the hours of around 8 a.m. and 2 p.m.

Recommendations:

- The early hours are the best times to do system updates because fewer orders will be impacted.
- Make that the server is operating well at night.

Distribution of total trip time taken

In [34]:
```
fig=plt.figure(figsize=(12,5))
sns.histplot(x=final_df['od_time_taken'])
plt.show()
```



Insight:

- Total time taken follows log-normal distribution

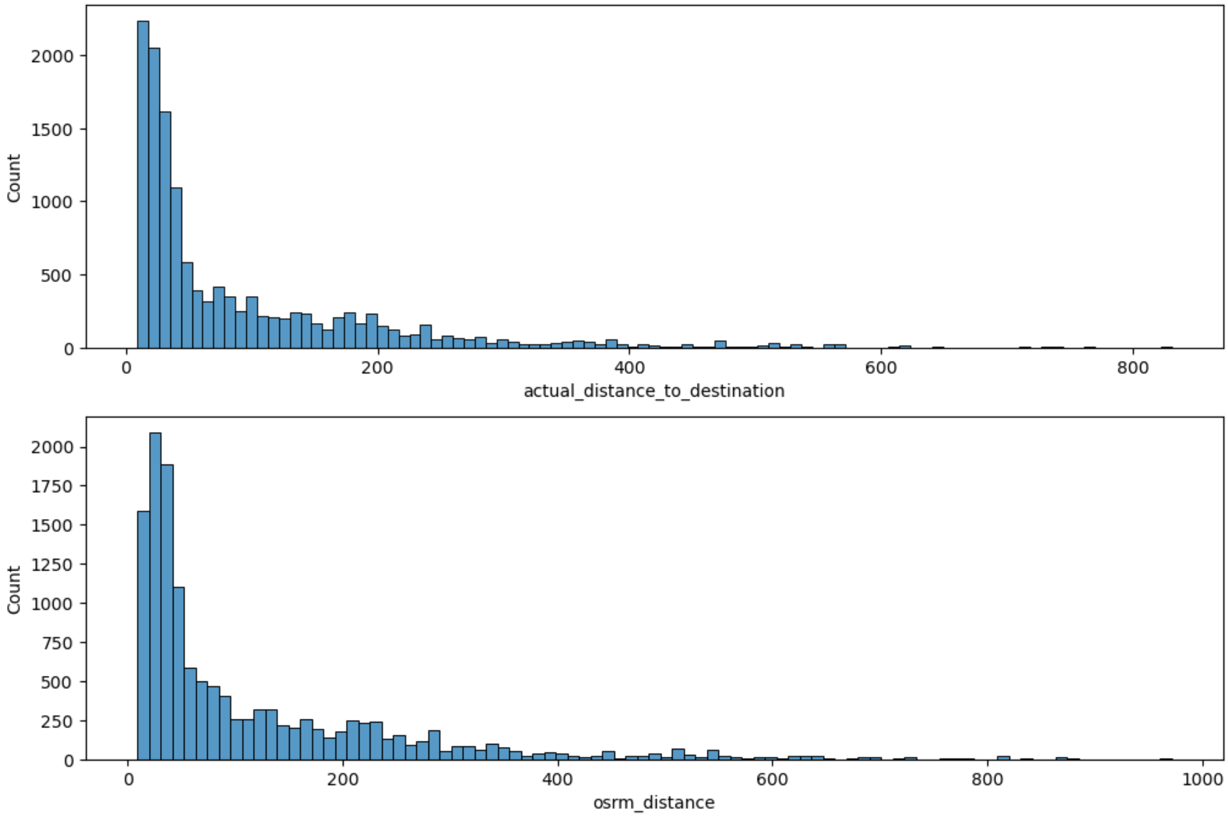
Distribution for Actual distance and OSRM Distance

In [35]:
```
fig=plt.figure(figsize=(12,8))

plt.subplot(2,1,1)
sns.histplot(x=final_df['actual_distance_to_destination'])

plt.subplot(2,1,2)
sns.histplot(x=final_df['osrm_distance'])

plt.show()
```



Insight:

- Both distance distributions follows log-normal distribution

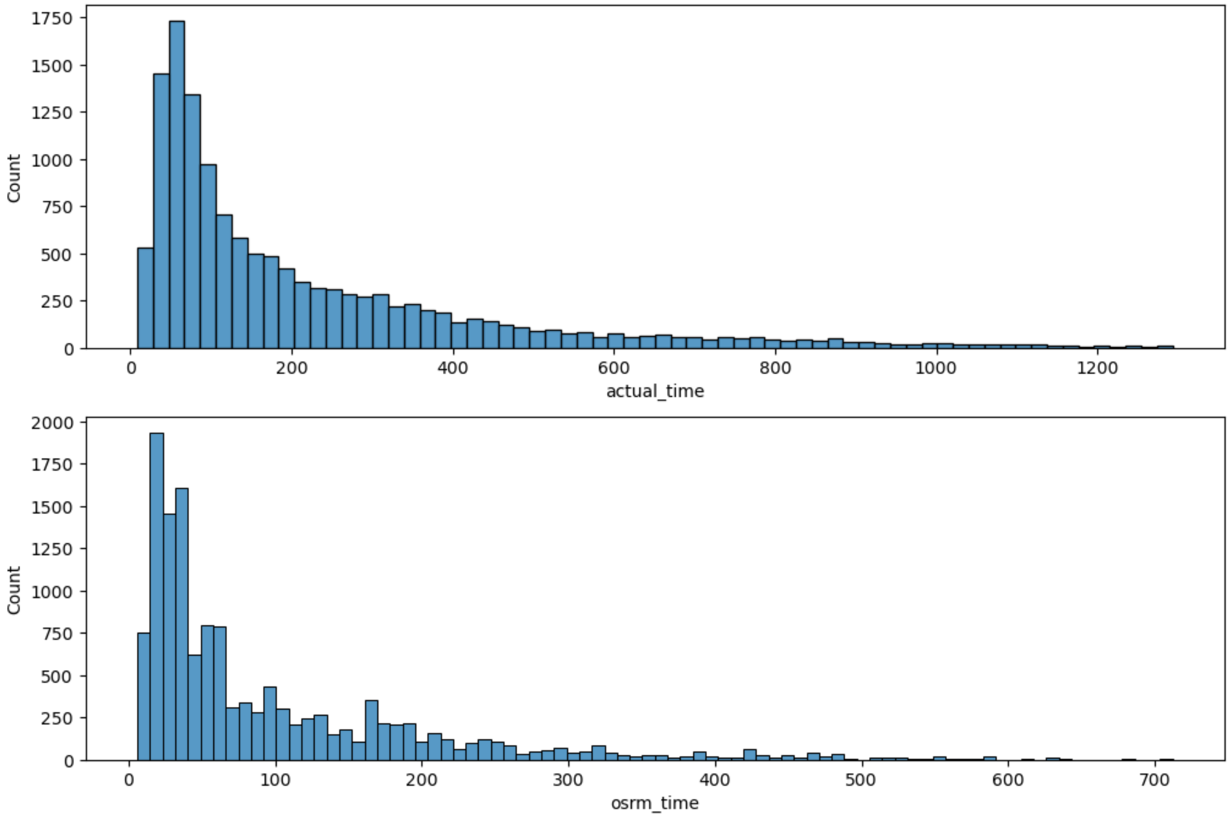
Distribution for Actual Time and OSRM Time

In [36]:
```
fig=plt.figure(figsize=(12,8))

plt.subplot(2,1,1)
sns.histplot(x=final_df['actual_time'])

plt.subplot(2,1,2)
sns.histplot(x=final_df['osrm_time'])

plt.show()
```

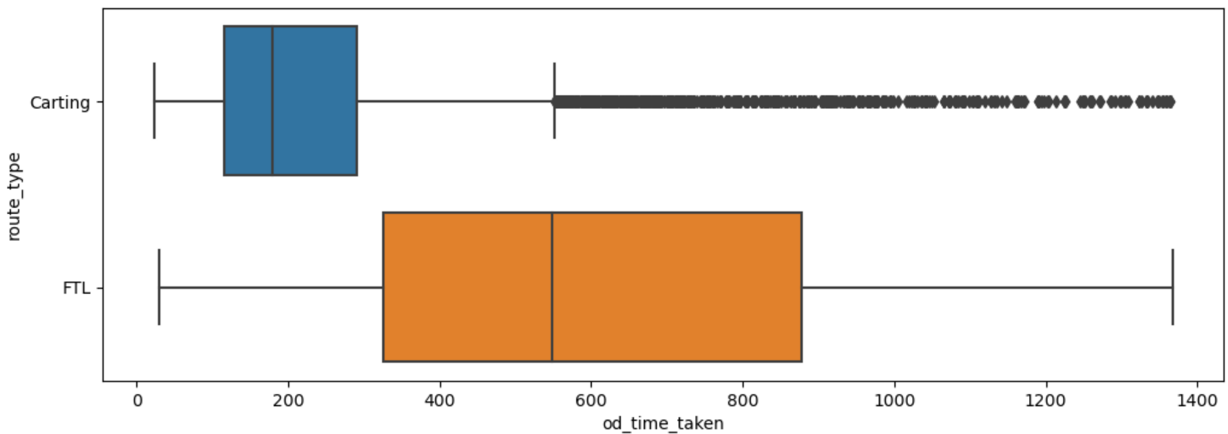


Insights:

- Both graphs follow log-normal distribution
- There is a significant difference between both graphs which seems Actual time differs highly from OSRM Time

Boxplot for each route type w.r.t. total time taken

In [37]:
```
fig=plt.figure(figsize=(12,4))
sns.boxplot(y='route_type',x='od_time_taken',data=final_df)
plt.show()
```

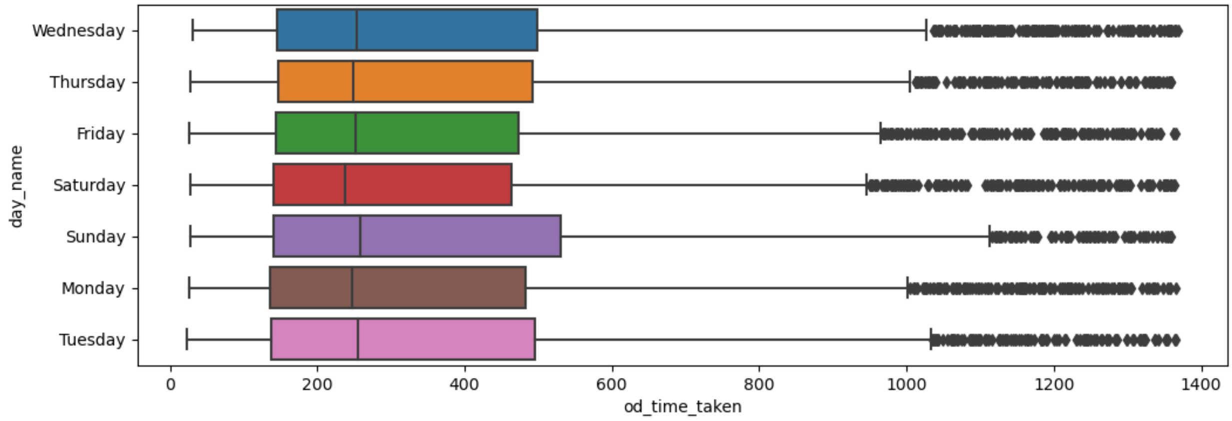


Insights:

- The median for the two route types varies significantly.
- The type of carting route has a lot of outliers.
- The median time for all trips to FTL is approximately nine hours.

Boxplot for various days w.r.t. to total trip time taken

In [38]:
```
fig=plt.figure(figsize=(12,4))
sns.boxplot(y='day_name',x='od_time_taken',data=final_df)
plt.show()
```



Insights:

- Every day's median trip time is nearly equal.
- The Sunday has the fewest outliers while the Saturday has the most.
- A trip took, on average, four hours to complete from start to finish.

One- Hot Encoding

Performing one-hot encoding for route_type category for making it useful for machine learning algorithms that require numerical input

In [39]:
```
def is_carting(x):
    if x=='Carting':
        return 1
    return 0
```

In [40]:
```
final_df['Carting']=final_df['route_type'].apply(lambda x:is_carting(x))
```

Standardization using StandardScaler

Performing Standardization to scale numerical features to a standard range, making them comparable and preventing some features from dominating others

In [41]:
```
def standardize(values):
    return (values - values.mean())/values.std()
```

In [42]:
```
cols = ['od_time_taken', 'start_scan_to_end_scan','actual_distance_to_destination',
        'osrm_distance', 'actual_time','osrm_time', 'segment_actual_time',
        'segment_osrm_time','segment_osrm_distance']

# Standardize the feature columns

standardized_df=final_df.copy(deep=True)
standardized_df[cols] = standardized_df[cols].apply(standardize)
```

Here, the final dataframe is not altered; rather, a copy of it is made, and the values in the corresponding columns are then substituted.

Hypothesis Testing

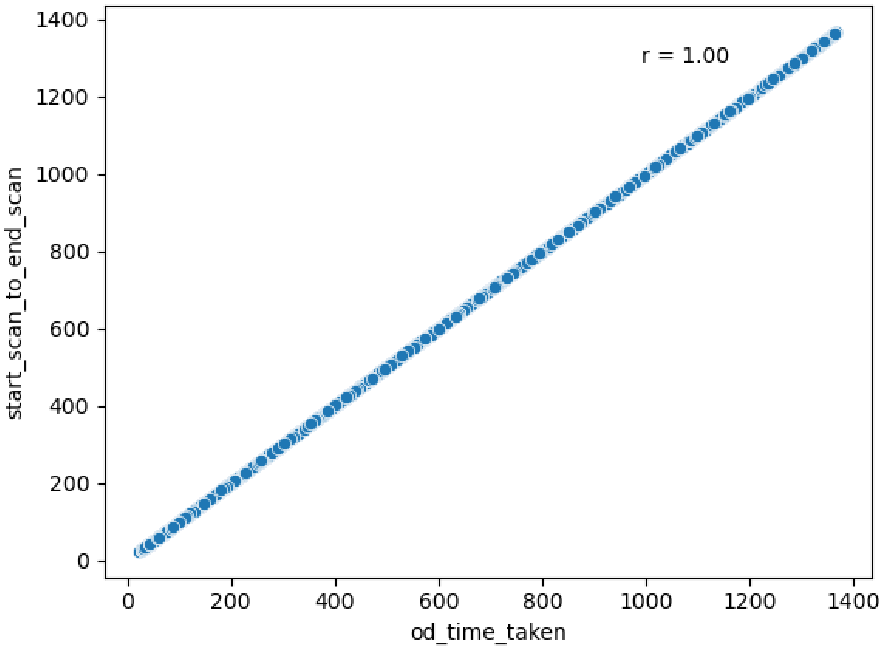
(We are checking the correlation between the columns with Pearson Technique)

In [43]:
```
# We take significance value (aplha) of 0.05

def hyp_checker(x):
    if x>0.05:
        print("Failed to Reject the Null Hypothesis")
    else:
        print("Reject the Null Hypothesis")
```

OD_time_taken vs Start_to_end_scan_time

In [44]:
```
sns.scatterplot(x='od_time_taken',y='start_scan_to_end_scan',data=final_df)
r, p = pearsonr(x=final_df['od_time_taken'],y=final_df['start_scan_to_end_scan'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [45]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['od_time_taken'],final_df['start_scan_to_end_scan'])
hyp_checker(p)
print(p)

# Hence, means of both columns are equal
```

```
Failed to Reject the Null Hypothesis
1.0
```

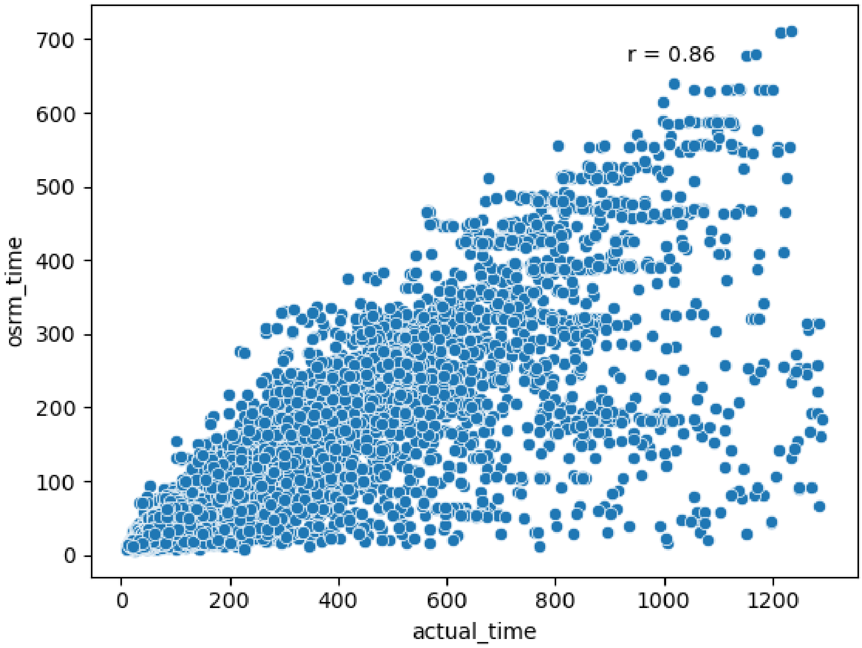
Insights:

- The means of the two columns are equal

- The total trip time and the start to finish scan have a strong correlation and are accurate

Actual_time vs OSRM time

In [46]:
```
sns.scatterplot(x='actual_time',y='osrm_time',data=final_df)
r, p = pearsonr(x=final_df['actual_time'],y=final_df['osrm_time'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [47]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['actual_time'],final_df['osrm_time'])
hyp_checker(p)
print(p)

# Means of both columns are not equal
```

```
Reject the Null Hypothesis
0.0
```

Insights:

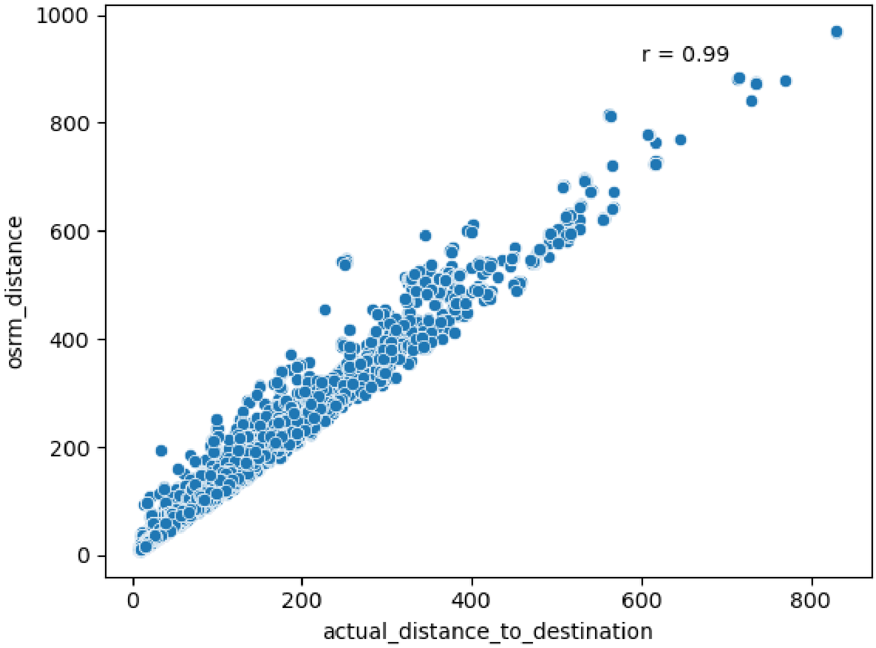
- Actual time and OSRM time are somewhat related but are not same. Correlation of .86 has been found.
- Means of both columns differ majorly.

Recommendation:

- Delhivery should improve or reconfigure the OSRM system of detecting time as in few cases they differ highly.
- Alternatively, Delhivery ought to resolve this issue with the drivers if they are to blame for being later than anticipated.

Actual_distance vs OSRM_distance

In [48]:
```
sns.scatterplot(x='actual_distance_to_destination',y='osrm_distance',data=final_df)
r, p = pearsonr(x=final_df['actual_distance_to_destination'],y=final_df['osrm_distance'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [49]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['actual_distance_to_destination'],final_df['osrm_distance'])
hyp_checker(p)
print(p)

# Hence, means of both columns are not equal
```

```
Reject the Null Hypothesis
6.843075736855715e-63
```

Insights:

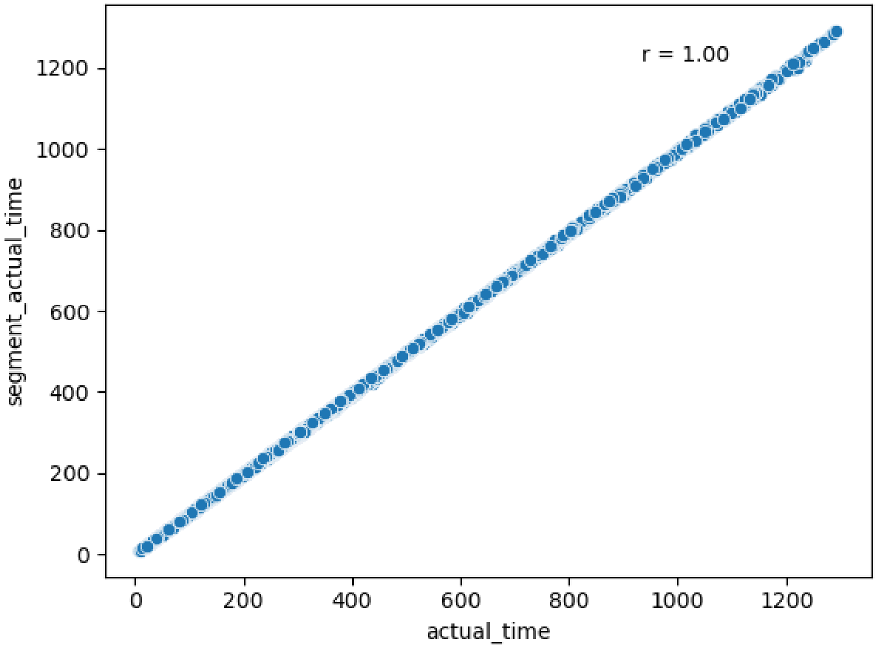
- The means of the two columns differ, despite the strong correlation between the OSRM and actual distances.
- The two columns' means are not equal.

Recommendation:

- Given the significant differences, Delhivery should modify or enhance the OSRM Distance Calculator.
- The system has to be modified since, in reality, as the distance grows, so does the gap between OSRM and Actual. They can add routes that actually exist but are not taken into consideration by OSRM in order to update it.

Actual_time vs Segment_Actual_time (Total)

In [50]:
```
sns.scatterplot(x='actual_time',y='segment_actual_time',data=final_df)
r, p = pearsonr(x=final_df['actual_time'],y=final_df['segment_actual_time'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [51]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['segment_actual_time'],final_df['actual_time'])
hyp_checker(p)
print(p)

# Hence, means of both columns are equal
```

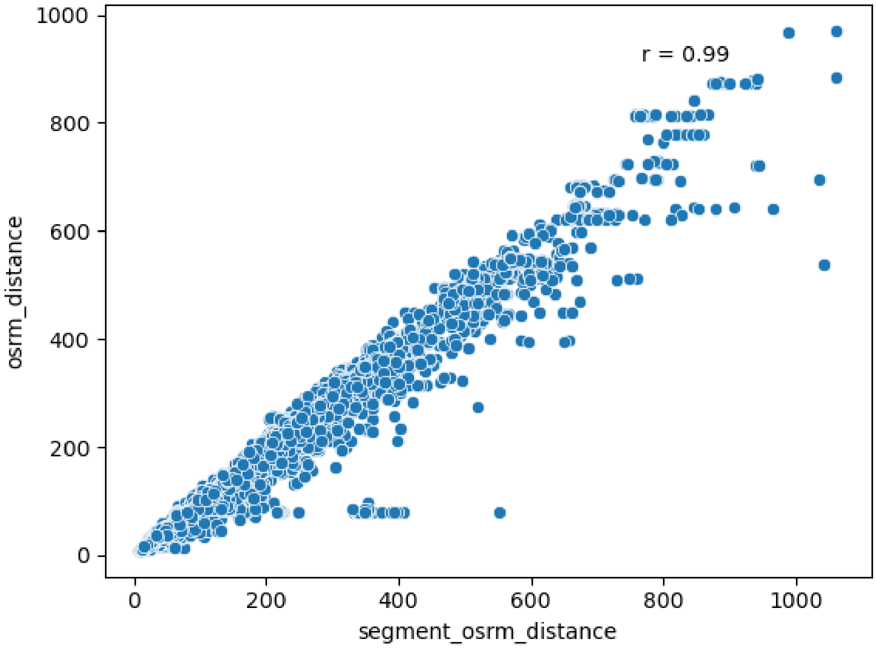
```
Failed to Reject the Null Hypothesis
0.48471236178400634
```

Insights:

- Total segment time and actual time are very highly correlated.
- Means of both columns are almost equal as well so we failed to reject the null hypothesis.

OSRM_Distance vs Segment_OSRM_distance (Total)

In [52]:
```
sns.scatterplot(x='segment_osrm_distance',y='osrm_distance',data=final_df)
r, p = pearsonr(x=final_df['segment_osrm_distance'],y=final_df['osrm_distance'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [53]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['segment_osrm_distance'],final_df['osrm_distance'])
hyp_checker(p)
print(p)

# Means of both columns are not equal
```

```
Reject the Null Hypothesis
1.0105786990635396e-06
```

Insights:

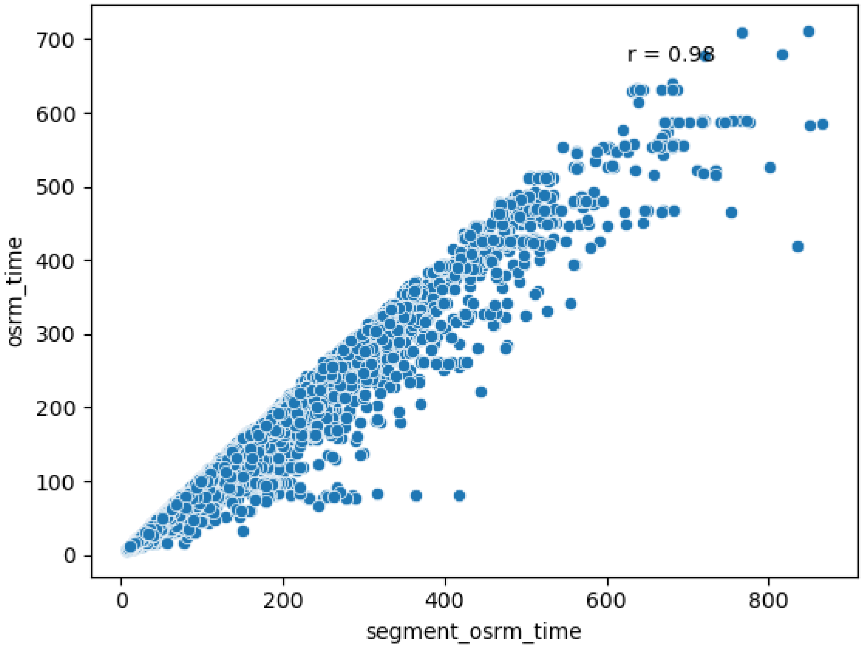
- There is a strong Pearson correlation between the two.
- Since we rejected the null hypothesis, the means of the two columns also differ.
- The OSRM Distance is greater than the sum of the segment OSRM Distance.

Recommendation:

- Since OSRM is calculating distance, there shouldn't be much of a difference between it and the segment total. However, this is where it differs greatly.
- Delhivery needs to investigate the cause of the issue and make improvements because the OSRM does not match the values.

OSRM_time vs Segment_OSRM_time (Total)

In [54]:
```
sns.scatterplot(x='segment_osrm_time',y='osrm_time',data=final_df)
r, p = pearsonr(x=final_df['osrm_time'],y=final_df['segment_osrm_time'])
plt.annotate('r = {:.2f}'.format(r), xy=(0.7, 0.9), xycoords='axes fraction')
plt.show()
```



In [55]:
```
# H0= Means of both columns are equal
# Ha= Means of both columns are not equal
stats,p=ttest_ind(final_df['osrm_time'],final_df['segment_osrm_time'])
hyp_checker(p)
print(p)

# Means of both columns are not equal
```

```
Reject the Null Hypothesis
4.2631844011033064e-13
```

Insights:

- Both are highly correlated as from pearson technique of correlation.
- By checking hypothesis, we failed to accept the null hypothesis and mean of both columns are not equal.
- In many cases total of segment ORSM times is higher than the overall OSRM time.

Recommendation:

- OSRM system needs to be updated or configured accordingly.

Highly Recommended:

- The routes in OSRM should be updated as sometimes actual distance is less than the OSRM.
- Either the time calculation of OSRM is faulty or drivers are longer than expected to deliver the product inspite having shorter routes in some cases.

Delhivery should take this into consideration

Now the dataset is ready for Data Scientists of Delhivery for building and training their Machine Learning Algorithms.