Applied Project Report
Analysis of Real-life Business Cases

By

Kalluru Srimannarayana Sharat Gupta

A Master's Project Report submitted to Scaler Neovarsity - Woolf in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

February, 2025

Scaler Mentee Email ID : kallurugupta@gmail.com

Thesis Supervisor : Shivank Agarwal

Date of Submission : 15/02/2025

Certification

I confirm that I have overseen / reviewed this applied project and, in my judgment, it adheres to the appropriate standards of academic presentation. I believe it satisfactorily meets the criteria, in terms of both quality and breadth, to serve as an applied project report for the attainment of Master of Science in Computer Science degree. This applied project report has been submitted to Woolf and is deemed sufficient to fulfill the prerequisites for the Master of Science in Computer Science degree.

Shivank Agarwal

……

Project Guide / Supervisor

DECLARATION

I confirm that this project report, submitted to fulfill the requirements for the Master of Science in Computer Science degree, completed by me from 20/02/2023 to 15/02/2025 is the result of my own individual endeavor. The Project has been made on my own under the guidance of my supervisor with proper acknowledgement and without plagiarism. Any contributions from external sources or individuals, including the use of AI tools, are appropriately acknowledged through citation. By making this declaration, I acknowledge that any violation of this statement constitutes academic misconduct. I understand that such misconduct may lead to expulsion from the program and/or disqualification from receiving the degree.

Kalluru Srimannarayana Sharat Gupta

Date: 15 February 2025

ACKNOWLEDGMENT

I would like to express my profound gratitude to all my Instructors & Teaching Assistants for their constant Support in the New Computer Science Journey.

A Big Shoutout to Scaler Team for Seamless integration of Leaning Programs and Creation of Wonderful platform for Enthusiastic Working Professional.

I would like to Thank all my batchmates who have made my learning into a wholesome Experience with their active participation and increasing enlightening by sharing Different Viewpoints with respect to their professional experience.

I would like to express my special thanks to my family for Support and efforts they provided throughout this Course. I am eternally grateful to them.

I would like to acknowledge that this project was completed entirely by me and not by someone else.

Kalluru Srimannarayana Sharat Gupta

Table of Contents

List of Tables

(To be written sequentially as they appear in the text)

List of Figures

(List of Images, Graphs, Charts sequentially as they appear in the text)

Applied Real-life Business Cases

ABSTRACT

Data science is the activity of analyzing vast amounts of unorganized and organized raw data to find patterns and draw conclusions that can be put to use. Data science is an interdisciplinary field, and the foundations include, inference, computer science, predictive analytics, the creation of machine learning techniques, and new tools for extracting information from large data sets.

With Increase in Digitalization, Each Business is generating huge amounts of Data, But Just generating Data does not benefit the Business. A Deep Analysis and Inference Extraction is Required to Support in Business Decisions and Machine Learning Model Development to Realize the Long-term Strategies with past Working Data

So Deep Analysis and Machine Learning Model Development, we need to Prepare the Data in systematic way else it will lead to Chaos.

Steps to be Followed:

1. Data Cleaning: it ensures the accuracy and consistency of your dataset, allowing for reliable results from any analysis you perform by removing errors, inconsistencies, and irrelevant data points, ultimately leading to more meaningful insights and better decision-making based on your findings.

   Ex: Removing Unknown Columns, Datatype Conversion, Duplicate Values Removal, Outlier Removing, Dealing Missing Values
2. Feature Engineering: it is process of transforming raw data into Meaningful features that can be used to train models.

   Ex: Extracting Day, week, month, from Date Data Type, Creating BMI from Weight and Height Features Etc
3. Exploratory Data Analysis: it is a method for analyzing data to identify patterns and relationships. It involves using visualizations, transformations, and models to understand the data.
4. Hypothesis Testing: It is a Statistical procedure to determine if there is enough evidence to support a claim about a population and find most effective Features for Model Building
5. Data Preparation: it is the Process where we fix our Data for Model Building
   a. by Splitting whole data into Training, validation and Testing Sets
   b. by Normalizing /Standardizing & Feature Encoding to Transform Categorical to Numerical
6. Model Building & Assumptions Checking: it the process where we
   a. Start Building our Models as per the Output Requirement like Continuous [Real Value] or Categorical [Probability] Output.
   b. After Building Model We need to Check if Data use meets the Assumptions for Selected model

      Ex: Linearity, No Multicollinearity, Homoscedasticity, Residuals Normality, No Auto Correlation to be verified for Regression Model
7. Validation & Testing: Once the Training is Done

a. Validation Dataset is Used to Tune the Hyperparameter of the Model to Achieve the Best Model Performance Metric
b. While Testing Data Set is used only once & That after Finalization of all parameter's ad Hyperparameters

8. Deployment & Monitoring: Once the Model is Ready
    a. We Containerize the Trained Model, Required Libraries and OS into a Container to make it run independent of it Deployed Environment and uniformity.
    b. Once Deployed we have Continuously monitor Model Performance for Data Shifting & Whenever need we have to Retrain, redeploy to meet Required Goal

Chapter 1 : Business Case Study 1

Problem Description

About Company

A Logistics and Supply Chain Company which is largest and fastest-growing fully integrated player in India by revenue in Fiscal 2021 aims 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.

Problem:

A Logistics and Supply Chain 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

DataSet:

- 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 wayCarting: 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) (Also Include Human Operation time between tranfer from one segment to other)
- 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 (Only time for that segmentg, No human operation time involved)
-

Libraries Used:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns



from scipy.stats import  ttest_ind

from scipy.stats import shapiro,levene
from statsmodels.graphics.gofplots import qqplot

from scipy.stats import mannwhitneyu

!pip install category_encoders
from category_encoders import TargetEncoder
!pip install sklearn
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,
StandardScaler, MinMaxScaler
```

Outcome of Analysis:

Basic data cleaning and exploration:
** Handle missing values in the data.
** Analyze the structure of the data.
** Try merging the rows using the hint mentioned above.
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
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 (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)
** Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)
** Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)
** Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)
** 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

Business Questions to be answered from Analysis

• 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

As Explained, Clean and effective Data is most important Inputs for Building best Model

If we do not clean data it build good Models and Also if we do not extract meaningful feature that impact the Output of Model, that model performance will not be as effective as it should be

So we will deeply Analyze the Data, Clean it & Extract Meaningful Features for Data Science team to build models on in this case study

Analysis

Let Import Data and Check the Basic Data Structure

```
[ ] data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhivery_data.csv")

[ ] data.shape

(144867, 24)

[ ] data.columns

Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

```
[ ] data.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destina
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_M
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_M
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_M
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_M
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat_M

Table 1 : Table 1 : Logistics Trip Record Raw Data Set

Data Cleaning:

Let's Clean the Dataset by Removing Unknown Columns:

- is_cutoff – Unknown field,
- cutoff_factor – Unknown field,
- cutoff_timestamp – Unknown field
- factor – Unknown field
- segment_factor – Unknown field

```
[ ] data.drop(columns = ["is_cutoff","cutoff_factor","cutoff_timestamp","factor","segment_factor"], axis = 1,inplace= True)
```

```
[ ] data.shape
```

```
(144867, 19)
```

```
data.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)
```

Unknown Columns have been Dropped.

Let’s Convert the Datatype of Features as per data:

Lets convert possible Column to "Cateory" datatype

- route_type – Transportation type
- data - tells whether the data is testing or training data

```
[ ] data["route_type"]=data["route_type"].astype("category")
    data["data"]=data["data"].astype("category")
```

Lets Convert Datetime Columns as datatime64[ns]

- trip_creation_time – Timestamp of trip creation
- od_start_time – Trip start time
- od_end_time – Trip end time

```
[ ] data["trip_creation_time"]=data["trip_creation_time"].astype("datetime64")
    data["od_start_time"]=data["od_start_time"].astype("datetime64")
    data["od_end_time"]=data["od_end_time"].astype("datetime64")
```

```
data.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  category
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  category
 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  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 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: category(2), datetime64[ns](3), float64(8), object(6)
```

Data type of 5 Features have been changed.

Let's Check the Duplicates Rows and Treat them:

```
[ ] data.loc[data.duplicated()]
```

data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	destination_name	od_start_time

No Duplicate row Found

Let's Check the Missing Values and Treat them:

```
[ ] data.isna().sum(axis = 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
```

- Missing Values are present only in source_name & Destination_name
- As there are not missing values in source_centre & destination_centre, we can check if Source_name & Destination_name existing in any of the rows

```
[ ] sourcenamemissingsourcesentre = data.loc[data["source_name"].isnull() == True]["source_center"].unique()
    sourcenamemissingsourcesentre
```

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

```
[ ] j = 1
    for i in sourcenamemissingsourcesentre:
      missedsourcename = data.loc[data["source_center"] == i]["source_name"].unique()
      if pd.isna(missedsourcename):
        data.loc[data["source_center"] == i,"source_name"] = "unknownsource "+str(j)
        j+=1
      else:
        data.loc[data["source_center"] == i,"source_name"]["source_name"] = missedsourcename[:1]
```

```
[ ] for i in sourcenamemissingsourcesentre:
      print(i, data.loc[data["source_center"] == i]["source_name"].unique())
```

```
IND342902A1B ['unknownsource 1']
IND577116AAA ['unknownsource 2']
IND282002AAD ['unknownsource 3']
IND465333A1B ['unknownsource 4']
IND841301AAC ['unknownsource 5']
IND509103AAC ['unknownsource 6']
IND126116AAA ['unknownsource 7']
IND331022A1B ['unknownsource 8']
IND505326AAB ['unknownsource 9']
IND852118A1B ['unknownsource 10']
```

We Identified and labelled the Unknown Source Name to utilize the Data points, Now lets do the same for Unknown Destination names

```
[ ] destinationnamemissingdestinationsentre = data.loc[data["destination_name"].isnull() == True]["destination_center"].unique()
    destinationnamemissingdestinationsentre
```

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

```
[ ] j = 1
    for i in destinationnamemissingdestinationsentre:
      misseddestinationname = data.loc[data["destination_center"] == i]["destination_name"].unique()
      if pd.isna(misseddestinationname):
        data.loc[data["destination_center"] == i,"destination_name"] = "unknowndestination "+str(j)
        j+=1
      else:
        data.loc[data["destination_center"] == i,"destination_name"]["destination_name"] = misseddestinationname[:1]
```

```
[ ] for i in destinationnamemissingdestinationsentre:
      print(i, data.loc[data["destination_center"] == i]["destination_name"].unique())
```

```
IND342902A1B ['unknowndestination 1']
IND577116AAA ['unknowndestination 2']
IND282002AAD ['unknowndestination 3']
IND465333A1B ['unknowndestination 4']
IND841301AAC ['unknowndestination 5']
IND505326AAB ['unknowndestination 6']
IND852118A1B ['unknowndestination 7']
IND126116AAA ['unknowndestination 8']
IND509103AAC ['unknowndestination 9']
IND221005A1A ['unknowndestination 10']
IND250002AAC ['unknowndestination 11']
IND331001A1C ['unknowndestination 12']
IND122015AAC ['unknowndestination 13']
```

```
[ ] data.isna().sum(axis=0)
```

```
data                              0
trip_creation_time                0
route_schedule_uuid               0
route_type                        0
trip_uuid                         0
source_center                     0
source_name                       0
destination_center                0
destination_name                  0
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
```

All the Missing values have been Resolved.

Let's Analyze Row Wise Data to See if Rows can be merged.

As can be seen from data row, there are multiple row for each "trip_uuid", we have to Merge multiple rows to get consolidated "od_start_time", "od_end_time", Source & Destination details with summed up Distance and time value

- Let us verify one trip_uuid & one route_schedule_uuid Details & see how multiple rows can be merged

```
[ ] data.loc[data["trip_uuid"] == data["trip_uuid"].unique()[0]]
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_center	source_name	destination_center	dest
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat
2	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat
3	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat
4	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND388121AAA	Anand_VUNagar_DC (Gujarat)	IND388620AAB	Khambhat

```
[ ] data.loc[data["route_schedule_uuid"] == data["route_schedule_uuid"].unique()[0]]["trip_uuid"].value_counts()
```

```
trip-153680339869927048    11
trip-153697725798753764    11
trip-153818828153597720    11
trip-153741093647649320    10
trip-153757917674683146    10
trip-153792558519954345    10
trip-153810169136762438    10
trip-153836091722390431    10
trip-153671811509671845    10
trip-153801468900715290    10
trip-153689022134280351     6
trip-153723562875380861     5
trip-153853302491268026     5
trip-153845108130043002     5
Name: trip_uuid, dtype: int64
```

```
[ ] data.loc[data["route_schedule_uuid"] == data["route_schedule_uuid"].unique()[0]]["source_name"].value_counts()
```

```
Khambhat_MotvdDPP_D (Gujarat)    69
Anand_VUNagar_DC (Gujarat)       54
Anand_Vaghasi_IP (Gujarat)        1
Name: source_name, dtype: int64
```

```
[ ] data.loc[data["route_schedule_uuid"] == data["route_schedule_uuid"].unique()[0]]["destination_name"].value_counts()
```

```
Anand_Vaghasi_IP (Gujarat)       69
Khambhat_MotvdDPP_D (Gujarat)    54
Anand_VUNagar_DC (Gujarat)        1
Name: destination_name, dtype: int64
```

- From above 4 cell we can say that "route_schedule_uuid" futher consists of many "trip_uuid"
- Further each "trip_uuid has many destination & sources -- Because of this we cannot directly "groupby" on "trip_uuid"
- First we will use "groupby" on 3 columns - ['trip_uuid','source_center','destination_center'] and use aggregations as needed for each feature & again then we will "groupby"" on 1 column "trip_uuid

```
[ ] data1 = data.groupby(['trip_uuid','source_center','destination_center']).agg({'data' : 'first',
    'route_schedule_uuid' : 'first' ,
    'route_type' : 'first',
    'trip_creation_time' : 'first',
    'source_name' : 'first',
    'destination_name' : 'last',
    'od_start_time' : 'first',
    'od_end_time' : 'first',
    'start_scan_to_end_scan' : 'first',
    'actual_distance_to_destination' : 'last',
    'actual_time' : 'last',
    'osrm_time' : 'last',
    'osrm_distance' : 'last',
    'segment_actual_time' : 'sum',
    'segment_osrm_time' : 'sum',
    'segment_osrm_distance' : 'sum'}).reset_index()
```

```
[ ] data.shape
```

```
(144867, 19)
```

```
[ ] data1.shape
```

```
(26368, 19)
```

- Now lets do second groupby with only "trip_uuid"
- Here we have to sort any of "od-start_time" or "od_end_time" for each "trip_uuid" to ensure we get correct initial source & final destination for each trip_uuid

```
[ ] data1.sort_values(by=['trip_uuid', 'od_start_time'], ascending=[True,True],inplace = True,ignore_index=True)
```

```
[ ] data2 = data1.groupby(['trip_uuid']).agg({'data' : 'first',
    'route_schedule_uuid' : 'first' ,
    'route_type' : 'first',
    'trip_creation_time' : 'first',
    'source_name' : 'first','source_center' : 'first',
    'destination_name' : 'last','destination_center' : 'last',
    'od_start_time' : 'first',
    'od_end_time' : 'last',
    'start_scan_to_end_scan' : 'sum',
    'actual_distance_to_destination' : 'sum',
    'actual_time' : 'sum',
    'osrm_time' : 'sum',
    'osrm_distance' : 'sum',
    'segment_actual_time' : 'sum',
    'segment_osrm_time' : 'sum',
    'segment_osrm_distance' : 'sum'}).reset_index()
```

```
[ ] data1.shape
(26368, 19)

[ ] data2.shape
(14817, 19)
```

So After Merging Row we were able to Consolidate all information of 144867 rows → 14817 rows
Now Lets cross check if we have achieved a Single Consolidate row for each “trip_uuid”

```
[ ] data.loc[data["trip_uuid"] == "trip-153671041653548748"]
```

124993	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124994	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124995	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124996	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124997	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124998	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	Kar
124999	training	2018-09-12	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-	FTL	trip-	IND462022AAA	Bhopal_Trnsport_H	IND209304AAA	Kar

```
[ ] data1.loc[data1["trip_uuid"] == "trip-153671041653548748"]
```

	trip_uuid	source_center	destination_center	data	route_schedule_uuid	route_type	trip_creation_time	source_name	destinati
0	trip-153671041653548748	IND462022AAA	IND209304AAA	training	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	2018-09-12 00:00:16.535741	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Cer (Uttar
1	trip-153671041653548748	IND209304AAA	IND000000ACB	training	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	2018-09-12 00:00:16.535741	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bila (I

```
[ ] data2.loc[data2["trip_uuid"] == "trip-153671041653548748"]
```

	trip_uuid	data	route_schedule_uuid	route_type	trip_creation_time	source_name	source_center	destination_name	destination_
0	trip-153671041653548748	training	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	2018-09-12 00:00:16.535741	Bhopal_Trnsport_H (Madhya Pradesh)	IND462022AAA	Gurgaon_Bilaspur_HB (Haryana)	IND0000

- As can be seen from above 3 cells now we can single row for each "trip_uuid" with proper od_start_time, od_end_time, Source & Destination details with summed up Distance and time value

Outlier Treatment:

Outlier Treatment Can be done only for Numerical features, so we will consolidate all Numerical features in one list for easy reference

```
[ ] numericalfeatures = ["start_scan_to_end_scan","od_total_time","actual_distance_to_destination","actual_time","segment_actual_time", "osrm_time",
     "osrm_distance","segment_osrm_time","segment_osrm_distance"]
```

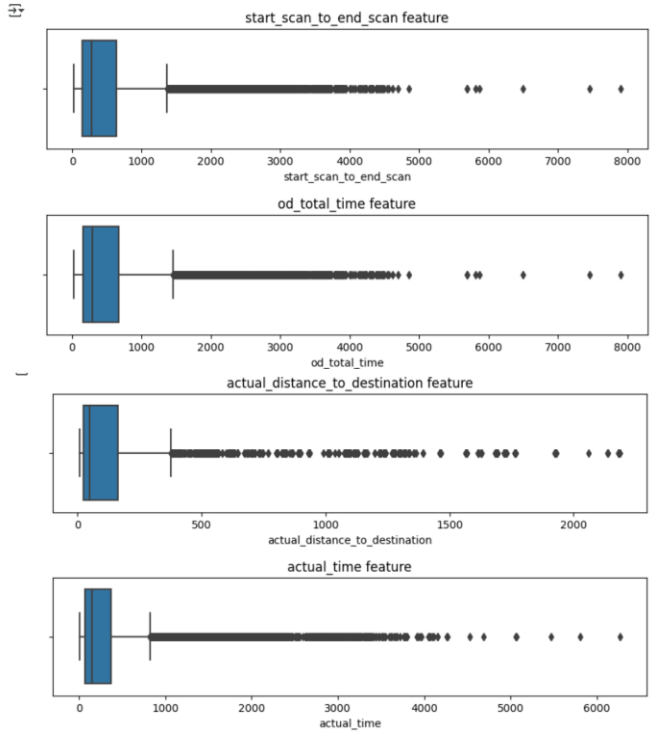
We will create a new Dataframe " outlier" which store Boolean value of IQR Analysis Result for each reading of all numerical feature columns

```
# Finding outlier for all Numerical features
outlier = pd.DataFrame() # creating as new dataframe to store outliers for each numerical feature
for z in numericalfeatures:
    print(z,"Feature")
    print("**********************************************************************")
    IQR = np.percentile(data2[z],75) - np.percentile(data2[z],25)
    lower_limit = max (np.percentile(data2[z],25) - 1.5*IQR,0)
    upper_limit = np.percentile(data2[z],75) + 1.5*IQR
    outlier[z] = (data2[z]>upper_limit) | (data2[z]<lower_limit)

    print("Minimum -->",data2[z].min())
    print("Lower Limit -->",lower_limit)
    print("Quantile25 -->",np.percentile(data2[z],25))
    print("Median -->",np.percentile(data2[z],50))
    print("Quantile75 -->",np.percentile(data2[z],75))
    print("Upper Limit -->",upper_limit)
    print("Maximum -->",data2[z].max())

    print()
    print("Out of ",len(data2[z]), "Data points , There are",outlier[z].sum(),"Outliers")
    print("Percentage of Outliers:",np.round((outlier[z].sum()/len(data2[z])*100),2),"%")
    print()
    print("**********************************************************************")
```

```
start_scan_to_end_scan Feature
**********************************************************************
Minimum --> 23.0
Lower Limit --> 0
Quantile25 --> 149.0
Median --> 280.0
Quantile75 --> 637.0
Upper Limit --> 1369.0
Maximum --> 7898.0

Out of  14817 Data points , There are 1267 Outliers
Percentage of Outliers: 8.55 %
```

Let us visualize the outliers with Box plot

```
[ ] for z in numericalfeatures:
      plt.figure(figsize=(10,2))
      sns.boxplot(data= data2, x = z)
      plt.title(z+" feature")
```



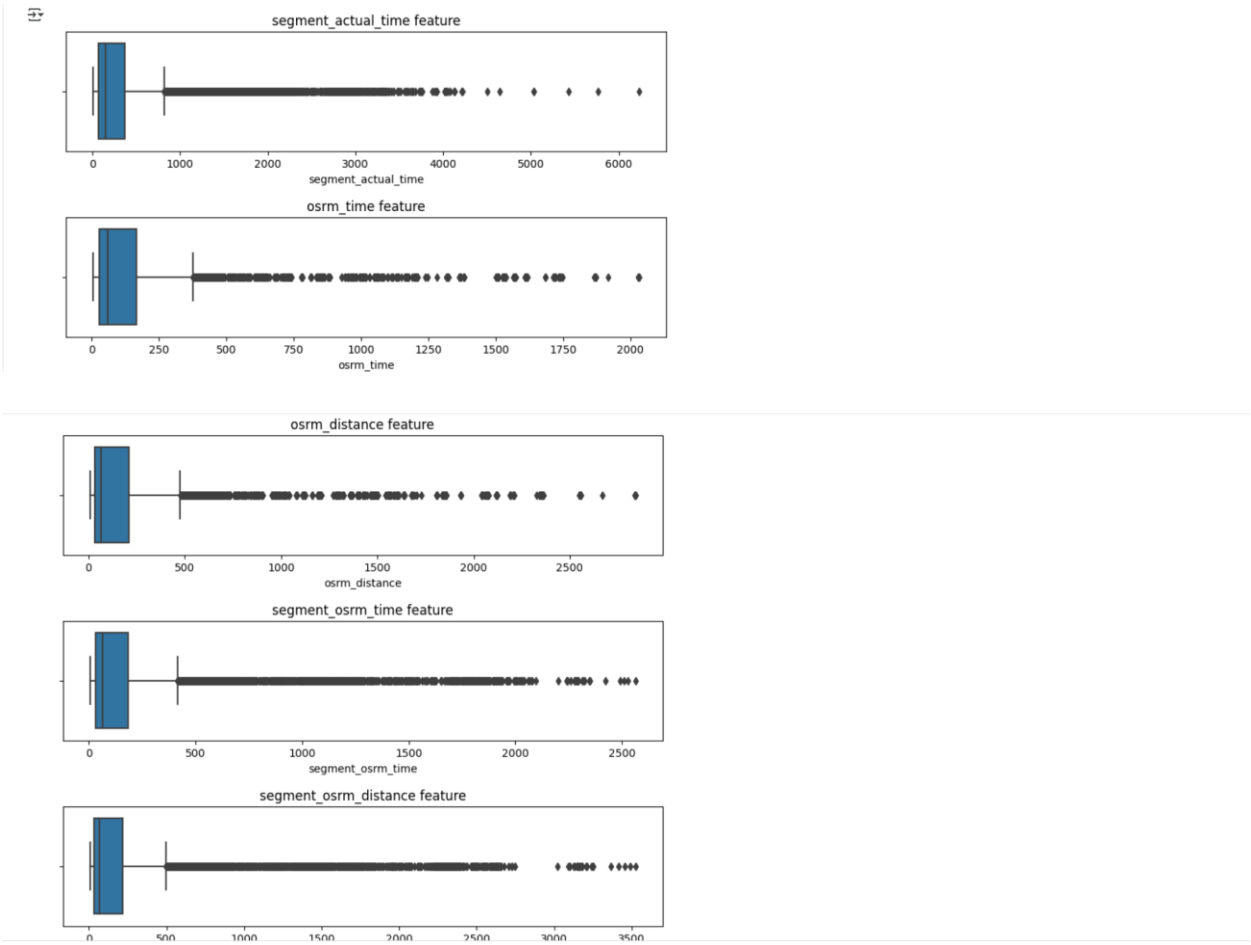


Figure 1 : Logistics Records Outlier

let us check if outliers are really outliers or they are occurring due to long distance/ Duration Courier

```
[ ] outlier.shape

(14817, 9)

[ ] outlier["count"] = outlier.sum(axis=1)

[ ] # Checking for rows which have come as outliers in all 9 numerical columns
    outlier[outlier["count"]==9].count()

start_scan_to_end_scan            943
od_total_time                     943
actual_distance_to_destination    943
actual_time                       943
segment_actual_time               943
osrm_time                         943
osrm_distance                     943
segment_osrm_time                 943
segment_osrm_distance             943
count                             943
dtype: int64
```

```
[ ] # Checking for rows which have come as outliers in all 9 numerical columns
    data2.iloc[outlier[outlier["count"]==9].index]
```

	trip_uuid	data	route_schedule_uuid	route_type	trip_creation_time	source_name	source_center	destination_name	destination_center	start_scan_to_end_scan	...	trip_cr
0	trip-153671041653548748	training	thanos::sroute:d7c989ba-a29b-4a0b-b2f4-288cdc6...	FTL	2018-09-12 00:00:16.535741	Bhopal_Trnsport_H (Madhya Pradesh)	IND462022AAA	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	2259.0	...	
2	trip-153671043369099517	training	thanos::sroute:de5e208e-7641-45e6-8100-4d9fb1e...	FTL	2018-09-12 00:00:33.691250	Bangalore_Nelmngla_H (Karnataka)	IND562132AAA	Chandigarh_Mehmdpur_H (Punjab)	IND160002AAC	3933.0	...	
41	trip-153671321710455800	training	thanos::sroute:951d77aa-4725-4c4e-882d-42acc35...	FTL	2018-09-12 00:46:57.104787	Bhiwandi_Mankoli_HB (Maharashtra)	IND421302AAG	Gurgaon_Bilaspur_HB (Haryana)	IND000000ACB	2338.0	...	
43	trip-153671328307356992	training	thanos::sroute:64d4c6c9-ffb9-4794-b9f1-05f064c...	FTL	2018-09-12 00:48:03.073766	Delhi_Airport_H (Delhi)	IND110037AAM	Bhiwandi_Mankoli_HB (Maharashtra)	IND421302AAG	2302.0	...	
62	trip-153671547254076660	training	thanos::sroute:2a713f58-e06f-4251-a6f0-f374373...	FTL	2018-09-12 01:24:32.541032	Hyderabad_Shamshbd_H (Telangana)	IND501359AAE	Hyderabad_Shamshbd_H (Telangana)	IND501359AAE	1792.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	...

- There are total 943 Rows which have come as outlier in all 9 numerical features
- But we cannot delete the rows just because of the IQR rule, these outliers might be trips which are having very long distance & duration
- So all these outliers can be good Data points only
- so we cannot drop these outliers data point
- we will use them as OK

<u>Feature Engineering 1:</u>

Now That We Have Cleaned all the Data and Consolidate our “Trip_uuid” Data, we will extract additional Relevant features for building good ML Model

Features Consolidation:

In this We try to Extract Relevant Single Feature from multiple Features to reduce the Dimensionality because Dimensionality Plays important role in the performance of the Models

- "od_start_time" & "od_end_time"can be converted to single feature by calcualting time difference between both

```
[ ] data2['od_total_time'] = (data2['od_end_time'] - data2['od_start_time']) / np.timedelta64(1, 'm')
    data2.drop(columns = ['od_end_time', 'od_start_time'], inplace = True)
    data2.head()
```

:an	actual_distance_to_destination	actual_time	osrm_time	osrm_distance	segment_actual_time	segment_osrm_time	segment_osrm_distance	od_total_time
9.0	824.732854	1562.0	717.0	991.3523	1548.0	1008.0	1320.4733	2260.109800
0.0	73.186911	143.0	68.0	85.1110	141.0	65.0	84.1894	181.611874
3.0	1927.404273	3347.0	1740.0	2354.0665	3308.0	1941.0	2545.2678	3934.362520
0.0	17.175274	59.0	15.0	19.6800	59.0	16.0	19.8766	100.494935
7.0	127.448500	341.0	117.0	146.7918	340.0	115.0	146.7919	718.349042

Feature Extraction:

Let Extract Multiple New Features from Single Feature

We can extract Day, Week, Month & Year from Trip_Creation_time to Give Grasp Seasonal Factor to our ML Model

Extract features like month, year, day & week from Trip_creation_time

```
[ ] data2['trip_creation_hour'] = data2['trip_creation_time'].dt.hour
    data2['trip_creation_day'] = data2['trip_creation_time'].dt.day
    data2['trip_creation_week'] = data2['trip_creation_time'].dt.isocalendar().week
    data2['trip_creation_month'] = data2['trip_creation_time'].dt.month
    data2['trip_creation_year'] = data2['trip_creation_time'].dt.year
```

All 5 above new Features are Cateogorical type, So lets convert them into Category Datatype

```
[ ] data2['trip_creation_hour'] = data2['trip_creation_hour'].astype("category")
    data2['trip_creation_day'] = data2['trip_creation_day'].astype("category")
    data2['trip_creation_week'] = data2['trip_creation_week'].astype("category")
    data2['trip_creation_month'] = data2['trip_creation_month'].astype("category")
    data2['trip_creation_year'] = data2['trip_creation_year'].astype("category")
```

```
[ ] data2.nunique()
```

```
trip_uuid                         14817
data                                  2
route_schedule_uuid                1504
route_type                            2
trip_creation_time                14817
source_name                         868
source_center                       868
destination_name                    956
destination_center                  956
start_scan_to_end_scan             2208
actual_distance_to_destination    14801
actual_time                        1853
osrm_time                           817
osrm_distance                     14734
segment_actual_time                1890
segment_osrm_time                  1242
segment_osrm_distance             14754
od_total_time                     14817
trip_creation_hour                   24
trip_creation_day                    22
trip_creation_week                    4
trip_creation_month                   2
trip_creation_year                    1
dtype: int64
```

- In this Dataset "trip_creation_year" is having only 1 unique value, this feature can be dropped
- However, we will keep the feature for now [so that this code can be generalized for any similar dataset in future]

- Now We Extract Feature like City,State from "source_name" & "destination_name"

```
[ ] data2["source_name"].unique()[:10]
```

```
array(['Bhopal_Trnsport_H (Madhya Pradesh)',
       'Tumkur_Veersagr_I (Karnataka)',
       'Bangalore_Nelmngla_H (Karnataka)', 'Mumbai Hub (Maharashtra)',
       'Bellary_Dc (Karnataka)', 'Chennai_Porur_DPC (Tamil Nadu)',
       'Chennai_Chrompet_DPC (Tamil Nadu)', 'HBR Layout PC (Karnataka)',
       'Surat_Central_I_4 (Gujarat)', 'Delhi_Lajpat_IP (Delhi)'],
      dtype=object)
```

- we will have to use multiple split function to separate State, City & Place_code
- We will create functions to commonly apply for destination_name & source_name

```
[ ] def state(x):
      y = x.split("(")
      if len(y) == 2:
        return y[1].replace(")","")
      elif len(y)== 1:
        return x
```

```
[ ] state('Mumbai Hub (Maharashtra)')
```

```
'Maharashtra'
```

```
[ ] state("unknownsource 1")
```

```
'unknownsource 1'
```

```
[ ] "city_place_code_state".split("_")
```

```
['city', 'place', 'code', 'state']
```

```
[ ] def city(x):
      if "unknown" in x:
        return x
      elif "_" in x:
        y=x.split("_")
      else:
        y=x.split(" ")
      if len(y)>1:
        return y[0]
```

```
[ ] city('Mumbai Hub (Maharashtra)')
```

```
'Mumbai'
```

```
[ ] city('Chennai_Porur_DPC (Tamil Nadu)')
```

```
'Chennai'
```

```
[ ] city("unknownsource 1")
```

```
'unknownsource 1'
```

```
[ ] "city_place_code_state".split("_",1)
```

```
['city', 'place_code_state']
```

```
[ ] def place(x):
      if "unknown" in x:
        return x
      elif "_" in x:
        y=x.split("_",1)
      else:
        y=x.split(" ",1)
      z = y[1].split(" (")
      return z[0]
```

```
[ ] place('Mumbai Hub (Maharashtra)')
```

```
'Hub'
```

```
[ ] place('Chennai_Porur_DPC (Tamil Nadu)')
```

```
'Porur_DPC'
```

```
[ ] place("unknownsource 1")
```

```
'unknownsource 1'
```

- All 3 User Functions state(),city(),place() are created and checked for proper functionality for ll types of inputs
- Now we will use these functions to create 3 new feature(State, City, Place) from sourc_name & destination_name

```
[ ] data2["source_state"] = data2["source_name"].apply(state)
    data2["source_city"] = data2["source_name"].apply(city)
    data2["source_place"] = data2["source_name"].apply(place)
    data2["destination_state"] = data2["destination_name"].apply(state)
    data2["destination_city"] = data2["destination_name"].apply(city)
    data2["destination_place"] = data2["destination_name"].apply(place)
```

```
[ ] data2["destination_state"]
```

```
0              Haryana
1            Karnataka
2               Punjab
3          Maharashtra
4            Karnataka
               ...
14812           Punjab
14813          Haryana
14814    Uttar Pradesh
14815       Tamil Nadu
14816        Karnataka
Name: destination_state, Length: 14817, dtype: object
```

6 New Features Have been Created for Hierarchy type Inputs to ML Team, Lets Convert them to Category.

```
[ ] data2["source_state"] = data2["source_state"].astype("category")
    data2["source_city"] = data2["source_city"].astype("category")
    data2["source_place"] = data2["source_place"].astype("category")
    data2["destination_state"] = data2["destination_state"].astype("category")
    data2["destination_city"] = data2["destination_city"].astype("category")
    data2["destination_place"] = data2["destination_place"].astype("category")
```

<u>Statistical Analysis, Graphical Visualization & Insights</u>

Unique Values in Features

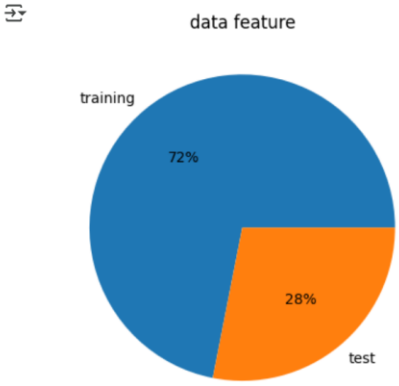
```
[ ] data2.nunique()
```

```
trip_uuid                         14817
data                                  2
route_schedule_uuid                1504
route_type                            2
trip_creation_time                14817
source_name                         868
source_center                       868
destination_name                    956
destination_center                  956
start_scan_to_end_scan             2208
actual_distance_to_destination    14801
actual_time                        1853
osrm_time                           817
osrm_distance                     14734
segment_actual_time                1890
segment_osrm_time                  1242
segment_osrm_distance             14754
od_total_time                     14817
trip_creation_hour                   24
trip_creation_day                    22
trip_creation_week                    4
trip_creation_month                   2
trip_creation_year                    1
source_state                         33
source_city                         668
```

Table 2 : Final Feature Extracted with Unique Values

Data Type:

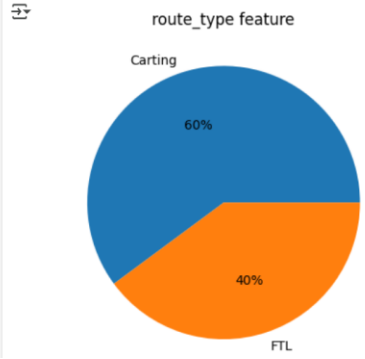
```
[ ]  plt.pie(x = data2["data"].value_counts().reset_index()["data"],
             labels = data2["data"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("data feature")
     plt.show()
```



- 72% Data is used for Training
- 28% Data is used for Test

Route_type Features

```
plt.pie(x = data2["route_type"].value_counts().reset_index()["route_type"],
        labels = data2["route_type"].value_counts().reset_index()["index"],
        autopct='%.0f%%')
plt.title("route_type feature")
plt.show()
```

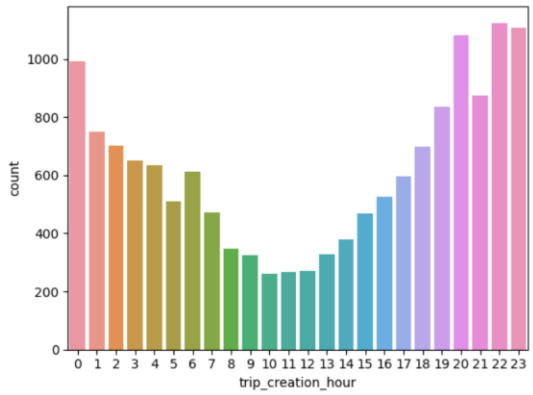


- 60% Trips are of Cart type
- 40% Trips are of FTL type

Trip_Creation_hour:

```
sns.countplot(data=data2, x= "trip_creation_hour")
```

```
<Axes: xlabel='trip_creation_hour', ylabel='count'>
```

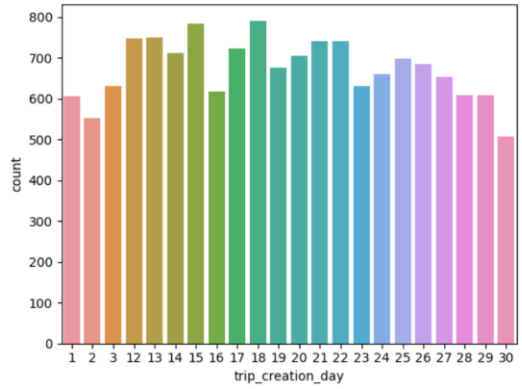


- Trip creations are lowest in the Noon and Starts to increase and reaches peak in midnight

Trip_creation_day:

```
[ ] sns.countplot(data=data2, x= "trip_creation_day")
```

```
<Axes: xlabel='trip_creation_day', ylabel='count'>
```

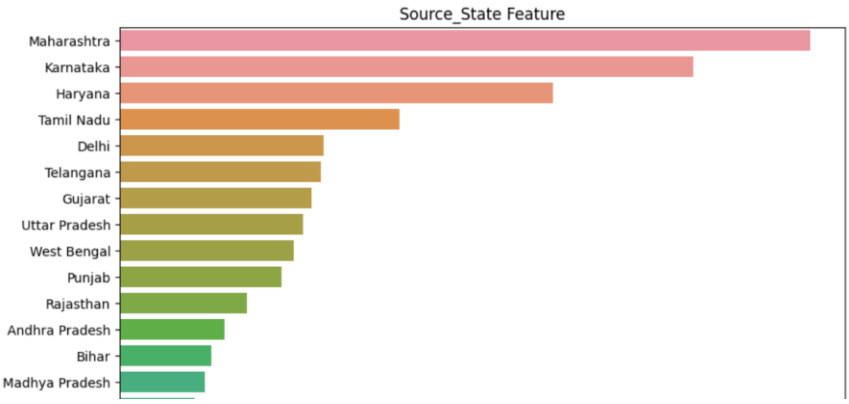


- More Number of Trip are created in the moiddle of the month and Number of trips are less at start and End of month

Source_State_Feature:

```
plt.figure(figsize =(10,12))
sns.countplot(data=data2, y= "source_state",order=data2['source_state'].value_counts().index)
plt.title("Source_State Feature")
```

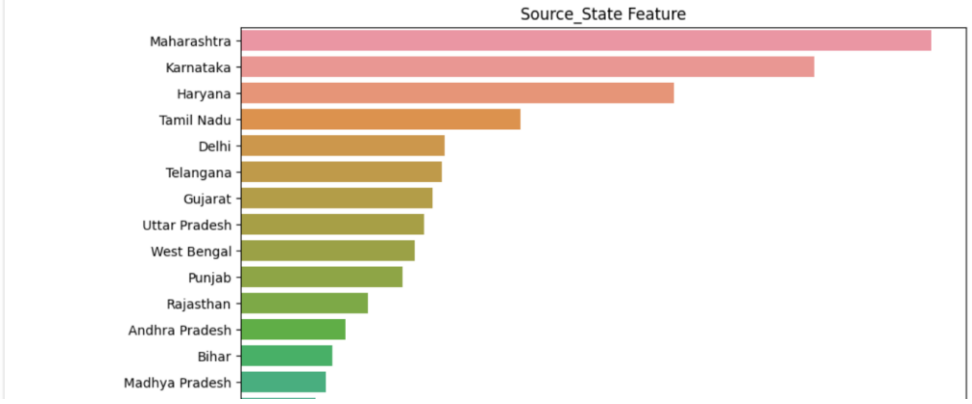
```
Text(0.5, 1.0, 'Source_State Feature')
```



Destination_State_Feature:

```
plt.figure(figsize =(10,12))
sns.countplot(data=data2, y= "source_state",order=data2['source_state'].value_counts().index)
plt.title("Source_State Feature")
```

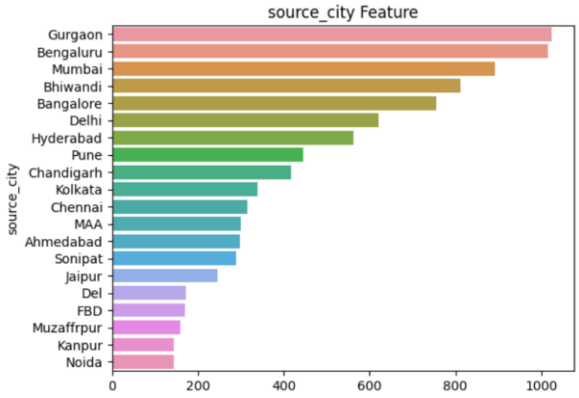
```
Text(0.5, 1.0, 'Source_State Feature')
```



Source_City_Feature:

```
[ ] sns.countplot(data=data2, y= "source_city",order=data2['source_city'].value_counts().index[:20])
    plt.title("source_city Feature")
```

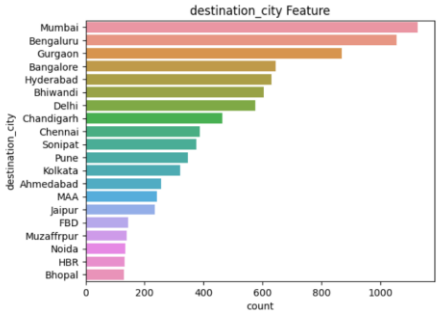
```
Text(0.5, 1.0, 'source_city Feature')
```



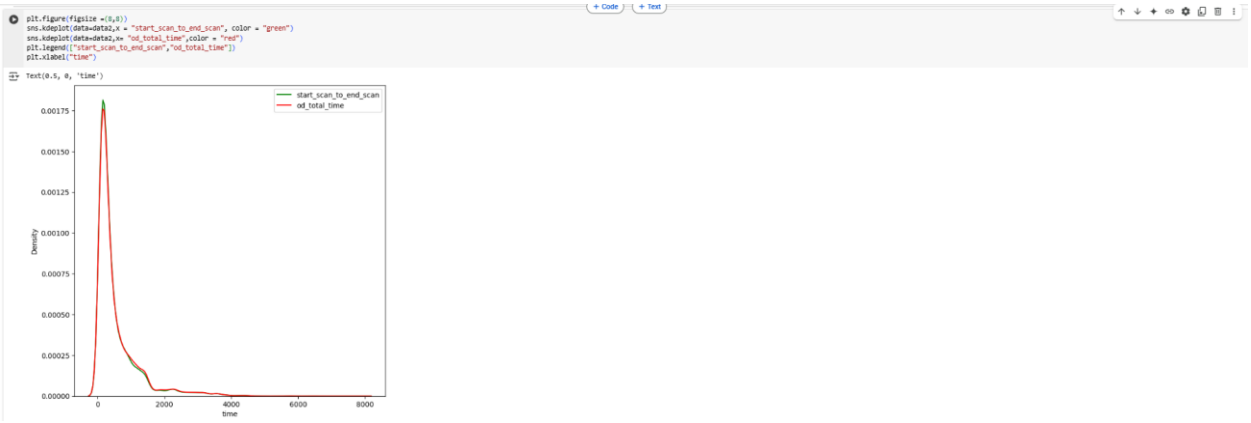
Destination_City_Feature:

```
sns.countplot(data=data2, y= "destination_city",order=data2['destination_city'].value_counts().index[:20])
plt.title("destination_city Feature")
```

```
Text(0.5, 1.0, 'destination_city Feature')
```

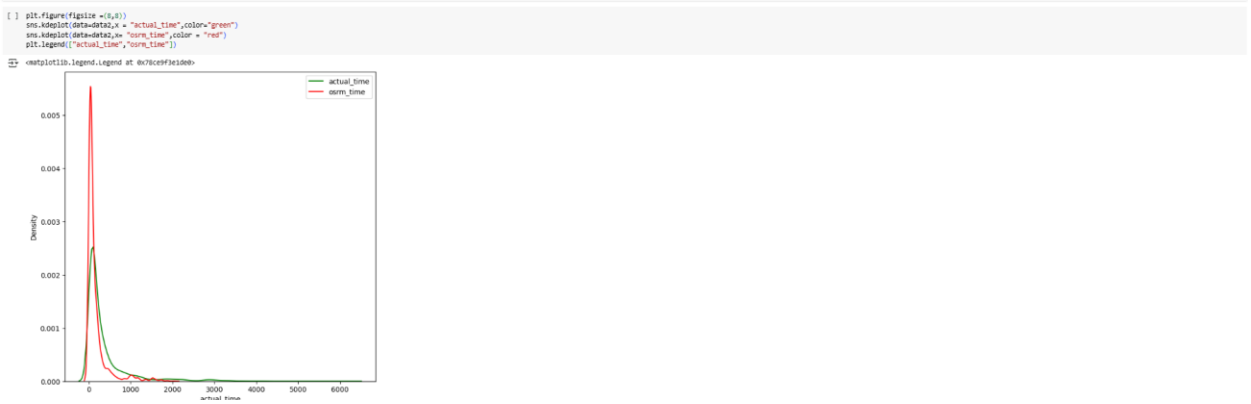


Start_scan_to_end_scan vs od_total_time:



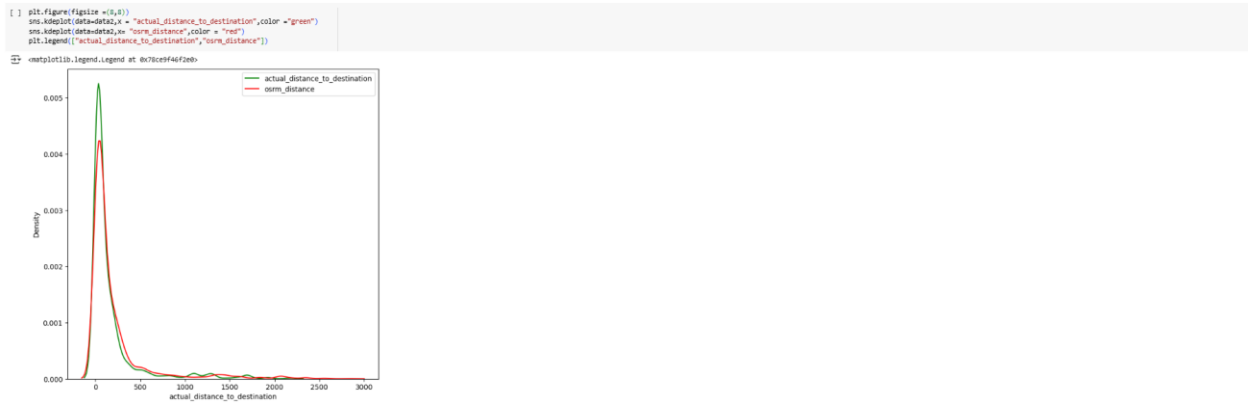
- od_total_time & start_scan_to_end_scan almost look like similar
- Further we can perform Hypothesis Testing to conclude

Actual_time VS OSRM_time:



- Actual_time is seems to lower than osrm_time

Actual_distance_to_distance Vs osrm_distance:



- Actual_distance_to_destination seems to be greater than osrm_distance

Correlation Matrix:

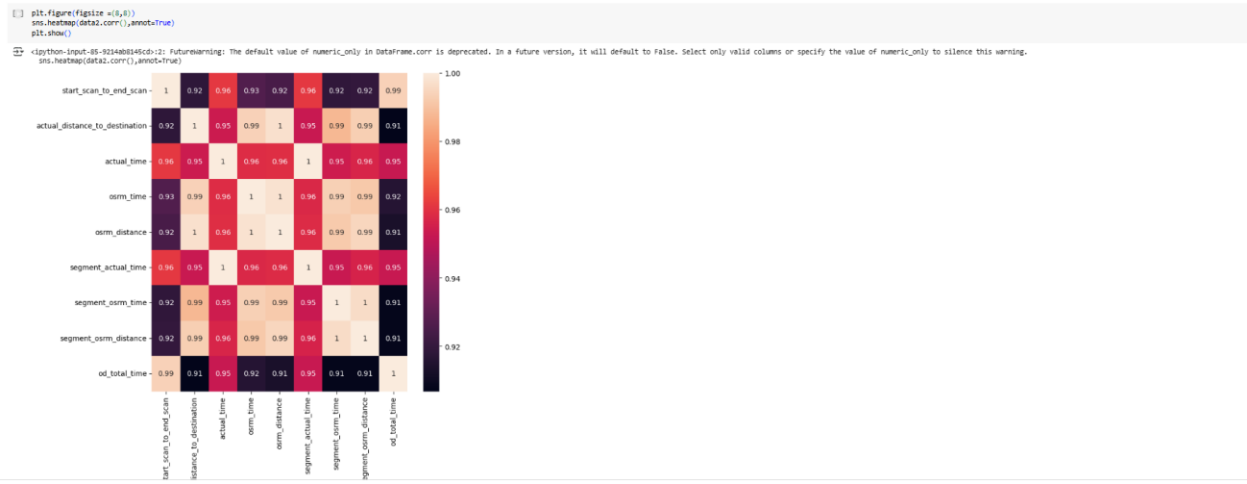
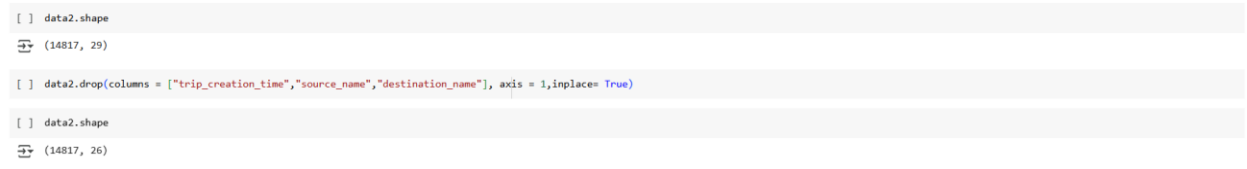


Figure 2 : Correlation Matrix for Logistics Records

- All Time & Distance Feature are very highly correlated.

<u>Feature Engineering 2:</u>

Eliminate Redundant features.

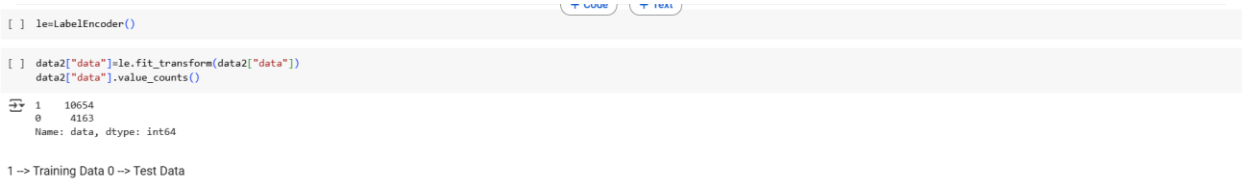


Encoding: Non-Numerical to Numerical

As Machine Learning Models are programs, they cannot understand Categorical Features, so we have to convert them to Numerical through Encoding.

- We have to Convert all Non Numerical to Numerical Before feeding to Model
- we will use following to to convert all the Categorical data to Numerical
- "One-Hot Encoding" for Non Numerical features with 2 values [in Problem statement "One-Hot Encoding" explicity Requested, else we can be "label Encoder" also]
- "Target Encoder" for Non Numerical features with more than 2 values with "actual_time" as Target"

Label Encoding:



One-Hot Encoding:

route_type

- As Explicity asked in the Problem statement, we are converting "route_type" Categorical to Numerical with One Hot Encoding

```
[ ] data2["route_type"].value_counts()
```

```
Carting    8908
FTL        5909
Name: route_type, dtype: int64
```

```
[ ] data2 = pd.get_dummies(data2, columns = ["route_type"])
```

```
[ ] data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 27 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   trip_uuid                     14817 non-null  object
 1   data                          14817 non-null  int64
 2   route_schedule_uuid           14817 non-null  object
 3   source_center                 14817 non-null  object
 4   destination_center            14817 non-null  object
 5   start_scan_to_end_scan        14817 non-null  float64
 6   actual_distance_to_destination  14817 non-null  float64
 7   actual_time                   14817 non-null  float64
 8   osrm_time                     14817 non-null  float64
 9   osrm_distance                 14817 non-null  float64
 10  segment_actual_time           14817 non-null  float64
 11  segment_osrm_time             14817 non-null  float64
 12  segment_osrm_distance         14817 non-null  float64
 13  od_total_time                 14817 non-null  float64
 14  trip_creation_hour            14817 non-null  category
 15  trip_creation_day             14817 non-null  category
 16  trip_creation_week            14817 non-null  category
 17  trip_creation_month           14817 non-null  category
 18  trip_creation_year            14817 non-null  category
 19  source_state                  14817 non-null  category
 20  source_city                   14817 non-null  category
 21  source_place                  14817 non-null  category
 22  destination_state             14817 non-null  category
 23  destination_city              14817 non-null  category
 24  destination_place             14817 non-null  category
 25  route_type_Carting            14817 non-null  uint8
 26  route_type_FTL                14817 non-null  uint8
dtypes: category(11), float64(9), int64(1), object(4), uint8(2)
memory usage: 1.9+ MB
```

- we can see "route_type" category columns has been One Hot Encoded --> it converted into 2 Numerical columns : route_type_Carting & route_type_FTL
- Now we will converts other Non-Numrical columns with >2 group using Target Encoder

Target Encoding:

As we are working to Build Forecast Models for "Actual_time", we will use it as Target for converting Categorical to Numerical using TargetEncoder

- Creating as TargetEncoder Object

```
[ ] te=TargetEncoder()
```

source_center

```
[ ] data2["source_center"].value_counts()
```

```
IND000000ACB    948
IND421302AAG    811
IND562132AAA    731
IND560099AAB    426
IND160002AAC    370
               ...
IND504215AAA      1
IND844101AAB      1
IND284403AAA      1
IND621212AAA      1
IND303338AAB      1
Name: source_center, Length: 868, dtype: int64
```

```
[ ] data2["source_center"]=te.fit_transform(data2["source_center"],data2["actual_time"])
    data2["source_center"].value_counts()
```

```
754.231013    948
469.427867    811
553.675787    731
99.046948     426
638.697297    370
             ...
339.820623      1
351.660494      1
364.931559      1
324.858149      1
344.894854      1
Name: source_center, Length: 789, dtype: int64
```

```
▾ others

[ ] data2.columns

Index(['trip_uuid', 'data', 'route_schedule_uuid', 'source_center',
       'destination_center', 'start_scan_to_end_scan',
       'actual_distance_to_destination', 'actual_time', 'osrm_time',
       'osrm_distance', 'segment_actual_time', 'segment_osrm_time',
       'segment_osrm_distance', 'od_total_time', 'trip_creation_hour',
       'trip_creation_day', 'trip_creation_week', 'trip_creation_month',
       'trip_creation_year', 'source_state', 'source_city', 'source_place',
       'destination_state', 'destination_city', 'destination_place',
       'route_type_Carting', 'route_type_FTL'],
      dtype='object')

[ ] nonnumerical = ['destination_center','trip_uuid','route_schedule_uuid','trip_creation_hour',
                    'trip_creation_day', 'trip_creation_week', 'trip_creation_month',
                    'trip_creation_year', 'source_state', 'source_city', 'source_place',
                    'destination_state', 'destination_city', 'destination_place']

[ ] for i in nonnumerical:
        data2[i]=te.fit_transform(data2[i],data2["actual_time"])

[ ] for i in data2.columns:
        print(i,"-->",data2[i].dtype,"-->",data2[i].nunique())

trip_uuid --> float64 --> 1853
data --> int64 --> 2
route_schedule_uuid --> float64 --> 1443
source_center --> float64 --> 789
destination_center --> float64 --> 840
start_scan_to_end_scan --> float64 --> 2208
actual_distance_to_destination --> float64 --> 14801
actual_time --> float64 --> 1853
osrm_time --> float64 --> 817
osrm_distance --> float64 --> 14734
segment_actual_time --> float64 --> 1890
segment_osrm_time --> float64 --> 1242
segment_osrm_distance --> float64 --> 14754
od_total_time --> float64 --> 14817
trip_creation_hour --> float64 --> 24
trip_creation_day --> float64 --> 22
trip_creation_week --> float64 --> 4
trip_creation_month --> float64 --> 2
trip_creation_year --> float64 --> 1
source_state --> float64 --> 33
source_city --> float64 --> 613
source_place --> float64 --> 659
destination_state --> float64 --> 41
destination_city --> float64 --> 684
destination_place --> float64 --> 714
route_type_Carting --> uint8 --> 2
route_type_FTL --> uint8 --> 2
```

Figure 3 : Final Feature after Feature Engineering

All Feature have been converted to Numerical data type

Scaling:

Before Scaling we will store the Row index for test and Tain data, so that we can separate data after Scaling

```
[ ] data2.shape

(14817, 27)

[ ] testtrain = data2["data"]
    data2.drop(columns = ["data"],axis=1,inplace=True)

[ ] data2.shape

(14817, 26)
```

- From PCA Mathematical Derivation, we know that if Any Feature's Mean is "0", it will be easy to do Computation.
- Also, If Feature's Mean = 0, then Model Development will be good
- So, we will do Standardization instead of Normalization.

Standardization

```
[ ] scaler = StandardScaler()
```

```
[ ] standardized = scaler.fit_transform(data2)
```

```
[ ] standardized
```

```
array([[ 2.14625072,  0.75265239,  0.92618362, ...,  1.46387707,
        -1.22781549,  1.22781549],
       [-0.38146143, -0.24588095, -0.29061509, ..., -0.20542008,
         0.81445462, -0.81445462],
       [ 5.32593091,  5.1735464 ,  0.79201088, ...,  0.78807576,
        -1.22781549,  1.22781549],
       ...,
       [-0.13385608, -0.39274303, -0.03958484, ...,  0.7209205 ,
         0.81445462, -0.81445462],
       [-0.16592008, -0.18628031, -0.45690199, ..., -0.42665845,
         0.81445462, -0.81445462],
       [-0.14632542, -0.13928955, -0.43218183, ..., -0.3540543 ,
        -1.22781549,  1.22781549]])
```

- lets convert standardized data into our initial DataFrame Format

```
[ ] datafinal = pd.DataFrame(standardized,columns = data2.columns)
```

```
[ ] datafinal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14817 entries, 0 to 14816
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   trip_uuid                       14817 non-null  float64
 1   route_schedule_uuid             14817 non-null  float64
 2   source_center                   14817 non-null  float64
 3   destination_center              14817 non-null  float64
 4   start_scan_to_end_scan          14817 non-null  float64
 5   actual_distance_to_destination  14817 non-null  float64
 6   actual_time                     14817 non-null  float64
 7   osrm_time                       14817 non-null  float64
 8   osrm_distance                   14817 non-null  float64
 9   segment_actual_time             14817 non-null  float64
 10  segment_osrm_time               14817 non-null  float64
 11  segment_osrm_distance           14817 non-null  float64
 12  od_total_time                   14817 non-null  float64
 13  trip_creation_hour              14817 non-null  float64
 14  trip_creation_day               14817 non-null  float64
 15  trip_creation_week              14817 non-null  float64
 16  trip_creation_month             14817 non-null  float64
 17  trip_creation_year              14817 non-null  float64
 18  source_state                    14817 non-null  float64
 19  source_city                     14817 non-null  float64
 20  source_place                    14817 non-null  float64
 21  destination_state               14817 non-null  float64
 22  destination_city                14817 non-null  float64
 23  destination_place               14817 non-null  float64
 24  route_type_Carting              14817 non-null  float64
 25  route_type_FTL                  14817 non-null  float64
dtypes: float64(26)
memory usage: 2.9 MB
```

Now we see:

- Data is Cleaned for Missing Values, Outlier, Data Type
- Data has been Converted to numerical for Model Building

Now we can Separate Data for Training and Testing

- we will use "testtrain" & divide data into X --> our Features Y --> Desired Outcome [Our desired outcome is "actual_time"]
- we will have 4 below datasets
  - Xtrain
  - Xtest
  - ytrain
  - ytest

```
[ ] y = data2["actual_time"]
    X = datafinal.drop(columns = ["actual_time"],axis=1)
```

```
[ ] X.shape, y.shape
```

```
((14817, 25), (14817,))
```

```
[ ] Xtrain = X.loc[testtrain == 1]
    Xtest = X.loc[testtrain == 0]
    ytrain = y.loc[testtrain == 1]
    ytest = y.loc[testtrain == 0]
```

```
[ ] Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape
```

```
((10654, 25), (4163, 25), (10654,), (4163,))
```

Business Insights

- The DataSet provided is between 2018-09-12 00:00:16.535741 & 2018-10-03 23:59:42.701692
- Training : Test Data Ratio use = 72:28
- 60% trips are of Cart Types, rest are of FTL type
- Trip creations are lowest in the Noon and Starts to increase and reaches peak in midnight
- More Number of Trip are created in the moiddle of the month and Number of trips are less at start and End of month
- There is not much effect of Week ina month for trip Creation
- Top 5 Source States:Maharastra,Karnataka,Harayana,Tamilnadu,Delhi
- Top 5 destination States: Maharastra,Karnataka,Harayana,Tamilnadu,Telangana
- Top5 Source Cities:Bengaluru,Gurgoan,Mumbai,Bhiwandi,Delhi
- Top5 Destination Cities: Bengaluru,Mumbai,Gurgoan,Hyderabad,Bhiwandi
- "od_total_time" is greater than "start_scan_to_end_scan" for a given trip_uuid
- "actual_time" is always greater than "osrm_time" for a given trip_uuid
- "actual_time" & "segment_actual_time" have approx same mean for a given trip_uuid
- "osrm_distance" is less than "segment_osrm_distance" for a given trip_uuid
- "osrm_time" is less than "segment_osrm_time" for a given trip_uuid
- "actual_distance_to_destination" is less than "osrm_distance" for a given trip_uuid

Recommendations:

- Major Traffic is found in Maharashtra, Karnataka, Haryana --> Appropriate infrastructure & Manpower needs to be maintained to reduce logistics Delays
- Carting Type Trip from among major Cities (Bengaluru, Gurgaon, Mumbai ,Hyderabad, Delhi, Bhiwandi) can be Converted to FTL by delhivery to further Optimize the Total trip Duration
- High Trip Creation goes on during nighttime, & in the middle of the month, so appropriate Resource to be maintained to reduce process bottle Necks.
- Actual Trip Time is always greater than Osrm predicted time, so Accordingly Buffer to be added, so that no false commitment at customer end
- Actual Distance_to_Destination is statistically lower than osrm distance, so correction might be needed for distance calculation so that price of trip can be optimized according & best value can be given to customer.

Chapter 2 : Business Case Study 2

Problem Description

About Company

A micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, company provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.

Company zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!

Problem:

Company wants to Determine factors affecting the demand for these shared electric cycles.

- Which variables are significant in predicting the demand for shared electric cycles?
- How well those variables describe the electric cycle demands.

Dataset:

- datetime: datetime
- season: season (1: spring, 2: summer, 3: fall, 4: winter)
- holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- weather:
    - 1: Clear, Few clouds, partly cloudy, partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: temperature in Celsius
- atemp: feeling temperature in Celsius
- humidity: humidity
- windspeed: wind speed
- casual: count of casual users
- registered: count of registered users
- total_count: count of total rental bikes including both casual and registered

Libraries Used:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import norm, binom, geom
```

```
from scipy.stats import ttest_1samp, ttest_ind,ttest_rel

from scipy.stats import chisquare,chi2,chi2_contingency

from scipy.stats import f_oneway,kruskal,shapiro,levene
from statsmodels.graphics.gofplots import qqplot

from scipy.stats import pearsonr,spearmanr

from scipy.stats import poisson,expon
!pip install category_encoders
from category_encoders import TargetEncoder
!pip install sklearn
from sklearn.preprocessing import LabelEncoder
```

Business Questions to be answered from Analysis

Statistically Determine the Significant Feature effecting the Demand of Shared Electric Cycle in Indian Market
Do Bivariate Analysis & Statistically conclude with Appropriate Hypothesis testing.

Hypothesis testing is important because it helps people make informed decisions based on Statistical Analysis. With Hypothesis testing we can Check Cause and Effect of Features on Outcome of ML Models and Reduce the Dimensionality of our Model

Example :

- 2- Sample T-Test to check if Working Day influences the number of electric cycles rented
- ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season
- Chi-square test to check if Weather is dependent on the season.

Analysis:

As we have seen Data Cleaning in Depth in Business Case 1, we will Do Univariate & Bivariate Analysis on Cleaned Data and Then Do Hypothesis testing to determine the Significant features effecting the demand of Shared cycle.

Dataset

```
[ ] data.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Table 3 : Dataset for Hypothesis testing

Here we will follow a Systematic Approach to Conclude w.r.t each Feature

1. Univariate Analysis
2. Bivariate Analysis
3. Hypothesis Testing

Univariate, Bivariate Analysis Set up

Lets define all the User defined functions for doing Bivariate analysis in this section

We will create user defined function to get Bivariate plots : "bivariateplot(info,cat,num)"

- info --> Dataframe
- cat --> any category column header
- num --> any numerical column header

```
[ ]  def bivariateplot(info,cat,num):
        plt.figure(figsize = (18,6))
        plt.subplot(1,4,1)
        plt.title("Mean Count value")
        sns.barplot(data=info,y=num,x = cat, estimator = "mean")
        plt.subplot(1,2,2)
        plt.title("Count Density")
        sns.kdeplot(data=info,x=num,hue = cat)
        plt.subplot(1,4,2)
        plt.title("Count box plot")
        sns.boxplot(data=info,y=num,x = cat)
        plt.show()
```

we will create another user defined function to get all statistical parameters of bivariate Boxplot

- cat --> any category column
- num --> any numerical column

```
def bivariateboxplotparameter(cat,num):
  A = data.groupby(cat)[num].describe()
  A["upper_Limit"] = (data.groupby(cat)[num].quantile(0.75)-data.groupby(cat)[num].quantile(0.25))*1.5+data.groupby(cat)[num].quantile(0.75)
  return A
```

Hypothesis Testing setup:

In this section let us fix the Confidence level & Level of Signification for all our Hypothesis testing

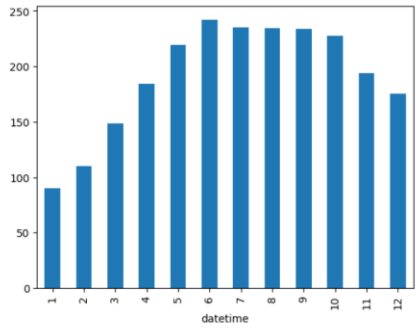
As this is with respect to predicting demand Situation

- let us Assume we want 95% Confidence to reject Null Hypothesis
- so, for every Hypothesis testing, Level of Significance will be "0.05 "

Impact of Datetime on User Count

```
data.groupby(data["datetime"].dt.month)["total_count"].mean().plot(kind="bar")
```

```
<Axes: xlabel='datetime'>
```



- User Count is highly active in the month from 5 ~ 10
- Lets us check if Datetime has any correlation with season

```
[ ] pd.crosstab(data["datetime"].dt.month,data["season"])
```

datetime	season	1	2	3	4
1		884	0	0	0
2		901	0	0	0
3		901	0	0	0
4		0	909	0	0
5		0	912	0	0
6		0	912	0	0
7		0	0	912	0
8		0	0	912	0
9		0	0	909	0
10		0	0	0	911
11		0	0	0	911
12		0	0	0	912

- It seems there is relation between Datetime Month & Season
- Lets conclude our inference with Hypothesis Testing

Lets do Hypothesis testing to check dependence of Seasons on datetime

Hypothesis testing:

As Datetime & Season are both Categorical Data, we will do Chi2_Contigency test to test their Dependence.

Null Hypothesis can be set as below.

- Ho: Datetime & Season are independent
- Ha: Datetime & Seasons are not Independent

```
[ ] A = pd.crosstab(data["datetime"].dt.month,data["season"])
```

```
[ ] # Ho: datetime & Season are independent
    # Ha: datetime & Seasons are not Independent

    chi_stat, p_value, df, exp_freq = chi2_contingency(A)

    print(p_value)
    print(exp_freq)
    if p_value < 0.05:
      print("Reject H0")
      print("datetime & Seasons are not Independent")
    else:
      print("Fail to Reject H0")
      print("datetime & Seasons are Independent")
```

```
0.0
[[218.11721477 221.93386    221.93386    222.01506522]
 [222.31177659 226.20181885 226.20181885 226.28458571]
 [222.31177659 226.20181885 226.20181885 226.28458571]
 [224.28568804 228.21027007 228.21027007 228.29377182]
 [225.02590483 228.96343928 228.96343928 229.04721661]
 [225.02590483 228.96343928 228.96343928 229.04721661]
 [225.02590483 228.96343928 228.96343928 229.04721661]
 [225.02590483 228.96343928 228.96343928 229.04721661]
 [224.28568804 228.21027007 228.21027007 228.29377182]
 [224.7791659  228.71238288 228.71238288 228.79606834]
 [224.7791659  228.71238288 228.71238288 228.79606834]
 [225.02590483 228.96343928 228.96343928 229.04721661]]
Reject H0
datetime & Seasons are not Independent
```

- with >95%, we can conclude that Datetime & Season are very much dependent on each other,So further we will continue our analysis only on Seasons

Impact of Season on User Count

Season:

1: spring, 2: summer, 3: fall, 4: winter

Univariate Analysis

```
[ ] data['season'].value_counts(normalize = True)
```

```
4    0.251148
2    0.251056
3    0.251056
1    0.246739
Name: season, dtype: float64
```

Season & total_count:

Bivariate Analysis

```
[ ] bivariateplot(info = data,cat ="season",num="total_count")
```

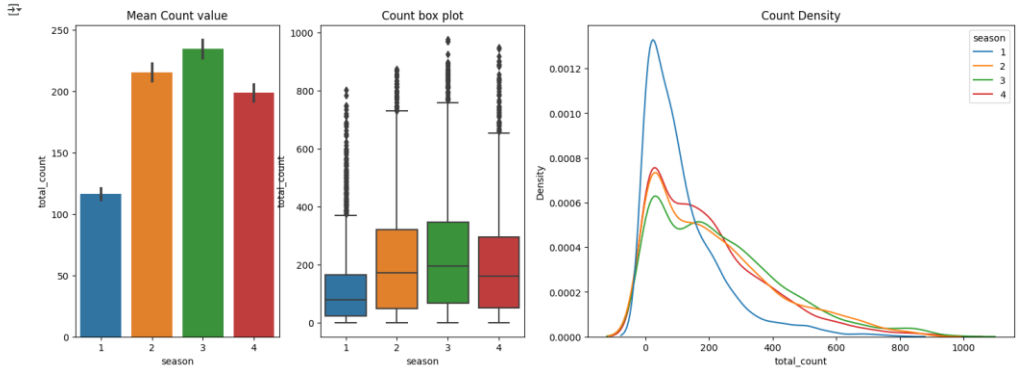


Figure 4 : Combined Visualization for Relation between Season & Total users

```
[ ] bivariateboxplotparameter(cat ="season",num="total_count")
```

season	count	mean	std	min	25%	50%	75%	max	upper_Limit
1	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0	374.0
2	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0	729.0
3	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0	765.5
4	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0	658.5

- No of user total_count in "1:Spring" season are very low with upper limit of 374

- whereas user total_count is highest in "3:fall" are high with upper limit count as 765

Let us do Hypothesis testing to check if Season has effect on Total_Count

Hypothesis testing:

As we have 4 groups in seasons Category --> we will use **ANNOVA** & check how does seasons effect on user count

- Ho --> All seasons have same user count mean
- Ha --> Some seasons user count Mean are different

Before using ANNOVA, we will check if all 4 season category data is following Normal Distribution & does they have equal variance

For Normal Distribution, we will use **Shapiro test** with below Hypothesis Condition

- H0 : Data is Gaussian
- Ha : Data is not Gaussian

For Equal Variances Check, we will use **Levene Test** with below Hypothesis condition

- H0: Variances are equal
- Ha: Variances are not equal

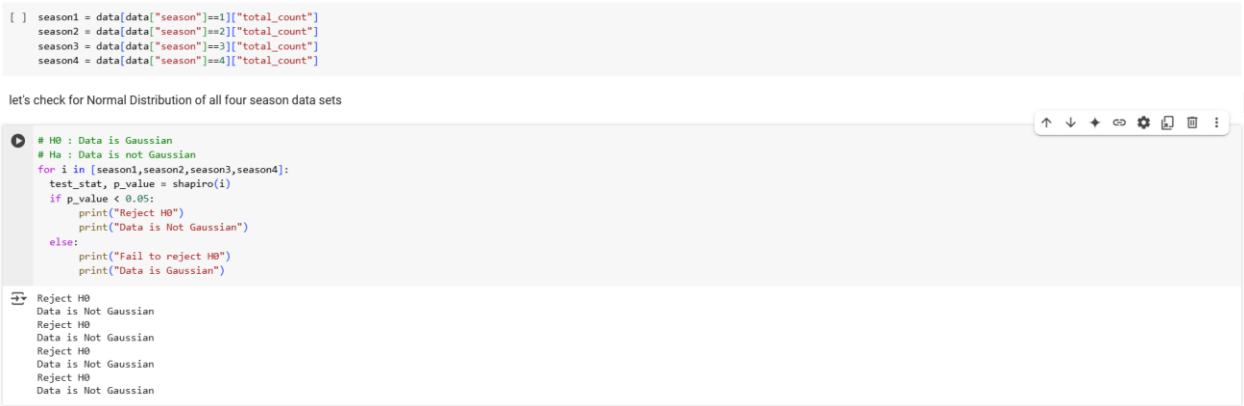
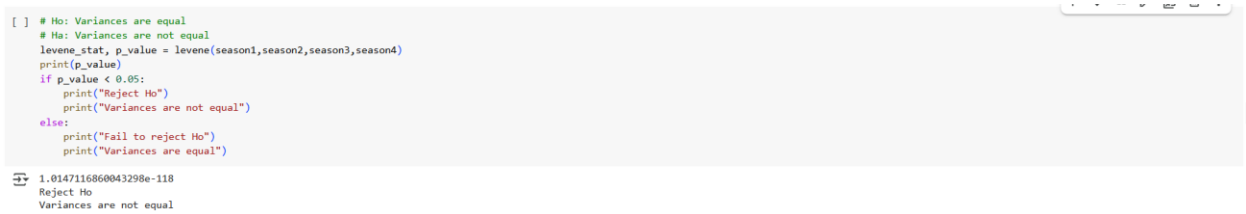


Figure 5 : Hypothesis Testing Set up

- All 4 Season Data sets are not following Normal Distribution
- Once will check Histogram plot.
- Now we will do Levene test to check for equal Variances.



- All four Seaons Datasets do not have equal Variance
- So 2 of ANNOVA assumption fails, we will do **kruskal Wallis** test

```
#H0 : All groups have same mean
#Ha : One or more groups have different mean

kruskal_stat, p_value= kruskal(season1,season2,season3,season4)
print(p_value)
if p_value < 0.05:
    print("Reject H0")
    print("One or more groups have different mean")
else:
    print("Fail to reject H0")
    print("All groups have same mean")
```

```
2.479008372608633e-151
Reject H0
One or more groups have different mean
```

From Kruskal Walli Test, With >95% confidence we can say that **Season has impact on total_user Count**

Impact of Weather on User Count

Weather

- 1: Clear, Few clouds, partly cloudy, partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

Univariate Analysis

```
data['weather'].value_counts(normalize = True)
```

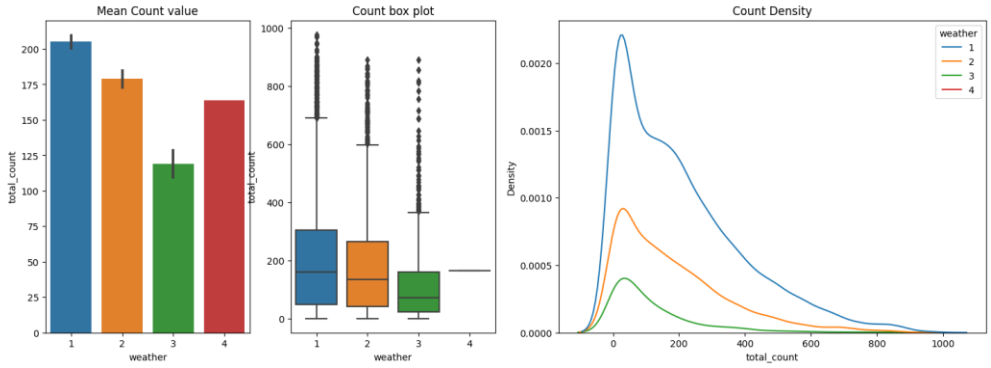
```
1    0.660665
2    0.260334
3    0.078909
4    0.000092
Name: weather, dtype: float64
```

Weather & total_count:

Bivariate Analysis

```
bivariateplot(info = data,cat ="weather",num="total_count")
```

```
<ipython-input-29-a4ea6cc46d6b>:8: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(data=info,x=num,hue = cat)
```




```
bivariateboxplotparameter(cat ="weather",num="total_count")
```

weather	count	mean	std	min	25%	50%	75%	max	upper_Limit
1	7192.0	205.236791	187.959566	1.0	48.0	161.0	305.0	977.0	690.5
2	2834.0	178.955540	168.366413	1.0	41.0	134.0	264.0	890.0	598.5
3	859.0	118.846333	138.581297	1.0	23.0	71.0	161.0	891.0	368.0
4	1.0	164.000000	NaN	164.0	164.0	164.0	164.0	164.0	164.0

- No of user total_count reduced as the weather became severe
- Weather:1 has total_count upper limit as 690
- weather:3 has total_count upper limit as 368

- There was only one booking for weather:4 with 164 users[registered : 158 & casual:6

Now let us do Hypothesis Testing to check if Weather has impact on total_Count

Hypothesis Testing:

As we have 4 groups in weather Category --> we will use **ANNOVA** & check how does weather effect on user count

- Ho --> All weather's have same user count mean
- Ha --> Some weather's user count Mean are different
-

Before using ANNOVA, we will check if all 4 weather category data is following Normal Distribution & does they have equal variance

For Normal Distribution, we will use **Shapiro test** with below Hypothesis Condition

- H0 : Data is Gaussian
- Ha : Data is not Gaussian

For Equal Variances Check, we will use **Levene Test** with below Hypothesis condition

- H0: Variances are equal
- Ha: Variances are not equal

```
[ ] weather1 = data[data["weather"]==1]["total_count"]
    weather2 = data[data["weather"]==2]["total_count"]
    weather3 = data[data["weather"]==3]["total_count"]
    weather4 = data[data["weather"]==4]["total_count"]
```

let's check for Normal Distribution of all four weather data sets, But weather4 has only 1 reading, so we cannot do Shapiro test

```
[ ] # H0 : Data is Gaussian
    # Ha : Data is not Gaussian
    for i in [weather1,weather2,weather3]:
      test_stat, p_value = shapiro(i)
      if p_value < 0.05:
          print("Reject H0")
          print("Data is Not Gaussian")
      else:
          print("Fail to reject H0")
          print("Data is Gaussian")
```

```
Reject H0
Data is Not Gaussian
Reject H0
Data is Not Gaussian
Reject H0
Data is Not Gaussian
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1816: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

- From Shapiro --> we can conclude that 4 weather data sets are not following Normal Distribution
- Now we will do Levene test to check for equal Variances.

```
[ ] # Ho: Variances are equal
    # Ha: Variances are not equal
    levene_stat, p_value = levene(weather1,weather2,weather3,weather4)
    print(p_value)
    if p_value < 0.05:
        print("Reject Ho")
        print("Variances are not equal")
    else:
        print("Fail to reject Ho")
        print("Variances are equal")
```

```
3.5049379468332380e-35
Reject Ho
Variances are not equal
```

- All four weather Datasets do not have equal Variance
- So 2 of ANNOVA assumption fails, we will do **kruskal Wallis** test

```
#H0 : All groups have same mean
#Ha : One or more groups have different mean

kruskal_stat, p_value= kruskal(weather1,weather2,weather3,weather4)
print(p_value)
if p_value < 0.05:
    print("Reject H0")
    print("One or more groups have different mean")
else:
    print("Fail to reject H0")
    print("All groups have same mean")
```

```
3.501611300708679e-44
Reject H0
One or more groups have different mean
```

From Kruskal Walli Test, With >95% confidence we can say that **weather has impact on total_user Count**

Dependency of Weather and Season

As Weather & Season are both Categorical Data, we will do **Chi- Squared Test** with below Hypothesis Condition

- Ho: Weather & Season are independent
- Ha: Weather & Seasons are not Independent

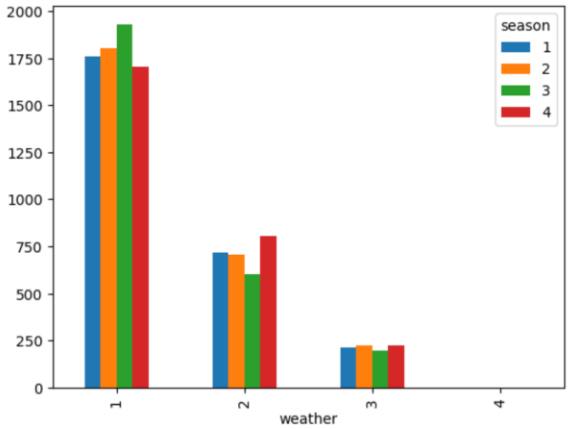
Bivariate Analysis:

```
A = pd.crosstab(data["weather"],data["season"])
print(A)
```

```
season      1     2     3     4
weather
1        1759  1801  1930  1702
2         715   708   604   807
3         211   224   199   225
4           1     0     0     0
```

```
A.plot(kind="bar")
```

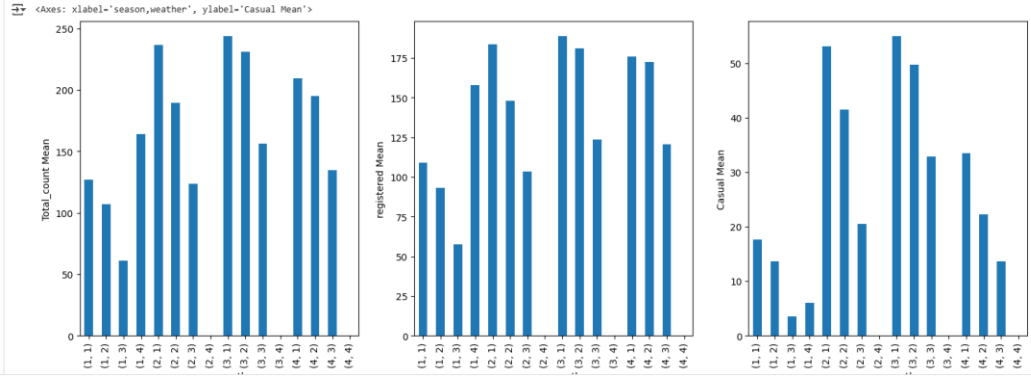
```
<Axes: xlabel='weather'>
```



Now let us check , how Mean of total_count varies in combination of Season & weather

```
plt.figure(figsize=(18,6))
plt.subplot(1,3,1)
plt.ylabel("Total_count Mean")
data.groupby(["season","weather"])["total_count"].mean().plot(kind = "bar")
plt.subplot(1,3,2)
plt.ylabel("registered Mean")
data.groupby(["season","weather"])["registered"].mean().plot(kind = "bar")
plt.subplot(1,3,3)
plt.ylabel("Casual Mean")
data.groupby(["season","weather"])["casual"].mean().plot(kind = "bar")
```

```
<Axes: xlabel='season,weather', ylabel='Casual Mean'>
```



From Bar plot we can see User Count Highly active in Weather1[Clear, Few clouds] & Season3[Fall]

Now Let us Check Dependency of Weather and Season with Hypothesis Testing

As Weather & Season are both Categorical Data, we will do Chi2_Contigency test to test their Dependence.

Null Hypothesis can be set as below.

- Ho: Weather & Season are independent
- Ha: Weather & Seasons are not Independent

But for Chi Sqaured test, Expected values in each cell has to be > 5,So we remove Weather4 data and do Chi2_Contingency test again
A[:3] will remove 4rth row from table

```
# Ho: Weather & Season are independent
# Ha: Weather & Seasons are not Independent

chi_stat, p_value, df, exp_freq = chi2_contingency(A[:3])

print(p_value)
print(exp_freq)
if p_value < 0.05:
  print("Reject H0")
  print("Weather & Seasons are not Independent")
else:
  print("Fail to Reject H0")
  print("Weather & Seasons are Independent")
```

```
2.8260014509929403e-08
[[1774.04869086 1805.76352779 1805.76352779 1806.42425356]
 [ 699.06201194  711.55920992  711.55920992  711.81956821]
 [ 211.8892972   215.67726229  215.67726229  215.75617823]]
Reject H0
Weather & Seasons are not Independent
```

However, result is same with >95% Confidence we can conclude that **Weather & Seasons are Dependent**

<u>Impact of Holiday on User Count:</u>

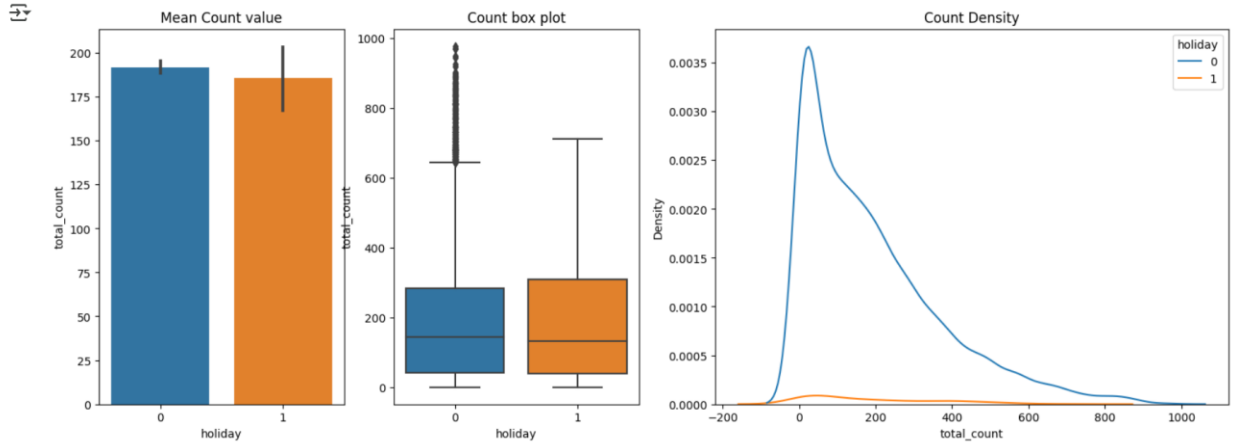
Univariate Analysis:

```
data['holiday'].value_counts(normalize = True)
```

```
0    0.971431
1    0.028569
Name: holiday, dtype: float64
```

Bivariate Analysis:

```
bivariateplot(info = data,cat ="holiday",num="total_count")
```



- No of user total_count reduced on a holiday compared to Non-Holiday
- Now lets do Hypothesis Testing to check if Holiday has impact on User Count

Hypothesis Testing:

As we have only 2 groups in Holiday day Category --> we will use **2 Sample/Independent T-test** & check how does Holiday day effect on user count, **First We will do two tailed test** & **later on if we reject Ho in two tailed test then we will do single tailed test**

- Ho --> User Count Mean is same on Holiday & Non Holiday
- Ha --> User Count Mean is not same on Holiday & Non Holiday

```
[ ] holiday_day = data[data["holiday"]==1]["total_count"]
    nonholiday_day = data[data["holiday"]==0]["total_count"]
```

Before using 2 Sample/Independent T test, we will check if 2 Samples have equal variance

For Equal Variances Check, we will use Levene Test with below Hypothesis condition

- H0: Variances are equal
- Ha: Variances are not equal

```
[ ] # Ho: Variances are equal
    # Ha: Variances are not equal
    levene_stat, p_value = levene(holiday_day,nonholiday_day)
    print(p_value)
    if p_value < 0.05:
        print("Reject Ho")
        print("Variances are not equal")
    else:
        print("Fail to reject Ho")
        print("Variances are equal")
```

```
0.9991178954732041
Fail to reject Ho
Variances are equal
```

- Levene hypothesis test proved that both samples have same Variance.
- So we can proceed with 2 Sample/Independent T-Test

```
[ ]  # Ho: mu1 = mu2
     # Ha: mu1 != mu2
     holiday_day = data[data["holiday"]==1]["total_count"]
     nonholiday_day = data[data["holiday"]==0]["total_count"]

     t_stat, p_value = ttest_ind(holiday_day,nonholiday_day)
     print(p_value)
     if p_value < 0.05:
         print("Reject H0")
     else:
         print("Fail to reject H0")
```

```
0.5736923883271103
Fail to reject H0
```

with 95% confidence we can that **Holiday day/Non Holiday day has no impact on Total_user count mean**

<u>Impact of Workingday on User Count:</u>

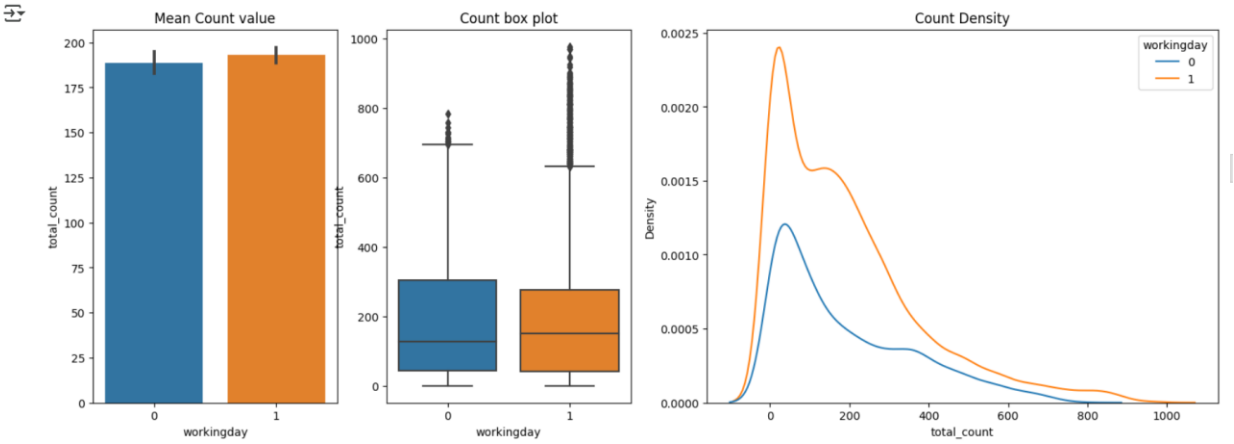
Univariate Analysis:

```
[ ]  data['workingday'].value_counts(normalize = True)
```

```
1    0.680875
0    0.319125
Name: workingday, dtype: float64
```

Bivariate Analysis:

```
[ ]  bivariateplot(info = data,cat ="workingday",num="total_count")
```



- Working day/Non working day Total_count Mean has very less difference
- Now lets do Hypothesis Testing to check if Workingday has impact on User Count

Hypothesis Testing:

As we have only 2 groups in Working day Category --> we will use 2 Sample/Independent T-test & check how does working day effect on user count, First We will do two tailed test & later on if we reject Ho in two tailed test then we will do single tailed test

- Ho --> User Count Mean is same on workinday & Non workinday
- Ha --> User Count Mean is not same on workinday & Non workinday

```
[ ]  working_day = data[data["workingday"]==1]["total_count"]
     nonworking_day = data[data["workingday"]==0]["total_count"]
```

Before using 2 Sample/Independent T test, we will check if 2 Samples have equal variance

For Equal Variances Check, we will use Levene Test with below Hypothesis condition

- H0: Variances are equal
- Ha: Variances are not equal

```
[ ] # Ho: Variances are equal
    # Ha: Variances are not equal
    levene_stat, p_value = levene(working_day,nonworking_day)
    print(p_value)
    if p_value < 0.05:
        print("Reject Ho")
        print("Variances are not equal")
    else:
        print("Fail to reject Ho")
        print("Variances are equal")
```

```
0.9437823280916695
Fail to reject Ho
Variances are equal
```

- Levene hypothesis test proved that both samples have same Variance
- So we can proceed with 2 Sample/Independent T-Test

```
[ ] # Ho: mu1 = mu2
    # Ha: mu1 != mu2

    t_stat, p_value = ttest_ind(working_day,nonworking_day)
    print(p_value)
    if p_value < 0.05:
        print("Reject H0")
    else:
        print("Fail to reject H0")
```

```
0.22644804226361348
Fail to reject H0
```

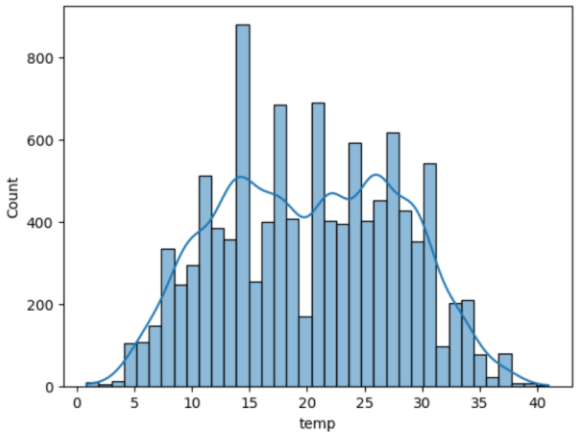
- with 95% confidence we can that **working day/Non Working day has no impact on Total_user count mean**

<u>Impact of Temp on User Count:</u>

Univariate Analysis:

```
[ ] sns.histplot(data["temp"],kde=True)
```

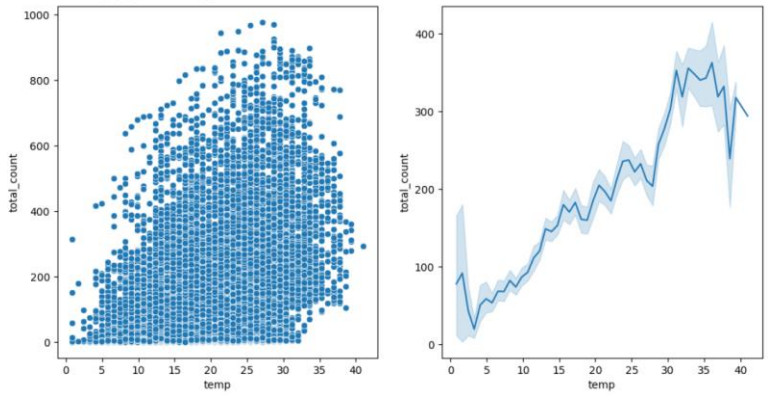
```
<Axes: xlabel='temp', ylabel='Count'>
```



Bivariate Analysis:

```
[ ] plt.figure(figsize=(12,6))
    plt.subplot(1,2,1)
    sns.scatterplot(x=data["temp"], y=data["total_count"])
    plt.subplot(1,2,2)
    sns.lineplot(x=data["temp"],y=data["total_count"])
```

```
<Axes: xlabel='temp', ylabel='total_count'>
```



- from lineplot, users Avg count increased as the temperature increased till 36 deg Celsius & then count dropped
- from scatter plot We can see there is no linear relationship between temperature and Total_count, so we will do spearman Correlation test

Hypothesis Testing:

As Temp and Users Count are both numerical, we will do Spearman Test as relation is somewhat nonlinear as seen in scatterplot

- Ho: No correlation between Temp & Total_count
- Ha: There is correlation between Temp & Total_count

```
[ ] # H0: No correlation
    # Ha: There is correlation

    spearman_coeff, p_value = spearmanr(data["temp"], data["total_count"])
    print(spearman_coeff)
    print(p_value)
    if p_value < 0.05:
        print("Reject H0")
        print("There is correlation")
    else:
        print("Fail to reject H0")
        print("There is no correlation")
```

```
0.40798939475098117
0.0
Reject H0
There is correlation
```

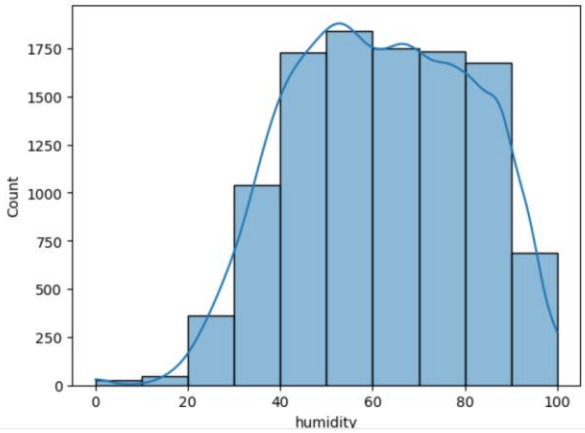
with >95% Confidence we can that **Temp has good positve correlation with Total_users count**

Impact of Humidity on User Count:

Univariate Analysis:

```
[ ] sns.histplot(data["humidity"],kde=True,bins=10)
```

```
<Axes: xlabel='humidity', ylabel='Count'>
```

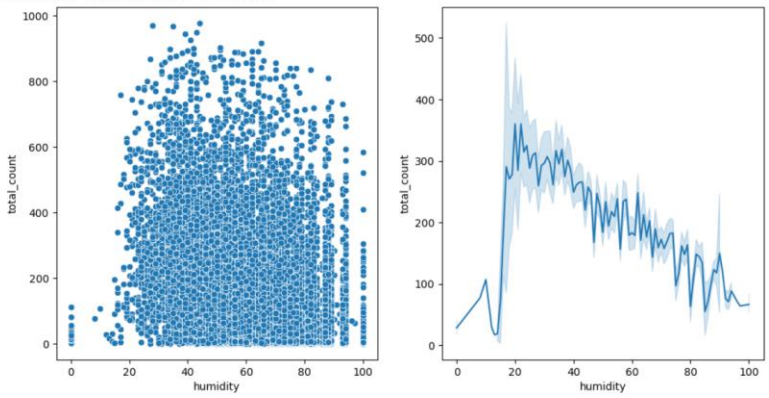


- humidity covers vast range from 0 to 100
- we can see distribution is left skewed, Humidity was mostly found in the range 40 ~ 100

Bivariate Analysis:

```
[ ] plt.figure(figsize=(12,6))
    plt.subplot(1,2,1)
    sns.scatterplot(x=data["humidity"], y=data["total_count"])
    plt.subplot(1,2,2)
    sns.lineplot(x=data["humidity"],y=data["total_count"])
```

```
<Axes: xlabel='humidity', ylabel='total_count'>
```



- from lineplot,Avg Users count reduced with increase in humididty
- Avg Users count peaked in the humidity range of 20-22 & then grdually dropped
- from scatter plot We can see there is no linear relationship between humidity and Total_count, but somewhat nonlinear relationship exists from humidity 20 to 100 -->Humidity increase, avg count decreases, so we will do spearman Correlation test

Hypothesis Testing:

As humidity and Users Count are both numerical, we will do Spearman Test as relation is somewhat nonlinear as seen in scatterplot

- Ho: No correlation between humidity & Total_count

- Ha: There is correlation between humidity & Total_count

```
[ ] # H0: No correlation
    # Ha: There is correlation

    spearman_coeff, p_value = spearmanr(data["humidity"], data["total_count"])
    print(spearman_coeff)
    print(p_value)
    if p_value < 0.05:
        print("Reject H0")
        print("There is correlation")
    else:
        print("Fail to reject H0")
        print("There is no correlation")
```

```
-0.35404912201756106
0.0
Reject H0
There is correlation
```

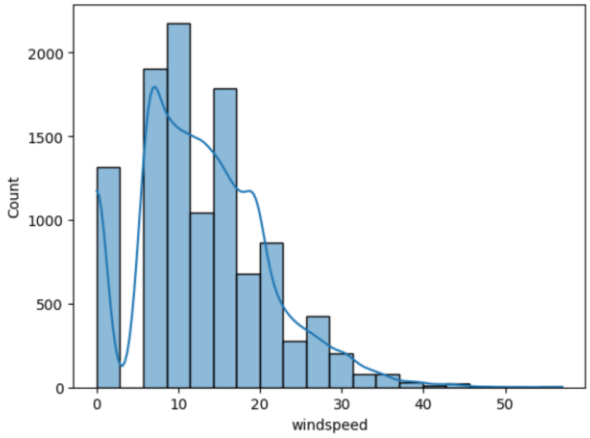
with >95% Confidence we can that **Humidity has good negative correlation with Total_users count**

<u>Impact of Windspeed on User Count</u>

Univariate Analysis:

```
[ ] sns.histplot(data["windspeed"],kde=True,bins=20)
```

```
<Axes: xlabel='windspeed', ylabel='Count'>
```

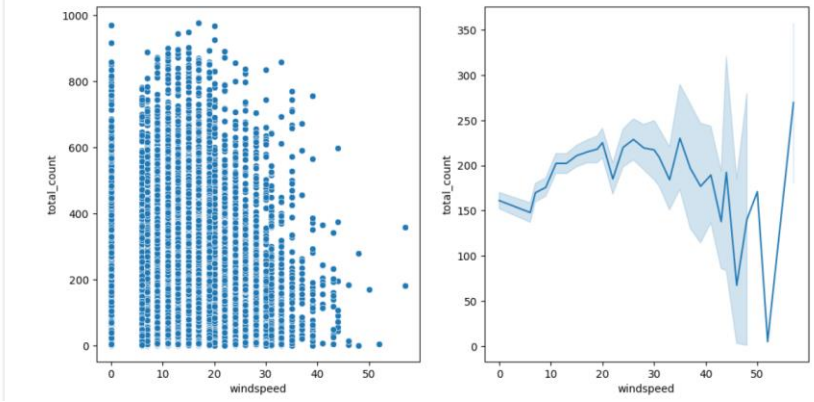


- windspeed covers vast range from 0 to 56
- Mostly wind are in the range 8 ~ 24

Bivariate Analysis:

```
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
sns.scatterplot(x=data["windspeed"], y=data["total_count"])
plt.subplot(1,2,2)
sns.lineplot(x=data["windspeed"],y=data["total_count"])
```

```
<Axes: xlabel='windspeed', ylabel='total_count'>
```



- from lineplot,Avg Users count reduced with increase in humididty
- Avg Users count peaked in the Windspeed range of 20-35
- from scatter plot We can see there is no linear relationship between windspeed and Total_count, but somewhat nonlinear relationship exists --> windspeed increase, avg count increases &decreases, so we will do spearman Correlation test

Hypothesis Testing:

As windspeed and Users Count are both numerical, we will do Spearman Test as relation is somewhat nonlinear as seen in scatterplot

- Ho: No correlation between windspeed & Total_count
- Ha: There is correlation between windspeed & Total_count

```
[ ] # H0: No correlation
    # Ha: There is correlation

    spearman_coeff, p_value = spearmanr(data["windspeed"], data["total_count"])
    print(spearman_coeff)
    print(p_value)
    if p_value < 0.05:
        print("Reject H0")
        print("There is correlation")
    else:
        print("Fail to reject H0")
        print("There is no correlation")
```

```
0.1357773747113304
5.9015220272171205e-46
Reject H0
There is correlation
```

with >95% Confidence we can that **Windspeed has very less positive correlation with Total_users count**

<u>Correlation matrix:</u>

- In this Section we will try solution through different approach i.e., feature Engineering.
- I will use "Label Encoder" & "Target encoder with Total_count as Target" to convert all the Categorical data to Numerical.

- Then i will use Correlation values to find the most Signification Feature to predict User_count on a given day.

```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   datetime      10886 non-null  datetime64[ns]
 1   season        10886 non-null  category
 2   holiday       10886 non-null  category
 3   workingday    10886 non-null  category
 4   weather       10886 non-null  category
 5   temp          10886 non-null  float64
 6   feeling_temp  10886 non-null  float64
 7   humidity      10886 non-null  int64
 8   windspeed     10886 non-null  float64
 9   casual        10886 non-null  int64
 10  registered    10886 non-null  int64
 11  total_count   10886 non-null  int64
dtypes: category(4), datetime64[ns](1), float64(3), int64(4)
memory usage: 808.6 KB
```

Lets Create Target Encoder and Label Encoder to Convert Categorical to Numerical

```
[ ] te=TargetEncoder()
```

```
[ ] data["season"]=te.fit_transform(data["season"],data["total_count"])
    data["season"].value_counts()
```

```
198.988296    2734
215.251372    2733
234.417124    2733
116.343261    2686
Name: season, dtype: int64
```

```
[ ] data["weather"]=te.fit_transform(data["weather"],data["total_count"])
    data["weather"].value_counts()
```

```
205.236791    7192
178.955540    2834
118.846333     859
187.986504       1
Name: weather, dtype: int64
```

```
[ ] le=LabelEncoder()
```

```
[ ] data["holiday"]=le.fit_transform(data["holiday"])
    data["holiday"].value_counts()
```

```
0    10575
1      311
Name: holiday, dtype: int64
```

```
[ ] data["workingday"]=le.fit_transform(data["workingday"])
    data["workingday"].value_counts()
```

```
1    7412
0    3474
Name: workingday, dtype: int64
```

```
[ ] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   datetime      10886 non-null  datetime64[ns]
 1   season        10886 non-null  float64
 2   holiday       10886 non-null  int64
 3   workingday    10886 non-null  int64
 4   weather       10886 non-null  float64
 5   temp          10886 non-null  float64
 6   feeling_temp  10886 non-null  float64
 7   humidity      10886 non-null  int64
 8   windspeed     10886 non-null  float64
 9   casual        10886 non-null  int64
 10  registered    10886 non-null  int64
 11  total_count   10886 non-null  int64
dtypes: datetime64[ns](1), float64(5), int64(6)
memory usage: 1020.7 KB
```

- All features have been converted to Numerical except for datetime.
- let's drop that datetime feature.

```
data.drop("datetime",axis=1,inplace=True)
```

```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,cmap = "Greens")
```

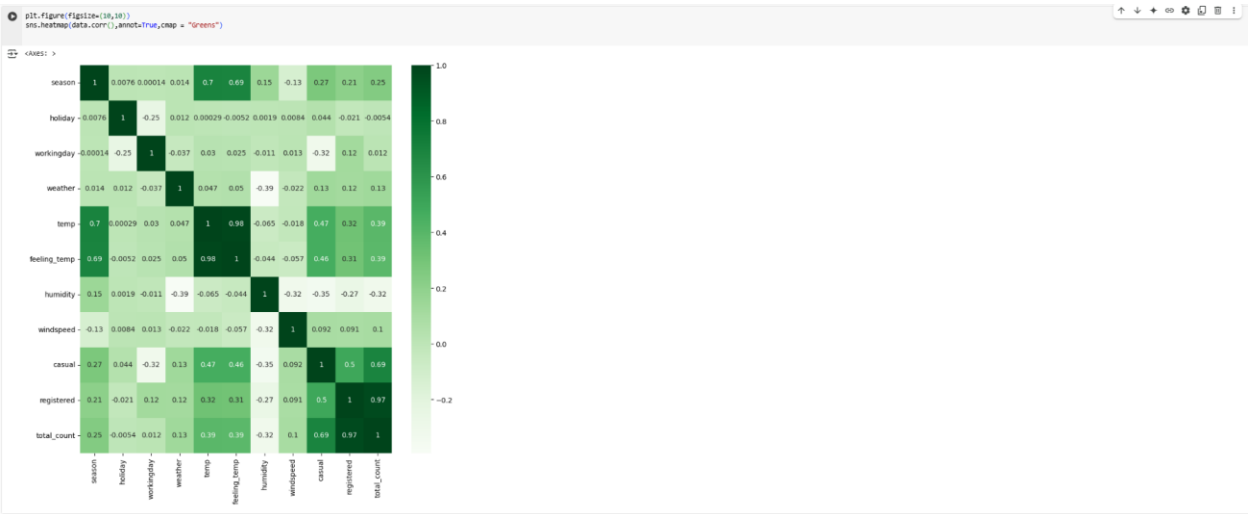


Figure 6 : Correlation Matrix to infer hypothesis Testing Result

Similar to what we have conclude in Hypothesis Testing for Total_count, Correlation matrix values are also Giving same output

- Holiday & working days has no impact on Total_count
- Season & Weather have impact on Total_count
- temp & feeling_temp are positively correlated on Total_count
- Humidity is negatively correlated on Total_count
- Windspeed has very less Positive Correlation on Total_count

As we Target encoded feature w.r.t Total_count, the above correlation matrix is giving correct correlation values for total_count only.

Business Insights

Overall Summary:

- total_count of user per day range between 0 to 647, however there are some unsual booking some days ranging from 648 to 977 per day
- registered user per day range between 0 to 501, however there are some unsual booking some days ranging from 502 to 886 per day
- casual user per day range between 0 to 116, however there are some unsual booking some days ranging from 117 to 367 per day

Seasons:

- No of user total_count in "1:Spring" season are very low with upper limit of 374 & Average Total_count of 116
- whereas user total_count is highest in "3:fall" are high with upper limit count as 765 & Average Total_count of 234
- With >95% Confidence we can say that Total_count, Registered & Casual User count are impacted by Seasons

Weather:

- No of user total_count reduced as the weather became severe
- No of user total_count in Weather:1 is high with upper limit of 690 & Average of 205
- No of user total_count in Weather:3 is low with upper limit of 368 & Average of 118
- There was only one booking for weather:4 with 164 users[registered : 158 & casual:6]
- With >95% Confidence we can say that Total_count, Registered & Casual User count are impacted by Weather

Holiday:

- Mean of Total_Count reduced on a holiday compared to Non-Holiday, However in Contradiction Hypothesis testing proved with 95% confidence that Holiday day/Non Holiday day has no impact on Total_user count mean
- Mean of Registered user reduced on a holiday compared to Non-Holiday & to Affirm Hypothesis testing proved with 95% confidence that Holiday day/Non Holiday day has impact on Registered count
- No of user Casual increase on a holiday compared to Non-Holiday & to affirm Hypothesis testing proved with 95% confidence that Casual user count is higher on Holiday compared to Non Holiday

Workingday:

- Total_count Mean of Working day & Non Workingday has not much diffeence and to affirm that Hypothesis testing proved with 95% confidence that Workingday/Non Workingday has no impact on Total_user count mean
- Registered user counts is higher on a working day compared to Non working day and to affirm Hypothesis testing proved with >95% confidence that registered user count is higher on Working day compared to Nonworking day

- Casual user count is higher on a non working day compared to working day and to affirm Hypothesis testing proved with >95% confidence that Casual user count is higher on Nonworking day compared to Working day

Humidity:

- Avg Users count peaked in the humidity range of 20-22 & then grdually dropped
- To affirm Hypothesis testing proved with >95% Confidence that Humidity has good negative correlation with Total_users, Registered & Casual User Count

Windspeed:

- Mostly wind are in the range 8 ~ 24 & Avg Users count peaked in the Windspeed range of 20-35
- To affirm Hypothesis testing proved with >95% Confidence that Windspeed has very less positive correlation with Total_users, Registered & Casual User Count

Recommendations

- Maximum usage is found in the Fall Season, its Better to avoid Possible Maintenance activity during that season & Keep More Electric bikes ready to use

- Similarly as Usage is less in Spring season, All Possible Maintenance to be done in fall season

- In weather3,4, user count is very less due to harsh weather conditions, Company can modify the design to instantly install sepal shield(motorcycle shield that protects riders from harsh weather conditions) to make it viable to use in harsh weather conditions also, so its can retain customer base even in harsh weather conditions

- Casual User increase on Holiday & Non working day, Company can better prepare its logistics to cater those unplanned Casual users & convert them to regular customers with some additional feel good service

- Temperature play quite some role for user to use, when temperature is in the range 0 ~ 20, Company can provide additional comfort mechanism[low cost handle heaters] to increase user count

Chapter 3 : Business Case Study 3

Problem Description:

About Company

A leading Indian education institute specializing in test preparation for study abroad exams like GMAT, GRE, SAT, TOEFL, and IELTS, providing comprehensive classroom training and admission counseling to help students enter top universities worldwide, it is known for its thorough research and focus on filling the information gap regarding education abroad, with multiple centers across India and Nepal.

Problem:

Company recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

- They want to understand what factors are important in graduate admissions and how these factors are interrelated among themselves.
- It will also help predict one's chances of admission given the rest of the variables.

Dataset:

- Serial No. (Unique row ID)
- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1) → **Output of Regression**

Libraries Used:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import mannwhitneyu
from scipy.stats import kruskal
from scipy.stats import spearmanr

import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.preprocessing import PolynomialFeatures
```

```
from scipy import stats
from scipy.stats import  kstest


from scipy.stats import shapiro,levene
from statsmodels.graphics.gofplots import qqplot
```

Business Questions to be answered from Analysis

- Use Non-graphical and graphical analysis for getting inferences about variables.
- Check correlation among independent variables and how they interact with each other.
- Use Linear Regression from (Statsmodel library) and explain the results.
- Test the assumptions of linear regression:
  - Multicollinearity check by VIF score
  - Mean of residuals
  - Linearity of variables (no pattern in residual plot)
  - Test for Homoscedasticity
  - Normality of residuals
- Do model evaluation- MAE, RMSE, R2 score, Adjusted R2.
- Try out different Linear Regression like Lasso, Ridge & Polynomial Features
- Provide actionable Insights & Recommendations

Analysis:

As we have seen Data Cleaning & Hypothesis Testing in Depth in previous Business Cases, we will Do Univariate & Bivariate Analysis on Cleaned Data and Then Do Linear Regression Modelling

```
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv")
```

+ Code + Text

```
[ ] data.shape
```

```
(500, 9)
```

```
[ ] data.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Table 4 : Dataset for Linear Regression

Statistical Analysis, Graphical Visualization:

Checking the Unique Values for each Variable

```
[ ] data.nunique()
```

```
GRE Score            49
TOEFL Score          29
University Rating     5
SOP                   9
LOR                   9
CGPA                184
Research              2
Chance of Admit      61
dtype: int64
```

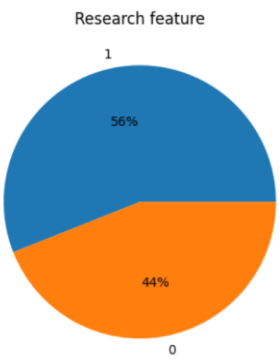
Statistical Analysis:

```
[ ] data.describe()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

Research:

```
[ ] plt.pie(x = data["Research"].value_counts().reset_index()["Research"],
            labels = data["Research"].value_counts().reset_index()["index"],
            autopct='%.0f%%')
    plt.title("Research feature")
    plt.show()
```



- 56% Students have Research Experience
- 44% Students do have Research Experience
- Let us Check what effect does Research have on “Chance of Admit”

```
[ ] sns.violinplot(data = data, y = "Chance of Admit", x = "Research")
```

```
<Axes: xlabel='Research', ylabel='Chance of Admit'>
```

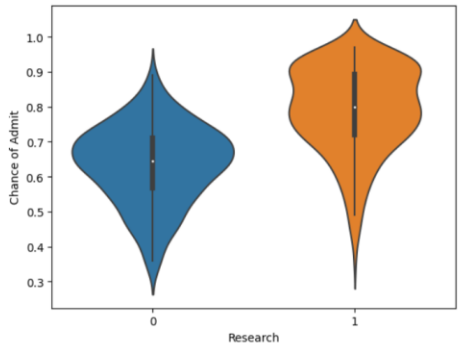


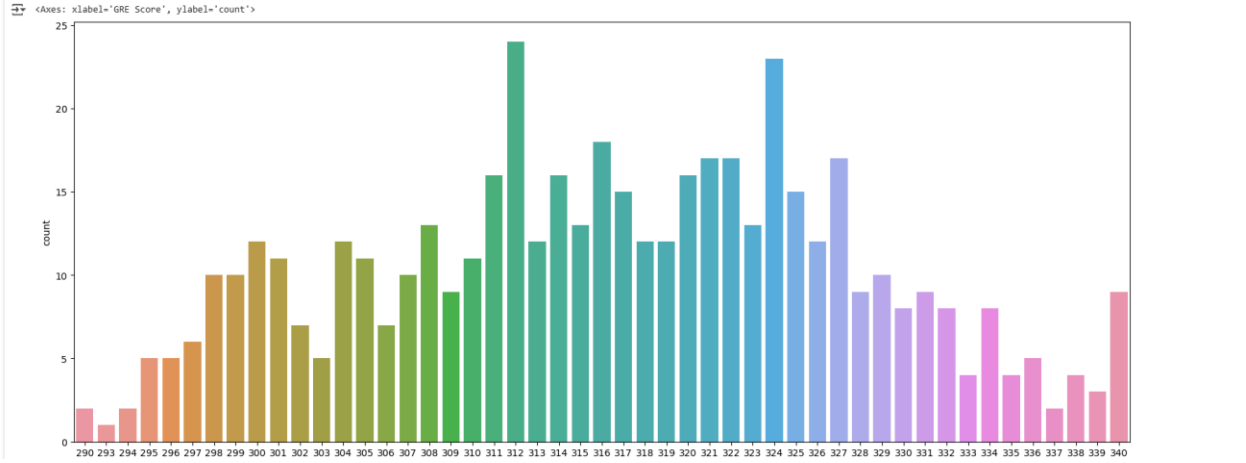
Figure 7 : Bivariate Analysis between Independent Variable and Predictor

- Chance of Admit is higher for student who has Research Experience

GRE Score

```
plt.figure(figsize = (20,8))
sns.countplot(data=data, x= "GRE Score")
```

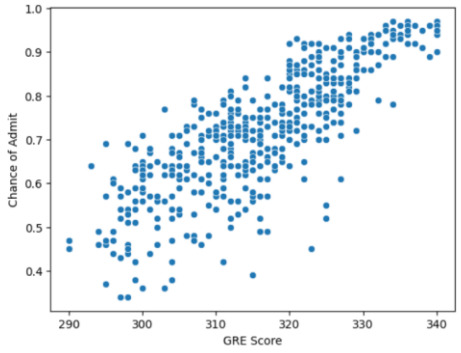
```
<Axes: xlabel='GRE Score', ylabel='count'>
```



- GRE Scores are greater than 290, More No of Data points are in that range 310 ~ 327
- Let us Check what effect does GRE Score have on “Chance of Admit”

```
[ ] sns.scatterplot(data=data, x='GRE Score',y='Chance of Admit')
```

```
<Axes: xlabel='GRE Score', ylabel='Chance of Admit'>
```

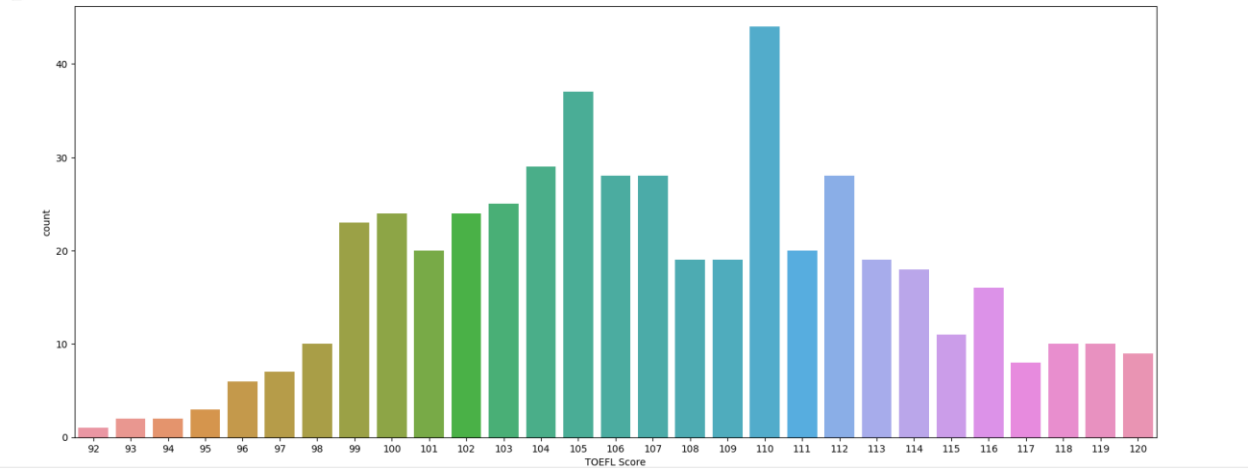


- Chance of Admit increased with increase in GRE Score

TOEFL Score

```
plt.figure(figsize = (20,8))
plt.xlim([0,340])
sns.countplot(data=data, x= "TOEFL Score")
```

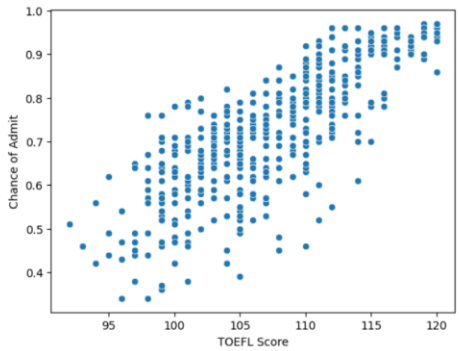
```
<Axes: xlabel='TOEFL Score', ylabel='count'>
```



- TOEFL Scores are greater than 91, More No of Data points are in that range 99~112
- Let us Check what effect does TOEFL have on “Chance of Admit”

```
sns.scatterplot(data=data, x='TOEFL Score',y='Chance of Admit')
```

```
<Axes: xlabel='TOEFL Score', ylabel='Chance of Admit'>
```

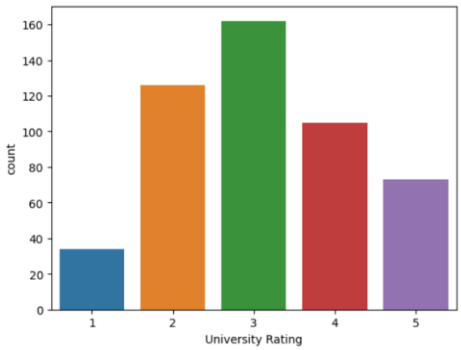


- Chance of Admit increased with increase in TOEFL Score

University Rating:

```
sns.countplot(data=data, x= 'University Rating')
```

```
<Axes: xlabel='University Rating', ylabel='count'>
```

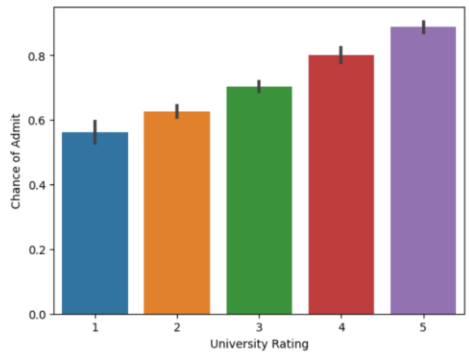


- More no of Student applied for University Ranking = 3

- Let us Check what effect does University ranking have on “Chance of Admit”

```
[ ] sns.barplot(data=data, x='University Rating',y='Chance of Admit',estimator = "mean")
```

```
<Axes: xlabel='University Rating', ylabel='Chance of Admit'>
```

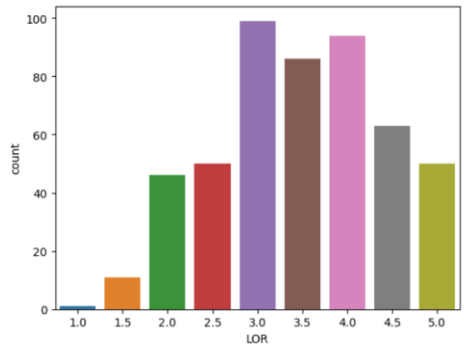


- Chance of Admit on an average increased with increase in University Rating

LOR

```
sns.countplot(data=data, x= 'LOR')
```

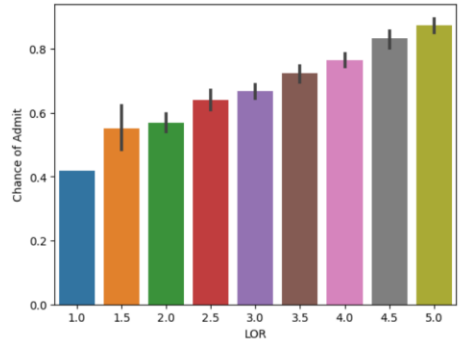
```
<Axes: xlabel='LOR', ylabel='count'>
```



- LOR ranged from 1 ~ 5
- Maximum rating lies between 3 ~ 4.5
- Let us Check what effect does LOR have on “Chance of Admit”

```
sns.barplot(data=data, x='LOR',y='Chance of Admit',estimator = "mean")
```

```
<Axes: xlabel='LOR', ylabel='Chance of Admit'>
```

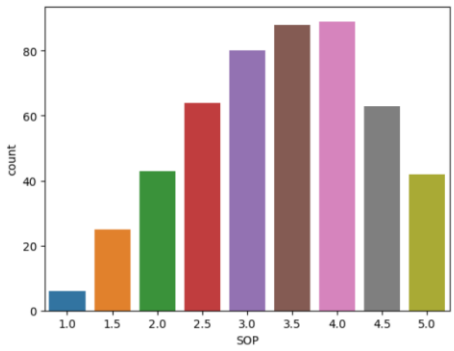


- Chance of Admit on an average increased with increase in LOR Rating

SOP

```
sns.countplot(data=data, x= 'SOP')
```

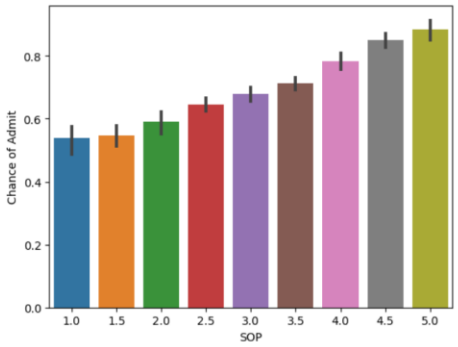
```
<Axes: xlabel='SOP', ylabel='count'>
```



- SOP ranged from 1 ~ 5
- Maximum rating lies between 2.5 ~ 4.5
- Let us Check what effect does SOP have on “Chance of Admit”

```
[ ] sns.barplot(data=data, x='SOP',y='Chance of Admit',estimator = "mean")
```

```
<Axes: xlabel='SOP', ylabel='Chance of Admit'>
```

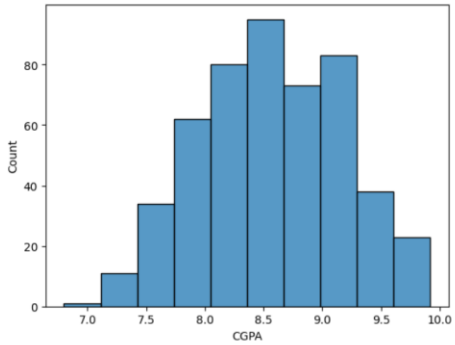


- Chance of Admit on an average increased with increase in SOP Rating

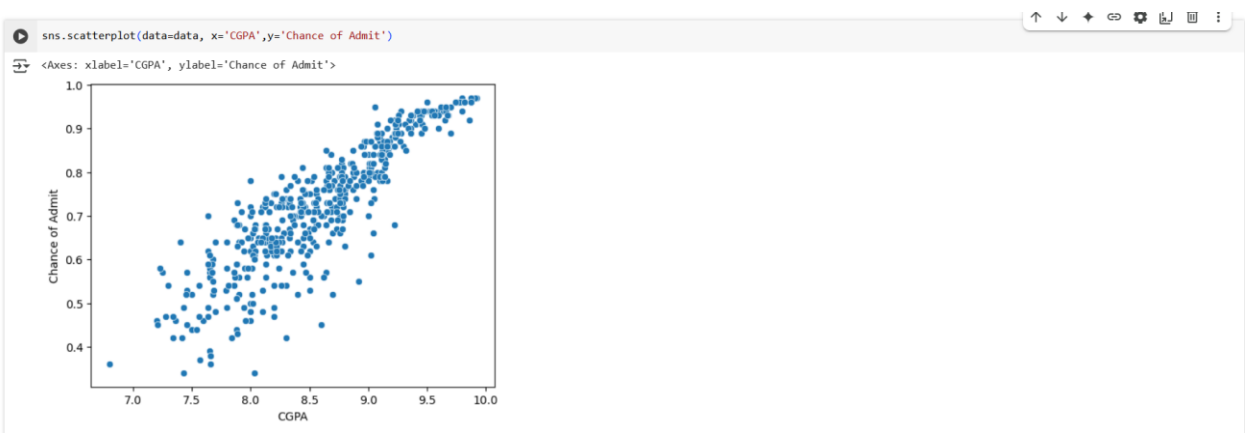
CGPA

```
[ ] sns.histplot(data=data, x= 'CGPA', bins=10)
```

```
<Axes: xlabel='CGPA', ylabel='Count'>
```



- CGPA of the Student are >= 7.0
- Maximum range lies between 7.5 ~ 9.5
- most of the Student applied are meritorious
- Let us Check what effect does CGPA have on “Chance of Admit”



- Chance of Admit increased with increase in CGPA

Overall Correlation:

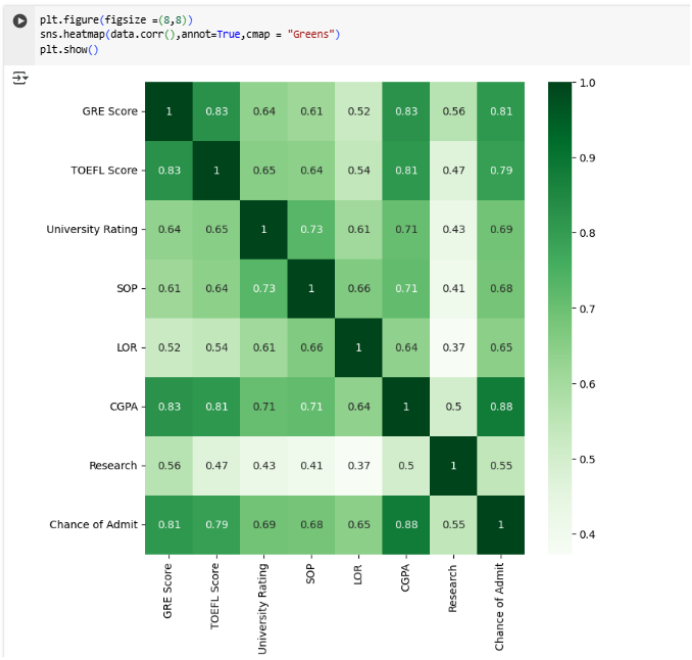


Figure 8 : Correlation Matrix for all independent & predictor variable

- All features have Positive correlation with "Chance of Admit."
- CGPA[0.88],GRE Score[0.81],TOEFL Score[0.79] has very high correlation with "Chance of Admit"
- Further SOP[0.68], LOR[0.65] has almost same Correlation with ""Chance of Admit"
- Comparatively Low Correlation is found w.r.t Research & "Chance of Admit"[0.55], However is positively impactful.

Hypothesis Testing:

In this Data Different test [Num-Num,Cat-Num] can be done
But our objective to correctly predict "Chance of Admit", so we will do Hypothesis testing for below combinations for check if infact feature has effect on "Chance of Admit"

- GRE Score and Chance of Admit
- TOEFL Score and Change of Admit
- CGPA & Chance of Admit
- LOP & Chance of Admit
- SOP & Chance of Admit
- University Rating & Chance of Admit
- Research & Chance of Admit

Test Setup:

In this section let us fix the Confidence level & Level of Signification for our Hypothesis testing.

As this is with respect to predicting Chance_of_Admit

- let us Assume we want 95% Confidence to reject Null Hypothesis, so for every Hypothesis testing, Level of Significance will be "0.05 "
- As our purpose of Hypothesis testing to just verify the relation of Feature not to give final judgement...We will use Non Parametric test to avoid assumptions checking
- However final judgment of impact of each feature will be given after training Linear Regression model
- for Num-Num, we will use "**spearmanr test"**
    - Ha: There is correlation
    - H0: No correlation

```
[ ] for i in data.columns[:-2]:
      spearman_coeff, p_value = spearmanr(data[i], data["Chance of Admit"])
      print("*******************************")
      print(p_value)
      if p_value < 0.05:
        print("Reject H0")
        print(i,"is correlated to Chance_of_admit")
      else:
        print("Fail to reject H0")
        print(i,"is not correlated to Chance_of_admit")
```

```
*******************************
5.734552105475668e-124
Reject H0
GRE Score is correlated to Chance_of_admit
*******************************
1.504956427966445e-109
Reject H0
TOEFL Score is correlated to Chance_of_admit
*******************************
5.88950055529750 6e-76
Reject H0
University Rating is correlated to Chance_of_admit
*******************************
1.1336315351749534e-75
Reject H0
SOP is correlated to Chance_of_admit
*******************************
8.1720720410888 56e-60
Reject H0
LOR is correlated to Chance_of_admit
*******************************
7.372294266325021e-171
Reject H0
CGPA is correlated to Chance_of_admit
```

From Hypothesis testing, we can say all feature are correlated to "Chance of Admit", we will proceed for data preparation for Model Building

<u>Data Preparation for Modelling:</u>

Scaling:

As we have seen Earlier, each of our features are in different scales we will have to **scale them** to

1. Improve Model Performance
2. Faster Convergence
3. Reduced Sensitivity
4. Enhanced Interpretability

5. Consistence Across Model

However Scaling can be done in 2 ways : Standardization and Normalization
But Standardization can be done only if Data is in Normal Distribution

So to check Normal Distribution we will do **KS Test**

```
# H0 : Data is Gaussian
# Ha : Data is not Gaussian
for i in data.columns:
  test_stat, p_value = kstest(data[i],stats.norm.cdf)
  print(p_value)
  if p_value < 0.05:
       print("Reject H0")
       print(i,"Data is Not Gaussian")
  else:
       print("Fail to reject H0")
       print(i,"Data is Gaussian")
  print()
```

```
0.0
Reject H0
GRE Score Data is Not Gaussian

0.0
Reject H0
TOEFL Score Data is Not Gaussian

0.0
Reject H0
University Rating Data is Not Gaussian

0.0
Reject H0
SOP Data is Not Gaussian

0.0
Reject H0
LOR Data is Not Gaussian

0.0
Reject H0
CGPA Data is Not Gaussian

4.1132799581816557e-116
Reject H0
Research Data is Not Gaussian

1.5256082690083004e-204
Reject H0
Chance of Admit Data is Not Gaussian
```

- we have observed that All the Numericals are not following Normal Distribution, so we cannot do Standardization
- Now we will do Normalization using Minmax Scaler

```
Normscaler = MinMaxScaler()
```

```
scaleddata = Normscaler.fit_transform(data)
```

```
scaleddata
```

```
array([[0.94      , 0.92857143, 0.75      , ..., 0.91346154, 1.        ,
        0.92063492],
       [0.68      , 0.53571429, 0.75      , ..., 0.66346154, 1.        ,
        0.66666667],
       [0.52      , 0.42857143, 0.5       , ..., 0.38461538, 1.        ,
        0.6031746 ],
       ...,
       [0.8       , 1.        , 1.        , ..., 0.88461538, 1.        ,
        0.93650794],
       [0.44      , 0.39285714, 0.75      , ..., 0.5224359 , 0.        ,
        0.61904762],
       [0.74      , 0.75      , 0.75      , ..., 0.71794872, 0.        ,
        0.79365079]])
```

```
scaleddata = pd.DataFrame(scaleddata,columns = data.columns)
```

```
scaleddata
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	0.94	0.928571	0.75	0.875	0.857143	0.913462	1.0	0.920635
1	0.68	0.535714	0.75	0.750	0.857143	0.663462	1.0	0.666667
2	0.52	0.428571	0.50	0.500	0.571429	0.384615	1.0	0.603175
3	0.64	0.642857	0.50	0.625	0.285714	0.599359	1.0	0.730159
4	0.48	0.392857	0.25	0.250	0.428571	0.451923	0.0	0.492063
...	...	...	...	...	...	...	...	...
495	0.84	0.571429	1.00	0.875	0.714286	0.711538	1.0	0.841270
496	0.94	0.892857	1.00	1.000	1.000000	0.983974	1.0	0.984127
497	0.80	1.000000	1.00	0.875	1.000000	0.884615	1.0	0.936508
498	0.44	0.392857	0.75	0.750	1.000000	0.522436	0.0	0.619048
499	0.74	0.750000	0.75	0.875	0.857143	0.717949	0.0	0.793651

500 rows × 8 columns

Test & Training Splitting

Splitting data into training and test sets is crucial to evaluate model performance on unseen data, prevent overfitting, and ensure the model generalizes well to new data. This helps in building robust and reliable machine learning models.

- Our desired Outcome is "Chance of Admit", So we will divide our scaleddata into X,y
- We will use 80:20 ratio for train & test. Further we will Divide X & y as below data sets
  - Xtrain
  - Xtest
  - ytrain
  - ytest

```
[ ] Xtrain, Xtest, ytrain, ytest = train_test_split(scaleddata.drop(["Chance of Admit"], axis = 1), scaleddata["Chance of Admit"], test_size=0.2, random_state=2)

[ ] Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape

((400, 7), (100, 7), (400,), (100,))
```

Model Building - Linear Regression [OLS]

```
[ ] Xtrain1 = sm.add_constant(Xtrain)  # Statmodels default is without intercept, to add intercept we need to add constant.

    model = sm.OLS(ytrain, Xtrain1)
    results = model.fit()
```

```
[ ] print(results.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:        Chance of Admit   R-squared:                       0.829
Model:                            OLS   Adj. R-squared:                  0.826
Method:                 Least Squares   F-statistic:                     272.0
Date:                Wed, 06 Dec 2023   Prob (F-statistic):          3.48e-146
Time:                        16:15:16   Log-Likelihood:                 388.56
No. Observations:                 400   AIC:                            -761.1
Df Residuals:                     392   BIC:                            -729.2
Df Model:                           7
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const                0.0224      0.015      1.475      0.141      -0.007       0.052
GRE Score            0.1692      0.044      3.888      0.000       0.084       0.255
TOEFL Score          0.1314      0.043      3.030      0.003       0.046       0.217
University Rating    0.0307      0.026      1.184      0.237      -0.020       0.082
SOP                  0.0136      0.031      0.437      0.662      -0.048       0.075
LOR                  0.1033      0.025      4.120      0.000       0.054       0.153
CGPA                 0.5615      0.053     10.634      0.000       0.458       0.665
Research             0.0392      0.011      3.477      0.001       0.017       0.061
==============================================================================
Omnibus:                       93.913   Durbin-Watson:                   1.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              230.334
Skew:                          -1.155   Prob(JB):                     9.63e-51
Kurtosis:                       5.912   Cond. No.                         23.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

- *From Summary R- Squared is 0.829 and R-Squared Adj is 0.826, from whcih we can say all the features in the model are relevant*
- *Prob (F-statistic):3.48e-146 --> is very low, which mean we can reject our null hypothesis [H0: Weights of all features are equal to Zero ; Ha: Weight of Some features are not equal to zero]*
- *from P>|t| of feature Weights , we can confidently say Weights of CGPA,GRE Score, TOEFL Score, LOR, Research are correct & significant to our model as their P>|t| is very low*
- *However weight of University Rating and SOP are not so significant to our model as their P>|t| is very high*

Lets Calculate Predicted Y values for Test and Train Data

```
[ ] # Calculating the Predicted Values (yhat) of training data
    yhattrain = results.predict(Xtrain1)

[ ] # Calculating the Predicted Values (yhat) of testing data
    Xtest1 = sm.add_constant(Xtest)
    yhattest = results.predict(Xtest1)
```

Let Calculate the Coefficients of Each Feature:

```
results.params.sort_values(ascending = False)

CGPA               0.561533
GRE Score          0.169155
TOEFL Score        0.131429
LOR                0.103283
Research           0.039244
University Rating  0.030733
const              0.022418
SOP                0.013592
dtype: float64
```

Table 5 : Feature Coefficients

```
weights = pd.DataFrame(list(zip(Xtest.columns,np.abs(results.params[1:]))),
                       columns=['feature', 'coeff'])
weights.sort_values(by = "coeff", ascending = False, inplace = True)
sns.barplot(y='feature', x='coeff', data=weights)
plt.xticks(rotation=90)
plt.show()
```

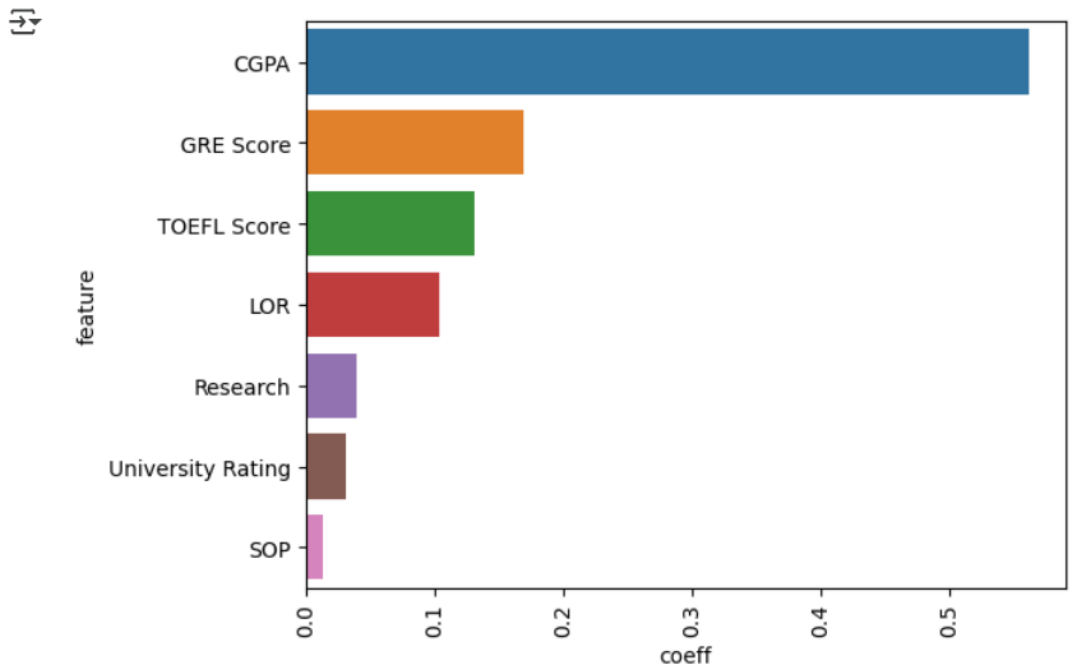


Figure 9 : Feature Importance thru Coefficient Values

- CGPA is the most important Feature for predicting "Chance of Admit"
- Next comes GRE Score, TOEFL Score, LOR
- Research, University Rating & SOP has least effect for Predicting "Chance of Admit"

Let us Plot Y & Yhat to See how good Model is Predicting:

```
[ ] plt.scatter(ytrain, yhattrain)
```

```
<matplotlib.collections.PathCollection at 0x7990bc8ab970>
```

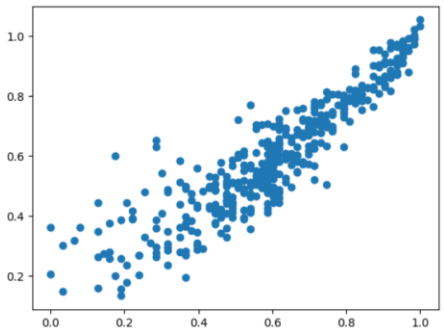
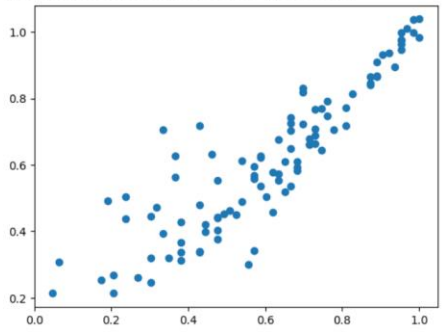


Figure 10 : Scatter plot between Ytrue and Ypredicted

```
plt.scatter(ytest, yhattest)
```

```
<matplotlib.collections.PathCollection at 0x7990bca69cc0>
```



- from Yhat and yactual plot we can say , model is predicting good
- Now lets check R-square of Testing & Training data

R-Square

R-squared, also known as the coefficient of determination, is important because it measures the proportion of the variance in the dependent variable that is predictable from the independent variables. This helps in assessing the goodness of fit of a regression model and understanding how well the model explains the variability of the outcome

```
[ ] # R-square Using OLS Attribute for Training data
    results.rsquared
```

```
0.8292848575410212
```

```
[ ] # R-square for training data
    1 - np.sum((ytrain - yhattrain)**2)/np.sum((ytrain - ytrain.mean())**2)
```

```
0.8292848575410212
```

```
[ ] # R-square for testing data
    1 - np.sum((ytest - yhattest)**2)/np.sum((ytest - ytest.mean())**2)
```

```
0.7927364356781469
```

- From R-square Values of Test & Train Data, we can say model is neither Overfit nor Underfit
- It is appropriately Fit

R-Square adjusted

Adjusted R-squared is important because it accounts for the number of predictors in a model, providing a more accurate measure of model fit by penalizing the inclusion of irrelevant variables. This helps in selecting models that truly improve predictive power without overfitting.

```
[ ]  # used defined function for R-Square Adjusted
     def R2adj(X,R2):
       return (1 - (1-R2) * ((X.shape[0]-1)/(X.shape[0]-X.shape[1]-1)))
```

```
[ ]  R2adj(Xtrain,results.rsquared)
```

```
0.8262363728542538
```

```
[ ]  # R-square adjusted Using OLS Attribute for Training data
     results.rsquared_adj
```

```
0.8262363728542538
```

```
[ ]  # R-square adjusted for training data
     R2= 1 - np.sum((ytrain - yhattrain)**2)/np.sum((ytrain - ytrain.mean())**2)
     1 - (1-R2) * ((Xtrain.shape[0]-1)/(Xtrain.shape[0]-Xtrain.shape[1]-1))
```

```
0.8262363728542538
```

```
[ ]  # R-square adjusted for testing data
     R2= 1 - np.sum((ytest - yhattest)**2)/np.sum((ytest - ytest.mean())**2)
     1 - (1-R2) * ((Xtest.shape[0]-1)/(Xtest.shape[0]-Xtest.shape[1]-1))
```

```
0.7769663818710494
```

- From Comparison of R-Square and R-Square Adjusted, we see difference b/w R-Square & R-Square adj for Training [0.829 - 0.826 = 0.003] , for testing [0.792 - 0.776 = 0.016] is very small
- So we can conclude there are no irrelevant features inline with our earlier Hypothesis Testing

Mean Absolute Error

Mean Absolute Error (MAE) is a metric used to measure the average magnitude of errors in a set of predictions, without considering their direction. It is calculated as the average of the absolute differences between predicted and actual values, providing a straightforward measure of prediction accuracy. Lower MAE values indicate better model performance

```
[ ]  # MAE for Training data
     (np.sum(np.abs(ytrain-yhattrain)))/len(ytrain)
```

```
0.06568270842637872
```

```
[ ]  # MAE for testing data
     (np.sum(np.abs(ytest-yhattest)))/len(ytest)
```

```
0.07509931877554368
```

- Both Test & Train data MAE Values are very low meaning our model is performing good

Root Mean Square Error

Root Mean Square Error (RMSE) is a metric used to measure the average magnitude of errors in a set of predictions. It is calculated as the square root of the average of the squared differences between predicted and actual values.

RMSE gives higher weight to larger errors, making it useful for identifying models with significant prediction errors. Lower RMSE values indicate better model performance

```
[ ] # RMSE for Training data
    ((np.sum((ytrain-yhattrain)**2))**0.5)/len(ytrain)

    0.004579977329232789

[ ] # RMSE for testing data
    ((np.sum((ytest-yhattest)**2))**0.5)/len(ytest)

    0.010564357955452694
```

Both Test & Train data RMSE Values are very low meaning our model is performing good

Linear Regression Assumptions

VIF

Variance Inflation Factor (VIF) is a measure used in linear regression to detect multicollinearity among predictor variables. High VIF values indicate that a predictor variable is highly correlated with other predictors, which can inflate the variance of the coefficient estimates and make the model unstable. Generally, a VIF value above 5 suggests significant multicollinearity that may need to be addressed.

Below is Iterative code for removing High VIF features by maintaining threshold for VIF<=5 [iterates code till none of features VIF >5

```
vif_thr =5
r2_thr = 0 # focusing only on VIF irrepsctive of feature removal effect on R-Square Adj
j = 1
feats_removed = []
cols2 = Xtrain.columns
while True:
    vif = pd.DataFrame()
    X_t = pd.DataFrame(Xtrain, columns=Xtrain.columns)[cols2]
    vif['Features'] = cols2
    vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)

    cols2 = vif["Features"][1:].values
    X2 = pd.DataFrame(Xtrain, columns=Xtrain.columns)[cols2] #Dropped the feature with high VIF & Again check Perfromance of the reamining data

    X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add intercept we need to add constant
    sm_model = sm.OLS(list(ytrain), X2_sm).fit()
    if (vif.iloc[0]['VIF'] < vif_thr) or (sm_model.rsquared_adj < r2_thr):
        print('Reached threshold')
        print('Highest vif:',vif.iloc[0])
        print('Current adj.R2',sm_model.rsquared_adj)
        print('Features removed:', j-1)
        print('List of features removed:', feats_removed)
        break
    feats_removed.append(vif.iloc[0]['Features']) # Addting the high VIF removed feature name to feats_removed
    j += 1
print()
print("*****************************************************************************************************************************************")
print()
# Final Assesment of Data after removing all possible high VIF feature with set Threshold values
print(vif)
print(sm_model.summary())
```

Figure 11 : Iterative Code to Eliminate Variance Inflation Factor

```
Reached threshold
Highest vif: Features     LOR
VIF         2.53
Name: 0, dtype: object
Current adj.R2 0.29447764265596066
Features removed: 5
List of features removed: ['CGPA', 'TOEFL Score', 'SOP', 'GRE Score', 'University Rating']

**********************************************************************************************************************************************************

   Features   VIF
0       LOR  2.53
1  Research  2.53
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.296
Model:                            OLS   Adj. R-squared:                  0.294
Method:                 Least Squares   F-statistic:                     167.5
Date:                Wed, 06 Dec 2023   Prob (F-statistic):           3.16e-32
Time:                        16:34:14   Log-Likelihood:                 105.27
No. Observations:                 400   AIC:                            -206.5
Df Residuals:                     398   BIC:                            -198.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4704      0.014     33.472      0.000       0.443       0.498
Research       0.2431      0.019     12.944      0.000       0.206       0.280
==============================================================================
Omnibus:                       23.026   Durbin-Watson:                   1.929
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               25.515
Skew:                          -0.612   Prob(JB):                     2.88e-06
Kurtosis:                       3.179   Cond. No.                         2.78
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 12 : Stats Model Summary

- By focusing only on VIF Score < 5 --> five Feaures ['CGPA', 'TOEFL Score', 'SOP', 'GRE Score', 'University Rating'] have been Removed
- As a results Rsqure_adj of the model came down to 0.294 --> Not a good Model for Prediction
- Now we will focus both on VIF & R-Square Adj

- Below is Iterative code for auto removing High VIF features by maintaing threshold for **VIF<=5 & Rsquare_adj >=0.8**

```
vif_thr =5
r2_thr = 0.8
j = 1
feats_removed = []
cols2 = Xtrain.columns
while True:
    vif = pd.DataFrame()
    X_t = pd.DataFrame(Xtrain, columns=Xtrain.columns)[cols2]
    vif['Features'] = cols2
    vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    if j == 1:
        print("****************************")
        print("Initial Condition")
        print("****************************")
        print(vif)
        print()

    cols2 = vif["Features"][1:].values
    X2 = pd.DataFrame(Xtrain, columns=Xtrain.columns)[cols2] #Dropped the feature with high VIF & Again check Perfromance of the reamining data

    X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add intercept we need to add constant
    sm_model = sm.OLS(list(ytrain), X2_sm).fit()
    if (vif.iloc[0]['VIF'] <= vif_thr) or (sm_model.rsquared_adj < r2_thr):

        print("Checking VIF")
        print("****************************")
        print('Reached threshold')
        print('Highest vif :',vif.iloc[0])
        print('Current adj.R2 if Highest VIF feature removed:',sm_model.rsquared_adj)
        print('Features removed :', j-1)
        print('List of features removed :', feats_removed)
        break
    feats_removed.append(vif.iloc[0]['Features']) # Addting the high VIF removed feature name to feats_removed
    j += 1
print("********************************************************************************")
# Final Assesment of Data after removing all possible high VIF feature with set Threshold values
```

```
****************************
Initial Condition
****************************
             Features    VIF
5                CGPA  40.15
1         TOEFL Score  29.57
0           GRE Score  28.94
3                 SOP  17.99
4                 LOR  11.28
2  University Rating  10.87
6            Research   3.28

Checking VIF
****************************
Reached threshold
Highest vif : Features     CGPA
VIF         40.15
Name: 5, dtype: object
Current adj.R2 if Highest VIF feature removed: 0.7766781427005298
Features removed : 0
List of features removed : []
********************************************************************************
```

- CGPA Score has very high VIF values
- But removal of these features is detoriating our Model,Bringing down Rsquare_adj to 0.776 which is less than threshold < 0.8

Mean of Residuals

The mean of residuals assumption in linear regression states that the average of the residuals (errors) should be zero. This ensures that the model's predictions are unbiased, meaning the predicted values are, on average, equal to the actual values. If this assumption holds, it indicates that the model is correctly specified

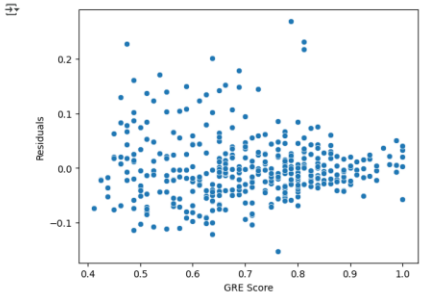
```
[ ] np.mean(yhattrain-ytrain)
-4.929390229335695e-16
[ ] np.mean(yhattest-ytest)
0.006100917484111618
```

- Mean of Residuals are almost equivalent to Zero

Linearity of Variables

The linearity of variables assumption in linear regression states that there should be a linear relationship between the independent variables and the dependent variable. This means that changes in the predictor variables should result in proportional changes in the response variable. Ensuring this assumption holds helps in accurately modeling the relationship and making reliable predictions.

```
[ ] for i in Xtrain.columns:
      sns.scatterplot(x=Xtrain[i],y=(yhattrain-ytrain))
      plt.xlabel(i)
      plt.ylabel("Residuals")
      plt.show()
```



- We can say No Rigid pattern observed for Residual w.r.t Features.

No Heteroskedasticity

Heteroskedasticity in regression analysis refers to the condition where the variance of the residuals (errors) is not constant across all levels of the independent variables. This violates one of the key assumptions of ordinary least squares (OLS) regression, which assumes homoscedasticity (constant variance). When heteroskedasticity is present, it can lead to inefficient estimates and unreliable hypothesis tests.

```
[ ] sns.scatterplot(x=yhattrain,y=yhattrain-ytrain)
    plt.xlabel("Predicted Chance of Admit")
    plt.ylabel("Residuals")
    plt.title("Predicted Chance of Admit vs Residuals")
    plt.show()
```

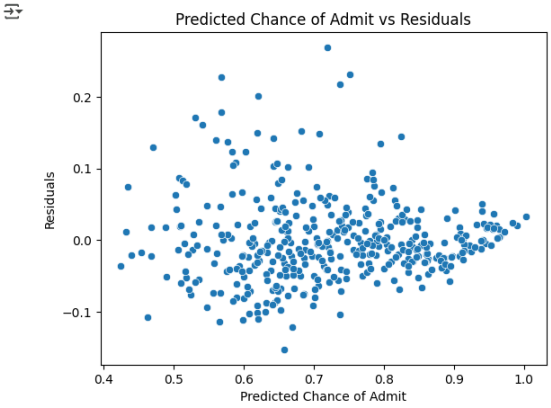


Figure 13 : Heteroskedasticity Check

- Very Slight Heteroskedasticity is Present
- However we can say variance is almost same for all data point

Normality of Residuals

The normality of residuals assumption in linear regression states that the residuals (errors) should be normally distributed. This is important for valid hypothesis testing and constructing confidence intervals. When this assumption holds, it ensures that the statistical tests for coefficients are reliable, and the model's predictions are accurate.

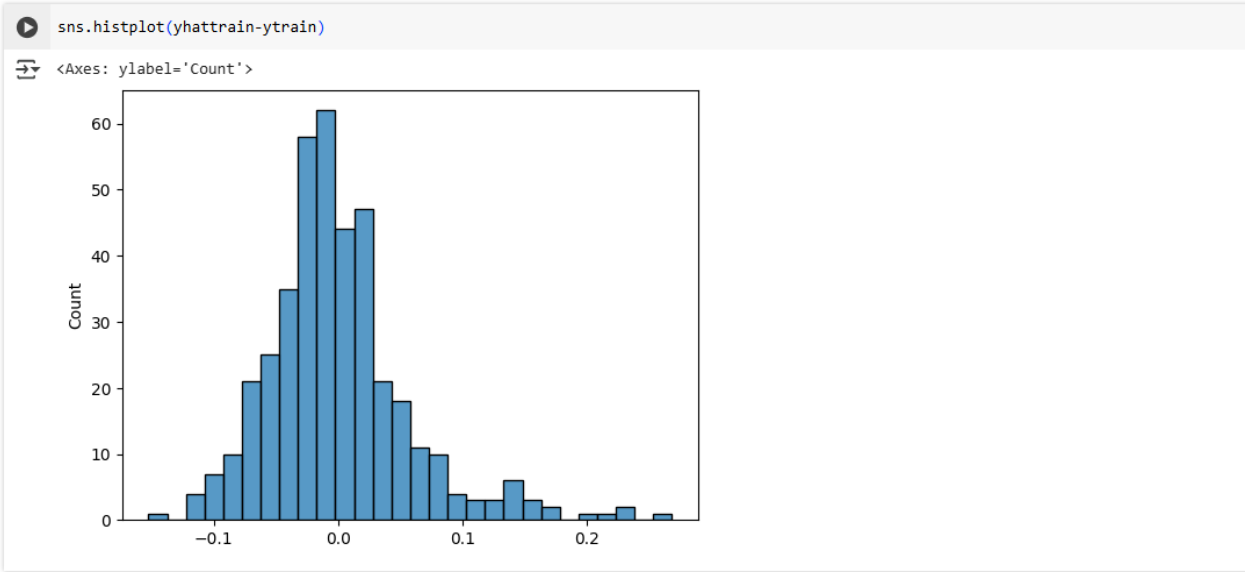


Figure 14 : Residuals Normality Check

- Almost bell Shape curve can be Visualized.
- Although it is not Perfect Normal Distribution, it is Skewed Normal
- We Can Further Consolidate the Normal Distribution using Hypothesis Testing or QQ Plots

So till now Have Build Linear Regression Model using Stats Model Library, Verified all Model Metric and Checked all Linear Regression Assumptions

Now we will Explore various Other Linear Regression Modelling Techniques

Model Building - Linear Regression [Lasso]

Lasso Linear Regression, or Lasso (Least Absolute Shrinkage and Selection Operator), is a type of linear regression that includes a regularization term to prevent overfitting and enhance model performance. The key feature of Lasso is that it can shrink some coefficients to zero, effectively performing variable selection and simplifying the model

```
[ ]  model1 = Lasso(alpha=0.0001)
     model1.fit(Xtrain,ytrain)
     print(model1.intercept_)
     print(model1.coef_)
```

```
0.02411503059989173
[0.16858803 0.13105956 0.03116347 0.01323098 0.10284886 0.55961982
 0.03946013]
```

```
[ ]  model1.score(Xtrain, ytrain)
```

```
0.8292787171347655
```

```
[ ]  # calculating RSquared Adjusted
     R2adj(Xtrain,model1.score(Xtrain, ytrain))
```

```
0.8262301227978863
```

```
[ ]  model1.score(Xtest, ytest)
```

```
0.7929714474185146
```

let us do L1 Regularization constant Hyperparameter Tuning

```
# let us do L1 Regularization constant Hyperparameter Tuning
R2testScore = []
R2trainScore = []
R2adjScore=[]
hyperparameter = [0.0001,0.001,0.01,0.1]
for i in hyperparameter:
  model1 = Lasso(alpha=i)
  model1.fit(Xtrain,ytrain)

  R2 = model1.score(Xtrain,ytrain)
  R2trainScore.append(R2)
  R2testScore.append(model1.score(Xtest,ytest))
  R2adjScore.append(R2adj(Xtrain,R2))

sns.lineplot(x=hyperparameter, y=R2trainScore, label = "R2trainScore")
sns.lineplot(x=hyperparameter, y=R2testScore , label = "R2testScore")
sns.lineplot(x=hyperparameter, y=R2adjScore , label = "R2adjScore")
plt.xscale('log')
plt.show()
```

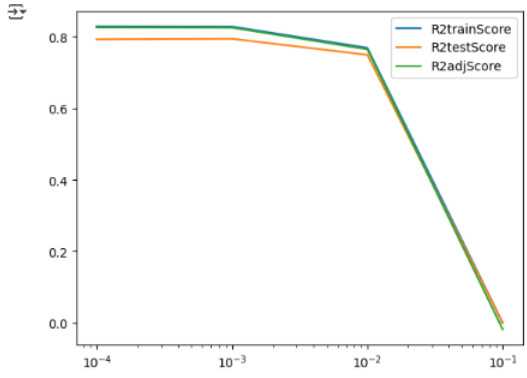


Figure 15 : Hyperparameter tuning for Lasso Model

- Lasso Linear_Regression is giving slightly better results with very low L1 Regularization Constant values.

- we are getting R-Square Detoriated with increase in L1 Regularization constant.
- Meaning with our Earlier Linear Regression Model was good with Given 7 features, further by increasing Regularization we are deteriorating our model by increasing underfitting.

Model Building - Linear Regression [Ridge]

Ridge Regression adds a penalty equal to the square of the magnitude of coefficients to the loss function. This penalty term is controlled by a parameter, alpha. The regularization term introduces some bias but reduces variance, leading to a more stable model that generalizes better to new data. Unlike Lasso, Ridge Regression shrinks the coefficients but does not set any of them to zero, meaning it retains all features in the model.

```
[ ] model2 = Ridge(alpha = 0.001)
    model2.fit(Xtrain,ytrain)
    print(model2.intercept_)
    print(model2.coef_)
```

```
0.022438486565023008
[0.16918593 0.13146555 0.03074948 0.01361192 0.1032931  0.56138234
 0.03924791]
```

```
[ ] model2.score(Xtrain, ytrain)
```

```
0.82928485385645
```

```
[ ] model2.score(Xtest, ytest)
```

```
0.7927324857587962
```

```
# let us do L1 Regularization constant Hyperparameter Tuning
R2testScore = []
R2trainScore = []
R2adjScore=[]
hyperparameter = [0.001,0.01,0.1,1,10,100]
for i in hyperparameter:
  model2 = Ridge(alpha=i)
  model2.fit(Xtrain,ytrain)

  R2 = model2.score(Xtrain,ytrain)
  R2trainScore.append(R2)
  R2testScore.append(model2.score(Xtest,ytest))
  R2adjScore.append(R2adj(Xtrain,R2))

sns.lineplot(x=hyperparameter, y=R2trainScore, label = "R2trainScore")
sns.lineplot(x=hyperparameter, y=R2testScore , label = "R2testScore")
sns.lineplot(x=hyperparameter, y=R2adjScore , label = "R2adjScore")
plt.xscale('log')
plt.show()
```

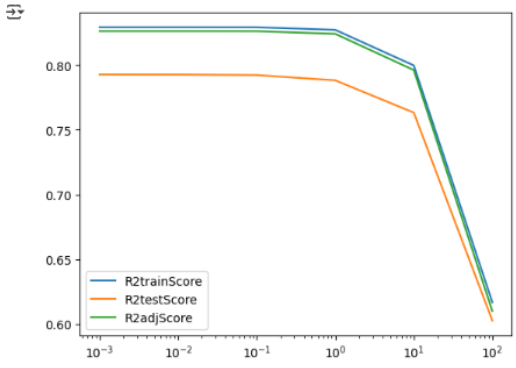


Figure 16 : Hyperparameter tuning for Ridge Model

- we are are getting R-Square Detoriated with increase in L2 Regularization constant
- Meaning with our Earlier Linear Regression Model was good with Given 7 features, further by increasing Regularization we are deteriorating our model by underfitting.
- so now we will use Polynomial Feature and check if our Linear Regression Model can be further Increased.

Model Building - Polynomial Regression

Polynomial Regression is a type of regression analysis where the relationship between the independent variable X and the dependent variable y is modeled as an nth degree polynomial. It extends linear regression by allowing for non-linear relationships between the variables.

Let us check if Different Polynomial will increase Rsquare & Rsquare_adj

```
R2testScore = []
R2trainScore = []
R2adjScore=[]
poly_hyper = np.arange(1,5) # Polynomial Degree Hyperparamter
for i in poly_hyper:
  poly = PolynomialFeatures(i)
  Xtrainpoly = poly.fit_transform(Xtrain)
  Xtestpoly = poly.fit_transform(Xtest)
  model4 = LinearRegression()
  model4.fit(Xtrainpoly,ytrain)

  R2 = model4.score(Xtrainpoly,ytrain)
  R2trainScore.append(R2)
  R2testScore.append(model4.score(Xtestpoly,ytest))
  R2adjScore.append(R2adj(Xtrainpoly,R2))



sns.lineplot(x=poly_hyper, y=R2trainScore, label = "R2trainScore")
sns.lineplot(x=poly_hyper, y=R2testScore , label = "R2testScore")
sns.lineplot(x=poly_hyper, y=R2adjScore , label = "R2adjScore")
plt.ylim((-1,1))
plt.show()

print()
maxR2poly = poly_hyper[np.argmax(R2adjScore)]
print("R-Square is Highest for Polynomial degree", maxR2poly)
print("R-Square train for polynomial degree ", maxR2poly , "is",R2trainScore[maxR2poly-1])
print("R-Square adj for polynomial degree ", maxR2poly , "is",R2adjScore[maxR2poly-1])
print("R-Square test polynomial degree ", maxR2poly , "is",R2testScore[maxR2poly-1])
```

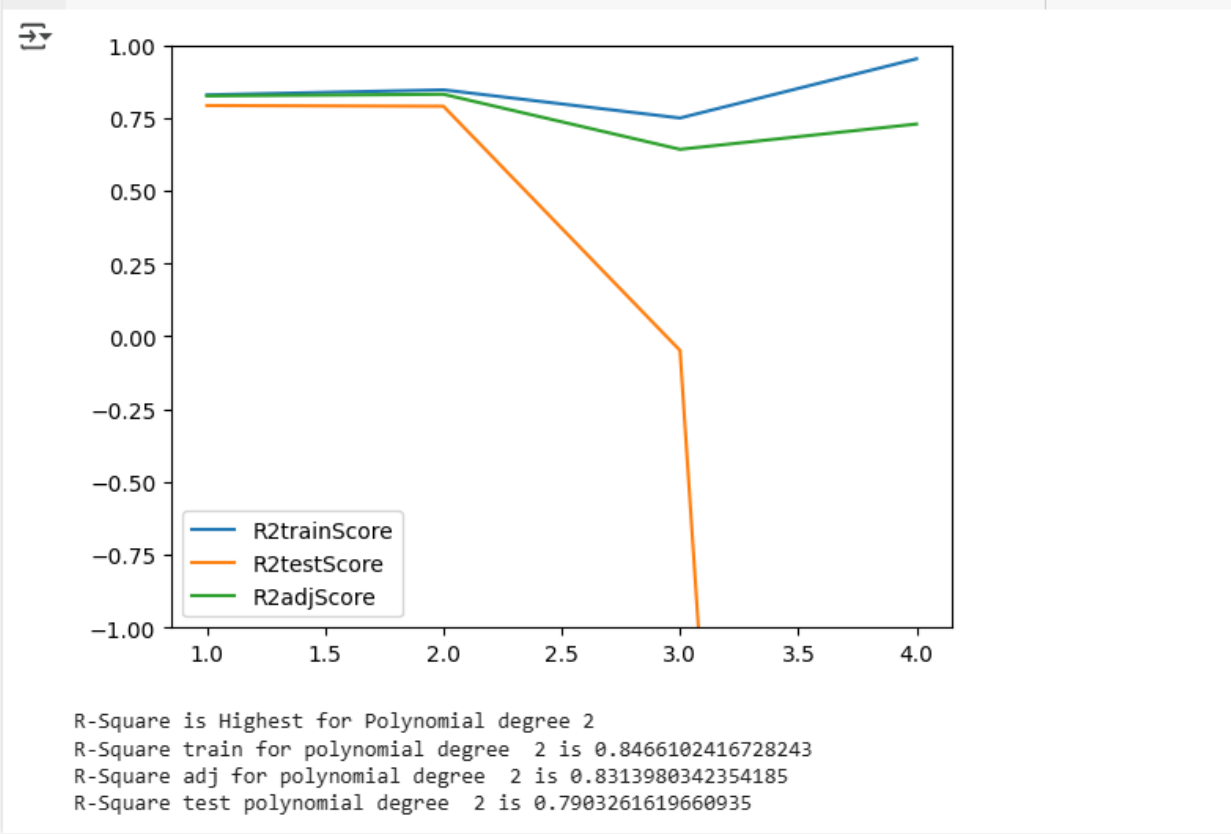
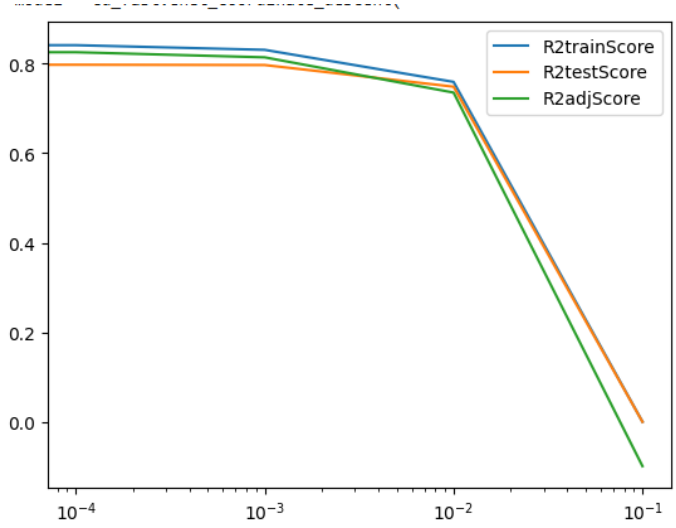


Figure 17 : Hyperparamter Tuning for Polynomial Degree

- Linear Regression Model is performing better with 2nd Degree Polynomial, further when degree increase it is leading to Overfit resulting in bad results for Testing data
- Further we will do L1 Regularization to on polynomial Features to to check if model performance can be further increased by reducing overfitting problem in higher polynomials

```
[ ]  R2testScore = []
     R2trainScore = []
     R2adjScore=[]
     hyperparameter = [ 0,0.0001,0.001,0.01,0.1] # L1 Regularization Constant
     for i in hyperparameter:


       poly = PolynomialFeatures(2)
       Xtrainpoly = poly.fit_transform(Xtrain)
       Xtestpoly = poly.fit_transform(Xtest)
       model5 = Lasso(alpha=i)
       model5.fit(Xtrainpoly,ytrain)

       R2 = model5.score(Xtrainpoly,ytrain)
       R2trainScore.append(R2)
       R2testScore.append(model5.score(Xtestpoly,ytest))
       R2adjScore.append(R2adj(Xtrainpoly,R2))

     sns.lineplot(x=hyperparameter, y=R2trainScore, label = "R2trainScore")
     sns.lineplot(x=hyperparameter, y=R2testScore , label = "R2testScore")
     sns.lineplot(x=hyperparameter, y=R2adjScore , label = "R2adjScore")
     plt.xscale('log')
     plt.show()

     print()
     maxR2hyper = hyperparameter[np.argmax(R2adjScore)]
     print("R-Square is Highest for L1 Regularization Constant", maxR2hyper)
     print("R-Square train :",R2trainScore[np.argmax(R2adjScore)])
     print("R-Square adj :",R2adjScore[np.argmax(R2adjScore)])
     print("R-Square test :",R2testScore[np.argmax(R2adjScore)])
```




```
R-Square is Highest for L1 Regularization Constant 0
R-Square train : 0.8464417191143481
R-Square adj : 0.8312127986959363
R-Square test : 0.7877397175497122
```

- Use of Regularization did not improve out 2nd Degree Polynomial Model

Business Insights:

- All the student applied are meritorious.
    - CGPA of the Student are >= 7.0
    - GRE Scores are greater than 290, More No of Data points are in that range 310 ~ 327
    - TOEFL Scores are greater than 91, More No of Data points are in that range 99~112
- All features have Positive correlation with "Chance of Admit"
    - CGPA[0.88],GRE Score[0.81],TOEFL Score[0.79] has very high correlation with "Chance of Admit"
    - Further SOP[0.68], LOR[0.65] has almost same Correlation with ""Chance of Admit"
    - Comparatively Low Correlation is found w.r.t Research & "Chance of Admit"[0.55], However is positively impactful
- From Stats OLS model Summary R- Squared is 0.829 and R-Squared Adj is 0.826, from which we can say all the features in the model are relevant
    - CGPA is the most important & Significant Feature for predicting "Chance of Admit"
    - Next significant feature in decreasing order comes: GRE Score, TOEFL Score, LOR
    - Research, University Rating & SOP are less significant for Predicting "Chance of Admit"
- From R-square Values of Train[0.829] & Test Data[0.792], we can say model is neither Overfit nor Underfit, It is Best Fit
- All Assumpitions of Linear Regression are almost Satisfied
- Further Use Regularization of Constant with Lasso and Ridge did not show much improvement as our OLS model is already Best Fit
- However, use of 2nd degree Polynomial feature very slightly improved Rsquare Adj from 0.826 --> 0.831

Recommendations:

- To improve the Model Following additional Features can be added
    - Under graduation College Rating
    - Extra Circular activity with an Unified Rating
    - Personal Essay Rating
    - Professional Working Experience
    - Financial Status
- further we can get actual data of Admission [yes or No] of past History records to understand the Actual Threshold for "Chance of Admit" to definitely get admitted
- With the help of above model, Jamboree can confidently guide the Students with exact course of action meaning what score/rating to improve to get admission to particular college

Chapter 4 : Business Case Study 4

Problem Description

About Company

A leading ride-sharing platform, aiming to provide reliable, affordable, and convenient urban transportation for everyone.

Problem:

- The constant challenge of ride sharing platform is the churn rate of its drivers. Ensuring driver loyalty and reducing attrition are crucial to the company's operation.
- Analyzing driver data can reveal patterns in driver behavior.
- Model Building for Driver Churn using Ensemble Techniques like Boosting and Bagging

Dataset:

- MMMM-YY : Reporting Date (Monthly)
- Driver_ID : Unique id for drivers
- Age : Age of the driver
- Gender : Gender of the driver – Male : 0, Female: 1
- City : City Code of the driver
- Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
- Income : Monthly average Income of the driver
- Date Of Joining : Joining date for the driver
- LastWorkingDate : Last date of working for the driver
- Joining Designation : Designation of the driver at the time of joining
- Grade : Grade of the driver at the time of reporting
- Total Business Value : The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
- Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

Libraries Used:

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

!pip install category_encoders
from category_encoders import TargetEncoder
from sklearn.preprocessing import LabelEncoder, StandardScaler,
MinMaxScaler

from scipy.stats import kstest
```

```
from scipy import stats

from sklearn.model_selection import train_test_split

from sklearn.impute import KNNImputer
from imblearn.over_sampling import SMOTE

from sklearn.ensemble import
RandomForestClassifier,GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import classification_report
```

Outcome of Analysis:

- Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset.
- Check for missing values and Prepare data for KNN Imputation, consider only numerical features for this purpose.
- Aggregate data to remove multiple occurrences of same driver data.
- Feature Engineering Steps:
    - Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1
    - Target variable creation: Create a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1
    - Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1
- Statistical summary of the derived dataset
- Check correlation among independent variables and how they interact with each other.
- One hot encoding of the categorical variable
- Class Imbalance Treatment
- Standardization of training data
- Using Ensemble learning - Bagging, Boosting methods with some hyper-parameter tuning
- Results Evaluation:
    - Classification Report
    - ROC AUC curve
- Provide actionable Insights & Recommendations

Business Questions to be answered from Analysis

- Understand correlation among independent variables and how they interact with each other.
- Verify Data imbalance and develop Model
- Develop bagging and Boosting Model to Predict the Driver Churn
- Evaluate the model Performance using Classification Report and ROC AUC Curve

Analysis

Let import Data and Check the Structure

```
[ ] data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.csv")
```

```
[ ] data.shape
```

```
(19104, 14)
```

```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            19104 non-null  int64  
 1   MMM-YY                19104 non-null  object 
 2   Driver_ID             19104 non-null  int64  
 3   Age                   19043 non-null  float64
 4   Gender                19052 non-null  float64
 5   City                  19104 non-null  object 
 6   Education_Level       19104 non-null  int64  
 7   Income                19104 non-null  int64  
 8   Dateofjoining         19104 non-null  object 
 9   LastWorkingDate       1616 non-null   object 
 10  Joining Designation   19104 non-null  int64  
 11  Grade                 19104 non-null  int64  
 12  Total Business Value  19104 non-null  int64  
 13  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(8), object(4)
memory usage: 2.0+ MB
```

```
[ ] data.head(10)
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381060	2
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665480	2
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	0	2
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	350000	1
9	9	04/01/20	4	43.0	0.0	C13	2	65603	12/07/19	27/04/20	2	2	0	1

```
[ ] data[data.Driver_ID == 4]
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	0	1
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	350000	1
9	9	04/01/20	4	43.0	0.0	C13	2	65603	12/07/19	27/04/20	2	2	0	1

Table 6 : Dataset for Logistics Regression

- Same Driver ID has Many Rows...we will merge all the Rows after extracting information from individual rows

Data Cleaning

Dropping Irrelevant Columns

- "Unnamed" 0" Feature is just an index , so value as a feature, So we will drop the feature

```
data.drop(['Unnamed: 0'], axis = 1,inplace= True)
```

Datatype Conversion

lets convert all the date features to datetime

```
[ ] data["MMM-YY"]=data["MMM-YY"].astype("datetime64")
    data["Dateofjoining"]=data["Dateofjoining"].astype("datetime64")
    data["LastWorkingDate"]=data["LastWorkingDate"].astype("datetime64")
```

Missing Value Treatment

```
[ ]  data.isna().sum(axis = 0)
```

```
MMM-YY                     0
Driver_ID                  0
Age                       61
Gender                    52
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
LastWorkingDate        17488
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
dtype: int64
```

- we will impute missing values through KNN Imputation for "Age" & "Gender"
- we need not treat "LastWorkingDate" becoz if driver has not left...it will be represented as Nan Only....later we will deal it by creating new feature

Gender

```
[ ]  data[data["Driver_ID"]==2774]
```

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
19023	2019-01-01	2774	40.0	0.0	C15	1	42313	2018-07-21	NaT	1	1	450010	4
19024	2019-02-01	2774	NaN	0.0	C15	1	42313	2018-07-21	NaT	1	1	1141280	4
19025	2019-03-01	2774	40.0	0.0	C15	1	42313	2018-07-21	NaT	1	1	2335840	4
19026	2019-04-01	2774	40.0	0.0	C15	1	42313	2018-07-21	NaT	1	1	106880	2
19027	2019-05-01	2774	40.0	0.0	C15	1	42313	2018-07-21	NaT	1	1	250000	2
19028	2019-06-01	2774	40.0	NaN	C15	1	42313	2018-07-21	NaT	1	1	0	2
19029	2019-07-01	2774	41.0	0.0	C15	1	42313	2018-07-21	2019-07-19	1	1	0	1

- As we can see rest of the Same Driver ID rows have Gender mentioned
- So we will use Nearest Neighbor as 1 as it is Categorical
- We will select only columns ["Driver_ID","Educational_Level","Income","Dateofjoining","Joining Designation"] for fitting of KNN Imputer

```
GenderKNN = KNNImputer(n_neighbors=1)
GenderKNN.fit(data[["Driver_ID","Education_Level","Gender","Income","Joining Designation"]])
data[["Driver_ID","Education_Level","Gender","Income","Joining Designation"]]=GenderKNN.transform(data[["Driver_ID","Education_Level","Gender","Income","Joining Designation"]])
```

```
[ ]  data.isna().sum(axis = 0)
```

```
MMM-YY                     0
Driver_ID                  0
Age                       61
Gender                     0
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
LastWorkingDate        17488
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
dtype: int64
```

Figure 18 : Missing Value Treatment

- All missing values in Gender have been imputed using KNN

Age

Like Gender, we will use Nearest Neighbor as 1

We will select only columns [Driver_ID", "Education_Level", "Age", "Gender", "Income", "Joining Designation"] for fitting of KNN Imputer

```
[ ] AgeKNN = KNNImputer(n_neighbors=1)
    AgeKNN.fit(data[["Driver_ID","Education_Level","Age","Gender","Income","Joining Designation"]])
    data[["Driver_ID","Education_Level","Age","Gender","Income","Joining Designation"]]=AgeKNN.transform(data[["Driver_ID","Education_Level","Age","Gender","Income","Joining Designation"]])
```

```
data.isna().sum(axis=0)
```

```
MMM-YY                    0
Driver_ID                 0
Age                       0
Gender                    0
City                      0
Education_Level           0
Income                    0
Dateofjoining             0
LastWorkingDate       17488
Joining Designation       0
Grade                     0
Total Business Value      0
Quarterly Rating          0
dtype: int64
```

All Missing Values in the Age are Imputed using KNN imputer

Duplicate rows

```
[ ] data.loc[data.duplicated()]
```

MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating

<u>Row Merging for Each Driver_ID</u>

Let us first sort the Data as per the Driver_ID and "MMM-YY"[Reporting Month], so that when mergin we get the correct data in last and first columns.

```
[ ] data.sort_values(by=["Driver_ID","MMM-YY"], ascending=[True,True],inplace = True,ignore_index=True)
```

- We will do 2 Nos Groupby on Driver ID to Collect First and Last Data for "Income",Grade","Quaterly Rating"
- Later we will merge them and use both data to create new features`

```
[ ] d1 = data.groupby(["Driver_ID"]).agg({'Age' : 'first',
    'Gender' : 'first' ,
    'City' : 'first',
    'Education_Level' : 'first',
    'Income' : 'first',
    'Dateofjoining' : 'first',
    'LastWorkingDate' : 'last',
    'Joining Designation' : 'first',
    'Grade' : 'first',
    'Total Business Value' : 'mean',
    'Quarterly Rating' : 'first'}).reset_index()
```

```
[ ] d2 = data.groupby(["Driver_ID"]).agg({
    'Income' : 'last',

    'Grade' : 'last',

    'Quarterly Rating' : 'last'}).reset_index()
```

```
[ ] data1 =d1.merge(d2,on="Driver_ID")
```

```
[ ] data1
```

	Driver_ID	Age	Gender	City	Education_Level	Income_x	Dateofjoining	LastWorkingDate	Joining Designation	Grade_x	Total Business Value	Quarterly Rating_x	Income_y	Grade_y	Quarterly Rating_y
0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-03-11	1.0	1	571860.000000	2	57387.0	1	2
1	2.0	31.0	0.0	C7	2.0	67016.0	2020-11-06	NaT	2.0	2	0.000000	1	67016.0	2	1
2	4.0	43.0	0.0	C13	2.0	65603.0	2019-12-07	2020-04-27	2.0	2	70000.000000	1	65603.0	2	1
3	5.0	29.0	0.0	C9	0.0	46368.0	2019-01-09	2019-03-07	1.0	1	40120.000000	1	46368.0	1	1
4	6.0	31.0	1.0	C11	1.0	78728.0	2020-07-31	NaT	3.0	3	253000.000000	1	78728.0	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2376	2784.0	33.0	0.0	C24	0.0	82815.0	2015-10-15	NaT	2.0	3	906200.833333	3	82815.0	3	4
2377	2785.0	34.0	1.0	C9	0.0	12105.0	2020-08-28	2020-10-28	1.0	1	0.000000	1	12105.0	1	1
2378	2786.0	44.0	0.0	C19	0.0	35370.0	2018-07-31	2019-09-22	2.0	2	312787.777778	2	35370.0	2	1
2379	2787.0	28.0	1.0	C20	2.0	69498.0	2018-07-21	2019-06-20	1.0	1	162971.666667	2	69498.0	1	1
2380	2788.0	29.0	0.0	C27	2.0	70254.0	2020-06-08	NaT	2.0	2	328320.000000	1	70254.0	2	2

2381 rows × 15 columns

Table 7 : Aggregated Data of each DriverID

Feature Engineering 1

- Let us create some flag for Quarterly Rating, Monthly Income, Grade by checking if Rating has increased or decreased --> if increase we will categorize it as "1" representing growth
- Next, we will create our Target Variable for each driver ID from last working day--> if it present value will be "1" meaning driver left

Income

```
[ ] def func1(x):
      if x["Income_x"] >= x["Income_y"] :
        return 0
      else:
         return 1
```

```
[ ] data1["Income"] = data1.apply(func1,axis=1)
```

```
data1[data1.Income == 1].head()
```

	Driver_ID	Age	Gender	City	Education_Level	Income_x	Dateofjoining	LastWorkingDate	Joining Designation	Grade_x	Total Business Value	Quarterly Rating_x	Income_y	Grade_y	Quarterly Rating_y	Income
18	26.0	41.0	0.0	C14	2.0	121529.0	2018-05-07	NaT	1.0	3	2.911162e+06	4	132577.0	4	2	1
40	54.0	33.0	0.0	C29	1.0	117993.0	2019-07-11	NaT	4.0	4	1.879072e+06	2	127826.0	5	1	1
46	60.0	46.0	1.0	C20	0.0	82126.0	2016-09-17	NaT	1.0	3	2.051063e+06	4	89592.0	4	2	1
80	98.0	24.0	0.0	C16	0.0	57977.0	2019-08-15	2020-12-25	2.0	2	1.259732e+06	3	63774.0	3	2	1
230	275.0	39.0	0.0	C20	0.0	89124.0	2016-05-02	NaT	1.0	3	1.404075e+06	3	97226.0	4	3	1

- We can see that if Income Increase --> it has be denoted as "1" in Income Column

Quarterly Rating

```
[ ] def func1(x):
      if x["Quarterly Rating_x"] >= x["Quarterly Rating_y"] :
        return 0
      else:
         return 1
```

```
[ ] data1["Quarterly Rating"] = data1.apply(func1,axis=1)
```

```
[ ] data1[data1["Quarterly Rating"]== 1]
```

	Driver_ID	Age	Gender	City	Education_Level	Income_x	Dateofjoining	LastWorkingDate	Joining Designation	Grade_x	Total Business Value	Quarterly Rating_x	Income_y	Grade_y	Quarterly Rating_y	Income	Quarterly Rating
4	6.0	31.0	1.0	C11	1.0	78728.0	2020-07-31	NaT	3.0	3	2.530000e+05	1	78728.0	3	2	0	1
17	25.0	29.0	0.0	C24	1.0	102077.0	2017-10-30	NaT	1.0	3	1.514630e+06	3	102077.0	3	4	0	1
21	31.0	32.0	1.0	C28	2.0	65189.0	2020-07-04	NaT	3.0	3	1.584900e+05	1	65189.0	3	2	0	1
29	41.0	33.0	0.0	C29	2.0	92372.0	2019-07-04	NaT	4.0	4	1.208662e+06	1	92372.0	4	2	0	1
33	45.0	35.0	0.0	C8	2.0	77544.0	2020-05-29	NaT	3.0	3	7.467629e+05	1	77544.0	3	3	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2352	2754.0	28.0	0.0	C6	2.0	62462.0	2019-06-08	NaT	1.0	1	3.018905e+05	1	62462.0	1	2	0	1
2358	2761.0	28.0	0.0	C20	2.0	131805.0	2020-07-17	NaT	3.0	3	5.035733e+05	1	131805.0	3	3	0	1
2374	2781.0	25.0	0.0	C23	2.0	46952.0	2020-02-17	NaT	2.0	2	7.848518e+05	1	46952.0	2	4	0	1
2376	2784.0	33.0	0.0	C24	0.0	82815.0	2015-10-15	NaT	2.0	3	9.062008e+05	3	82815.0	3	4	0	1
2380	2788.0	29.0	0.0	C27	2.0	70254.0	2020-06-08	NaT	2.0	2	3.283200e+05	1	70254.0	2	2	0	1

358 rows × 17 columns

- We can see that if Quaterly Rating Increase --> it has be denoted as "1" in Quaterly Rating Column

Grade

```
[ ] def func1(x):
      if x["Grade_x"] >= x["Grade_y"] :
        return 0
      else:
         return 1
```

```
[ ] data1["Grade"] = data1.apply(func1,axis=1)
```

```
[ ] data1[data1["Grade"]== 1]
```

	Driver_ID	Age	Gender	City	Education_Level	Income_x	Dateofjoining	LastWorkingDate	Joining Designation	Grade_x	Total Business Value	Quarterly Rating_x	Income_y	Grade_y	Quarterly Rating_y	Income	Quarterly Rating	Grade
18	26.0	41.0	0.0	C14	2.0	121529.0	2018-05-07	NaT	1.0	3	2.911162e+06	4	132577.0	4	2	1	0	1
40	54.0	33.0	0.0	C29	1.0	117993.0	2019-07-11	NaT	4.0	4	1.879072e+06	2	127826.0	5	1	1	0	1
46	60.0	46.0	1.0	C20	0.0	82126.0	2016-09-17	NaT	1.0	3	2.051063e+06	4	89592.0	4	2	1	0	1
80	98.0	24.0	0.0	C16	0.0	57977.0	2019-08-15	2020-12-25	2.0	2	1.259732e+06	3	63774.0	3	2	1	0	1
230	275.0	39.0	0.0	C20	0.0	89124.0	2016-05-02	NaT	1.0	3	1.404075e+06	3	97226.0	4	3	1	0	1
256	307.0	37.0	0.0	C26	0.0	80856.0	2018-10-05	2020-10-26	2.0	3	2.583608e+06	4	88207.0	4	1	1	0	1
267	320.0	27.0	1.0	C20	0.0	56813.0	2018-07-13	NaT	1.0	1	1.256403e+06	4	63126.0	2	4	1	0	1
312	368.0	43.0	0.0	C23	1.0	46719.0	2018-09-18	NaT	1.0	1	1.286220e+06	4	51911.0	2	4	1	0	1

- We can see that if Grade Increase --> it has be denoted as "1" in Grade Column

Churn [Target Feature]

```
[ ] data1["Churn"] = data1["LastWorkingDate"].notnull()
```

```
[ ] data1[data1["Churn"]== True]
```

	Driver_ID	Age	Gender	City	Education_Level	Income_x	Dateofjoining	LastWorkingDate	Joining Designation	Grade_x	Total Business Value	Quarterly Rating_x	Income_y	Grade_y	Quarterly Rating_y	Income	Quarterly Rating	Grade	Churn
0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-03-11	1.0	1	571860.000000	2	57387.0	1	2	0	0	0	True
2	4.0	43.0	0.0	C13	2.0	65603.0	2019-12-07	2020-04-27	2.0	2	70000.000000	1	65603.0	2	1	0	0	0	True
3	5.0	29.0	0.0	C9	0.0	46368.0	2019-01-09	2019-03-07	1.0	1	40120.000000	1	46368.0	1	1	0	0	0	True
5	8.0	34.0	0.0	C2	0.0	70656.0	2020-09-19	2020-11-15	3.0	3	0.000000	1	70656.0	3	1	0	0	0	True
7	12.0	35.0	0.0	C23	2.0	28116.0	2019-06-29	2019-12-21	1.0	1	434530.000000	4	28116.0	1	1	0	0	0	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

```
[ ] data1['Churn'] = data1['Churn'].map({True: 1,False: 0})
```

```
[ ] data1.Churn.value_counts()
```

```
1    1616
0     765
Name: Churn, dtype: int64
```

```
[ ] data.Driver_ID.nunique()
```

```
2381
```

- We have created out Target Feature for all Driver ID's

As we have created our New Features, lets drop all irrelevant Columns.

```
[ ] data1.drop(["Income_x","Income_y","Quarterly Rating_x","Quarterly Rating_y","Grade_x","Grade_y","Driver_ID",
                "LastWorkingDate","Dateofjoining"],inplace = True,axis=1)
```

```
[ ] data1.shape
```

```
(2381, 10)
```

```
[ ] data1.head()
```

	Age	Gender	City	Education_Level	Joining Designation	Total Business Value	Income	Quarterly Rating	Grade	Churn
0	28.0	0.0	C23	2.0	1.0	571860.0	0	0	0	1
1	31.0	0.0	C7	2.0	2.0	0.0	0	0	0	0
2	43.0	0.0	C13	2.0	2.0	70000.0	0	0	0	1
3	29.0	0.0	C9	0.0	1.0	40120.0	0	0	0	1
4	31.0	1.0	C11	1.0	3.0	253000.0	0	1	0	0

Feature Conversion to Categorical

```
[ ] cat = ["Age","Gender","City","Education_Level","Joining Designation","Income","Quarterly Rating","Grade","Churn"]
```

```
[ ] for i in cat:
      data1[i].astype("category")
```

```
[ ] data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2380
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    2381 non-null   float64
 1   Gender                 2381 non-null   float64
 2   City                   2381 non-null   object
 3   Education_Level        2381 non-null   float64
 4   Joining Designation    2381 non-null   float64
 5   Total Business Value   2381 non-null   float64
 6   Income                 2381 non-null   int64
 7   Quarterly Rating       2381 non-null   int64
 8   Grade                  2381 non-null   int64
 9   Churn                  2381 non-null   int64
dtypes: float64(5), int64(4), object(1)
memory usage: 204.6+ KB
```

Univariate & Bivariate Analysis

Churn

```
[ ]  plt.pie(x = data1["Churn"].value_counts().reset_index()["Churn"],
             labels = data1["Churn"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("Churn feature")
```

```
Text(0.5, 1.0, 'Churn feature')
```

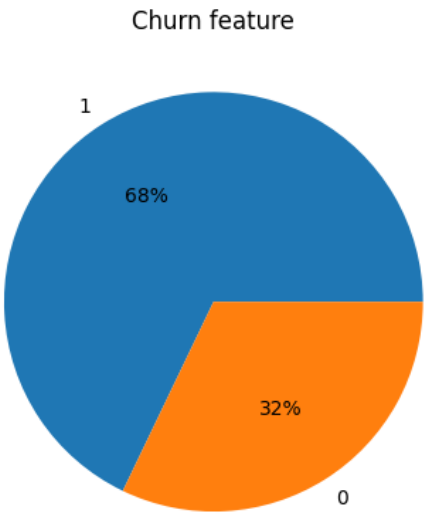


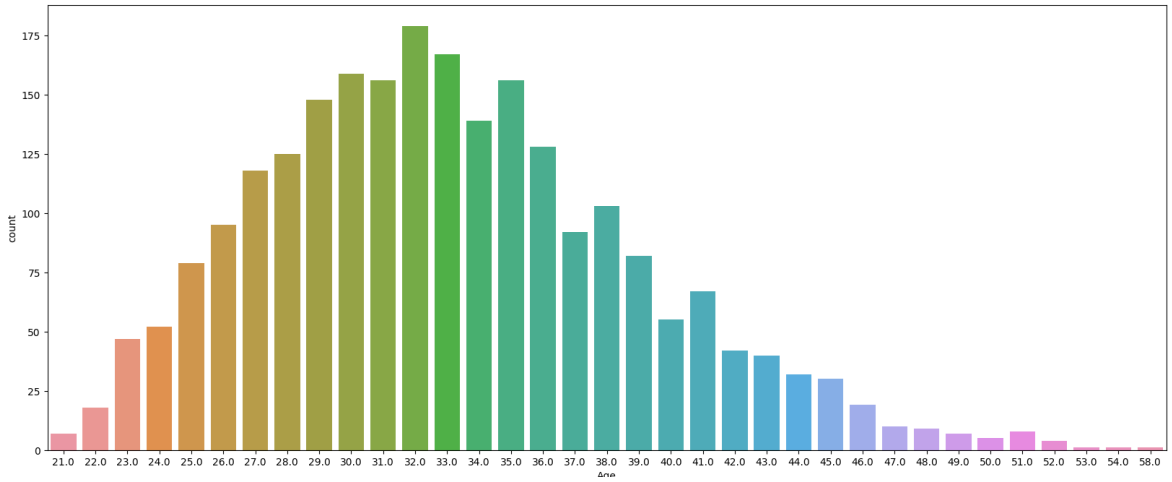
Figure 19 : Univariate Analysis of Target Variable

- In data Shared 68% of the Drivers have attritioned only 32% have retained
- Huge Imbalance in the Data, we have to address imbalance before model training.

Age

```
[ ]  plt.figure(figsize = (20,8))
     sns.countplot(x=data1["Age"])
```

```
<Axes: xlabel='Age', ylabel='count'>
```



- 50% of driver have age range between 29~37

- Min age of Driver is 21 Years
- Max age of Driver is 58 Years
- Lets bin the Age ranges to Reduce the No of Categories in Age

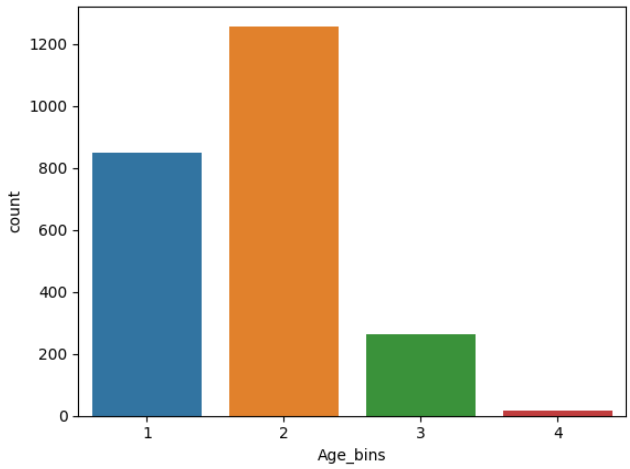
```
[ ] data1["Age_bins"] = pd.cut(data1["Age"], bins = [20,30,40,50,60],
                               labels=(1,2,3,4))
```

```
[ ] data1["Age_bins"].value_counts()
```

```
2    1257
1     848
3     261
4      15
Name: Age_bins, dtype: int64
```

```
[ ] sns.countplot(x=data1["Age_bins"])
```

```
<Axes: xlabel='Age_bins', ylabel='count'>
```




```
[ ] pd.crosstab(data1["Age_bins"],data1["Churn"],normalize = "index")
```

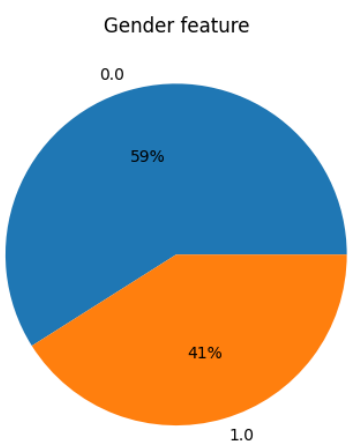
Churn	0	1
Age_bins		
1	0.298349	0.701651
2	0.340493	0.659507
3	0.306513	0.693487
4	0.266667	0.733333

Table 8 : Bivariate Analysis using crosstab

- Majority of Driver lie in the Age range 20~30
- Attrition almost same in all Age range...comparatively slightly high in 50~60 range

Gender

```
[ ]  plt.pie(x = data1["Gender"].value_counts().reset_index()["Gender"],
             labels = data1["Gender"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("Gender feature")
     plt.show()
```




```
[ ]  pd.crosstab(data1["Gender"],data1["Churn"],normalize = "index")
```

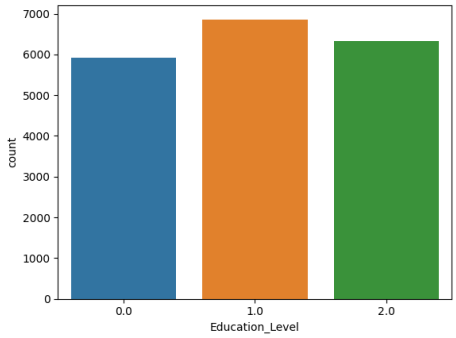
Churn	0	1
Gender		
0.0	0.324786	0.675214
1.0	0.316274	0.683726

- 59% of Drivers are Male, 41% are Female.
- Attrition rate is almost same irrespective of Gender.

Education_level

```
[ ]  sns.countplot(x= data["Education_Level"])
```

```
<Axes: xlabel='Education_Level', ylabel='count'>
```




```
[ ]  pd.crosstab(data1["Education_Level"],data1["Churn"],normalize = "index")
```

Churn	0	1
Education_Level		
0.0	0.308673	0.691327
1.0	0.337107	0.662893
2.0	0.317955	0.682045

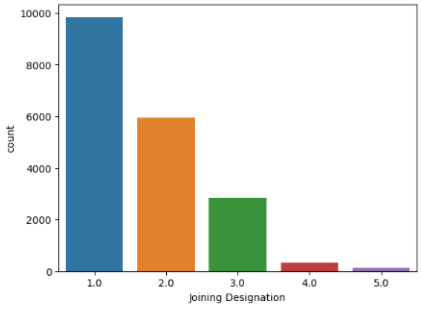
- Driver with Education 12+ are comparatively high.

- Attrition rate is almost same w.r.t Education level.

Joining Destination

```
[ ] sns.countplot(x= data["Joining Designation"])
```

```
<Axes: xlabel='Joining Designation', ylabel='count'>
```




```
[ ] pd.crosstab(data1["Joining Designation"],data1["Churn"],normalize = "index")
```

Churn	0	1
Joining Designation		
1.0	0.267057	0.732943
2.0	0.312883	0.687117
3.0	0.444219	0.555781
4.0	0.388889	0.611111
5.0	0.272727	0.727273

- Majority of Driver are having Joining Grade as "1"
- Attrition rate is highest with Driver having Joining Grade as "1"
- Attrition rate is least with Driver having Joining Grade as "3

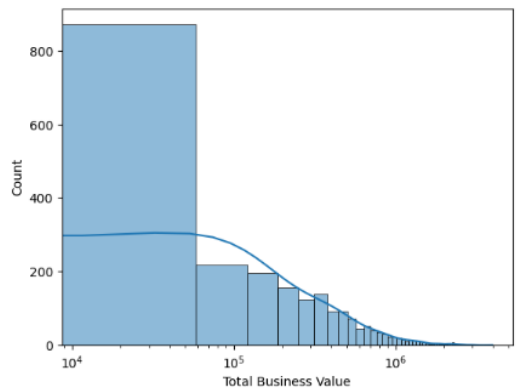
Total Business Value

- Total Bussines Value is Average amount of Value brought by the driver to the Company

```
data1["Total Business Value"].describe()
```

```
count    2.381000e+03
mean     3.120854e+05
std      4.495705e+05
min     -1.979329e+05
25%      0.000000e+00
50%      1.506244e+05
75%      4.294988e+05
max      3.972128e+06
Name: Total Business Value, dtype: float64
```

```
sns.histplot(data1["Total Business Value"],kde = True )
plt.xscale("log")
```



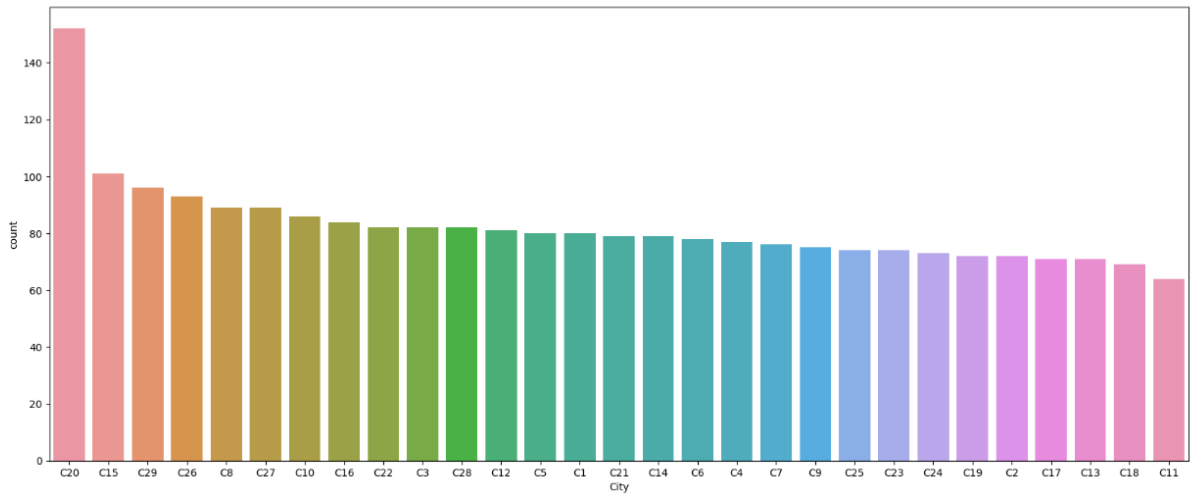
- Average Total Business values brought by the Driver range from -197,932 ~ 3,972,128

- 25% of Driver have Negative Average value to Company
- 50% of Driver have Average Values to Company < 429,498

Income

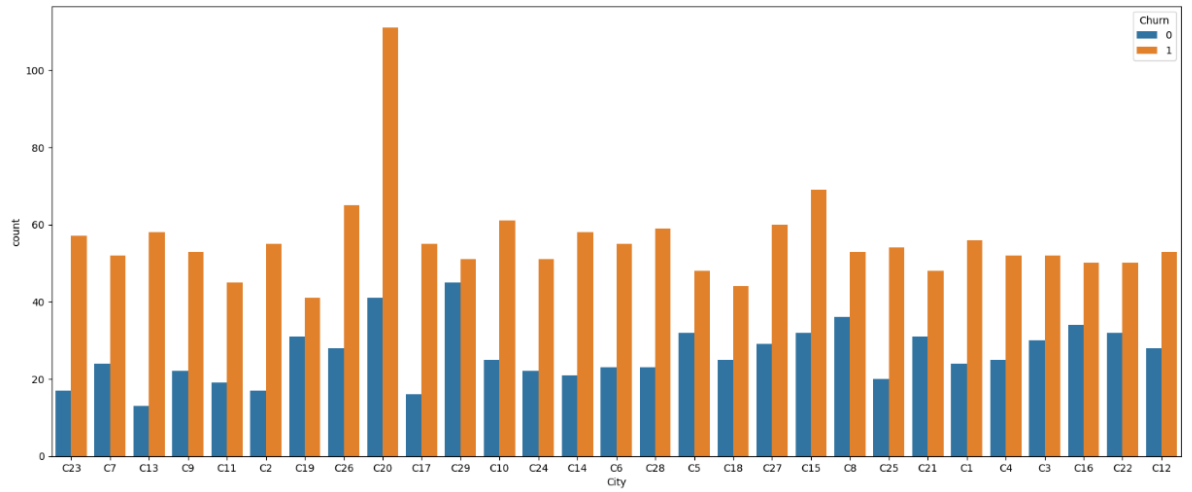
```
[ ] plt.figure(figsize = (20,8))
    sns.countplot(data=data1, x= "City" ,order = data1["City"].value_counts().index)
```

```
<Axes: xlabel='City', ylabel='count'>
```




```
[ ] plt.figure(figsize = (20,8))
    sns.countplot(data=data1, hue="Churn", x= "City")
```

```
<Axes: xlabel='City', ylabel='count'>
```




```
[ ] pd.crosstab(data1["City"],data1["Churn"],normalize = "index").reset_index().sort_values(1,ascending = False)[:10]
```

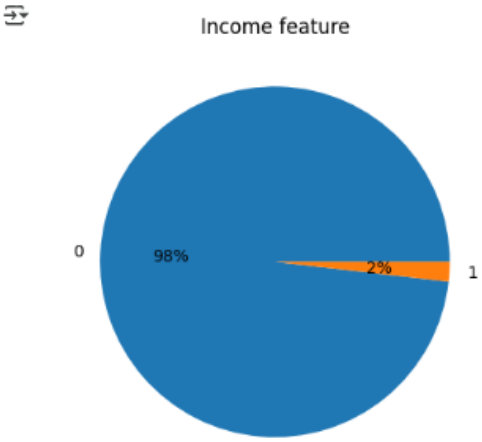
	Churn	City	0	1
4		C13	0.183099	0.816901
8		C17	0.225352	0.774648
15		C23	0.229730	0.770270
11		C2	0.236111	0.763889
5		C14	0.265823	0.734177
12		C20	0.269737	0.730263
17		C25	0.270270	0.729730
20		C28	0.280488	0.719512
1		C10	0.290698	0.709302
28		C9	0.293333	0.706667

- Most no of Drivers are from city "C20"

- Least no of Driver from "C11"
- Driver Attrition rate is highest in the City "C13"

Income

```
[ ]  plt.pie(x = data1["Income"].value_counts().reset_index()["Income"],
             labels = data1["Income"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("Income feature")
     plt.show()
```



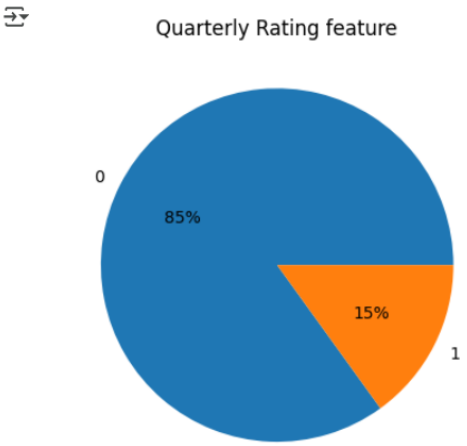

```
[ ]  pd.crosstab(data1["Income"],data1["Churn"],normalize = "index")
```

Churn	0	1
Income		
0	0.310094	0.689906
1	0.930233	0.069767

- only 2 % of Driver had their Income Growth
- Very low 6.9% Attrition rate observed with Driver of Increase Income
- 68.9% Attrition rate observed in Driver with No Income Growth

Quarterly Rating

```
[ ]  plt.pie(x = data1["Quarterly Rating"].value_counts().reset_index()["Quarterly Rating"],
             labels = data1["Quarterly Rating"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("Quarterly Rating feature")
     plt.show()
```



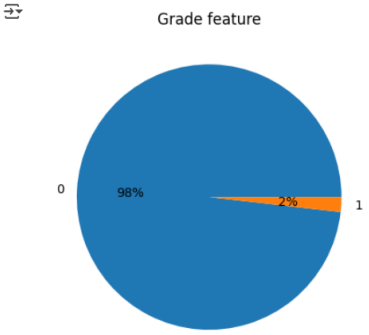

```
[ ]  pd.crosstab(data1["Quarterly Rating"],data1["Churn"],normalize = "index")
```

Churn	0	1
Quarterly Rating		
0	0.24172	0.75828
1	0.77095	0.22905

- Only 15 % of Driver had their Quarterly Rating Increase
- 23% Attrition rate observed with Driver Quarterly Rating has Increase.
- whereas 77% Attrition rate observed in Driver with Quarterly Rating Increase

Grade

```
[ ]  plt.pie(x = data1["Grade"].value_counts().reset_index()["Grade"],
             labels = data1["Grade"].value_counts().reset_index()["index"],
             autopct='%.0f%%')
     plt.title("Grade feature")
     plt.show()
```



```
[ ] pd.crosstab(data1["Grade"],data1["Churn"],normalize = "index")
```

Grade	0	1
0	0.310094	0.689906
1	0.930233	0.069767

- Only 2 % of Driver had their Grade Growth
- Very low 6.9% Attrition rate observed with Driver of Increase in Grade
- 68.9% Attrition rate observed in Driver with No Grade Growth

Feature Engineering 2

Eliminate Redundant Features

We can drop features - "Age", "Total Business Value " as we have already created similar feature using pd.cut

```
[ ] data1.drop(["Age","Total Business Value"], inplace = True, axis=1)
```

Encoding : Non-Numerical to Numerical

- we will follow below encoding Strategy for rest of categorical features
- if Feature wise unique Value[n]
    - n<=2 --> Label Encoder
    - 3<= n <=5 --> One Hot Encoding
    - n >5 -->Target Encoding

In Feature "City", we have 29 Categories, so we will use Target Encoder

```
[ ] data1.City.nunique()
```

```
29
```

```
[ ] te=TargetEncoder()
    data1["City"]=te.fit_transform(data1["City"],data1["Churn"])
```

Scaling of Numerical Categories

We will do Normalization here as data does not have Normal Distribution

```
[ ] Normscaler = MinMaxScaler()
```

```
[ ] scaleddata = Normscaler.fit_transform(data1)
```

```
[ ] scaleddata
```

```
array([[0.        , 0.83772787, 1.        , ..., 1.        , 0.        ,
        0.75      ],
       [0.        , 0.53686301, 1.        , ..., 0.        , 0.33333333,
        0.        ],
       [0.        , 1.        , 1.        , ..., 1.        , 0.66666667,
        0.25      ],
       ...,
       [0.        , 0.13598406, 0.        , ..., 1.        , 0.66666667,
        0.5       ],
       [1.        , 0.6986695 , 1.        , ..., 1.        , 0.        ,
        0.5       ],
       [0.        , 0.50164366, 1.        , ..., 0.        , 0.        ,
        0.5       ]])
```

```
[ ] scaleddata = pd.DataFrame(scaleddata,columns = data1.columns)
```

```
[ ] scaleddata
```

	Gender	City	Education_Level	Joining Designation	Income	Quarterly Rating	Grade	Churn	Age_bins	TBV_bins
0	0.0	0.837728	1.0	0.00	0.0	0.0	0.0	1.0	0.000000	0.75
1	0.0	0.536863	1.0	0.25	0.0	0.0	0.0	0.0	0.333333	0.00
2	0.0	1.000000	1.0	0.25	0.0	0.0	0.0	1.0	0.666667	0.25
3	0.0	0.615400	0.0	0.00	0.0	0.0	0.0	1.0	0.000000	0.25
4	1.0	0.602321	0.5	0.50	0.0	1.0	0.0	0.0	0.333333	0.50
...	...	...	...	...	...	...	...	...	...	...
2376	0.0	0.587228	0.0	0.25	0.0	1.0	0.0	0.0	0.333333	0.75
2377	1.0	0.615400	0.0	0.00	0.0	0.0	0.0	1.0	0.333333	0.00
2378	0.0	0.135984	0.0	0.25	0.0	0.0	0.0	1.0	0.666667	0.50
2379	1.0	0.698669	1.0	0.00	0.0	0.0	0.0	1.0	0.000000	0.50
2380	0.0	0.501644	1.0	0.25	0.0	1.0	0.0	0.0	0.000000	0.50

2381 rows × 10 columns

Table 9 : Scaled Data for Model Building

Correlation Matrix:

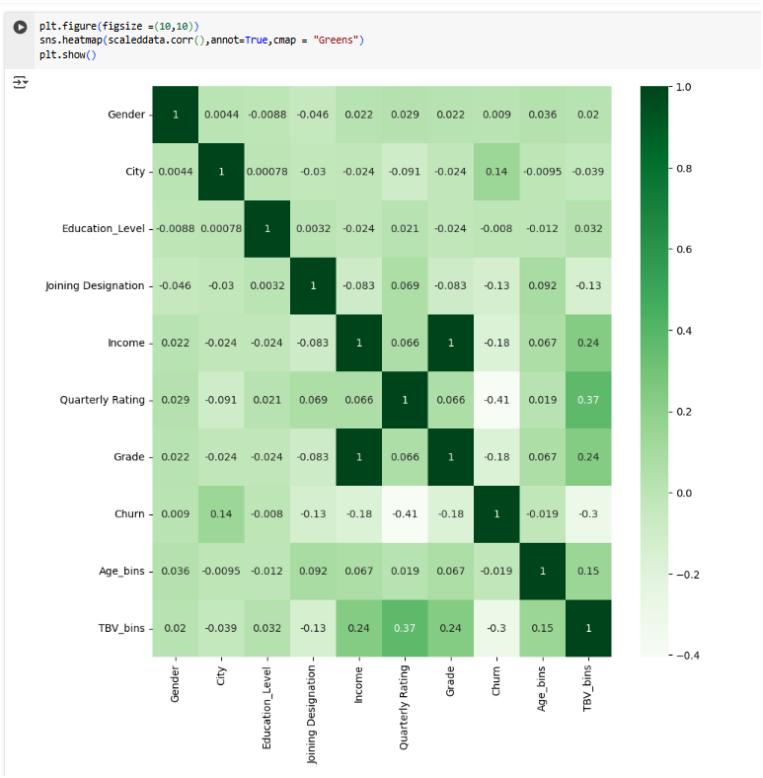


Figure 20 : Correlation Matrix of all Features for Logistics Regression

- Income and Grade Feature feature are Perfectly Correlated
- Basically these are Flag --> if "1" in Income feature means Driver Income increases similarily if "1" in Grade Feature means Driver Grade Increase
- So we will drop Grade Feature

```
[ ] scaleddata.drop(["Grade"], inplace = True, axis=1)
```

```
scaleddata.head()
```

	Gender	City	Education_Level	Joining Designation	Income	Quarterly Rating	Churn	Age_bins	TBV_bins
0	0.0	0.837728	1.0	0.00	0.0	0.0	1.0	0.000000	0.75
1	0.0	0.536863	1.0	0.25	0.0	0.0	0.0	0.333333	0.00
2	0.0	1.000000	1.0	0.25	0.0	0.0	1.0	0.666667	0.25
3	0.0	0.615400	0.0	0.00	0.0	0.0	1.0	0.000000	0.25
4	1.0	0.602321	0.5	0.50	0.0	1.0	0.0	0.333333	0.50

Data Prep for Modelling

Final data - Dividing Data for Training & Test

- Our desired Outcome is "Churn"
- So we will divide our scaleddata into X,y
- we will use 80:20 ratio for train & test
- Further we will Divide X & y as below data sets
    - Xtrain
    - Xtest
    - ytrain
    - ytest

```
[ ]  Xtrain, Xtest, ytrain, ytest = train_test_split(scaleddata.drop(["Churn"], axis = 1), scaleddata["Churn"], test_size=0.2, random_state=32)
```

```
[ ]  Xtrain.shape, Xtest.shape, ytrain.shape, ytest.shape
```

```
((1904, 8), (477, 8), (1904,), (477,))
```

Data Balancing

```
ytrain.value_counts()
```

```
1.0    1286
0.0     618
Name: Churn, dtype: int64
```

```
[ ]  1286/618
```

```
2.0809061488673137
```

Table 10 : Data Imbalance

- Data is Highly Imbalance, More data points are available for Attrition Driver
- So we will use SMOTE to Balance Data

```
[ ]  from imblearn.over_sampling import SMOTE
     print('Before SMOTE')
     print(ytrain.value_counts())

     Xtrainsm, ytrainsm = SMOTE().fit_resample(Xtrain,ytrain)
     print('After Oversampling')
     print(ytrainsm.value_counts())
```

```
Before SMOTE
1.0    1286
0.0     618
Name: Churn, dtype: int64
After Oversampling
1.0    1286
0.0    1286
Name: Churn, dtype: int64
```

Now the Data is Balanced

Model Training - Random Forest [Ensemble - Bagging]

Random Forest is an ensemble learning method used for classification and regression tasks. It builds multiple decision trees during training and merges their results to improve accuracy and control overfitting. Here are the key points:

- Ensemble Method: Combines the predictions of several base estimators (decision trees) to enhance the model's performance.
- Bootstrap Aggregation (Bagging): Each tree is trained on a random subset of the data, which helps in reducing variance and improving generalization.
- Feature Randomness: At each split in the tree, a random subset of features is considered, which helps in creating diverse trees and reducing correlation among them.

One of the Important things required for good Model is **Hyperparameter Tuning**

In Random Forest Some of the important Hyperparameters are:

- 'n_estimators'
- 'max_depth'
- 'max_features'
- 'bootstrap'

But Directly We cannot do Hyperparameter in Such n Dimensional Space, it will take very long time

So First we will do Random Search Multiple time and them we will do Grid Search

Random Search

```
[ ]  params = {
        'n_estimators' : [100,300,500],
        'max_depth' : [3,5,10],
        'max_features' : [3,5,8],
        'bootstrap' : [True,False]
        }
```

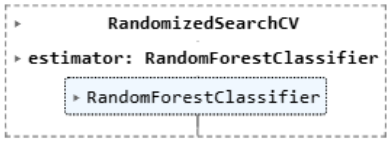
```
[ ]  from sklearn.model_selection import RandomizedSearchCV
     from sklearn.ensemble import RandomForestClassifier


     random = RandomizedSearchCV(estimator = RandomForestClassifier(random_state=7),
                                 param_distributions = params,
                                 scoring = 'recall',
                                 cv = 3,
                                 n_iter=15,
                                 n_jobs=-1
                                 )
```

Figure 21:Random Search Hyperparameter Tuning

```
[ ]  # 1st Random search
     random.fit(Xtrainsm, ytrainsm)

     print("Best param: ", random.best_params_)
     print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 100, 'max_features': 5, 'max_depth': 3, 'bootstrap': False}
Best score:  0.8802874176705954
```

```
[ ]  # 2nd Random search
     random.fit(Xtrainsm, ytrainsm)
```




```
[ ]  print("Best param: ", random.best_params_)
     print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 500, 'max_features': 3, 'max_depth': 5, 'bootstrap': False}
Best score:  0.8825948195107074
```

```
[ ]  # 3rd Random Search
     random.fit(Xtrainsm, ytrainsm)

     print("Best param: ", random.best_params_)
     print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 300, 'max_features': 5, 'max_depth': 3, 'bootstrap': False}
Best score:  0.8826184200015975
```

From above 3 Random Search, we can finalize Small Space in 4-dimensional Space as below

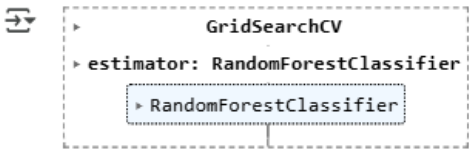
```
[ ]  params1 = {
          'n_estimators' : [100,300,500],
          'max_depth' : [3,5,6],
          'max_features' : [3,5,6],
          'bootstrap' : [False]
          }
```

Now we will do Grid Search

Grid Search

```
[ ]  from sklearn.model_selection import GridSearchCV
     grid = GridSearchCV(estimator = RandomForestClassifier(random_state=7),
                         param_grid = params1,
                         scoring = 'recall',
                         cv = 3,
                         n_jobs=-1,
                         )
```

```
[ ]  grid.fit(Xtrainsm, ytrainsm)
```




```
[ ]  print("Best params: ", grid.best_params_)
     print("Best score: ", grid.best_score_)
```

```
Best params:  {'bootstrap': False, 'max_depth': 3, 'max_features': 5, 'n_estimators': 300}
Best score:  0.8826184200015975
```

Figure 22 : Grid Search Hyperparameter Tuning

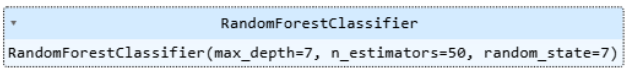
From Random and Grid Search we were able to fix out Hyper parameters

- max_depth = 3
- max_features = 5
- n_estimators = 300
- bootstrap = False

Model Training and Evaluation

```
[ ]  from sklearn.ensemble import RandomForestClassifier

     rf_clf = RandomForestClassifier(random_state=7,max_features = 5,max_depth=3, n_estimators=300, bootstrap=false,criterion='gini')
```

```
[ ]  rf_clf.fit(Xtrainsm,ytrainsm)
```



Accuracy

```
[ ] rf_clf.score(Xtrainsm,ytrainsm)
```

```
0.8114307931570762
```

```
[ ] ypred = rf_clf.predict(Xtest)
```

```
[ ] rf_clf.score(Xtest,ytest)
```

```
0.7945492662473794
```

- 79% Predictions by our model are accurate.

Precision

```
[ ] precision_score(ytest, ypred)
```

```
0.8411764705882353
```

- 84% of all Positive prediction are actually positve

Recall

```
[ ] recall_score(ytest, ypred)
```

```
0.8666666666666667
```

- Our Model predicts only 87% of all actual Positives as postives rest 13% of actual Positives aredetected as Negative[False Negatives]

F1 Score

```
[ ] f1_score(ytest,ypred)
```

```
0.853731343283582
```

- From F1_score of 0.85, we can say Our model has almost good balance between reducing False Positives [Detect as Churner but not] and False Negative [Actually Churner but detect as not Churner]
- However, there is need to Further improve F1_Score

Confusion Matrix

```
[ ]  conf_matrix = confusion_matrix(ytest, ypred)
     conf_matrix
```

```
array([[ 93,  54],
       [ 44, 286]])
```

```
[ ]  fig, ax = plt.subplots(figsize=(5,5))
     ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d1499934d90>
```

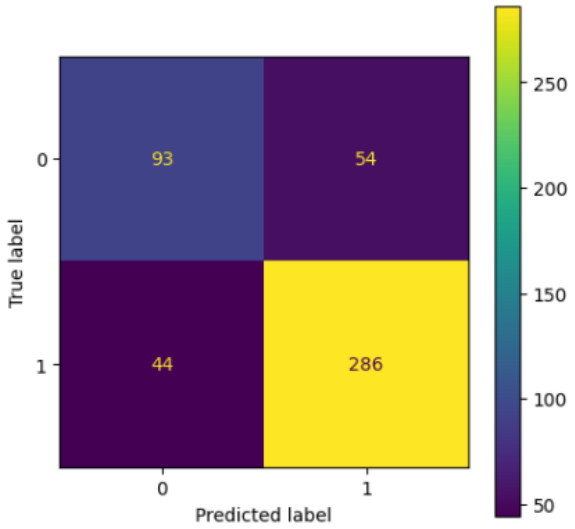
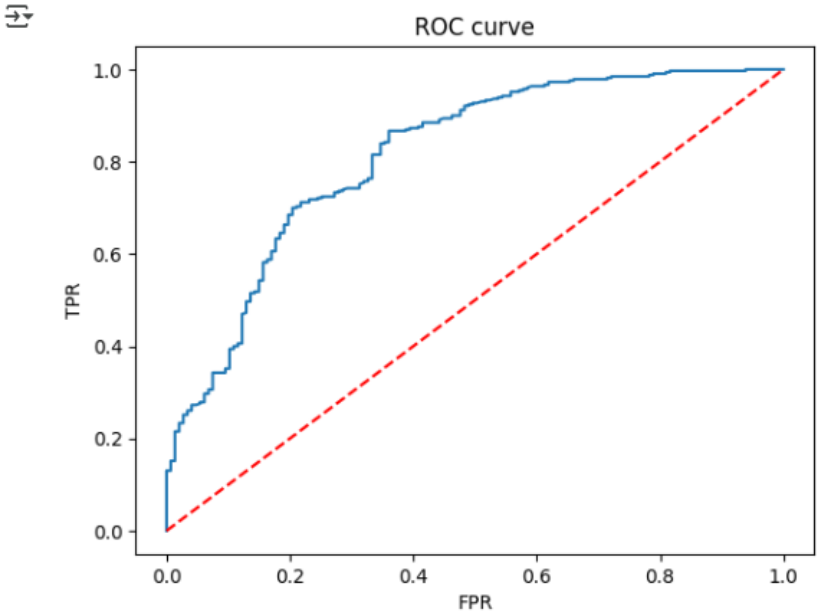


Figure 23 : Confusion matrix

- As Seen Earlier from High Recall and Better Precision values, same is reflected in Confusion Matrix
- How ever our model is detecting FP & FN at almost same
- But as per the business problem our Focus is to identify all attrition driver as acquiring new drivers is more expensive than retaining existing ones --> So we want to reduce False Negtives
- So we will focus on Improving Recall at moderate expense of Precision by changing the threshold value for Classification

ROC

```
[ ]  probability = rf_clf.predict_proba(Xtest)
     probabilites = probability[:,1]
     fpr, tpr, thr = roc_curve(ytest,probabilites)

     plt.plot(fpr,tpr)

     #random model for reference
     plt.plot(fpr,fpr,'--',color='red' )
     plt.title('ROC curve')
     plt.xlabel('FPR')
     plt.ylabel('TPR')
     plt.show()
```




```
[ ]  roc_auc_score(ytest,probabilites)
```

```
0.8134611420325706
```

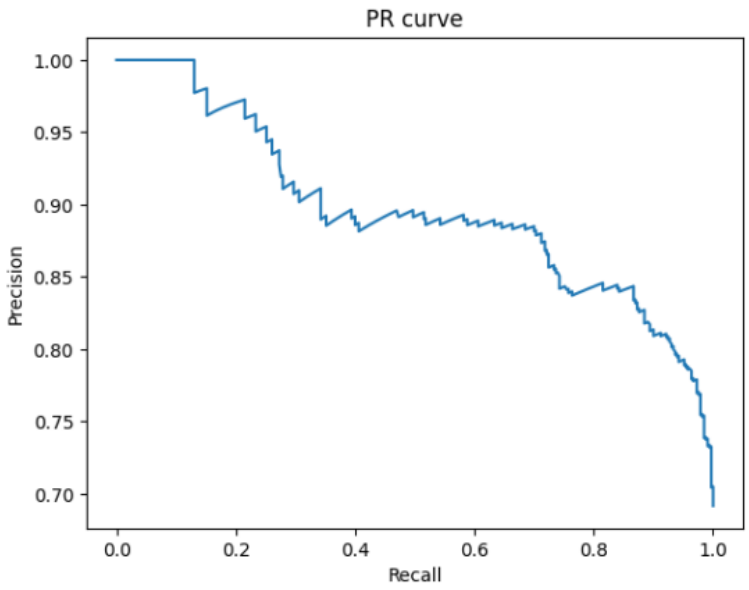
Figure 24 : Receiver Operating Curve

- roc_auc_score is 0.81 meaning our model can 81% efficiently Distinguish Positive and Negative Classes

Precision Recall Curve

```
[ ]  precision, recall, thr = precision_recall_curve(ytest, probabilites)
```

```
[ ]  plt.plot(recall, precision)

     plt.xlabel('Recall')
     plt.ylabel('Precision')
     plt.title('PR curve')
     plt.show()
```




```
[ ]  auc(recall, precision)
```

```
0.8957804551237599
```

Figure 25 : Precision Recall Curve

- 0.895 AUC under PR curve represents that our model has high Presicion and Recall values

Classification Report

```
[ ]  print(classification_report(ytest, ypred))
```

	precision	recall	f1-score	support
0.0	0.68	0.63	0.65	147
1.0	0.84	0.87	0.85	330
accuracy			0.79	477
macro avg	0.76	0.75	0.75	477
weighted avg	0.79	0.79	0.79	477

Table 11 : Classification Report

Optimizing Classification Threshold for Business Objective [Recall]

- As per the business problem, acquiring new drivers is more expensive than retaining existing ones
- So our focus should to predict TP and Reduce FN, so that we can correctly identify prospective Attrition Drivers
- Now let us plot Recall and Precision Variation w.r.t Different Classification Threshold values
- So that we can select best threshold for our Business Problem

```
[ ]  def precision_recall_curve_plot(y_test, pred_proba_c1):
       precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

       threshold_boundary = thresholds.shape[0]
       # plot precision
       plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--',
                label="Precision")
       # plot recall
       plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

       start, end = plt.xlim()
       plt.xticks(np.round(np.arange(start, end, 0.1), 2))

       plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall Value')
       plt.legend(); plt.grid()
       plt.show()

     precision_recall_curve_plot(ytest, probabilites)
```

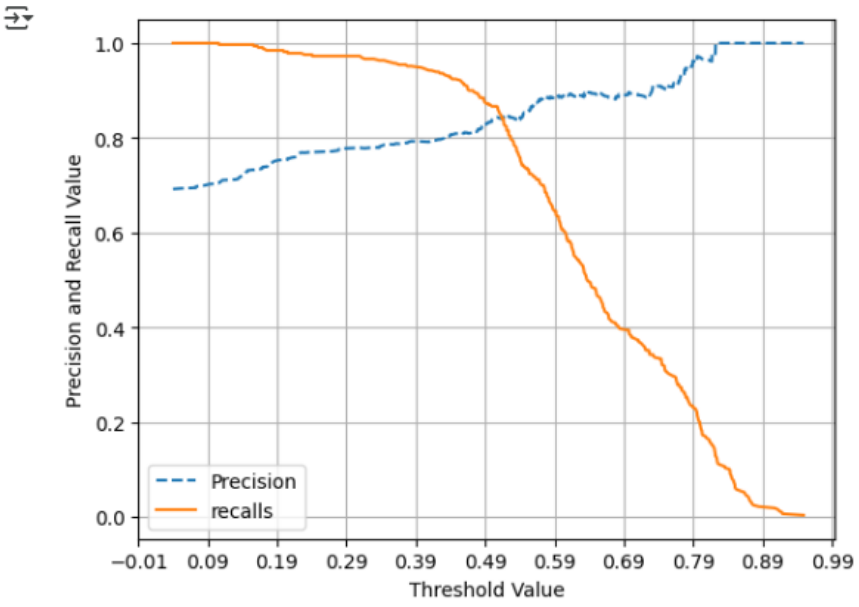


Figure 26 : Threshold Tuning for Business Objective

- So in order to increase Recall of our Model, we can reduce the Threshold from Default 0.5 --> 0.43 to get better Recall result at little degrading precision result[which is ok as per company assessment]
- No we will check the Model performance on test data with New threshold

```
[ ] ypredthres_043 = (rf_clf.predict_proba(Xtest)[:, 1] >= 0.43).astype(int)
```

```
[ ] recall_score(ytest, ypredthres_043)
```

```
0.9333333333333333
```

```
[ ] precision_score(ytest, ypredthres_043)
```

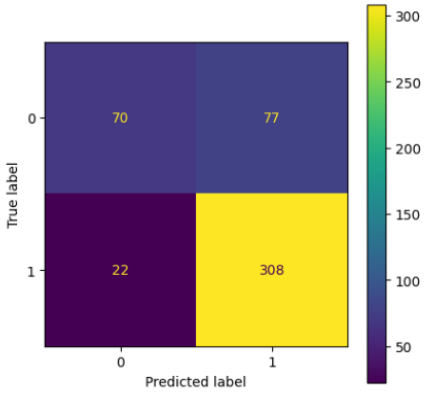
```
0.8
```

```
[ ] f1_score(ytest,ypredthres_043)
```

```
0.8615384615384616
```

```
[ ] conf_matrix = confusion_matrix(ytest, ypredthres_043)
    fig, ax = plt.subplots(figsize=(5,5))
    ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d149a06f820>
```



- After Threshold Change from 0.5 --> 0.43 ,our Random Forest Model can detect attrition driver with
- - Precision: 84% --> 80%
- - **Recall : 87% --> 93%**
- - Accuracy: No change 79%
- - F1-Score : 85% --> 86%
- - No of False Neagtive Reduced from 42 --> 22 [Nearly 100% Improvment]

Model Training - Gradient Boosting DT [Ensemble - Boosting]

Hyperparameter Tuning

```
params = {
    'n_estimators' : [100,300,500],
    'max_depth' : [3,5,10],
    'max_features' : [3,5,8],
    'learning_rate' : [0.01,0.05,0.1,0.5]
    }
```

```
[ ] from sklearn.model_selection import RandomizedSearchCV
    from sklearn.ensemble import GradientBoostingClassifier


    random = RandomizedSearchCV(estimator = GradientBoostingClassifier(random_state=7),
                                param_distributions = params,
                                scoring = 'recall',
                                cv = 3,
                                n_iter=15,
                                n_jobs=-1
                                )
```

```
[ ] # 1st Random search
    random.fit(Xtrainsm, ytrainsm)

    print("Best param: ", random.best_params_)
    print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 300, 'max_features': 8, 'max_depth': 3, 'learning_rate': 0.1}
Best score:  0.8646602618565234
```

```
[ ] # 2nd Random search
    random.fit(Xtrainsm, ytrainsm)

    print("Best param: ", random.best_params_)
    print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 100, 'max_features': 3, 'max_depth': 3, 'learning_rate': 0.5}
Best score:  0.8428879012991163
```

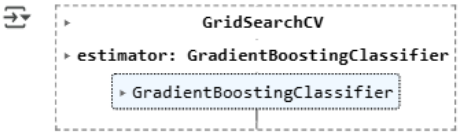
```
[ ] # 3rd Random Search
    random.fit(Xtrainsm, ytrainsm)

    print("Best param: ", random.best_params_)
    print("Best score: ", random.best_score_)
```

```
Best param:  {'n_estimators': 100, 'max_features': 5, 'max_depth': 3, 'learning_rate': 0.01}
Best score:  0.8725101482110827
```

```
[ ] params1 = {
        'n_estimators' : [100,200,300],
        'max_depth' : [3],
        'max_features' : [5,6,8],
        'learning_rate' : [0.01,0.025,0.05,0.1]
        }
```

```
[ ] from sklearn.model_selection import GridSearchCV
    grid = GridSearchCV(estimator = GradientBoostingClassifier(random_state=7),
                        param_grid = params1,
                        scoring = 'recall',
                        cv = 3,
                        n_jobs=-1,
                        )
```

```
[ ] grid.fit(Xtrainsm, ytrainsm)
```




```
[ ] print("Best params: ", grid.best_params_)
    print("Best score: ", grid.best_score_)
```

```
Best params:  {'learning_rate': 0.01, 'max_depth': 3, 'max_features': 5, 'n_estimators': 100}
Best score:  0.8725101482110827
```

From Random and Grid Search we were able to fix out Hyper parameters for GBDT as below

- max_depth = 3
- max_features = 5
- n_estimators = 100
- learning rate = 0.01

Model Training and Evaluation

```
[ ] from sklearn.ensemble import GradientBoostingClassifier
    gb_clf = GradientBoostingClassifier(n_estimators=100,max_depth=3,max_features = 5,learning_rate=0.01,random_state=7)
    gb_clf.fit(Xtrainsm,ytrainsm)
    ypred = gb_clf.predict(Xtest)
```

Accuracy

```
[ ] gb_clf.score(Xtrainsm,ytrainsm)
```

```
0.9284603421461898
```

```
[ ] ypred = gb_clf.predict(Xtest)
```

```
[ ] gb_clf.score(Xtest,ytest)
```

```
0.7651991614255765
```

- 76% Predictions by our model are accurate

Precision

```
[ ]  precision_score(ytest, ypred)
```

```
0.807909604519774
```

- 80% of all Positive prediction are actually positve

Recall

```
[ ]  recall_score(ytest, ypred)
```

```
0.8666666666666667
```

- Our Model predicts only 87% of all actual Positives as postives remaining 13% as Negative[Which are False Negatives]

F1 Score

```
[ ]  f1_score(ytest,ypred)
```

```
0.8362573099415205
```

- From F1_score of 0.83, we can say Our model has almost good balance between reducing False Positives[Detect as attrition driver but not] and False Negative[Actually attrition driver but detect as not attrition]
- However there is need to Further improve F1_Score

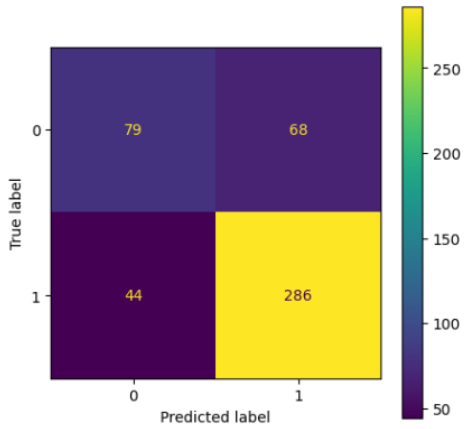
Confusion matrix

```
[ ]  conf_matrix = confusion_matrix(ytest, ypred)
     conf_matrix
```

```
array([[ 79,  68],
       [ 44, 286]])
```

```
[ ]  fig, ax = plt.subplots(figsize=(5,5))
     ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
```

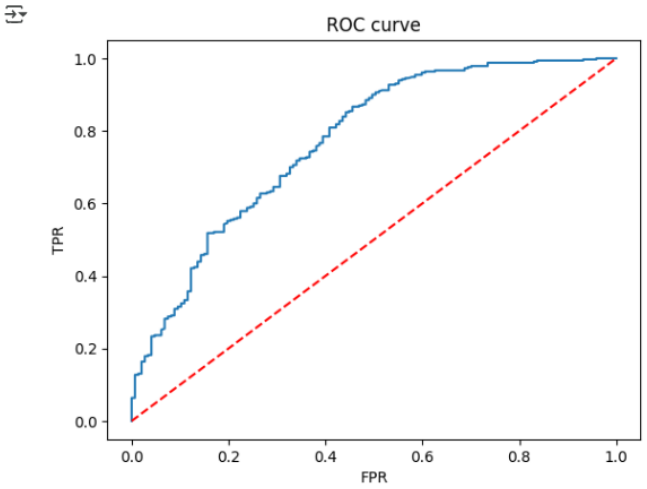
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d149e09c9d0>
```



- As Seen Earlier from High Recall and Better Precision values, same is reflected in Confusion Matrix
- How ever our model is detecting FP & FN at almost 10% & 6% , But as per the business problem our Focus is to identify all attrition driver as acquiring new drivers is more expensive than retaining existing ones
- So we will focus on Improving Recall at moderate expense of Precision

ROC

```
[ ]  probability = gb_clf.predict_proba(Xtest)
     probabilites = probability[:,1]
     fpr, tpr, thr = roc_curve(ytest,probabilites)

     plt.plot(fpr,tpr)

     #random model for reference
     plt.plot(fpr,fpr,'--',color='red' )
     plt.title('ROC curve')
     plt.xlabel('FPR')
     plt.ylabel('TPR')
     plt.show()
```




```
[ ]  roc_auc_score(ytest,probabilites)
```

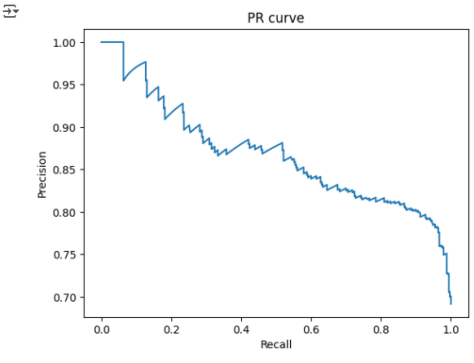
```
0.7706761492475778
```

roc_auc_score is 0.77 meaning our model can 77% efficiently Distinguish Positve and Negative Classes

Precision Recall Curve

```
[ ]  precision, recall, thr = precision_recall_curve(ytest, probabilites)
```

```
[ ]  plt.plot(recall, precision)

     plt.xlabel('Recall')
     plt.ylabel('Precision')
     plt.title('PR curve')
     plt.show()
```



```
[ ]  auc(recall, precision)

0.8678787549538943
```

- 0.867 AUC under PR curve represents that our model has high Presicion and Recall values

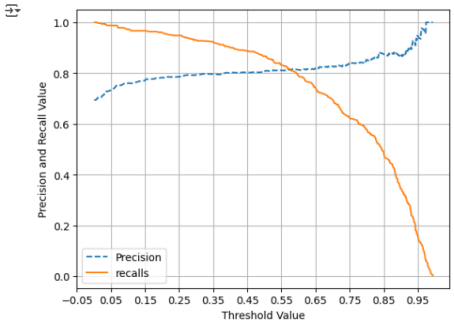
Classification Report

```
[ ]  print(classification_report(ytest, ypred))

              precision    recall  f1-score   support

         0.0       0.64      0.54      0.59       147
         1.0       0.81      0.87      0.84       330

    accuracy                           0.77       477
   macro avg       0.73      0.70      0.71       477
weighted avg       0.76      0.77      0.76       477
```

Optimizing Classification Threshold for Business Objective [Recall]

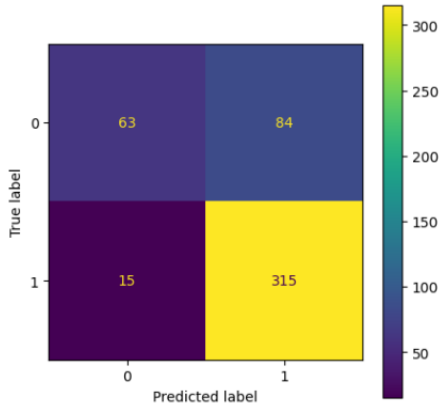
- As per the business problem, acquiring new drivers is more expensive than retaining existing ones
- So our focus should to predict TP and Reduce FN, so that we can correctly identify prospective Attrition Drivers
- Now let us plot Recall and Precision Variation w.r.t Different Classification Threshold values
- So that we can select best threshold for our Business Problem

```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--',
             label="Precision")
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall Value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(ytest, probabilites)
```



- So in order to increase Recall of our Model, we can reduce the Threshold from Default 0.5 --> 0.37 to get better Recall result at little degrading precision result [which is ok as per company assessment]
- Now we will check the Model performance on test data with New threshold

```
[ ] ypredthres_037 = (rf_clf.predict_proba(Xtest)[:, 1] >= 0.37).astype(int)

    recall_score(ytest, ypredthres_037)

    0.9545454545454546

[ ] precision_score(ytest, ypredthres_037)

    0.7894736842105263

[ ] f1_score(ytest,ypredthres_037)

    0.8641975308641975

[ ] conf_matrix = confusion_matrix(ytest, ypredthres_037)
    fig, ax = plt.subplots(figsize=(5,5))
    ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d149a1ae500>
```



After Threshold Change from 0.5 --> 0.37 ,our GBDT Model can detect attrition driver with

- Precision: 81% --> 78%
- Recall : 87% --> 95%
- Accuracy: No change 77%
- F1-Score : 84% --> 86%
- False Negative Reduced from 44 --> 15 [approx. 150% Improvement]

Business Insights

Bagging:

- From Hyperparameter tuning through Random and Grid Search we were able to fix out Hyper parameters
  - max_depth = 3
  - max_features = 5
  - n_estimators = 300
  - bootstrap = False
- 79% Predictions by our model are accurate
- 84% of all Positive prediction are actually positve
- Our Model predicts only 87% of all actual Positives as postives rest 13% of actual Positives aredetected as Negative[False Negatives]

- From F1_score of 0.85, we can say Our model has almost good balance between reducing False Positives[Detect as Churner but not] and False Negative[Actually Churner but detect as not Churner]
- However there is need to Further improve F1_Score
- How ever our model is detecting FP & FN at almost same
- roc_auc_score is 0.81 meaning our model can 81% efficiently Distinguish Positve and Neagative Classes
- 0.895 AUC under PR curve represents that our model has high Presicion and Recall values
- As per the bussiness problem, acquiring new drivers is more expensive than retaining existing ones
- So our focus should to predict TP and Reduce FN, so that we can correctly idenitfy prospective Attrition Drivers
- so, From Classification Threshold Versus Presicion& Recall Curve, we Checked and changed Classification threshold from 0.5 --> 0.39 for Better Results
- After Threshold Change from 0.5 --> 0.43 ,our Random Forest Model can detect attrition driver with
    - Precision: 84% --> 80%
    - Recall : 87% --> 93%
    - Accuracy: No change 79%
    - F1-Score : 85% --> 86%
- No of False Negative Reduced from 42 --> 22 [Nearly 100% Improvement]

Boosting:

- From Hyperparameter tuning through Random and Grid Search we were able to fix out Hyper parameters for GBDT as below
    - max_depth = 3
    - max_features = 5
    - n_estimators = 100
    - learning rate = 0.01
- 76% Predictions by our model are accurate
- 80% of all Positive prediction are actually positve
- Our Model predicts only 87% of all actual Positives as postives remaining 13% as Negative[Which are False Negatives]
- From F1_score of 0.83, we can say Our model has almost good balance between reducing False Positives[Detect as attrition driver but not] and False Negative[Actually attrition driver but detect as not attrition]
- However there is need to Further improve F1_Score
- As Seen Earlier from High Recall and Better Precision values, same is reflected in Confusion Matrix
- roc_auc_score is 0.77 meaning our model can 77% efficiently Distinguish Positve and Negative Classe
- 0.867 AUC under PR curve represents that our model has high Presicion and Recall value
- How ever our model is detecting FP & FN at almost 10% & 6% , But as per the business problem our Focus is to identify all attrition driver as acquiring new drivers is more expensive than retaining existing one
- so, From Classification Threshold Versus Presicion& Recall Curve, we Checked and changed Classification threshold from 0.5 --> 0.39 for Better Results
- After Threshold Change from 0.5 --> 0.37 ,our GBDT Model can detect attrition driver with
    - Precision: 81% --> 78%
    - Recall : 87% --> 95%
    - Accuracy: No change 77%
    - F1-Score : 84% --> 86%
- False Negative Reduced from 44 --> 15 [approx 150% Improvement]

Recommendations

- More Information could be inputed into Model for better prediction like below
    - Average No of Hours working per Day/month
    - General working hours
    - Customer Feeback Score for the Driver
    - Regularity Discipline Parameters in working
- Further Kfold technique can be used along with Random Forest and GBDT Model as no of data point are very less
- Intensive Random & Grid Search can be performed for more Hyper Parameter tuning

Chapter 5 : Business Case Study 5

Problem Description

About Company

A subscription video on demand and over-the-top streaming service. It was launched in India with content in 12 languages.

Problem:

- Create a Recommender System to show personalized movie recommendations based on ratings given by a user and other users like them to improve user experience.

Dataset:

- MMMM-YY : Reporting Date (Monthly)
- Driver_ID : Unique id for drivers
- Age : Age of the driver
- Gender : Gender of the driver – Male : 0, Female: 1
- City : City Code of the driver
- Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
- Income : Monthly average Income of the driver
- Date Of Joining : Joining date for the driver
- LastWorkingDate : Last date of working for the driver
- Joining Designation : Designation of the driver at the time of joining
- Grade : Grade of the driver at the time of reporting
- Total Business Value : The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
- Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

Libraries Used:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime

import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter("ignore")

import re

!pip install category_encoders
from category_encoders import TargetEncoder
```

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,
StandardScaler, MinMaxScaler

from scipy.stats import pearsonr

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.neighbors import NearestNeighbors

!pip install cmfrec
from cmfrec import CMF

from sklearn.metrics import (
    mean_squared_error as mse,
    mean_absolute_error as mae,
    mean_absolute_percentage_error as mape)

from sklearn.model_selection import train_test_split

from sklearn.manifold import TSNE
!pip install umap-learn
!pip install umap-learn[plot]
import umap
import umap.plot
```

Business Questions to be Answered from Analysis

- leveraging user ratings and similarities among users to create a robust, personalized movie recommender system.
- Utilizing a comprehensive dataset of movie ratings, user demographics, and movie details, develop a system that can accurately predict user preferences and suggest movies accordingly.
- The insights gained from this system are expected to drive user engagement, increase satisfaction, and foster a more intuitive user experience.
- Visualizing the data with respect to different categories to get a better understanding of the underlying distribution.
- Creating a pivot table of movie titles & user id and imputing the NaN values with a suitable value
- Follow the Item-based approach and
- Pearson Correlation
    - Take a movie name as input from the user.
    - Recommend 5 similar movies based on Pearson Correlation
- Cosine Similarity
    - Print the item similarity matrix and user similarity matrix.
- Create a CSR matrix using the pivot table. Write a function to return top 5 recommendations for a given item.
- Use Matrix Factorization to create Movie and User Embedding to develop Recommendation System

Analysis

Data Load & Cleaning:

```
[ ] movies=pd.read_fwf("zee-movies.dat",encoding='ISO-8859-1')
```

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Movie ID::Title::Genres  3883 non-null   object
 1   Unnamed: 1              100 non-null    object
 2   Unnamed: 2              51 non-null     object
dtypes: object(3)
memory usage: 91.1+ KB
```

```
[ ] movies.shape
```

```
(3883, 3)
```

```
[ ] movies.head(5)
```

	Movie ID::Title::Genres	Unnamed: 1	Unnamed: 2
0	1::Toy Story (1995)::Animation|Children's|Comedy	NaN	NaN
1	2::Jumanji (1995)::Adventure|Children's|Fantasy	NaN	NaN
2	3::Grumpier Old Men (1995)::Comedy|Romance	NaN	NaN

```
[ ] ratings=pd.read_fwf("zee-ratings.dat",encoding='ISO-8859-1')
```

```
[ ] ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 1 columns):
 #   Column                              Non-Null Count    Dtype
---  ------                              --------------    -----
 0   UserID::MovieID::Rating::Timestamp  1000209 non-null  object
dtypes: object(1)
memory usage: 7.6+ MB
```

```
[ ] ratings.shape
```

```
(1000209, 1)
```

```
[ ] ratings.head(5)
```

	UserID::MovieID::Rating::Timestamp
0	1::1193::5::978300760
1	1::661::3::978302109
2	1::914::3::978301968
3	1::3408::4::978300275

```
[ ] users=pd.read_fwf("zee-users.dat",encoding='ISO-8859-1')
```

```
[ ] users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 1 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   UserID::Gender::Age::Occupation::Zip-code  6040 non-null   object
dtypes: object(1)
memory usage: 47.3+ KB
```

```
[ ] users.shape
```

```
(6040, 1)
```

```
[ ] users.head(5)
```

	UserID::Gender::Age::Occupation::Zip-code
0	1::F::1::10::48067
1	2::M::56::16::70072
2	3::M::25::15::55117
3	4::M::45::7::02460

Table 12 : Datasets for Recommender System

Let drop the Columns with Incompletely Filled Data

```
[ ] a= movies["Unnamed: 1"].isnull() & movies["Unnamed: 2"].isnull()
    a.sum()
```

```
3783
```

```
[ ] movies = movies[movies["Unnamed: 1"].isnull() & movies["Unnamed: 2"].isnull()]
```

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3783 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Movie ID::Title::Genres  3783 non-null   object
 1   Unnamed: 1              0 non-null      object
 2   Unnamed: 2              0 non-null      object
dtypes: object(3)
memory usage: 118.2+ KB
```

```
[ ] movies.drop(columns = ["Unnamed: 1","Unnamed: 2"], inplace=True)
```

```
movies.shape
```

```
(3783, 1)
```

```
movies.head(5)
```

	Movie ID::Title::Genres
0	1::Toy Story (1995)::Animation|Children's|Comedy
1	2::Jumanji (1995)::Adventure|Children's|Fantasy
2	3::Grumpier Old Men (1995)::Comedy|Romance
3	4::Waiting to Exhale (1995)::Comedy|Drama
4	5::Father of the Bride Part II (1995)::Comedy

Lets Extract Features from “Movies” File

```
movies["Movie ID"] = movies["Movie ID::Title::Genres"].str.split("::").transform(lambda x : x[0])
movies["Title"] = movies["Movie ID::Title::Genres"].str.split("::").transform(lambda x : x[1])
movies["Genres"] = movies["Movie ID::Title::Genres"].str.split("::").transform(lambda x : x[-1])
```

```
movies.head()
```

	Movie ID::Title::Genres	Movie ID	Title	Genres
0	1::Toy Story (1995)::Animation|Children's|Comedy	1	Toy Story (1995)	Animation|Children's|Comedy
1	2::Jumanji (1995)::Adventure|Children's|Fantasy	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3::Grumpier Old Men (1995)::Comedy|Romance	3	Grumpier Old Men (1995)	Comedy|Romance
3	4::Waiting to Exhale (1995)::Comedy|Drama	4	Waiting to Exhale (1995)	Comedy|Drama
4	5::Father of the Bride Part II (1995)::Comedy	5	Father of the Bride Part II (1995)	Comedy

```
movies["Genressplit"] = movies["Genres"].str.split("|")
```

```
movies = movies.explode("Genressplit")
```

```
movies.drop(columns= ["Movie ID::Title::Genres"], inplace = True)
```

```
movies.head()
```

	Movie ID	Title	Genres	Genressplit
0	1	Toy Story (1995)	Animation|Children's|Comedy	Animation
0	1	Toy Story (1995)	Animation|Children's|Comedy	Children's
0	1	Toy Story (1995)	Animation|Children's|Comedy	Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	Adventure
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	Children's

- Lets Extract Release Year Features also

```
movies["Releaseyear"] = movies["Title"].str.split("(").transform(lambda x : x[-1])
movies["Title"] = movies["Title"].str.split("(").transform(lambda x : x[0])
```

```
movies["Releaseyear"] = movies["Releaseyear"].str.split(")").transform(lambda x : x[0])
```

```
[ ] movies["Genres"]=movies["Genres"].astype("category")
    movies["Releaseyear"]=movies["Releaseyear"].astype("int")
```

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6174 entries, 0 to 3882
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   Movie ID     6174 non-null   object  
 1   Title        6174 non-null   object  
 2   Genres       6174 non-null   category
 3   Genressplit  6174 non-null   object  
 4   Releaseyear  6174 non-null   int64   
dtypes: category(1), int64(1), object(3)
memory usage: 263.6+ KB
```

```
movies.head()
```

	Movie ID	Title	Genres	Genressplit	Releaseyear
0	1	Toy Story	Animation|Children's|Comedy	Animation	1995
0	1	Toy Story	Animation|Children's|Comedy	Children's	1995
0	1	Toy Story	Animation|Children's|Comedy	Comedy	1995
1	2	Jumanji	Adventure|Children's|Fantasy	Adventure	1995

Table 13 : Feature Extraction from Single Column

Similarly, We can Extract Feature from Both Ratings and User, Now Let's See Final Datasets

```
[ ] users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   USerID      6040 non-null   object  
 1   Gender      6040 non-null   category
 2   Age         6040 non-null   category
 3   Occupation  6040 non-null   category
 4   Zipcode     6040 non-null   category
dtypes: category(4), object(1)
memory usage: 233.9+ KB
```

```
[ ] users.head()
```

	USerID	Gender	Age	Occupation	Zipcode
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

```
[ ] ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 9 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   USerID     1000209 non-null  object
 1   Movie ID   1000209 non-null  object
 2   Rating     1000209 non-null  int64
 3   Timestamp  1000209 non-null  int64
 4   hour       1000209 non-null  int64
 5   dayofweek  1000209 non-null  int64
 6   month      1000209 non-null  int64
 7   year       1000209 non-null  int64
 8   day        1000209 non-null  int64
dtypes: int64(7), object(2)
memory usage: 68.7+ MB
```

```
[ ] ratings.head()
```

	USerID	Movie ID	Rating	Timestamp	hour	dayofweek	month	year	day
0	1	1193	5	978300760	22	6	12	2000	31
1	1	661	3	978302109	22	6	12	2000	31
2	1	914	3	978301968	22	6	12	2000	31
3	1	3408	4	978300275	22	6	12	2000	31

Exploratory Data Analysis

Movies

```
[ ] for i in movies.columns:
      print("No of Unique {} are {}".format(i,movies[i].nunique()))
```

```
No of Unique Movie ID are 3783
No of Unique Title are 3733
No of Unique Genres are 296
No of Unique Genressplit are 25
No of Unique Releaseyear are 81
```

Some of the Genres are not completely spelled, we will correct them so that they do no behave as separate Genre

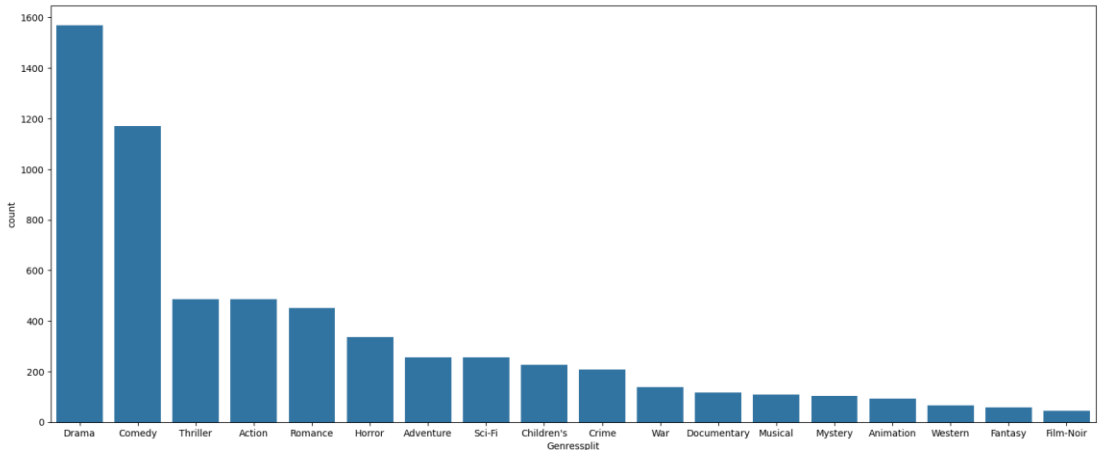
```
[ ] movies.loc[movies["Genressplit"] =="Horro","Genressplit"] = "Horror"
    movies.loc[movies["Genressplit"] =="Fantas","Genressplit"] = "Fantasy"
    movies.loc[movies["Genressplit"] =="Dram","Genressplit"] = "Drama"
    movies.loc[movies["Genressplit"] =="Wester","Genressplit"] = "Western"
    movies.loc[movies["Genressplit"] =="Sci-F","Genressplit"] = "Sci-Fi"
    movies.loc[movies["Genressplit"] =="Thrille","Genressplit"] = "Thriller"
    movies.loc[movies["Genressplit"] =="Wa","Genressplit"] = "War"
```

```
[ ] movies["Genressplit"].nunique()
```

```
18
```

```
[ ] plt.figure(figsize =(20,8))
    sns.countplot(data= movies, x="Genressplit", order = movies["Genressplit"].value_counts().reset_index()["index"])
```

```
<Axes: xlabel='Genressplit', ylabel='count'>
```




```
[ ] movies.Releaseyear.describe()
```

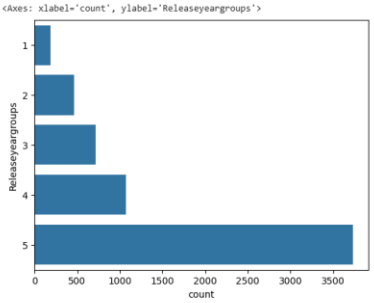
```
count    6174.000000
mean     1986.229997
std        16.547598
min      1919.000000
25%      1983.000000
50%      1994.000000
75%      1997.000000
max      2000.000000
Name: Releaseyear, dtype: float64
```

More than 75% of the movies have release year after 1983
lets Convert Release year into bins as Release year recency factor

```
[ ] bins = [1919,1940,1960,1980,1990,2000]
    movies["Releaseyeargroups"] = pd.cut(movies["Releaseyear"], bins, labels=[1,2,3,4,5])
```

```
[ ] sns.countplot( movies["Releaseyeargroups"])
```

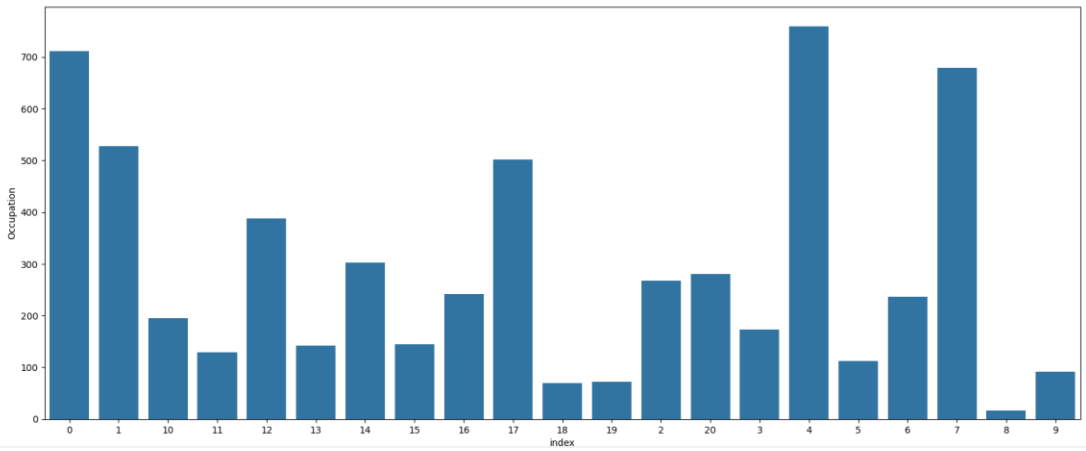
```
<Axes: xlabel='count', ylabel='Releaseyeargroups'>
```



Users

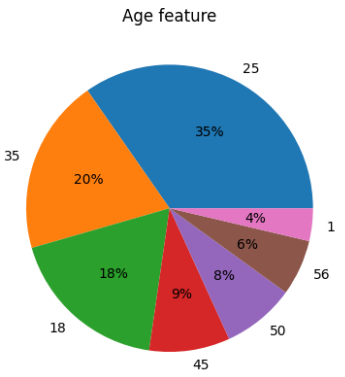
```
[ ] for i in users.columns:
      print("No of Unique {} are {}".format(i,users[i].nunique()))

No of Unique UserID are 6040
No of Unique Gender are 2
No of Unique Age are 7
No of Unique Occupation are 21
No of Unique Zipcode are 3439
```

```
[ ] plt.figure(figsize =(20,8))
    sns.barplot(data= users["Occupation"].value_counts().reset_index(), x= "index", y="Occupation")

<Axes: xlabel='index', ylabel='Occupation'>
```



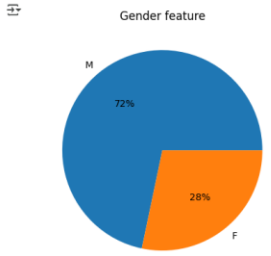
- Majority of the Users occupation
  - college/grad student
  - executive/managerial
  - academic/educator
- Minimum no of users are from below Occupations
  - Farmers
  - tradesman/craftsman
  - homemakers

```
[ ] plt.pie(x = users["Age"].value_counts().reset_index()["Age"],
            labels = users["Age"].value_counts().reset_index()["index"],
            autopct='%.0f%%')
    plt.title("Age feature")
    plt.show()
```



- 35% of the users are in the Age group "25 ~ 34"
- 20% of the users are in the Age group "35 ~ 44"
- only 4 % of the users are from age group "1~17"
- only 6% fo the users are from group >56

```
plt.pie(x = users["Gender"].value_counts().reset_index()["Gender"],
        labels = users["Gender"].value_counts().reset_index()["index"],
        autopct='%.0f%%')
plt.title("Gender feature")
plt.show()
```

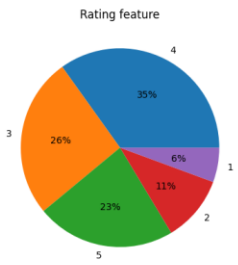


Ratings

```
[ ] for i in ratings.columns:
      print("No of Unique {} are {}".format(i,ratings[i].nunique()))
```

```
No of Unique UserID are 6040
No of Unique Movie ID are 3706
No of Unique Rating are 5
No of Unique Timestamp are 458455
No of Unique hour are 24
No of Unique dayofweek are 7
No of Unique month are 12
No of Unique year are 4
No of Unique day are 31
```

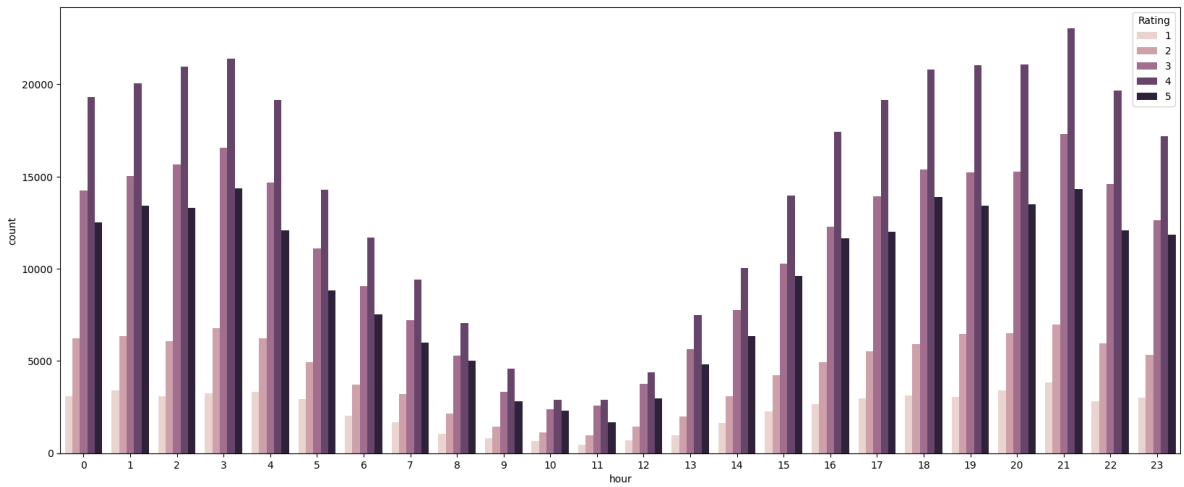
```
[ ] plt.pie(x = ratings["Rating"].value_counts().reset_index()["Rating"],
            labels = ratings["Rating"].value_counts().reset_index()["index"],
            autopct='%.0f%%')
    plt.title("Rating feature")
    plt.show()
```



- 35% of the ratings are given as 4
- only 23% of ratings are given as 5

```
[ ] plt.figure(figsize =(20,8))
    sns.countplot(data= ratings, x= "hour",hue = "Rating")
```

```
<Axes: xlabel='hour', ylabel='count'>
```



- Most of the movies are rated/watched in the Night and midnight
- Movies are least watched in morning and noon time

Creating Matrices for Recommendation Systems

Combine 3 Dataframes into 1

As we have three different data sets, we will use merge Functions to Combine them into one data set as below

```
[ ] data = ratings.merge(users[["USerID","Gender","Age","Occupation"]],on = "USerID", how = "right")

[ ] data = data.merge(movies[["Movie ID","Title","Genres","Releaseyeargroups","Releaseyear"]],on = "Movie ID", how = 'right')

[ ] data.shape

(1978767, 16)

[ ] data.head(10)
```

	USerID	Movie ID	Rating	Timestamp	hour	dayofweek	month	year	day	Gender	Age	Occupation	Title	Genres	Releaseyeargroups	Releaseyear
0	1	1	5.0	978824268.0	23.0	5.0	1.0	2001.0	6.0	F	1	10	Toy Story	Animation|Children's|Comedy	5	1995
1	6	1	4.0	978237008.0	4.0	6.0	12.0	2000.0	31.0	F	50	9	Toy Story	Animation|Children's|Comedy	5	1995
2	8	1	4.0	978233496.0	3.0	6.0	12.0	2000.0	31.0	M	25	12	Toy Story	Animation|Children's|Comedy	5	1995
3	9	1	5.0	978225952.0	1.0	6.0	12.0	2000.0	31.0	M	25	17	Toy Story	Animation|Children's|Comedy	5	1995
4	10	1	5.0	978226474.0	1.0	6.0	12.0	2000.0	31.0	F	35	1	Toy Story	Animation|Children's|Comedy	5	1995
5	18	1	4.0	978154768.0	5.0	5.0	12.0	2000.0	30.0	F	18	3	Toy Story	Animation|Children's|Comedy	5	1995
6	19	1	5.0	978555994.0	21.0	2.0	1.0	2001.0	3.0	M	1	10	Toy Story	Animation|Children's|Comedy	5	1995
7	21	1	3.0	978139347.0	1.0	5.0	12.0	2000.0	30.0	M	18	16	Toy Story	Animation|Children's|Comedy	5	1995
8	23	1	4.0	978463614.0	19.0	1.0	1.0	2001.0	2.0	M	35	0	Toy Story	Animation|Children's|Comedy	5	1995
9	26	1	3.0	978130703.0	22.0	4.0	12.0	2000.0	29.0	M	25	7	Toy Story	Animation|Children's|Comedy	5	1995

Table 14 : Combined Dataset from 3 Tables

Now we have all the required information in one Dataframe, from this Dataframe we can extract information for User-User and Item-Item Based Recommendation Approach

Due to Limitation of Hardware Capabilities, in this Use case we will take top 1000 watched Movies and 1000 Frequent users

```
[ ] select_movies = ratings['Movie ID'].value_counts()[:1000].index.to_list()

[ ] movies = movies.loc[movies["Movie ID"].isin(select_movies)]
    movies.shape

(2024, 6)

[ ] select_users = ratings['USerID'].value_counts()[:1000].index.to_list()

[ ] users = users.loc[users["USerID"].isin(select_users)]
    users.shape

(1000, 5)
```

Item-Item Similarity Matrix

Let's create a Matrix for Item [Movie]

```
item_item = movies[['Movie ID','Releaseyear']].drop_duplicates()
```

```
[ ] item_item.head()
```

	Movie ID	Releaseyear
0	1	1995
1	2	1995
2	3	1995
5	6	1995
6	7	1995

```
[ ] item_item.shape
```

```
(976, 2)
```

Adding Average Rating of an Item

```
[ ] item_item = item_item.merge(data.groupby('Movie ID').Rating.mean().reset_index(), on='Movie ID', how = "left")
```

```
[ ] item_item.head()
```

	Movie ID	Releaseyear	Rating
0	1	1995	4.146846
1	2	1995	3.201141
2	3	1995	3.016736
3	6	1995	3.878723
4	7	1995	3.410480

Adding Genre By doing sort of OHE

```
[ ] m = movies.pivot(index='Movie ID', columns='Genressplit', values='Title')
    m = ~m.isna()
    m = m.astype(int)
```

```
[ ] m.head()
```

Genressplit	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
Movie ID																		
1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1020	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1022	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1027	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

```
[ ] item_item = item_item.merge(m, on='Movie ID', how = "left")
```

Adding Gender wise Ratings for an item

```
[ ] item_item = item_item.merge(
        data.groupby(['Movie ID', 'Gender'])['Rating'].mean().
        reset_index().pivot(index='Movie ID', columns='Gender', values='Rating').
        reset_index(), on='Movie ID', how = "left")
```

```
[ ] item_item.head()
```

	Movie ID	Releaseyear	Rating	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	...	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	F	M
0	1	1995	4.146846	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	4.187817	4.130552
1	2	1995	3.201141	0	1	0	1	0	0	0	...	0	0	0	0	0	0	0	0	3.278409	3.175238
2	3	1995	3.016736	0	0	0	0	1	0	0	...	0	0	0	1	0	0	0	0	3.073529	2.994152
3	6	1995	3.878723	1	0	0	0	0	1	0	...	0	0	0	0	0	1	0	0	3.682171	3.909988
4	7	1995	3.410480	0	0	0	0	1	0	0	...	0	0	0	1	0	0	0	0	3.588235	3.267717

5 rows × 23 columns

Adding Age wise Ratings for an item

```
item_item = item_item.merge(
    data.groupby(['Movie ID', 'Age'])['Rating'].mean().
    reset_index().pivot(index='Movie ID', columns='Age', values='Rating').
    reset_index(), on='Movie ID', how = "left")
```

Adding Occupation wise Ratings for an item

```
item_item = item_item.merge(
    data.groupby(['Movie ID', 'Occupation'])['Rating'].mean().
    reset_index().pivot(index='Movie ID', columns='Occupation', values='Rating').
    reset_index(), on='Movie ID', how = "left")
```

```
item_item.shape
```

```
(976, 51)
```

Final item_item Matrix

```
item_item.shape
```

```
(976, 51)
```

```
item_item.columns
```

```
Index(['Movie ID', 'Releaseyear', 'Rating', 'Action', 'Adventure', 'Animation',
       'Children's', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
       'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
       'Thriller', 'War', 'Western', 'F', 'M', '1_x', '18_x', '25', '35', '45',
       '50', '56', '0', '1_y', '10', '11', '12', '13', '14', '15', '16', '17',
       '18_y', '19', '2', '20', '3', '4', '5', '6', '7', '8', '9'],
      dtype='object')
```

```
item_item =item_item.set_index("Movie ID")
```

```
item_item.fillna(0, inplace=True)
```

```
item_item.head(10)
```

Movie ID	Releaseyear	Rating	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	...	19	2	20	3	4	5	6	7	8	9
1	1995	4.146846	0	0	1	1	1	0	0	0	...	3.960000	4.000000	4.046296	4.492537	4.040404	4.025641	4.394737	4.189055	4.500000	4.028571
2	1995	3.201141	0	1	0	1	0	0	0	0	...	3.100000	2.970588	3.071429	3.285714	3.116505	3.400000	3.107143	3.115942	2.666667	3.500000
3	1995	3.016736	0	0	0	0	1	0	0	0	...	3.333333	2.777778	2.789474	2.909091	3.060606	2.750000	2.714286	3.045455	3.000000	2.875000
6	1995	3.878723	1	0	0	0	0	1	0	0	...	4.000000	3.853659	3.368421	3.625000	4.034965	3.909091	4.068966	3.923810	4.000000	4.500000

```
scaler = MinMaxScaler()
item_item_scaled = pd.DataFrame(scaler.fit_transform(item_item), columns=item_item.columns, index=item_item.index)
item_item_scaled.head()
```

Movie ID	Releaseyear	Rating	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	...	19	2	20	3	4	5	6	7	8	9
1	0.932432	0.861485	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	...	0.740000	0.779528	0.836279	0.927919	0.799284	0.805128	0.905427	0.835363	0.900000	0.805714
2	0.932432	0.540193	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.525000	0.467462	0.562667	0.553710	0.510963	0.680000	0.520807	0.474001	0.533333	0.700000
3	0.932432	0.477544	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.583333	0.409011	0.483532	0.436927	0.493519	0.550000	0.403455	0.450265	0.600000	0.575000
6	0.932432	0.770394	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.750000	0.735164	0.646023	0.658915	0.797586	0.781818	0.808115	0.746044	0.800000	0.900000
7	0.932432	0.611314	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.406250	0.539534	0.568135	0.527132	0.630640	0.714286	0.552812	0.496809	0.600000	0.850000

5 rows × 50 columns

Table 15 : Item-Item Matrix - Scaled

User-User Similarity Matrix

- Lets Create a Matric for User

```
users.columns
```

```
Index(['UserID', 'Gender', 'Age', 'Occupation', 'Zipcode'], dtype='object')
```

+ Code + Text

```
user_user = users[['UserID', 'Gender', 'Age', 'Occupation', 'Zipcode']].drop_duplicates()
```

```
user_user.head()
```

	UserID	Gender	Age	Occupation	Zipcode
9	10	F	35	1	95370
17	18	F	18	3	95825
21	22	M	18	15	53706
22	23	M	35	0	90049
25	26	M	25	7	23112

Adding Average Rating of a User

+ Code + Text

```
user_user = user_user.merge(data.groupby('UserID').Rating.mean().reset_index(), on='UserID', how = "left")
```

Adding count of Movies rated by a user

```
[ ] user_user = user_user.merge(data.groupby('UserID').Rating.count().reset_index(), on='UserID', how = "left")
```

```
[ ] user_user.columns
```

```
Index(['UserID', 'Gender', 'Age', 'Occupation', 'Zipcode', 'Rating_x',
       'Rating_y'],
      dtype='object')
```

```
[ ] user_user.rename(columns = {'Rating_x':'Ave.rating','Rating_y':'movie_count'}, inplace = True)
```

```
[ ] user_user.columns
```

```
Index(['UserID', 'Gender', 'Age', 'Occupation', 'Zipcode', 'Ave.rating',
       'movie_count'],
      dtype='object')
```

Adding Average Hour, weekday, day, month for rating for a user

```
[ ] data.columns
```

```
Index(['UserID', 'Movie ID', 'Rating', 'Timestamp', 'hour', 'dayofweek',
       'month', 'year', 'day', 'Gender', 'Age', 'Occupation', 'Title',
       'Genres', 'Releaseyeargroups', 'Releaseyear'],
      dtype='object')
```

```
[ ] user_user = user_user.merge(data.groupby('UserID').hour.mean().reset_index(), on='UserID', how = "left")
    user_user = user_user.merge(data.groupby('UserID').dayofweek.mean().reset_index(), on='UserID', how = "left")
    user_user = user_user.merge(data.groupby('UserID').day.mean().reset_index(), on='UserID', how = "left")
    user_user = user_user.merge(data.groupby('UserID').month.mean().reset_index(), on='UserID', how = "left")
```

Label Encoding Gender

```
le=LabelEncoder()
user_user["Gender"]=le.fit_transform(user_user["Gender"])
user_user["Gender"].value_counts()
```

```
1    769
0    231
Name: Gender, dtype: int64
```

Target Encoding Occupation and ZipCode

```
[ ] te=TargetEncoder()
    user_user["Occupation"]=te.fit_transform(user_user["Occupation"],user_user["Ave.rating"])
    user_user["Zipcode"]=te.fit_transform(user_user["Zipcode"],user_user["Ave.rating"])
```

```
[ ] user_user.head()
```

	UserID	Gender	Age	Occupation	Zipcode	Ave.rating	movie_count	hour	dayofweek	day	month
0	10	0	35	3.428073	3.570591	4.135338	798	7.181704	4.958647	25.969925	8.953634
1	18	0	18	3.555448	3.499044	3.585434	714	5.134454	5.000000	30.000000	12.000000
2	22	1	18	3.556099	3.462524	3.059486	622	3.824759	4.840836	28.225080	10.271704
3	23	1	35	3.447147	3.465551	3.328012	689	17.968070	1.203193	4.432511	1.568940
4	26	1	25	3.523273	3.423584	3.005457	733	5.570259	5.188267	29.991814	11.909959

Final user_user Matrix

```
[ ] user_user.shape
(1000, 11)

[ ] user_user.columns
Index(['UserID', 'Gender', 'Age', 'Occupation', 'Zipcode', 'Ave.rating',
       'movie_count', 'hour', 'dayofweek', 'day', 'month'],
      dtype='object')

[ ] user_user =user_user.set_index("UserID")

[ ] item_item.fillna(0, inplace=True)

[ ] scaler1 = MinMaxScaler()
    user_user_scaled = pd.DataFrame(scaler1.fit_transform(user_user), columns=user_user.columns, index=user_user.index)
    user_user_scaled.head()
```

UserID	Gender	Age	Occupation	Zipcode	Ave.rating	movie_count	hour	dayofweek	day	month
10	0.0	0.618182	0.000000	0.826929	0.827281	0.091762	0.311618	0.826441	0.832331	0.707952
18	0.0	0.309091	0.740129	0.600966	0.601777	0.067136	0.221629	0.833333	0.966667	1.000000
22	1.0	0.309091	0.743912	0.485626	0.386096	0.040164	0.164059	0.806806	0.907503	0.834313
23	1.0	0.618182	0.110832	0.495187	0.496213	0.059807	0.785749	0.200532	0.114417	0.000000
26	1.0	0.436364	0.553172	0.362644	0.363940	0.072706	0.240785	0.864711	0.966394	0.991368

Table 16 : User-User Matrix

User-Item Interaction Matrix

```
ratings.columns
Index(['UserID', 'Movie ID', 'Rating', 'Timestamp', 'hour', 'dayofweek',
       'month', 'year', 'day'],
      dtype='object')

[ ] user_item = ratings.pivot(index = 'UserID',columns ='Movie ID',values = 'Rating' )

[ ] user_item.fillna(0,inplace = True)

[ ] user_item.shape
(6040, 3706)

[ ] user_item.head()
```

Movie ID	1	10	100	1000	1002	1003	1004	1005	1006	1007	...	99	990	991	992	993	994	996	997	998	999
UserID																					
1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1001	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 3706 columns

Table 17 : User-Item Matrix

```
[ ] # Findind No of Ratings
    (user_item >0).sum().sum()
1000209

[ ] # Finding Sprsity in User_Item Interaction Matrix

    (user_item >0).sum().sum() / (user_item.shape[0]*user_item.shape[1])
0.044683625622312845
```

Pearson Correlation for Item Item Recommendation

In this Section, we will use Item-Item Matrix and Pearson Correlation to Recommend an Item based on Given Query Item

```
item_item_scaled.shape
```

```
(976, 50)
```

```
item_item_scaled.index
```

```
Index(['1', '2', '3', '6', '7', '10', '11', '16', '17', '19',
       ...
       '3863', '3868', '3869', '3893', '3897', '3911', '3916', '3927', '3948',
       '3952'],
      dtype='object', name='Movie ID', length=976)
```

- Let’s Create a DataFrame for a Query and All Candidates with Pearson Correlation

```
ranks = []

for query in item_item_scaled.index:
    for candidate in item_item_scaled.index:
        if candidate == query:
            continue
        ranks.append([query, candidate, pearsonr(item_item_scaled.loc[query], item_item_scaled.loc[candidate])[0]])


Pearson_Item_df = pd.DataFrame(ranks, columns=['query', 'candidate', 'Pearson'])
Pearson_Item_df.head()
```

	query	candidate	Pearson
0	1	2	0.604590
1	1	3	0.664959
2	1	6	0.559148
3	1	7	0.719536
4	1	10	0.482636

```
Pearson_Item_df = Pearson_Item_df.merge(movies[['Movie ID', 'Title']], left_on='query', right_on='Movie ID').rename(columns={'Title': 'query_title'}).drop(columns=['Movie ID'])
Pearson_Item_df = Pearson_Item_df.merge(movies[['Movie ID', 'Title']], left_on='candidate', right_on='Movie ID').rename(columns={'Title': 'Rec_candidate_title'}).drop(columns=['Movie ID'])
Pearson_Item_df = Pearson_Item_df.sort_values(by=['query', 'Pearson'], ascending = [True, False])
Pearson_Item_df.drop_duplicates(inplace=True)
Pearson_Item_df.head()
```

	query	candidate	Pearson	query_title	Rec_candidate_title
3442565	1	3114	0.991886	Toy Story	Toy Story 2
2668242	1	2355	0.991738	Toy Story	Bug's Life, A
3984409	1	3751	0.983111	Toy Story	Chicken Run
1311869	1	1223	0.923581	Toy Story	Grand Day Out, A

```
Pearson_Item_df.drop_duplicates(inplace=True)
```

```
Pearson_Item_df.shape
```

```
(951600, 5)
```

```
Pearson_Item_df = Pearson_Item_df.sort_values(by=['query', 'Pearson'], ascending = [True, False])
```

```
Pearson_Item_df.head()
```

	query	candidate	Pearson	query_title	Rec_candidate_title
3442565	1	3114	0.991886	Toy Story	Toy Story 2
2668242	1	2355	0.991738	Toy Story	Bug's Life, A
3984409	1	3751	0.983111	Toy Story	Chicken Run
1311869	1	1223	0.923581	Toy Story	Grand Day Out, A
1180488	1	1148	0.923447	Toy Story	Wrong Trousers, The

Function for Item-Item Recommendation using Pearson Correlation Coefficient

```
# This function work only if we precalculate all possible pearson correlation values for Item vs Item
# But this is efficient as we have to calcuate Pearson Correlation values only once
# and we can query through below function multiple time

def Item_Rec(data,Movie_ID,k):
  # Data is pearson Correlation Dataframe for all item vs item
  # Movie_ID --> Item for which we are finding simialar movies
  # k --> No of Recommenadions Required
  return (data.loc[data["query"] == Movie_ID])[:k]
```

```
Item_Rec(data = Pearson_Item_df ,Movie_ID = "1",k=5)
```

	query	candidate	Pearson	query_title	Rec_candidate_title
3442565	1	3114	0.991886	Toy Story	Toy Story 2
2668242	1	2355	0.991738	Toy Story	Bug's Life, A
3984409	1	3751	0.983111	Toy Story	Chicken Run
1311869	1	1223	0.923581	Toy Story	Grand Day Out, A
1180488	1	1148	0.923447	Toy Story	Wrong Trousers, The

```
[ ] movies[movies["Title"] == "Liar Liar "]["Movie ID"]

1455    1485
Name: Movie ID, dtype: object
```

```
[ ] Item_Rec(data = Pearson_Item_df ,Movie_ID = "1485",k=3)
```

	query	candidate	Pearson	query_title	Rec_candidate_title
492051	1485	441	0.985896	Liar Liar	Dazed and Confused
1753444	1485	1517	0.981324	Liar Liar	Austin Powers: International Man of Mystery
3117903	1485	2759	0.979604	Liar Liar	Dick

Cosine Similarity Matrix for item Recommendation

In this Section, we will use Item-Item Matrix and Cosine Similarity to Recommend an Item based on Given Query Item

```
[ ] Cosine_Item_Matrix= cosine_similarity(item_item_scaled)

[ ] Cosine_Item_Matrix.shape

(976, 976)

[ ] Cosine_Item_Matrix

array([[1.        , 0.87006646, 0.88116146, ..., 0.84282108, 0.90654986,
        0.82580961],
       [0.87006646, 1.        , 0.77969586, ..., 0.84228991, 0.82098621,
        0.77555389],
       [0.88116146, 0.77969586, 1.        , ..., 0.7742923 , 0.88098996,
        0.76826642],
       ...,
       [0.84282108, 0.84228991, 0.7742923 , ..., 1.        , 0.83453074,
        0.82651856],
       [0.90654986, 0.82098621, 0.88098996, ..., 0.83453074, 1.        ,
        0.88169918],
       [0.82580961, 0.77555389, 0.76826642, ..., 0.82651856, 0.88169918,
        1.        ]])

[ ] Cosine_Item_Mat_df = pd.DataFrame(Cosine_Item_Matrix, index = item_item_scaled.index, columns = item_item_scaled.index )
```

```
Cosine_Item_Mat_df
```

Movie ID	1	2	3	6	7	10	11	16	17	19	...	3863	3868	3869	3893	3897	3911	3916	3927	3948	3952
Movie ID																					
1	1.000000	0.870066	0.881161	0.863783	0.905579	0.836112	0.902030	0.856726	0.876303	0.844698	...	0.825611	0.951386	0.937018	0.894715	0.936804	0.953418	0.860025	0.842821	0.906550	0.825810
2	0.870066	1.000000	0.779696	0.805213	0.805253	0.857140	0.806551	0.816625	0.818021	0.734305	...	0.796189	0.848157	0.836826	0.793096	0.834635	0.849304	0.820522	0.842290	0.820986	0.775554
3	0.881161	0.779696	1.000000	0.813927	0.983843	0.790526	0.949329	0.796574	0.882326	0.880996	...	0.771802	0.924591	0.938131	0.877770	0.898160	0.922586	0.800100	0.774292	0.880990	0.768266
6	0.863783	0.805213	0.813927	1.000000	0.844156	0.936083	0.846084	0.898857	0.865581	0.756206	...	0.871590	0.894587	0.869301	0.892155	0.884865	0.897956	0.850961	0.824869	0.846936	0.862837
7	0.905579	0.805253	0.983843	0.844156	1.000000	0.820446	0.962502	0.836780	0.909549	0.872103	...	0.806187	0.948002	0.949328	0.906529	0.924366	0.944451	0.835908	0.802116	0.910420	0.794295
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3911	0.953418	0.849304	0.922586	0.897956	0.944451	0.867288	0.939398	0.889948	0.910967	0.896029	...	0.848359	0.990592	0.980210	0.929954	0.972405	1.000000	0.885320	0.867956	0.939537	0.853665
3916	0.860025	0.820522	0.800100	0.850961	0.835908	0.843586	0.906543	0.947713	0.906447	0.734827	...	0.839612	0.890384	0.849674	0.876370	0.939827	0.885320	1.000000	0.836448	0.928057	0.944069
3927	0.842821	0.842290	0.774292	0.824869	0.802116	0.866246	0.812453	0.829250	0.864123	0.712889	...	0.856874	0.869017	0.827917	0.813153	0.860942	0.867956	0.836448	1.000000	0.834531	0.826519
3948	0.906550	0.820986	0.880990	0.846936	0.910420	0.843845	0.905495	0.893697	0.855408	0.855983	...	0.835680	0.953801	0.933481	0.944095	0.938663	0.939537	0.928057	0.834531	1.000000	0.881699
3952	0.825810	0.775554	0.768266	0.862837	0.794295	0.870092	0.865012	0.969553	0.871778	0.725391	...	0.851016	0.850637	0.807674	0.906585	0.902291	0.853665	0.944069	0.826519	0.881699	1.000000

976 rows × 976 columns

Table 18 : Item Cosine Similarity Matrix

Function for Item-Item Recommendation using Cosine Similarity

```
[ ] # This function work only if we precalculate Cosine matrix for Item vs Item
    # But this is efficient as we have to calcuate Cosine Matric values only once
    # and we can query through below function multiple time

    def Item_Rec_cosine(data,Movie_ID,k):
      # Data is pearson Correlation Dataframe for all item vs item
      # Movie_ID --> Item for which we are finding simialar movies
      # k --> No of Recommenadions Required
      iloc_indices = np.argsort(data.loc[Movie_ID,:].values)[::-1]
      return data.index[iloc_indices][1:k+1]
```

```
[ ] Item_Rec_cosine(Cosine_Item_Mat_df,"1",5)
```

```
Index(['3114', '2355', '3751', '1148', '1223'], dtype='object', name='Movie ID')
```

```
[ ] movies1= movies.drop(["Genressplit"], axis=1).drop_duplicates()
```

```
[ ] movies1[movies1["Movie ID"].isin(Item_Rec_cosine(Cosine_Item_Mat_df,"1",5))]
```

	Movie ID	Title	Genres	Releaseyear	Releaseyeargroups
1132	1148	Wrong Trousers, The	Animation|Comedy	1993	5
1205	1223	Grand Day Out, A	Animation|Comedy	1992	5
2286	2355	Bug's Life, A	Animation|Children's|Comedy	1998	5
3045	3114	Toy Story 2	Animation|Children's|Comedy	1999	5
3682	3751	Chicken Run	Animation|Children's|Comedy	2000	5

- We can see that above recommendations are similar to Movie_ID "1"
- Movies recommended of Movie_ID "1" are exactly same what was observed in Pearson Correlation Method [3114,2355,3751,1148,1223]

```
[ ] # Checking Simialar Movies for "Liar Liar" - Movie_ID : 1485
    movies1[movies1["Movie ID"].isin(Item_Rec_cosine(Cosine_Item_Mat_df,"1485",5))]
```

	Movie ID	Title	Genres	Releaseyear	Releaseyeargroups
139	141	Birdcage, The	Comedy	1996	5
437	441	Dazed and Confused	Comedy	1993	5
1155	1171	Bob Roberts	Comedy	1992	5
1482	1517	Austin Powers: International Man of Mystery	Comedy	1997	5
2690	2759	Dick	Comedy	1999	5

Cosine Similarity Matrix for Similar User Recommendation

```
[ ] Cosine_User_Matrix= cosine_similarity(user_user_scaled)
```

```
[ ] Cosine_User_Matrix.shape
```

```
(1000, 1000)
```

```
[ ] Cosine_User_Matrix
```

```
array([[1.        , 0.89519699, 0.75665161, ..., 0.71989674, 0.81829577,
        0.65972244],
       [0.89519699, 1.        , 0.86142648, ..., 0.60438336, 0.87533041,
        0.75412339],
       [0.75665161, 0.86142648, 1.        , ..., 0.76970319, 0.75548841,
        0.89926712],
       ...,
       [0.71989674, 0.60438336, 0.76970319, ..., 1.        , 0.70726427,
        0.79418284],
       [0.81829577, 0.87533041, 0.75548841, ..., 0.70726427, 1.        ,
        0.78261326],
       [0.65972244, 0.75412339, 0.89926712, ..., 0.79418284, 0.78261326,
        1.        ]])
```

```
[ ] Cosine_User_Mat_df = pd.DataFrame(Cosine_User_Matrix, index = user_user_scaled.index, columns = user_user_scaled.index )
```

Function for Similar User Recommendation

```
# This function work only if we precalculate Cosine matrix for User vs User
# But this is efficient as we have to calcuate Cosine Matric values only once
# and we can query through below function multiple time

def User_Rec_cosine(data,User_ID,k):
  # Data is pearson Correlation Dataframe for all User vs User
  # User_ID --> User for which we are finding simialar Users
  # k --> No of Recommenadions Required
  iloc_indices = np.argsort(data.loc[User_ID,:].values)[::-1]
  return data.index[iloc_indices][1:k+1]
```

```
User_Rec_cosine(Cosine_User_Mat_df,"10",5)
```

```
Index(['151', '5271', '919', '1701', '1164'], dtype='object', name='USerID')
```

```
users.columns
```

```
Index(['USerID', 'Gender', 'Age', 'Occupation', 'Zipcode'], dtype='object')
```

```
users[users["USerID"].isin(User_Rec_cosine(Cosine_User_Mat_df,"10",5))]
```

	USerID	Gender	Age	Occupation	Zipcode
150	151	F	25	20	85013
918	919	F	35	1	92056
1163	1164	F	25	19	90020
1700	1701	F	25	4	97233
5270	5271	F	25	0	94110

- We can see that above recommendations are similar to User_ID "10"

<u>Nearest Neighbors Algorithm for Item Recommendation with Cosine Metric</u>

```
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
```

```
model_knn.fit(item_item_scaled)
```

```
NearestNeighbors(algorithm='brute', metric='cosine')
```

- Now we will find 5 Nearest Neighbour for Movie_ID = 1

```
# k in the Movie_ID
k = "1"
distances, indices = model_knn.kneighbors(item_item_scaled.loc[k,:].values.reshape(1, -1), n_neighbors = 6)
```

```
distances
```

```
array([[0.        , 0.00248231, 0.00257019, 0.00528172, 0.02423075,
        0.02435134]])
```

```
indices
```

```
array([[  0, 806, 602, 950, 266, 295]])
```

```
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(item_item_scaled.index[i]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, item_item_scaled.index[indices.flatten()[i]], distances.flatten()[i]))
```

```
Recommendations for 1:

1: 3114, with distance of 0.00248231289646883:
2: 2355, with distance of 0.0025701919525846773:
3: 3751, with distance of 0.005281715411900256:
4: 1148, with distance of 0.024230750204634388:
5: 1223, with distance of 0.024351342474249527:
```

```
# k in the Movie_ID
# Checking Simialar Movies for "Liar Liar" - Movie_ID : 1485
k = "1485"
distances, indices = model_knn.kneighbors(item_item_scaled.loc[k,:].values.reshape(1, -1), n_neighbors = 6)
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(item_item_scaled.index[i]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, item_item_scaled.index[indices.flatten()[i]], distances.flatten()[i]))
```

```
Recommendations for 1:

1: 441, with distance of 0.005334599396921047:
2: 1517, with distance of 0.0070791118083948566:
3: 2759, with distance of 0.007760380205664719:
4: 141, with distance of 0.01000774462089804:
5: 1171, with distance of 0.010126389346876263:
```

- Nearest Neighbor Algorithm also Recommended same Movies for Movie_ID "1" similar to Pearson Correlation and Cosine similarity

<u>Matrix Factorization with k(Embeddings)= 4</u>

Matrix factorization involves decomposing a large user-item interaction matrix into the product of two lower-dimensional matrices. These matrices represent latent factors for users and items, capturing underlying patterns in the data.

```
[ ] ratings.columns
```

```
Index(['USerID', 'Movie ID', 'Rating', 'Timestamp', 'hour', 'dayofweek',
       'month', 'year', 'day'],
      dtype='object')
```

```
[ ] user_item1 = ratings[['USerID', 'Movie ID', 'Rating']]
```

```
user_item1['USerID'].nunique()
```

```
6040
```

```
[ ] user_item1['Movie ID'].nunique()
```

```
3706
```

```
[ ] user_item1.shape
```

```
(1000209, 3)
```

```
[ ] #  Changing columns as per library requirement
    user_item1.columns = ['UserId', 'ItemId', 'Rating']
```

```
[ ] user_item1.head()
```

	UserId	ItemId	Rating
0	1	1193	5
1	1	661	3
2	1	914	3
3	1	3408	4
4	1	2355	5

```
[ ] model = CMF(method="als", k=4, lambda_=0.1, user_bias=True, item_bias=True, verbose=False)
    model.fit(user_item1)
```

```
Collective matrix factorization model
(explicit-feedback variant)
```

```
[ ] model.A_.shape
```

```
(6040, 4)
```

```
model.B_.shape
```

```
(3706, 4)
```

```
[ ] # Finding Predicted Rating for all Interactions
    np.dot(model.A_, model.B_.T)
```

```
array([[-0.22948895, -0.38242546,  0.07186198, ..., -0.16327427,
        -0.23036666,  0.04428816],
       [-0.36907816, -0.37764624,  0.47913522, ..., -0.05196466,
        -0.10314548,  0.02455359],
       [-0.47322673, -0.36404192, -0.20490578, ..., -0.22136272,
        -0.20130685, -0.00387301],
       ...,
       [-0.99852836, -0.12333941,  0.15010752, ..., -0.00315326,
         0.20635262, -0.13794099],
       [-0.11629342,  0.00380674,  0.1834784 , ...,  0.04070282,
         0.02041265, -0.01076825],
       [ 0.36193278,  0.56550443, -0.00564478, ...,  0.23328021,
         0.19428274, -0.0382481 ]], dtype=float32)
```

```
[ ] # Final Predicted Rating matrix  [dot(User, Item)+ user Bias + Item bias + Global Rating Mean]
    user_item_pred = np.dot(model.A_, model.B_.T) + model.user_bias_.reshape(-1, 1)+ model.item_bias_.reshape(1, -1) + model.glob_mean_
```

```
[ ] user_item_pred = pd.DataFrame(user_item_pred, columns = user_item.columns, index = user_item.index)
```

```
[ ] user_item_pred
```

Movie ID	1	10	100	1000	1002	1003	1004	1005	1006	1007	...	99	990	991	992	993	994	996	997	998	999
UserID																					
1	4.333495	3.188670	4.219245	4.184110	3.803981	4.290605	4.594391	4.422247	4.590399	4.315201	...	3.534152	4.313815	4.035079	3.595278	3.862863	4.864362	3.105049	1.412619	4.176008	4.017798
10	3.969983	2.969526	4.402596	4.185825	3.709589	4.065962	4.367632	4.136885	4.570137	4.346762	...	3.288252	4.159964	3.848906	3.367805	3.637520	4.320831	2.851356	1.300005	4.079306	3.774141
100	4.128202	3.245499	3.980923	4.167925	3.631841	4.364059	3.930409	4.118989	3.828007	4.023709	...	3.375077	4.982054	4.058735	3.615774	3.917556	4.154002	3.176496	1.392975	4.243513	4.008082
1000	4.808551	3.847477	4.297471	3.459938	4.111794	4.249291	4.312610	4.487895	4.468925	4.805585	...	3.084459	4.512967	3.810835	3.274061	3.515534	3.189830	2.636032	1.623991	4.318335	3.679024
1001	3.997905	3.117397	3.031715	2.593992	2.945951	3.552661	2.987324	3.546302	3.117285	3.520792	...	2.378847	4.075969	3.091537	2.632554	2.911162	2.812785	2.125735	0.751135	3.389384	3.039845
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	4.446848	3.490171	3.820152	3.310775	3.719384	4.062055	3.911000	4.156456	3.901362	4.274743	...	2.900009	4.479270	3.629959	3.125385	3.383956	3.147267	2.544312	1.304223	4.050930	3.527829
996	4.192039	3.149032	3.812319	3.518573	3.619547	3.980602	4.058711	4.085215	3.972115	4.122580	...	2.961611	4.265916	3.618758	3.130378	3.389986	3.557965	2.571515	1.164302	3.960178	3.537775
997	3.868599	3.751900	4.601635	4.674826	3.927784	4.753085	2.981449	3.752574	3.081795	4.365089	...	3.209965	6.621392	4.388346	3.836277	4.195067	2.140124	3.397312	1.876884	4.916871	4.139713
998	4.219825	3.348037	4.103997	3.832288	3.753444	4.191797	3.967757	4.118134	3.998971	4.290276	...	3.113029	4.706343	3.842623	3.344982	3.622274	3.484600	2.811747	1.389731	4.199922	3.735877
999	4.257422	3.469105	3.474244	2.948009	3.502455	3.914452	3.305850	3.823320	3.228119	4.004505	...	2.529815	4.720342	3.410211	2.885222	3.156685	2.212729	2.300619	1.141678	3.933162	3.267767

6040 rows × 3706 columns

Table 19 : User Item Matrix using Matrix Factorization

```
[ ] user_item_pred.loc["1000"]["1"]
```

```
4.808551
```

```
[ ] user_item.loc["1000"]["1"]
```

```
5.0
```

MAPE Metric

```
[ ]  user_item.values[user_item > 0]
```

```
array([5., 5., 5., ..., 2., 2., 2.])
```

```
[ ]  user_item_pred.values[user_item > 0]
```

```
array([4.333495 , 4.4208403, 3.5126076, ..., 0.2085023, 2.641737 ,
       2.3006194], dtype=float32)
```

```
[ ]  mape(user_item.values[user_item > 0],user_item_pred.values[user_item > 0] )
```

```
0.427507060167678
```

MSE

```
[ ]  mse(user_item.values[user_item > 0],user_item_pred.values[user_item > 0] )
```

```
2.235840644716731
```

```
[ ]  (mse(user_item.values[user_item > 0],user_item_pred.values[user_item > 0]))**0.5
```

```
1.4952727659917875
```

MAE

```
mae(user_item.values[user_item > 0],user_item_pred.values[user_item > 0] )
```

```
1.1930984183148428
```

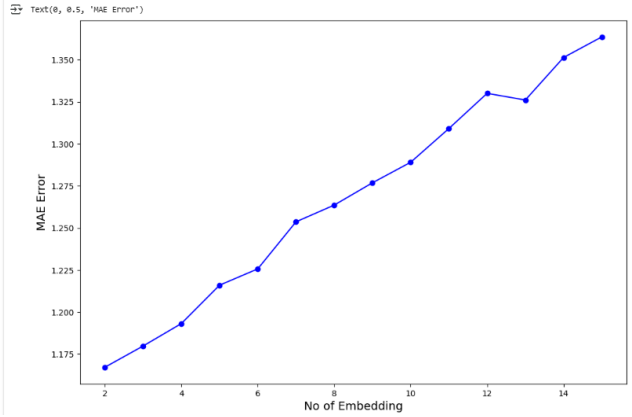
Embedding Dimension Hyperparameter Tuning:

- Looking at the Error value, we can say that embeddings have to be increased to get better results.
- Let's do Hyperparameter Tuning for No of Embeddings

```
[ ]  MAE = []
     for i in range(2,16):
       model = CMF(method="als", k=i, lambda_=0.1, user_bias=True, item_bias=True, verbose=False)
       model.fit(user_item1)
       user_item_pred = np.dot(model.A_, model.B_.T) + model.user_bias_.reshape(-1, 1)+ model.item_bias_.reshape(1, -1) + model.glob_mean_
       MAE.append(mae(user_item.values[user_item > 0],user_item_pred[user_item > 0]))
```

```
plt.figure(figsize=(12, 8))
plt.plot(range(2, 16), MAE, "bo-")
plt.xlabel("No of Embedding", fontsize=14)
plt.ylabel("MAE Error", fontsize=14)
```

```
Text(0, 0.5, 'MAE Error')
```



- MAE Error is increasing with increase in No of Embeddings
- As Sparsity of the Matrix is very high, we are not getting Better Results
- To get Better Results, we have increase more User-Item Ratings

Embedding Visualization with k=4

```
model = CMF(method="als", k=4, lambda_=0.1, user_bias=True, item_bias=True, verbose=False)
model.fit(user_item1)
user_item_pred = np.dot(model.A_, model.B_.T) + model.user_bias_.reshape(-1, 1)+ model.item_bias_.reshape(1, -1) + model.glob_mean_
```

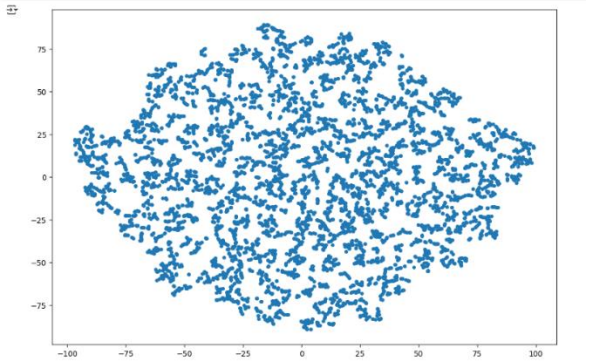
```
# Users
model.A_.shape
```

```
(6040, 4)
```

```
# Items
model.B_.shape
```

```
(3706, 4)
```

```
# User Visualization using tSNE
usertsne = TSNE(n_components = 2, perplexity = 15, random_state = 0)
Z = usertsne.fit_transform(model.A_)

fig, ax = plt.subplots(figsize=(12, 8))
im = ax.scatter(Z[:,0],Z[:,1], s=15)
```



```
# Items Visualization using tSNE
itemtsne = TSNE(n_components = 2, perplexity = 15, random_state = 0)
Z = usertsne.fit_transform(model.B_)

fig, ax = plt.subplots(figsize=(12, 8))
im = ax.scatter(Z[:,0],Z[:,1],s=15)
```

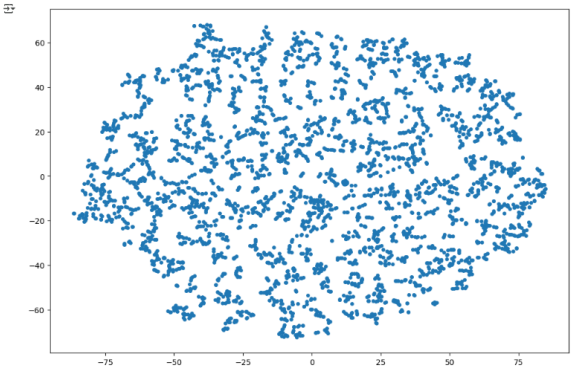


Figure 27 : User Visualization using tSNE

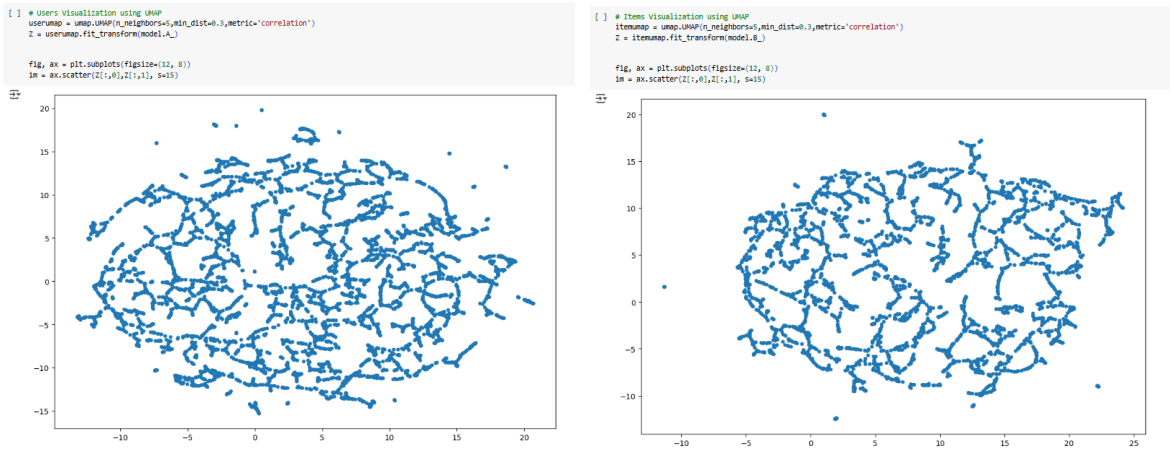


Figure 28 : User Visualization using tSNE

- From MF Embedding Visualization, we are not able to clearly see Clusters but can observe some eccentric Users and Items are present from UMAP.

Business Insights

- Gender has no Bias in Rating Viewpoint for Movies
- Age Groups 18 ~ 25, 25 ~ 34, 35 ~ 44, 45 ~ 49 are comparatively critical in rating movies in comparison with other age groups
- All Age groups have almost preference order for Genres : Drama--> Comedy --> Action --> Thriller
- Female has more preference in Genres - Romance, Comedy, Drame in Comparison with Male
- Male has more preference in Genres - Action, Mystery in comparison with Female
- Comparatively Retired & Homemaker People less preference for Action & have more preference for Comedy and Drama
- Comparatively 1~18 Age user watch/rate movie in the late night between 21:00 ~ 23:00
- Occupation:"college/grad student" have rated most Movies, next most rated are others, executive/managerial,academic/educator,technician/engineer
- Most of the Movie Rating are given by 25 ~ 34 Age Group
- Nearly 75% of ratings are done by Males
- Men in Black (1997) is most no. of times rated movie

Item Item Recommendation System:

Item-Item Recommendation are similar in all 3 Models

- Pearson Correlation
- Cosine Similarity
- Nearest Neighbors Approach

How ever little Variation is Observed in Order of Recommendation May due to Type of Metric Used in 3 Models

- Pearson Correlation : Correlation(Variance) Comparison
- Cosine Similarity : Direction in N-Dim Hyperspace & Comparison of Difference in Angle
- Nearest Neighbors : Purely on the Basis of Distance
-

So there is slight Difference in Order of Recommendation

For Example, Movie_ID: 1, Recommendation Order as below

- Pearson Correlation: 3114,2355,3751,1223,1148
- Cosine Similarity: 1148,1223,2355,3114,3751
- Nearest Neighbors Approach: 3114, 2355,3751,1148,1223

User-User Recommendation System:

User-User Recommendation using Cosine Similarity has Given Comparable results when verified with User Data

Matric Factorization Recommendation System

- Only 4.46 % of ratings are filled in User_Item Interaction Matrix
- Sparsity is Very High for User-Item Interaction Matrix
- So MAPE [42%], MAE [1.19] Observed... which is very High..Not Reliable
- Even with Increasing the no of Embeddings fomr 2 --> 16, MAE increase from 1.165 --> 1.37
- So Need to improve Sparity for use of Matrix Factorization

Drawbacks of Matric Factorization Algorithm

- Predicted rating are not able to get for movies which do not have any Earlier Rating [Kind of Cold Start Problem for Non Rated Movies]
- Need Decent Sparsity to get Relaible Values

Visualization of MF Embedding

From MF Embdedding Visualization, we are not able to clearly see Clusters but can observe some eccentric Users and Items are present from UMAP

Recommendations

Genres like Thriller, Action Romance can be increased as they are 3~ 5th most watched Movies

- Mostly in this Plaform Content is watched by Students or White Collar Job Users ....less users from Blue Collar users like Farmer & trader Segment....Need to Figure Schemes & Content to increase theit Viewership
- Users Have Less Tendency to watch Movies on Fridays , Even if it is Weekend.. So in order not loose User Focus on Fridays, some Exlusive releases or Rent basis Schemes can be provided to capture user even on Friday
- Inorder to capture Young Age Users, Party Streaming cn alos be provided for capturing Threatre & Together watching Feeling
- Make sure that Each Movie has altleast 1 rating in order not ot loose out in MAtrix Factorization Algorithm
- As the User Base is more,User-User Simialrity can be used to Improve REcommendnation to avoid Cold Start Trouble

CONCLUSION

The Conclusion should include some key points as elaborated below -

- Key Takeaways: Highlight the most important points or lessons learned from the Project
- Practical Applications: Significance of methodologies / tools with their real-world applications
- Limitations: Limitations of the methodologies / approaches / tools and suggestions for improvement

References

1. Websites
    a. Geeks for Geeks Website
    b. W3Schools
    c. Wikipedia
    d. Medium
2. Python Libraries
    a. NumPy
    b. Pandas
    c. Matplotlib
    d. Seaborn
    e. Scipy
    f. Sklearn
    g. Umap
    h. statsmodel

Format Guidelines

1) Detailed and Elaborate report of 100 pages at least is expected
2) Margins - Every page of your document must meet the margin requirements of 1.25 inches on the left and right, and 1 inch on the top and bottom.
3) Font:
    a) Style: Times New Roman,
    b) Font Size:14 (For Headings), 12 (For body text) in black colored text.
    c) All text must be the same justification, like left justified or fully justified.
4) Line Spacing:
    a) Body of the text: 1.5
    b) List of Tables/graphs/charts/bibliography: single line.
5) Alignment:
    a) Title page: Centre
    b) Chapter Heading: Centre
    c) Subheading: Left
    d) Body of the text: Justify
6) Titles: All titles and subtitles should be in bold. All tables/graphs/charts/figures should have appropriate titles.
7) Numbering of the tables, charts, graphs should be in the following fashion: Second table/graph/chart in the second chapter should be numbered as Table/graph/chart no. 2.02; where the first digit stands for chapter no. and digits after (.) stands for number of table/graph/charts in that chapter. Same numbering should be followed for all other chapters.