Title: Recommendation Project

About Dataset

This dataset contains all Jio prepaid plans. But it does not includes the plan with Hotstar subscription. Each plan is first converted for single day the it is converted for 365 days or a year. Lets make some interesting insights from it.

Link:
https://www.kaggle.com/datasets/dhamur/jioprepaidplans

```
[6] # Supress Warnings
    import warnings
    warnings.filterwarnings('ignore')
```

```
[5] # Import libraries
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from matplotlib import axes
    import plotly.express as px
    import matplotlib.colors as mcolors
```

```
# Read the given file and view sample records
df = pd.read_csv("/content/Jio - Jio Prepaid Planes.csv")
df.head()
```

	Days	Price	Price/Day	For 365 Days	Profit per customer	From 400 Million users	Data_per_day
0	28	209	7.464286	2724.464286	5.214286	2085.714286	1.0
1	24	179	7.458333	2722.291667	3.041667	1216.666667	1.0
2	20	149	7.450000	2719.250000	0.000000	0.000000	1.0
3	336	2545	7.574405	2764.657738	0.000000	0.000000	1.5
4	84	666	7.928571	2893.928571	129.270833	51708.333330	1.5

1. Libraries and Loading dataset

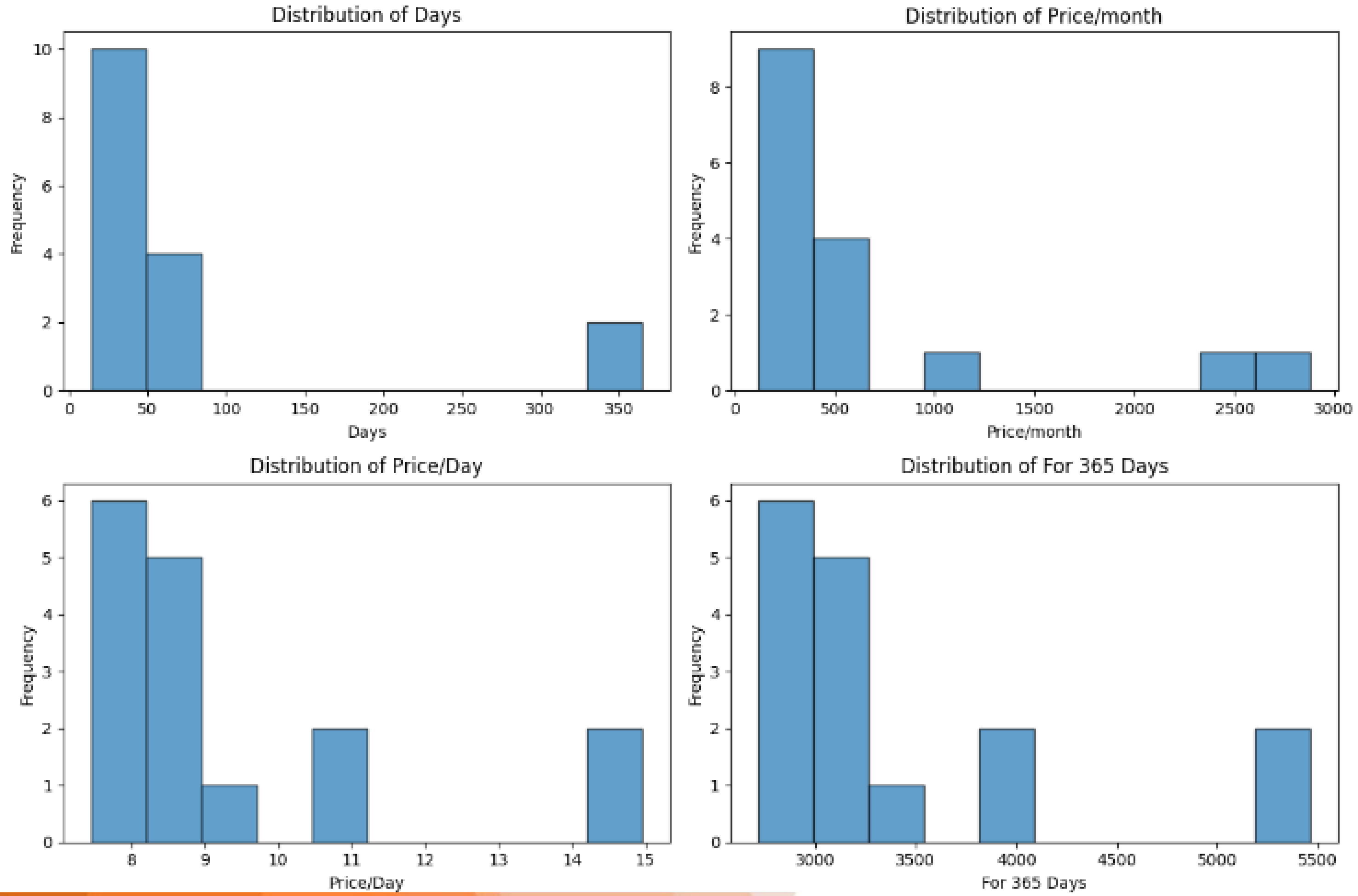
As a start I imported the required libraries and displayed the Head() of our data set to better understand and visualize the given data

```
df.describe()
```

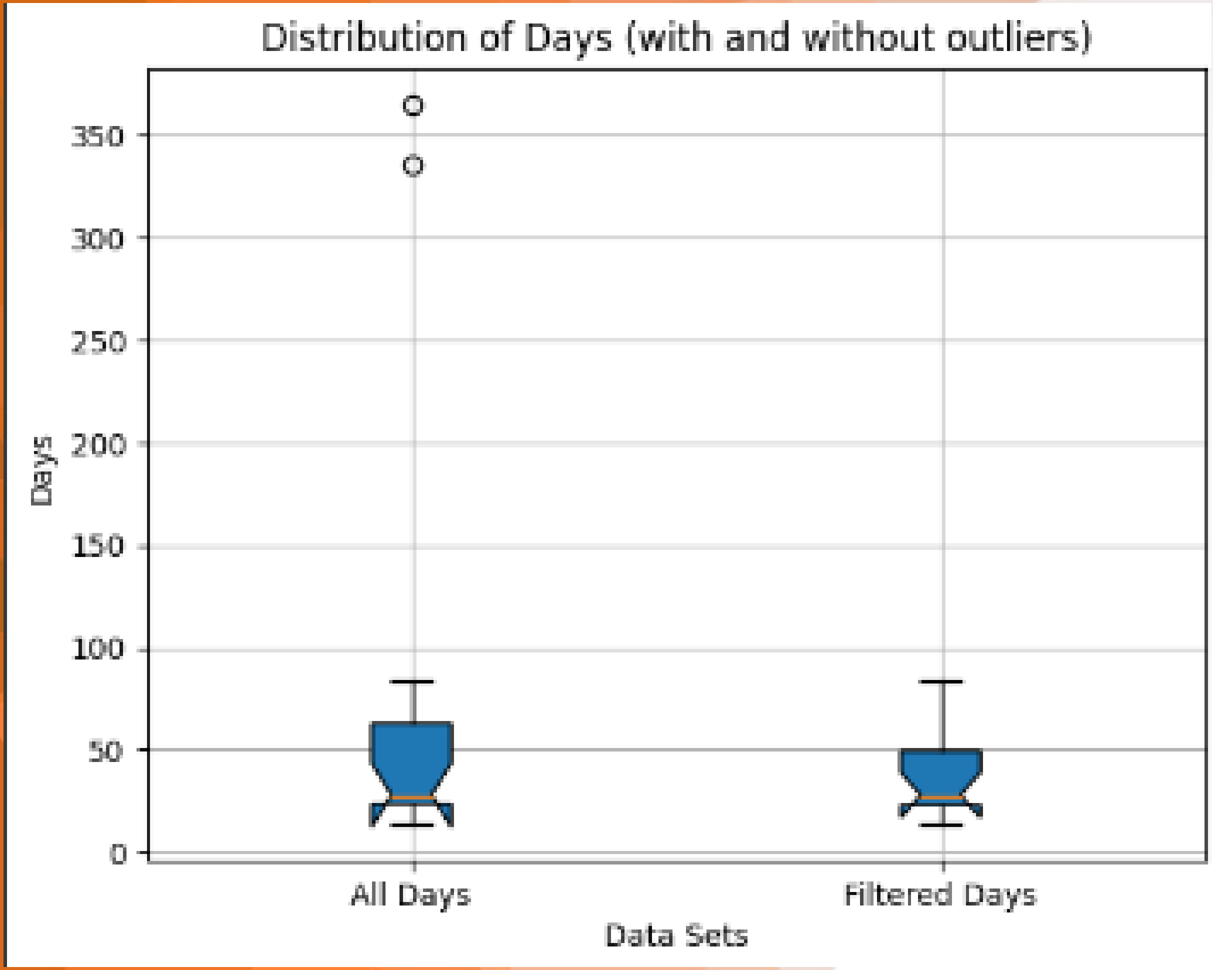
	Days	Price	Price/Day	For 365 Days	Profit per customer	From 400 Million users	Data_per_day
count	16.000000	16.000000	16.000000	16.000000	16.000000	16.000000	16.000000
mean	76.687500	663.812500	9.306167	3396.750914	306.361219	122544.487586	1.718750
std	109.140716	845.097645	2.320658	847.040309	345.420462	138168.184724	0.604669
min	14.000000	119.000000	7.450000	2719.250000	0.000000	0.000000	1.000000
25%	23.750000	206.500000	7.809355	2850.414434	2.281250	912.500000	1.500000
50%	28.000000	279.000000	8.544643	3118.794643	294.933036	117973.214300	1.500000
75%	63.000000	566.250000	9.808036	3579.933035	388.228132	155291.252575	2.000000
max	365.000000	2879.000000	14.964286	5461.964286	1072.521739	429008.695700	3.000000

2. Describing the data

by providing the statistical description to better know the ranges the data is variating with



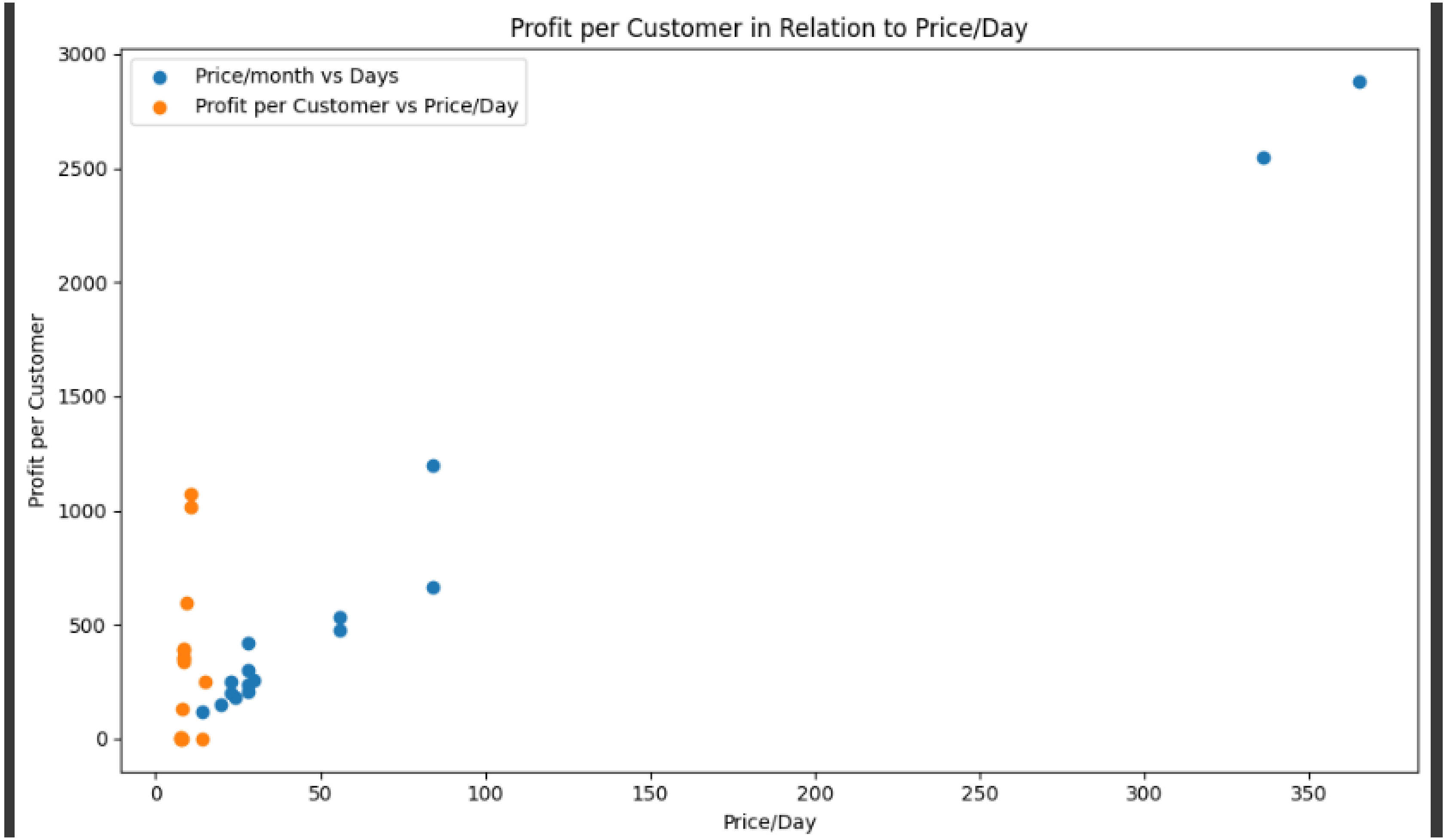
3. Applying basic Visualizations on our given Data to provide better insights



4. Data Exploration and Preprocessing:

Perform exploratory data analysis (EDA) to understand the distribution of different features (e.g., Days, Price, Price/Day, For 365 Days, Profit per customer, Data_per_day). Handle missing values, if any, and perform necessary data cleaning in addition to removing outliers.

5. Data Visualizations
Insights about Relations between data



```
df1 = df.copy()
print(df1['Data_per_day'])

low = df1[df1['Data_per_day'] < 2]
medium = df1[df1['Data_per_day'] ==2]
high = df1[df1['Data_per_day'] ==3]

print("Category Low with Data per day lesser than 2 GB the count is:" , low.shape[0])
print("Category Medium with Data per day equal to 2 GB the count is:" , medium.shape[0])
print("Category High with Data per day equal to 3 GB the count is:" , high.shape[0])
```

```
0     1.0
1     1.0
2     1.0
3     1.5
4     1.5
5     1.5
6     1.5
7     1.5
8     1.5
9     1.5
10    2.0
11    2.0
12    2.0
13    2.0
14    3.0
15    3.0
Name: Data_per_day, dtype: float64
Category Low with Data per day lesser than 2 GB the count is: 10
Category Medium with Data per day equal to 2 GB the count is: 4
Category High with Data per day equal to 3 GB the count is: 2
```

6. Customer Segmentation:

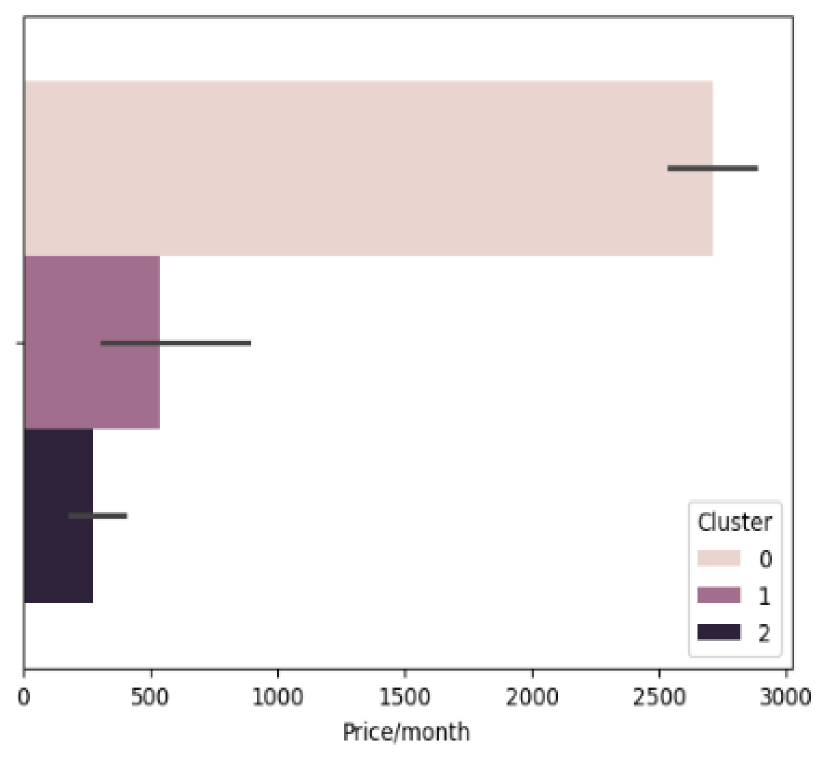
In here we Segmented the Data according to Data_per_day where we had it organized into
3 categories Low, Medium, and High, each having respectively:

* data<2
* data=2
* data>2

K means Clustering

```
sns.barplot(x = 'Price/month', hue = 'Cluster', data = df1)
```

```
<Axes: xlabel='Price/month'>
```




```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(df1[['Days', 'Price/Day', 'Data_per_day', 'Profit per customer']])

kmeans = KMeans(n_clusters=3, random_state=42)


kmeans.fit(scaled_features)

df1['Cluster'] = kmeans.labels_

cluster_centroids = scaler.inverse_transform(kmeans.cluster_centers_)
cluster_df = pd.DataFrame(cluster_centroids, columns=['Days', 'Price/Day', 'Data_per_day', 'Profit per customer'])
print("Cluster Centroids:")
print(cluster_df)

print("Inertia (within-cluster sum of squares):", kmeans.inertia_)

print("Cluster Sizes:")
print(df1['Cluster'].value_counts())

print("Original Data with Clusters:")
print(df1)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.preprocessing import StandardScaler  # Optional for scaling


features = ['Days', 'Profit per customer', 'Price/Day', 'Data_per_day']

# Split data (adjust test size based on your needs)
X_train, X_test, y_train, y_test = train_test_split(df1[features],
                                                    df1['Churn'],
                                                    test_size=0.2, random_state=42)

# Feature scaling (optional, based on model choice)
use_scaling = True  # Set to False if not using scaling

if use_scaling:
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
else:
    X_train_scaled = X_train
    X_test_scaled = X_test

# Model training and evaluation
models = [LogisticRegression(), DecisionTreeClassifier(), RandomForestClassifier()]
model_names = ["Logistic Regression", "Decision Tree", "Random Forest"]

for model, name in zip(models, model_names):
    model.fit(X_train_scaled if use_scaling else X_train, y_train)
    y_pred = model.predict(X_test_scaled if use_scaling else X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    print(f"\n**{name} Results:**")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"ROC-AUC: {roc_auc:.4f}")
```

```
**Logistic Regression Results:**
Accuracy: 0.2500
Precision: 0.3333
Recall: 0.5000
ROC-AUC: 0.2500

**Decision Tree Results:**
Accuracy: 0.2500
Precision: 0.0000
Recall: 0.0000
ROC-AUC: 0.2500

**Random Forest Results:**
Accuracy: 0.5000
Precision: 0.5000
Recall: 0.5000
ROC-AUC: 0.5000
```

7. Predictive Modeling:

Built a predictive model to forecast customer churn based on plan attributes and usage patterns. Used models like logistic regression, decision trees, or random forests

8. Summary:

As we can see that according to the Plans available and taking into consideration
the different categories of Data_per_day.

Plan 8 Data Category 1
Plan 13 Data Category 2
Plan 15 Data Category 3
Are the top performing