Analysis of Booking Cancellations at Yourcabs.com

HUMBER INSTITUTE OF TECHNOLOGY AND ADVANCED LEARNING

(HUMBER COLLEGE)

Submitted to: Professor, Sarama Shehmir

Submitted by: Minh Thu Nguyen

Date: 22-06-2023

Contents

Introduction 2

Goal of the Analysis 2

Executive Summary 2

Dataset Attributes 3

Inputs, outputs for Machine Learning 4

Data Pre-processing (Question 1 + 2) 5

  Rename columns using map 6

  Handling with null-values 6

  Handling Time field 9

Data Division and Encoding Categories Variables (Question 3) 15

Predictive Models (Question 4+5) 17

  Neural Network 19

  K-Nearest Neighbors (KNN) 21

Conclusion 23

Reference 23

Introduction

In late 2013, the taxi company Yourcabs.com in Bangalore, India was facing a problem with the drivers using their platform—not all drivers were showing up for their scheduled calls. Drivers would cancel their acceptance of a call, and, if the cancellation did not occur with adequate notice, the customer would be delayed or even left high and dry.

Bangalore is a key tech center in India, and technology was transforming the taxi industry. Yourcabs.com featured an online booking system (though customers could phone in as well), and presented itself as a taxi booking portal. The Uber ride sharing service started its Bangalore operations in mid-2014.

Yourcabs.com had collected data on its bookings from 2011 to 2013, and posted a contest on Kaggle, in coordination with the Indian School of Business, to see what it could learn about the problem of cab cancellations.

The data presented for this case are a randomly selected subset of the original data, with 10,000 rows, one row for each booking. There are 17 input variables, including user (customer) ID, vehicle model, whether the booking was made online or via a mobile app, type of travel, type of booking package, geographic information, and the date and time of the scheduled trip. The target variable of interest is the binary indicator of whether a ride was canceled. The overall cancellation rate is between 7% and 8%.

Goal of the Analysis

Our goal is to predict cab cancellations to reduce the cost incurred by the company. By predicting possible cancellations an hour before the pickup time, YourCabs will be better able to manage its drivers by providing them up-to-date information about customer cancellations and reduce the cost incurred from sending a cab to a booking location that has been cancelled by the customer.

Executive Summary

This report is an analysis of YourCabs cancellation problem. The data set consisted of 43.431 trips, most of the cancellations were travel type 2 (Point to Point) so the focus of this analysis to predict the cab booking cancellation based on the days of week, hour of a day, type of travelling…

The proportion of cancellations was highest on Saturday, but all of days of the week excluding Saturday were relatively high as well. This is most likely due to rush hour traffic and the high demand for transportation that comes with it as well as drivers choosing to take trip scheduled through their vendor rather than YourCabs. The highest proportion of bookings were from late morning (11AM – 1PM). The number of cancellations on weekdays doubled that of weekend.

YourCabs should put a few different plans into action to lessen cancellations in the future. In order to increase the supply of drivers and slightly balance the demand for trips, YourCabs should enact surge pricing on Friday and Saturday between the hours of 11 a.m. and 1 p.m. and 3 p.m. and 7 p.m. Consequences should be implemented for drivers who frequently cancel their YourCabs trips in order to complete a trip through their vendor. This will help to address the issue of schedule conflicts between vendors. If a vendor has drivers who routinely postpone YourCabs trips, YourCabs should sever ties with that vendor. YourCabs should experiment with requiring drivers to allow for longer trips than what might seem necessary. The use of these techniques will enable YourCabs to significantly lower cancellations.

Dataset Attributes

This dataset was used because it is meant to build and evaluate predictive models. It includes our output car_cancellation and the misclassification costs in Cost_of_error.

Attribute	Description	Data Type
id	Booking id	Numeric/Integer
user_id	The ID of the customer (based on mobile number)	Numeric/Integer
vehicle_model_id	Vehicle model type	Numeric/Integer
package_id	Type of package (1=4hrs & 40kms, 2=8hrs & 80kms, 3=6hrs & 60kms, 4= 10hrs & 100kms, 5=5hrs & 50kms, 6=3hrs & 30kms, 7=12hrs & 120kms)	Numeric/Integer
travel_type_id	Type of travel (1=long distance, 2= point to point, 3= hourly rental).	Numeric/Integer
from_area_id	Unique identifier of area. Applicable only for point-to-point travel and packages	Numeric/Integer
to_area_id	Unique identifier of area. Applicable only for point-to-point travel	Numeric/Integer
from_city_id	Unique identifier of city	Numeric/Integer
to_city_id	Unique identifier of city (only for intercity)	Numeric/Integer
from_date	Time stamp of requested trip start	Date/DateTime
To_date	Time stamp of trip end	Date/DateTime
online_booking	If booking was done on desktop website	Numeric/Integer
mobile_site_booking	If booking was done on mobile website	Numeric/Integer
booking_created	Time stamp of booking	
from_lat	Latitude of from area	Numeric/Integer
from_long	Longitude of from area	Numeric/Integer
to_lat	Latitude of to area	Numeric/Integer
to_long	Longitude of to area	Numeric/Integer
Car_Cancellation	Whether the booking was cancelled (1) or not (0) due to unavailability of a car.	Numeric/Integer
Cost_of_error	The cost incurred if the booking is misclassified. For an un-cancelled booking, the cost of misclassificaiton is 1. For a cancelled booking, the cost is a function of the cancellation time relative to the trip start time	Numeric/Integer

Inputs, outputs for Machine Learning

- **Inputs (Predictors):**
  - desktop_booking
  - mobile_booking
  - from_date:is_weekend
  - from_date:hour
  - booking_created:hour
  - time_diff
  - travel_type_
  - Long Distance
  - travel_type_Point to Point
  - from_date_day_name_Monday
  - from_date_day_name_Saturday
  - from_date_day_name_Sunday
  - from_date_day_name_Thursday
  - from_date_day_name_Tuesday
  - from_date_day_name_Wednesday
  - from_date_month_name_August
  - from_date_month_name_December
  - from_date_month_name_February
  - from_date_month_name_January
  - from_date_month_name_July
  - from_date_month_name_June
  - from_date_month_name_March
  - from_date_month_name_May
  - from_date_month_name_November
  - from_date_month_name_October
  - from_date_month_name_September
  - booking_created_day_name_Monday
  - booking_created_day_name_Saturday
  - booking_created_day_name_Sunday
  - booking_created_day_name_Thursday
  - booking_created_day_name_Tuesday
  - booking_created_day_name_Wednesday
  - booking_created_month_name_August
  - booking_created_month_name_February
  - booking_created_month_name_January
  - booking_created_month_name_July
  - booking_created_month_name_June
  - booking_created_month_name_March
  - booking_created_month_name_May
  - booking_created_month_name_November

- booking_created_month_name_October
    - booking_created_month_name_September
    - booking_created:is_weekend_Weekend
    - from_date:day_part_dawn
    - from_date:day_part_early morning
    - from_date:day_part_evening
    - from_date:day_part_late morning
    - from_date:day_part_midnight
    - from_date:day_part_night
    - from_date:day_part_noon
    - booking_created:day_part_dawn
    - booking_created:day_part_early morning
    - booking_created:day_part_evening
    - booking_created:day_part_late morning
    - booking_created:day_part_midnight
    - booking_created:day_part_night
    - booking_created:day_part_noon

- **Outputs (Targets):**
    - Car_Cancellation

Data Pre-processing (Question 1 + 2)

The data set contains a total of 43.431 entries and 20 columns. Each row in the DataFrame represents a specific entry in the cab booking. The columns in the data set contain various information related to the booking information.

To ensure the data is in a suitable format for analysis, several data preprocessing steps are performed. These steps include renaming columns, handling non-numeric values, converting date columns to datetime format, handling missing data and duplicates, dropping unnecessary columns, filtering invalid data and create new columns. These preprocessing steps help in cleaning and transforming the data set, making it ready for further analysis and prediction modeling.

Rename columns using map

Columns name

8]:

```
#Rename columns
column_mapping = {
    'id': 'id',
    'user_id': 'user_id',
    'vehicle_model_id': 'vehicle_model_type',
    'package_id': 'package_type',
    'travel_type_id': 'travel_type',
    'from_area_id': 'from_area',
    'to_area_id': 'to_area',
    'from_city_id':'from_city_id',
    'to_city_id':'to_city_id',
    'from_date': 'from_date',
    'to_date':'to_date',
    'online_booking':'desktop_booking',
    'mobile_site_booking':'mobile_booking',
    'booking_created': 'booking_dated',
    'from_lat': 'from_lat',
    'from_long': 'from_long',
    'to_lat': 'to_lat',
    'to_long': 'to_long',
    'Car_Cancellation': 'Car_Cancellation',
    'Cost_of_error': 'Cost_of_error'
}
df = df.rename(columns =column_mapping)

#Check updated columns
df.info()
```

Handling with null-values

In [10]:

```
#Drop columns before handling with missing values
columns_to_drop = ['id','user_id','package_type','to_area','from_city_id','to_city_id','to_date']
df.drop(columns_to_drop, axis=1, inplace=True)
df.columns
```

Out[10]:

```
Index(['vehicle_model_type', 'travel_type', 'from_area', 'from_date',
       'desktop_booking', 'mobile_booking', 'booking_dated', 'from_lat',
       'from_long', 'to_lat', 'to_long', 'Car_Cancellation', 'Cost_of_error'],
      dtype='object')
```

I dropped some columns have more than 80% of missing values and unneeded columns

1]:
```
df.isnull().sum()
```
1]:
```
vehicle_model_type       0
travel_type              0
from_area               88
from_date                0
desktop_booking          0
mobile_booking           0
booking_dated            0
from_lat                93
from_long               93
to_lat                9138
to_long               9138
Car_Cancellation         0
Cost_of_error            0
dtype: int64
```

With rows with small percentage of null values, I dropped non-values in 'from_area', 'from_lat', 'from_long' columns.

With 'to_lat', 'to_long' having 9138 null rows, I filled non-values with mean values.

In [12]:

```
#Drop non-values in 'from_area', 'from_lat', 'from_long' columns because of small percenatge
#Fill non-values in 'to_lat', 'to_long' with mean values with numeric data
df.dropna(subset=['from_area','from_lat','from_long'], inplace=True)
df['to_lat'].fillna(df['to_lat'].mean(),inplace=True)
df['to_long'].fillna(df['to_long'].mean(),inplace=True)
```

In [13]:

```
#Confirm the changes
print("Missing values")
df.isnull().sum()
```

```
Missing values
```

Out[13]:

```
vehicle_model_type    0
travel_type           0
from_area             0
from_date             0
desktop_booking       0
mobile_booking        0
booking_dated         0
from_lat              0
from_long             0
to_lat                0
to_long               0
Car_Cancellation      0
Cost_of_error         0
dtype: int64
```

Handling Time field

In [14]:

```
#Convert date columns into datetime format
df['from_date_dt'] = pd.to_datetime(df['from_date']).dt.strftime('%m/%d/%Y')
df['from_date_time'] = pd.to_datetime(df['from_date']).dt.strftime('%H:%M')
df['booking_created_dt'] = pd.to_datetime(df['booking_dated']).dt.strftime('%m/%d/%Y')
df['booking_created_time']=pd.to_datetime(df['booking_dated']).dt.strftime('%H:%M')
```

In [15]:

```
#Extract day month components from 'from_date_dt'
df['from_date_day_name'] = pd.to_datetime(df['from_date_dt']).dt.day_name()
df['from_date_day_num'] = pd.to_datetime(df['from_date_dt']).dt.day_of_week #The day of the week wit
h Monday=0, Sunday=6.
df['from_date_month_name'] = pd.to_datetime(df['from_date_dt']).dt.month_name()
df['from_date_month_num'] = pd.to_datetime(df['from_date_dt']).dt.month
df[['from_date_dt','from_date_day_name', 'from_date_day_num','from_date_month_name','from_date_month
_num']].head(10)
```

Out[15]:

	from_date_dt	from_date_day_name	from_date_day_num	from_date_month_name	from_date_month_num
0	01/01/2013	Tuesday	1	January	1
1	01/01/2013	Tuesday	1	January	1
2	01/01/2013	Tuesday	1	January	1
3	01/01/2013	Tuesday	1	January	1
4	01/01/2013	Tuesday	1	January	1
5	01/01/2013	Tuesday	1	January	1
6	01/01/2013	Tuesday	1	January	1
7	01/01/2013	Tuesday	1	January	1
8	01/01/2013	Tuesday	1	January	1
9	01/01/2013	Tuesday	1	January	1

In [16]:

```
#Extract day month components from 'from_date_dt'
df['booking_created_day_name'] = pd.to_datetime(df['booking_created_dt']).dt.day_name()
df['booking_created_day_num'] = pd.to_datetime(df['booking_created_dt']).dt.day_of_week #The day of
the week with Monday=0, Sunday=6.
df['booking_created_month_name'] = pd.to_datetime(df['booking_created_dt']).dt.month_name()
df['booking_created_month_num'] = pd.to_datetime(df['booking_created_dt']).dt.month
df[['booking_created_dt','booking_created_day_name', 'booking_created_day_num','booking_created_mont
h_name','booking_created_month_num']].head(10)
```

Out[16]:

	booking_created_dt	booking_created_day_name	booking_created_day_num	booking_created_month_name	booking_created_month_
0	01/01/2013	Tuesday	1	January	1
1	01/01/2013	Tuesday	1	January	1
2	01/01/2013	Tuesday	1	January	1
3	01/01/2013	Tuesday	1	January	1
4	01/01/2013	Tuesday	1	January	1
5	01/01/2013	Tuesday	1	January	1
6	01/01/2013	Tuesday	1	January	1
7	01/01/2013	Tuesday	1	January	1
8	01/01/2013	Tuesday	1	January	1
9	01/01/2013	Tuesday	1	January	1

There are 2 datetime fields ('from_date' and 'booking_dated'). In order to using datetime functions, I extracted date columns to datetime formate using pandas (pd.to_datetime). In these above codes, I extracted datetime format, day-month components (days of week and month) to support further analysis.

In order to analyze deeply in time field, I created weekend flag and day part flag to specific time range in a day/ week as below:

In [17]:

```
#Create Weekend Flag
df['from_date:is_weekend'] = np.where(df['from_date_day_num'].isin([5,6]), 1,0)
df['booking_created:is_weekend'] = np.where(df['booking_created_day_num'].isin([5,6]), 1,0)

df[['from_date_dt','from_date_day_num','from_date:is_weekend']].head()
df[['booking_created_dt','booking_created_day_num','booking_created:is_weekend']].head()
```

Out[17]:

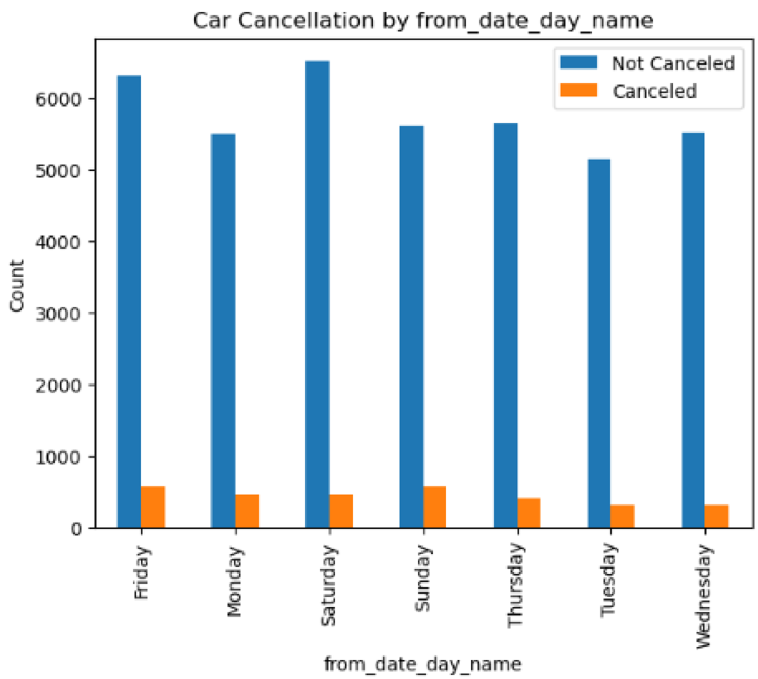
	booking_created_dt	booking_created_day_num	booking_created:is_weekend
0	01/01/2013	1	0
1	01/01/2013	1	0
2	01/01/2013	1	0
3	01/01/2013	1	0
4	01/01/2013	1	0

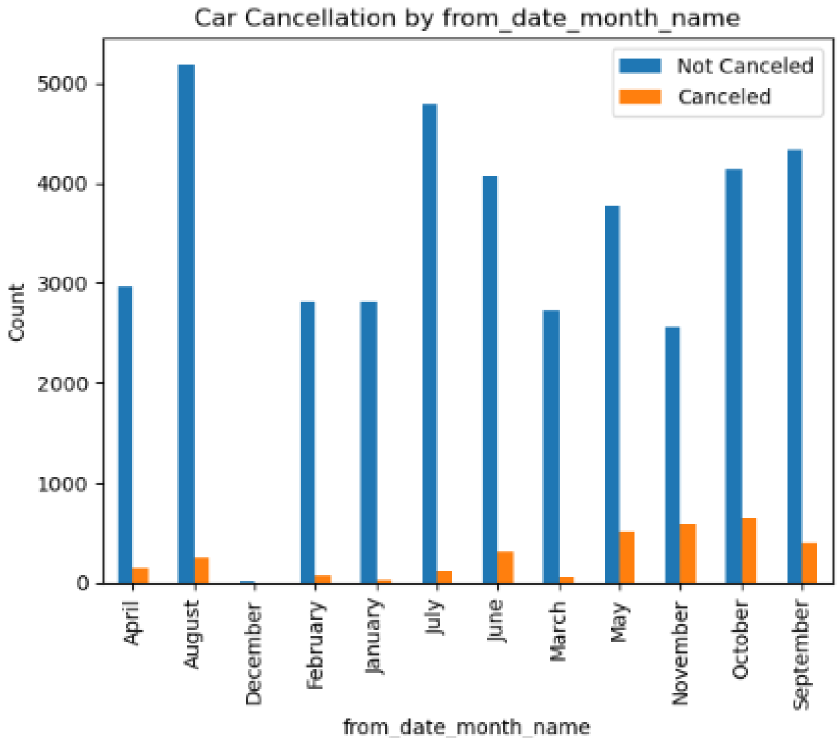
In [18]:

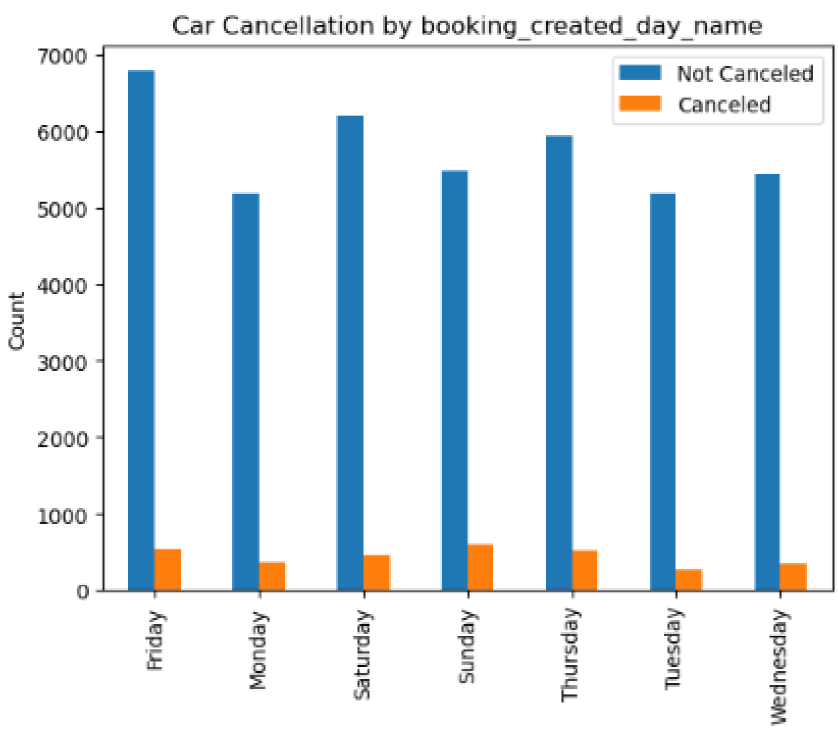
```
#Create Day Part Flag
def day_part(hour):
    if hour in [4,5]:
        return "dawn"
    elif hour in [6,7]:
        return "early morning"
    elif hour in [8,9,10]:
        return "late morning"
    elif hour in [11,12,13]:
        return "noon"
    elif hour in [14,15,16]:
        return "afternoon"
    elif hour in [17, 18,19]:
        return "evening"
    elif hour in [20, 21, 22]:
        return "night"
    elif hour in [23,24,1,2,3]:
        return "midnight"


# utilize it along with apply method in 'from_date_time'
df['from_date:hour'] = pd.to_datetime(df['from_date_time']).dt.hour
df['from_date:day_part'] = df['from_date:hour'].apply(day_part)
df.head()
```

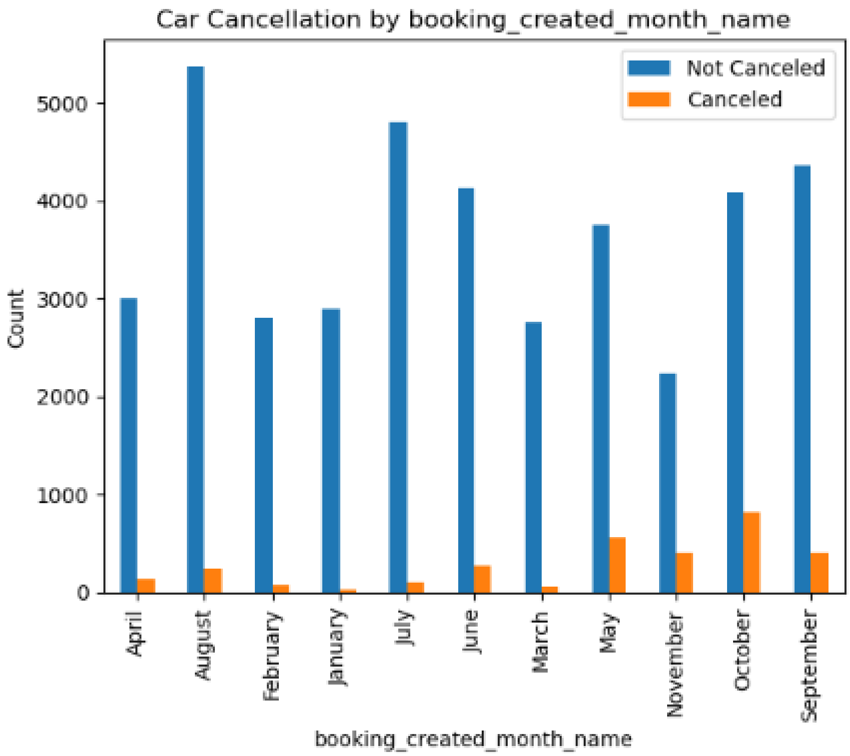
In [26]:

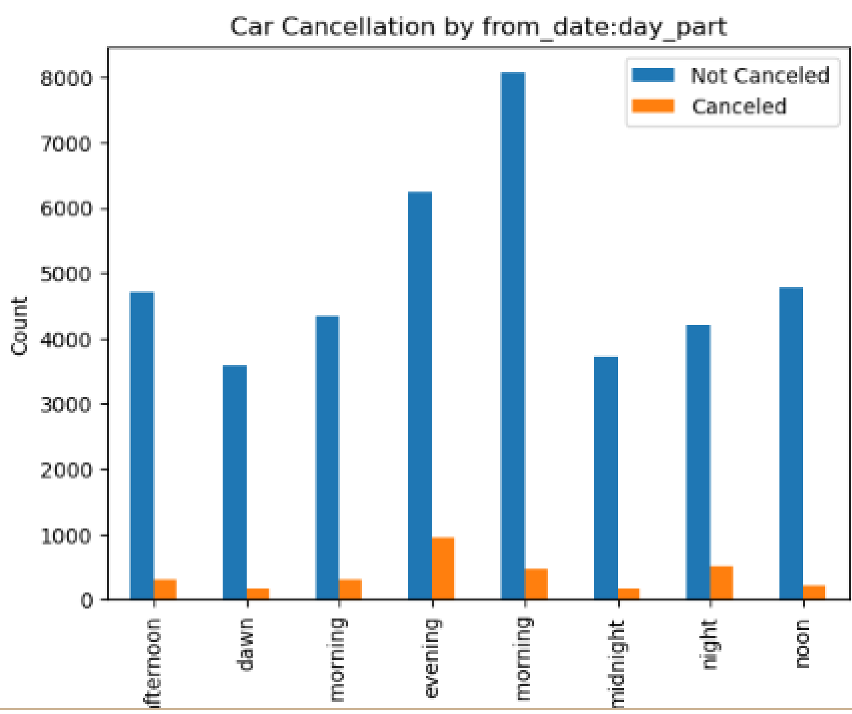
```
# Select object columns from the dataframe
object_columns = df.select_dtypes(include='object').columns

# Loop through each object column
for column in object_columns:
# Count the occurrences of car_cancellation for each category in the column
    cancellation_counts = df.groupby(column)['Car_Cancellation'].value_counts().unstack().fillna(0)

# Plot the bar chart
    cancellation_counts.plot(kind='bar', stacked=False)
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.title(f'Car Cancellation by {column}')
    plt.legend(['Not Canceled', 'Canceled'])
    plt.show()
```




Car Cancellation by from_date_month_name
Not Canceled
Canceled
Count
0
1000
2000
3000
4000
5000
April
August
December
February
January
July
June
March
May
November
October
September
from_date_month_name

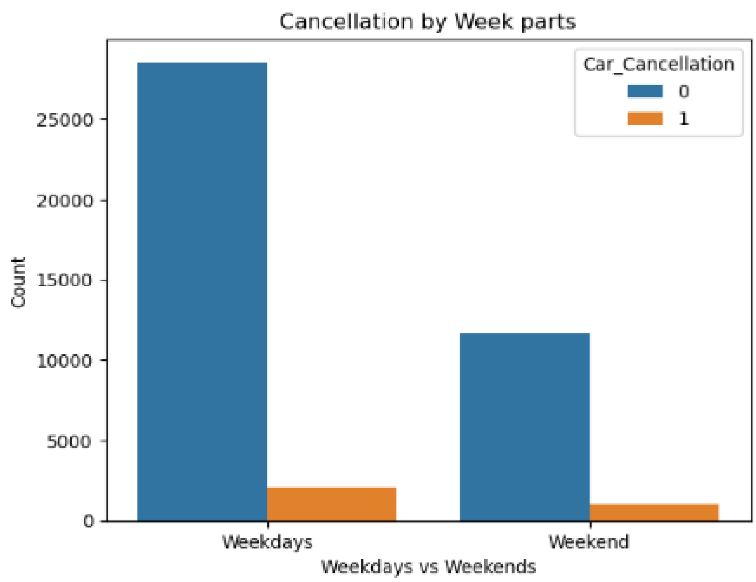

Car Cancellation by booking_created_day_name
Not Canceled
Canceled
Count
0
1000
2000
3000
4000
5000
6000
7000
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday


Car Cancellation by booking_created_month_name
Not Canceled
Canceled
Count
0
1000
2000
3000
4000
5000
April
August
February
January
July
June
March
May
November
October
September
booking_created_month_name


Car Cancellation by from_date:day_part
Not Canceled
Canceled
Count
0
1000
2000
3000
4000
5000
6000
7000
8000
fternoon
dawn
morning
evening
morning
midnight
night
noon

Those above visualization are the plot showing relation between the number of cancellation per day part time, per day in a week and per month. It is evident that evening time is the time experiencing highest percentage of cancellation. In regards of month, October is the most cancelled month followed by November. This observation may result from the high demand of commuting in the autumn time in Indian when weather is good for outside activity.

In [29]:
```
##Weekdays and Weekend Cancellations
# Create a dictionary mapping numeric values to categorical labels
booking_week = {0: 'Weekdays', 1: 'Weekend'}
df['booking_created:is_weekend'] = df['booking_created:is_weekend'].map(booking_week)

cancellation_week_count = df.groupby('booking_created:is_weekend')['Car_Cancellation'].sum().reset_i
ndex()
# Plot the countplot
sns.countplot(data=df, x='booking_created:is_weekend', hue='Car_Cancellation')

# Set labels and title
plt.xlabel('Weekdays vs Weekends')
plt.ylabel('Count')
plt.title('Cancellation by Week parts')

# Display the plot
plt.show()
```



As can be seen from this figure, the number of cancellations in weekdays is as twice as that of weekend. The possible reasons are on weekdays, people often have demanding work schedules and commitments. They might book cabs for their commute to work, meetings or other professional obligations.

Data Division and Encoding Categories Variables (Question 3)

After transforming the data, this is how data frame looks like

In [30]:

```
num = df.select_dtypes(include='number')
char = df.select_dtypes(include='object')
```

In [31]:

```
ml_df = df.copy()
ml_df
```

Out[31]:

	vehicle_model_type	travel_type	from_area	desktop_booking	mobile_booking	from_lat	from_long	to_lat	to_lo
0	28	Point to Point	83.0	0	0	12.924150	77.672290	12.927320	77.6
1	12	Point to Point	1010.0	0	0	12.966910	77.749350	12.927680	77.6
2	12	Point to Point	1301.0	0	0	12.937222	77.626915	13.047926	77.5
3	12	Point to Point	768.0	0	0	12.989990	77.553320	12.971430	77.6
4	12	Point to Point	1365.0	0	0	12.845653	77.677925	12.954340	77.6
...	...	...	...	...	...	...	...	...	...
43426	12	Point to Point	1147.0	1	0	13.030640	77.649100	12.952780	77.5
43427	12	Point to Point	393.0	1	0	13.199560	77.706880	13.017436	77.6
43428	12	Hourly Rental	974.0	0	0	13.075570	77.559040	13.026648	77.6
43429	87	Point to Point	1263.0	0	0	12.968970	77.594560	12.938230	77.6
43430	12	Point to Point	689.0	0	1	12.976720	77.649270	13.199560	77.7

43338 rows × 26 columns

There are 43.338 rows and 26 columns after cleaning and do some transformation

For numeric variables, I only encode the travelling type for further learning and change new columns into traveltype_pointtopoint and traveltype_hourly

In [32]:

```python
#Encoding
traveltype = pd.get_dummies(ml_df['travel_type'],drop_first=True)
ml_df = pd.concat([ml_df,traveltype],axis=1)
ml_df = ml_df.drop(['travel_type'],axis=1)
ml_df.rename(columns={2:'traveltype_pointtopoint',3:'traveltype_hourly'},inplace=True)
ml_df
```

Out[32]:

	vehicle_model_type	from_area	desktop_booking	mobile_booking	from_lat	from_long	to_lat	to_long	Car_
0	28	83.0	0	0	12.924150	77.672290	12.927320	77.635750	0
1	12	1010.0	0	0	12.966910	77.749350	12.927680	77.626640	0
2	12	1301.0	0	0	12.937222	77.626915	13.047926	77.597766	0
3	12	768.0	0	0	12.989990	77.553320	12.971430	77.639140	0
4	12	1365.0	0	0	12.845653	77.677925	12.954340	77.600720	0
...	...	...	...	...	...	...	...	...	...
43426	12	1147.0	1	0	13.030640	77.649100	12.952780	77.590880	0
43427	12	393.0	1	0	13.199560	77.706880	13.017436	77.644580	0
43428	12	974.0	0	0	13.075570	77.559040	13.026648	77.640595	0
43429	87	1263.0	0	0	12.968970	77.594560	12.938230	77.622890	0
43430	12	689.0	0	1	12.976720	77.649270	13.199560	77.706880	0

43338 rows × 27 columns

For categorical columns, I generated them into numeric values (using get_dummies) to focus on the day part/ day of week and month. Other categorical columns I dropped because they are unused. To proceed with neural networks I need the input to be in numerical format, so I converted the categorical columns to numerical with the help of get_dummies method (Pandas.get_dummies — Pandas 1.2.4 Documentation, n.d.).

In [33]:

```python
# generate numeric values from categorical columns
ml_df = pd.get_dummies(char, drop_first = True) # dropping the first category

print(ml_df.shape)
ml_df
```

```
(43338, 50)
```

Out[33]:

	travel_type_Long Distance	travel_type_Point to Point	from_date_day_name_Monday	from_date_day_name_Saturday	from_date_day_name_Sunday
0	0	1	0	0	0
1	0	1	0	0	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	1	0	0	0
...	...	...	...	...	...
43426	0	1	0	0	1
43427	0	1	0	0	0
43428	0	0	0	0	1
43429	0	1	0	0	1
43430	0	1	1	0	0

After that, I combined both numeric and categorical columns into new data frame ready for further algorithm analysis

In [34]:

```
#Concate numeric and categorical variables
ml_df = pd.concat([num,ml_df],axis=1,join='inner')
ml_df
```

Out[34]:

arly	booking_created:day_part_evening	booking_created:day_part_late morning	booking_created:day_part_midnight	booking_created:day_part_night
	0	0	1	0
	0	0	1	0
	0	0	1	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

43338 rows × 68 columns

Predictive Models (Question 4+5)

The models that I have looked at are Neural Network and K-Nearest Neighbors (KNN) with using the same predictors and target

In [36]:

```python
# target and predictors
predictors = ml_df.drop(['vehicle_model_type', 'from_area','from_lat', 'from_long', 'to_lat', 'to_lo
ng', 'from_date_day_num', 'from_date_month_num','Car_Cancellation', 'Cost_of_error','booking_created
_day_num','booking_created_month_num'], axis = 1)
target = ml_df[['Car_Cancellation']]

predictors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43338 entries, 0 to 43430
Data columns (total 56 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   desktop_booking                 43338 non-null  int64  
 1   mobile_booking                  43338 non-null  int64  
 2   from_date:is_weekend            43338 non-null  int64  
 3   from_date:hour                  43338 non-null  int64  
 4   booking_created:hour            43338 non-null  int64  
 5   time_diff                       43338 non-null  float64
 6   travel_type_Long Distance       43338 non-null  uint8  
 7   travel_type_Point to Point      43338 non-null  uint8  
 8   from_date_day_name_Monday       43338 non-null  uint8  
 9   from_date_day_name_Saturday     43338 non-null  uint8  
 10  from_date_day_name_Sunday       43338 non-null  uint8  
 11  from_date_day_name_Thursday     43338 non-null  uint8  
 12  from_date_day_name_Tuesday      43338 non-null  uint8  
 13  from_date_day_name_Wednesday    43338 non-null  uint8  
 14  from_date_month_name_August     43338 non-null  uint8  
 15  from_date_month_name_December   43338 non-null  uint8  
 16  from_date_month_name_February   43338 non-null  uint8  
 17  from_date_month_name_January    43338 non-null  uint8  
 18  from_date_month_name_July       43338 non-null  uint8  
 19  from_date_month_name_June       43338 non-null  uint8  
 20  from_date_month_name_March      43338 non-null  uint8  
 21  from_date_month_name_May        43338 non-null  uint8  
```

Target is car_cancellation and Predictors are 55 columns

Neural Network

In [37]:
```
X = predictors
y = target
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [38]:
```
clf = MLPClassifier(hidden_layer_sizes=(3), activation='logistic', solver='lbfgs',random_state=1)
clf.fit(train_X, train_y)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:541: Co
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

Out[38]:
```
MLPClassifier
MLPClassifier(activation='logistic', hidden_layer_sizes=3, random_state=1,
              solver='lbfgs')
```

First I start with the hidden layers in neural networks, 3 hidden layer refers to one or more layers of nodes located between the input layer and the output layer. These hidden layers extract and transform features from the input data, enabling the network to learn and make predictions or classifications.

Each node or neuron in a hidden layer receives inputs from the previous layer, applies an activation function to the weighted sum of those inputs, and produces an output. The outputs from the nodes in the hidden layer then serve as inputs to the next layer until the final output layer is reached. The word "hidden" comes from the fact that the hidden layer nodes' outputs cannot be directly seen or accessed as a component of the neural network's input or output. They are temporary versions of the data that incorporate the features the network has learned (Sandhyakrishnan, 2021).

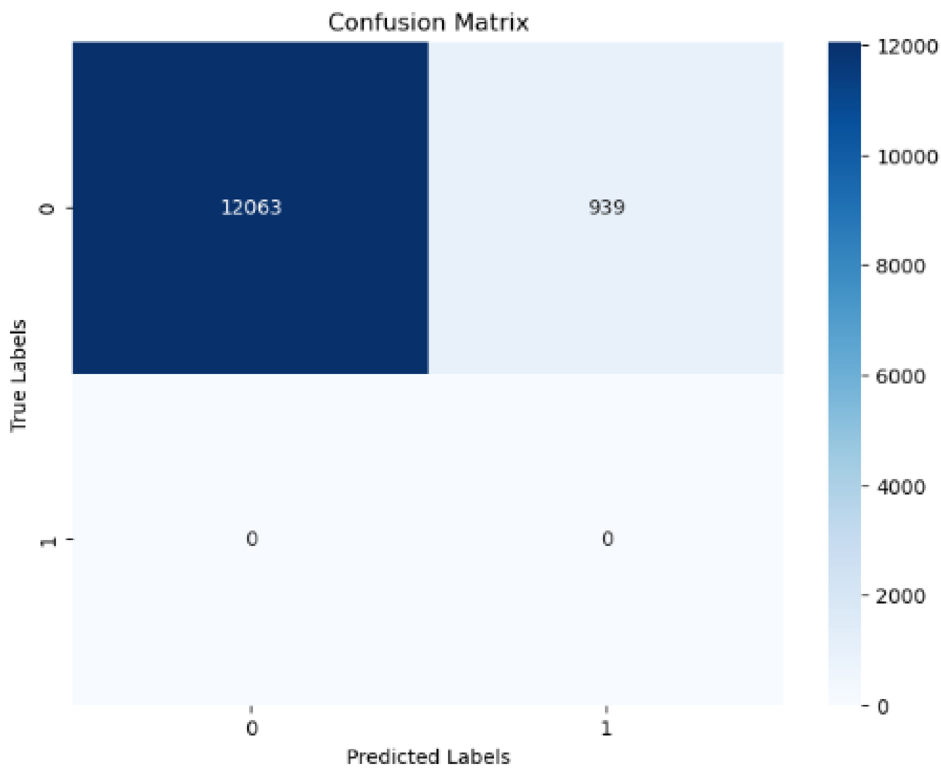
In [39]:

```python
# Network structure
print('Intercepts')
print(clf.intercepts_)
print('Weights')
print(clf.coefs_)
```

```
Intercepts
[array([1.0407515 , 1.12245564, 0.1982206 ]), array([1.37643551])]
Weights
[array([[-2.03199094e+00, -1.64064109e+00, -1.49560381e-01],
       [-1.57112121e+00, -5.62512797e-01, -1.47162911e-01],
       [ 2.75230744e-02,  3.61358259e-01,  4.17052937e-02],
       [-2.24406684e-02,  1.42932683e+00,  2.24296949e+00],
       [ 5.79147013e-02, -1.19565739e+00,  2.55160993e+00],
       [ 2.14193770e-03, -1.19293109e-01, -3.20399561e-03],
       [ 9.02370759e-02, -5.07928378e-02,  1.24366784e-01],
       [-8.11115852e-01, -1.56578258e-01,  2.78391961e-01],
       [ 2.23666870e-01,  3.72374311e-01, -1.28971828e-01],
       [ 2.20484513e-02,  8.92143013e-02,  1.92236607e-01],
       [-1.96617281e-01,  1.78390879e-01,  1.95309610e-01],
       [ 3.37110141e-01,  3.11965424e-01, -2.57691883e-02],
       [ 2.45562703e-01,  2.09191410e-01, -1.69314568e-01],
       [ 3.71730686e-01,  3.89146763e-01,  1.47635381e-01],
       [ 7.05693667e-02,  9.63967195e-02, -1.07715933e-01],
       [-1.91858097e-02,  1.50446480e-01, -7.49526075e-02],
       [ 5.74059854e-01,  1.89619689e-01, -1.38913311e-01],
       [ 6.42167348e-01,  1.85033408e-01, -8.02132542e-02],
       [ 6.34053865e-01,  2.71110055e-01,  6.58246504e-02],
       [-1.35192223e-01, -6.95862891e-02,  9.33998138e-02],
       [ 4.02546080e-01,  1.68810898e-01,  9.67619478e-02],
       [-4.61595649e-01, -4.15916383e-01,  4.64350954e-02],
       [-6.18717810e-01, -5.24691576e-02,  1.34833913e-01],
       [-7.06898807e-01,  7.17617384e-02, -1.12304215e-01],
```

In [40]:

```python
#Predicting y for X_test
y_pred = clf.predict(valid_X)

# validation performance
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_pred, valid_y)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

from sklearn.metrics import accuracy_score
# Calculate accuracy score
accuracy = accuracy_score(valid_y, y_pred)

# Print the accuracy score
print("Accuracy:", accuracy)
```




```
Accuracy: 0.9277803414859253
```

The accuracy rate of the model is 92.77%

Overall Error rate is = 100% - 92.77% = 7.23%

K-Nearest Neighbors (KNN)

Build Machine Learning - K-Nearest Neighbors (KNN)

[45]:

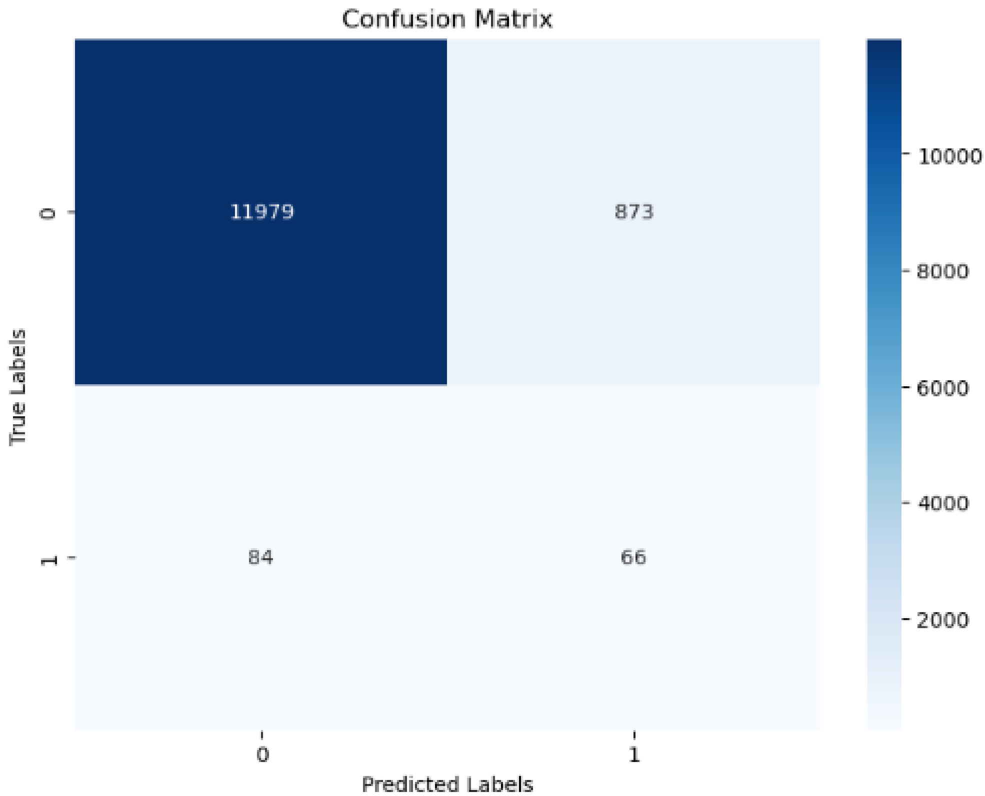
```
knn = KNeighborsClassifier(weights='distance', n_neighbors=10)
knn.fit(train_X, train_y)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/neighbors/_classification.py:215: DataConversionWarni
  return self._fit(X, y)
```

[45…

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10, weights='distance')
```

```python
y_pred2 = knn.predict(valid_X)

# validation performance
cm = confusion_matrix(y_pred2, valid_y)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Calculate accuracy score
accuracy = accuracy_score(valid_y, y_pred2)

# Print the accuracy score
print("Accuracy:", accuracy)
```




```
Accuracy: 0.9263959390862944
```

When applying KNN algorithm, the accuracy rate is 92.63% and the overall error rate is 7.37%

Conclusion

The provided dataset, which focuses on the 'Car_Cancellation' target variables, provides insightful information for enhancing the supply chain process. Organizations can improve cost management, maximize operational efficiency, and lower operating costs by utilizing neural network and KNN models.

The accuracy of Neural Network is 92.77% and KNN is 92.63%, slightly different which partly indicates that the models work quite well in practice.

By analyzing the dataset using neural networks and KNN, organizations can achieve cost optimization by identifying which booking stand high chance of cancellation based on factors such as day part, day of week and month and type of travelling. The neural network models and KNN can predict the car cancellation accurately, enabling organizations to equip them with up-to-date information about customer cancellations, optimize cost and transportation expenses.

Reference

pandas.get_dummies — pandas 1.2.4 documentation. (n.d.). Pandas.pydata.org. Retrieved June 23, 2023, from https://pandas.pydata.org/pandas-docs/version/1.2.4/reference/api/pandas.get_dummies.html

Sandhyakrishnan. (2021, September 10). *How to determine the number of layers and neurons in the hidden layer?* Medium. https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3