UrbanClap Assignment

This is the solution to the Assignment 1 for the Business Analytics intern position in UrbanClap, Gurgaon

Importing the libraries

In [1]:
```
import numpy as np                                      # For performing opera
tions on numpy arrays
import pandas as pd                                     # For storing and mani
pulating the dataset
import pickle                                           # For storing the inte
rmediate results
import random                                           # For generating rando
m distributions
import time, datetime, calendar                         # For working with tim
estamps

import matplotlib.pyplot as plt                         # For plotting the gra
phs
import seaborn as sns                                   # For generating visua
lisation
import plotly                                           # For interactive Grap
hing
plotly.tools.set_credentials_file(
    username='revantg',
    api_key='0ox6fKEY5TPFHc5DSjNF'                      # API Key has been men
tioned deliberately for using Plotly
)

from imblearn.over_sampling import SMOTE                # For sampling the unb
alanced dataset for classification
from sklearn.preprocessing import LabelEncoder          # For encoding labels
to numbers
from sklearn.preprocessing import OneHotEncoder         # For one-hot encoding
categorical variables
from sklearn.model_selection import train_test_split    # For Splitting into T
rainSet and TestSet
from sklearn.model_selection import RepeatedKFold       # For KFold Cross Vali
dation
from sklearn.model_selection import TimeSeriesSplit     # For CrossValidation
of Time-Series Datasets
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression    # For Logistic Regress
ion
from sklearn.metrics import accuracy_score              # For calculating accu
racy
```

In [2]:
```
from warnings import simplefilter                       # For ignoring Warning
s
simplefilter(action='ignore')

pd.set_option('display.max_columns', 500)
```

Importing the dataset

This dataset belongs to a company providing home services via online discovery platform.
The dataset contains records of 30,940 transactions of 16,711 customers.

- Each row corresponds to one transaction (transaction/order/request will be used interchangeably) placed by a customer tracked by `Transaction_ID` unique to the order placed

Fields

- **`Profile ID`** : The unique identity value of each customer. For example, transactions with IDs `BBCHH`, `CHWFD` are placed by the same customer identified through `Profile ID 1`.
- **`Date Of Booking`** : Date on which customer placed the order.
- **`Date of Service Request`** : Date on which the customer needs the service delivered at his/her house
- **`Slot Of Booking`** : Hour of the service requested date when the customer wants the servie to be served
- **`Source`** : The channel of Customer (eg: Customer came to the app/web via Google, Facebook etc)

In [3]:
```
df_orig = pd.read_excel("dataset.xlsx",
                        sheet_name = "Data",
                        parse_dates = ['Date_of_Booking', 'Date_of_Service_Reques
ted']
                       )

df = df_orig.copy()
```

In [4]:
```
df.head()
```

Out[4]:

	Transaction_ID	Profile ID	Date_of_Booking	Date_of_Service_Requested	Source	Slot of Booking (Hour of the Day)
0	BBCHH	1	2018-05-20	2018-05-21	D	14
1	CHWFD	1	2018-09-23	2018-09-23	C	8
2	DYDMF	2	2018-11-10	2018-11-13	C	11
3	EZYSA	3	2018-04-12	2018-04-16	B	8
4	HWAKX	3	2018-08-05	2018-08-06	B	11

The following queries have been solved in `Python3`

1. **Plot new users acquired every month on a bar chart (New user in a month = a customer who has placed a request for the first time ever)**
2. **30-Day repeat rate is defined as percentage of new users who have placed a 2nd order within 30 days of placing their first order.**
   What is the 30-day repeat rate of users acquired in December 2017?
3. **What is the 90-day repeat rate of users acquired in Jan,Feb,March 2018?**
4. **Use logistic regression to predict the 90-day repeat of users acquired in November 2018.**
5. **Plot the distribution of users by frequency of their 90-day repeat (Number of times user repeated within first 90 days)**

Solution 1 : Users acquired every month on a bar chart

1. Finding the first `Date_of_Booking` for every user

```
grouped_df = df[['Profile ID', 'Date_of_Booking']].groupby([df['Profile ID']]).min(
)
```

In [5]:
```
grouped_df = df[['Profile ID', 'Date_of_Booking']].groupby([df['Profile ID']
]).min()[['Date_of_Booking']]
grouped_df.rename(columns = {'Date_of_Booking' : 'First_Day_of_Booking'}, in
place = True)
```

2. Finding the number of new users with respect to every month and year

```
grouped_df.groupby([grouped_df['First_Day_of_Booking'].dt.year, grouped_df['First_D
ay_of_Booking'].dt.month_name()]).count()
```

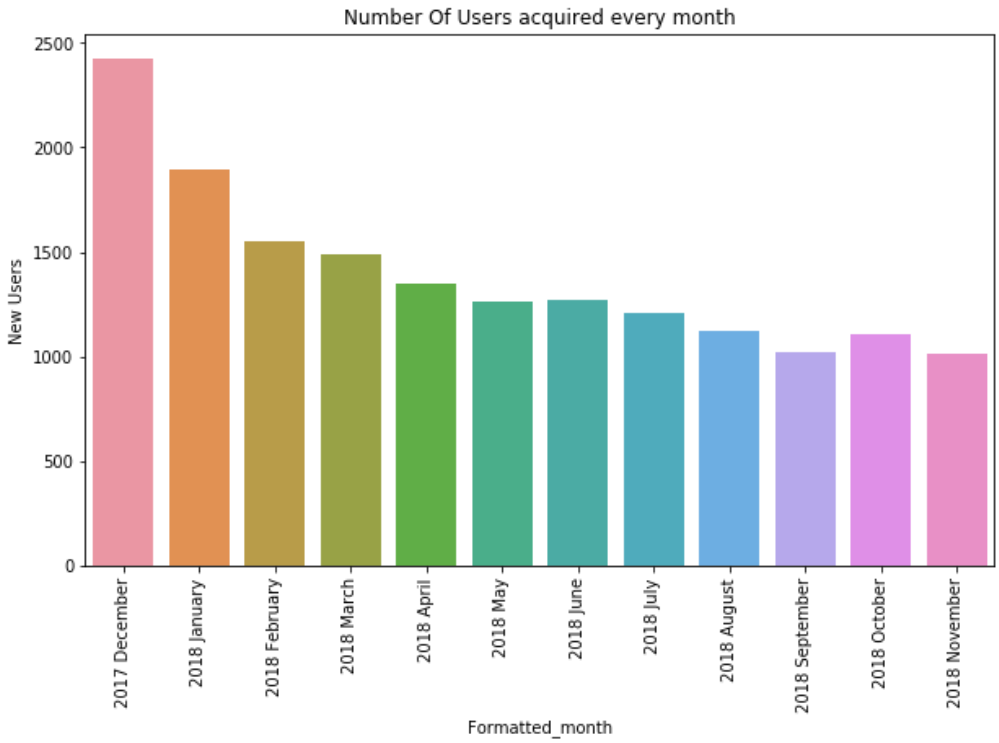
In [6]:
```
num_of_new_users = grouped_df.groupby(
                                        [
                                            grouped_df['First_Day_of_Booking
'].dt.year,
                                            grouped_df['First_Day_of_Booking
'].dt.month
                                        ]
                                      ).count()
num_of_new_users.index.names = ['Year Of Booking', 'Month Of Booking']
num_of_new_users.rename(columns = {'First_Day_of_Booking' : "New Users"}, in
place = True)

num_of_new_users.sort_index(by = ['Year Of Booking', 'Month Of Booking'], in
place = True)

num_of_new_users
```

Out[6]:

		New Users
Year Of Booking	**Month Of Booking**	
2017	**12**	2424
2018	**1**	1892
	2	1549
	3	1490
	4	1346
	5	1264
	6	1273
	7	1208
	8	1120
	9	1020
	10	1110
	11	1015

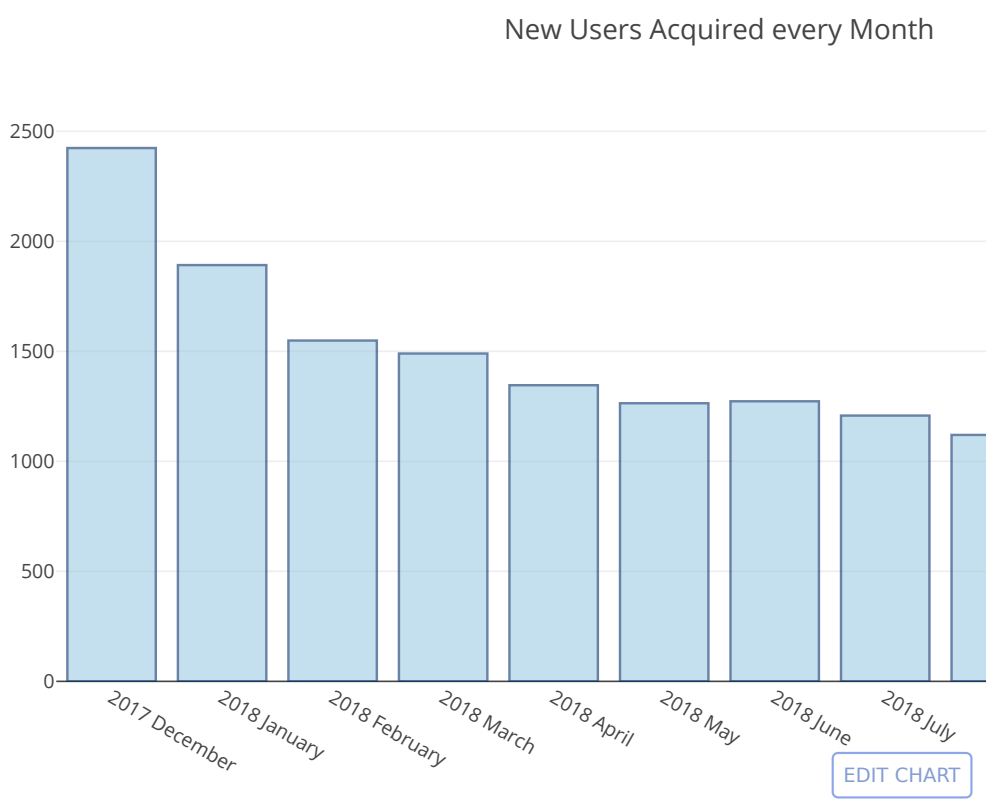
3. Plotting / Visualising the data for `num_of_new_users` (Number of new users) w.r.t. month and year

In [7]:
```
num_of_new_users.reset_index(inplace = True)
num_of_new_users['Formatted_month'] = num_of_new_users['Year Of Booking'].as
type(str) + ' ' + \
                                      num_of_new_users['Month Of Booking'].a
pply(
                                          lambda x : time.strftime("%B", tim
e.strptime(str(x), '%m'))
                                      )

axes = plt.figure(figsize = (10, 6))
ax = sns.barplot(
    x = 'Formatted_month',
    y = 'New Users',
    data = num_of_new_users
)
ax.set_xticklabels(labels = num_of_new_users['Formatted_month'], rotation=90
)
ax.set_title('Number Of Users acquired every month')

plt.show()
```



In [8]:
```
trace0 = plotly.graph_objs.Bar(
    x=num_of_new_users['Formatted_month'],
    y=num_of_new_users['New Users'],
    text = (num_of_new_users['New Users'] / num_of_new_users['New Users'].su
m() * 100).round(2).astype(str).values + "% of new-users were acquired this
month",
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='New Users Acquired every Month',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='New Users Acquired every Month')
```

Out[8]:

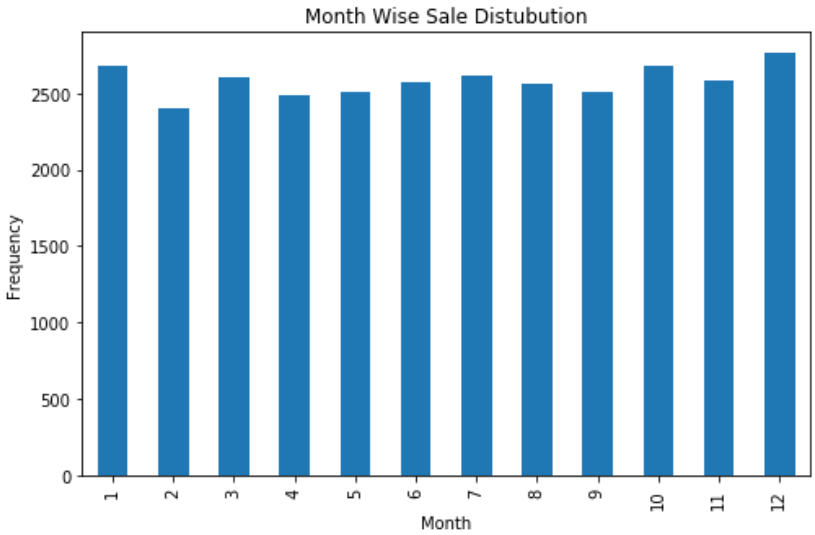


In [9]:
```
# %age changes every month
for i in (np.divide(num_of_new_users['New Users'].values[1:] - num_of_new_us
ers['New Users'].values[:-1],num_of_new_users['New Users'].values[1:]) * 100
):
    print(np.round(i, 3), "%age decrease from the previous month")
```

```
-28.118 %age decrease from the previous month
-22.143 %age decrease from the previous month
-3.96 %age decrease from the previous month
-10.698 %age decrease from the previous month
-6.487 %age decrease from the previous month
0.707 %age decrease from the previous month
-5.381 %age decrease from the previous month
-7.857 %age decrease from the previous month
-9.804 %age decrease from the previous month
8.108 %age decrease from the previous month
-9.36 %age decrease from the previous month
```

In [10]:
```
ax = plt.figure(figsize = (8, 5))
plt.title("Month Wise Sale Distubution")
plt.xlabel("Month")
plt.ylabel("Frequency")
df['Date_of_Booking'].dt.month.value_counts().sort_index().plot(kind = 'bar'
)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0763d2e630>



Insights

- Number of new users in January 2018 dropped the most ie 28.118 %
- Number of New Users in May had dropped almost about to 52.14% of sales in Deceember 2017 **However**, the total users engaging on the service remained roughly the same as seen in the above graph

Solution 2 : What is the 30-day repeat rate of users acquired in December 2017?

All of the following statements have been written in a more explanatory manner.
The code can be compressed to be of 4-5 lines.
None of the code mentioned below makes use of an explicit `for loop` , thereby maintaining the awesome runtime of `numpy` and `pandas` resulting in an extremely optimised code

1. Finding the first `Date_of_Booking` for each user

In [11]:
```
first_booking_data = df.groupby(['Profile ID'])['Date_of_Booking'].min().to_
frame().reset_index()
```

2. Get user data of users who have their first `Date_of_Booking` 's Month == 12 and `Date_of_Booking` 's Year = 2017

In [12]:
```
dec_users_id = first_booking_data[(first_booking_data['Date_of_Booking'].dt.
month == 12) & (first_booking_data['Date_of_Booking'].dt.year == 2017)]['Pro
file ID'].values
dec_users_data = df.loc[df['Profile ID'].isin(dec_users_id)]
dec_users_data.sort_values(['Profile ID', 'Date_of_Booking']).head()
```

Out[12]:

	Transaction_ID	Profile ID	Date_of_Booking	Date_of_Service_Requested	Source	Slot of Booking (Hour of the Day)
8	BGNZX	7	2017-12-07	2017-12-12	D	16
25	IXAJB	18	2017-12-30	2018-01-03	C	14
26	UVFIY	18	2018-11-06	2018-11-07	C	13
35	RCZNH	25	2017-12-30	2018-01-03	A	16
36	IJJUZ	25	2018-05-14	2018-05-15	D	14

3. Filtering the users who've booked more than 1 orders

In [13]:
```
freq_dec_users = dec_users_data.groupby(['Profile ID']).size()
repeat_dec_users = list(freq_dec_users[freq_dec_users > 1].index)
dec_users_data = df.loc[df['Profile ID'].isin(repeat_dec_users)]
```

In [14]:
```
a = dec_users_data.groupby(['Profile ID'])['Date_of_Booking'].nsmallest(2).g
roupby(level = 'Profile ID')
print("30 Day Repeat Rate (for < 30 days)  :", ((a.last() - a.first()).dt.da
ys < 30).sum() * 100 / len(dec_users_id))
print("30 Day Repeat Rate (for <= 30 days) :", ((a.last() - a.first()).dt.da
ys <= 30).sum() * 100 / len(dec_users_id))
```

```
30 Day Repeat Rate (for < 30 days)  : 16.625412541254125
30 Day Repeat Rate (for <= 30 days) : 16.831683168316832
```

Solution 3 : What is the 90-day repeat rate of users acquired in Jan, Feb, March 2018?

Reusing the above code with the following modifications

- `repeat_rate_mon = ['01', '02', '03']`
- `repeat_rate_year = 2018`
- `rep_rate_interval = 90`

In [15]:
```
repeat_rate_mon, repeat_rate_year = ['01', '02', '03'], [2018]
rep_rate_interval = 90

first_booking_data = df.groupby(['Profile ID'])['Date_of_Booking'].min().to_
frame().reset_index()

dec_users_id = first_booking_data[(first_booking_data['Date_of_Booking'].dt.
month.isin(repeat_rate_mon)) & (first_booking_data['Date_of_Booking'].dt.yea
r.isin(repeat_rate_year))]['Profile ID'].values
dec_users_data = df.loc[df['Profile ID'].isin(dec_users_id)]
dec_users_data.sort_values(['Profile ID', 'Date_of_Booking']).head()

freq_dec_users = dec_users_data.groupby(['Profile ID']).size()
repeat_dec_users = list(freq_dec_users[freq_dec_users > 1].index)
dec_users_data = df.loc[df['Profile ID'].isin(repeat_dec_users)]

a = dec_users_data.groupby(['Profile ID'])['Date_of_Booking'].nsmallest(2).g
roupby(level = 'Profile ID')
print("30 Day Repeat Rate (for < 90 days)  :", ((a.last() - a.first()).dt.da
ys < rep_rate_interval).sum() / len(dec_users_id) * 100)
print("30 Day Repeat Rate (for <= 90 days) :", ((a.last() - a.first()).dt.da
ys <= rep_rate_interval).sum() / len(dec_users_id) * 100)
```

```
30 Day Repeat Rate (for < 90 days)  : 20.54350030419793
30 Day Repeat Rate (for <= 90 days) : 20.74629892516731
```

Solution 4 : Predict the 90-day repeat of users acquired in November 2018 using Logistic Regression

1. Finding the indexes of rows containing the record of All Users's first booking.
2. Selecting the above acquired indexes to get their record of first purchase/use of service.

In [16]:
```
vals = df.groupby(['Profile ID'])['Date_of_Booking'].idxmin().values
profile_booking_data = df.loc[vals]
```

3. Finding the number of purchases of each user.
4. Filtering in the users whose number of purchases > 1.
5. Finding the `first_date_of_booking` for each of the above filtered users.
6. Finding the `second_date_of_booking` for each of the above filtered users.
7. Filtering the users whose difference between `second_date_of_booking` and `second_date_of_booking`

- `second_date_of_booking - first_date_of_booking < 90`

In [17]:
```
grouped_df = df[['Profile ID', 'Date_of_Booking']].groupby(['Profile ID']).s
ize().to_frame().rename(columns = {0 : 'No. of times shopped'})

repeat_users = list(grouped_df[grouped_df['No. of times shopped'] > 1].index
)
repeat_users_data = df.loc[df['Profile ID'].isin(repeat_users), ['Profile ID
', 'Date_of_Booking']]

first_two_dates_of_booking_df =  repeat_users_data[['Profile ID', 'Date_of_B
ooking']].groupby(['Profile ID'])['Date_of_Booking'].nsmallest(2).groupby(le
vel = 'Profile ID')
first_date_of_booking = first_two_dates_of_booking_df.first()
second_date_of_booking = first_two_dates_of_booking_df.last()

repeat_rate = ((second_date_of_booking - first_date_of_booking).dt.days < re
p_rate_interval)
repeat_users = repeat_rate[repeat_rate].index.tolist()
```

8. Displaying the data for each users

9. Adding the column `repeat_within_90_days`

- `repeat_within_90_days` is set to `0` if user did not place a second order within 90 days
- `repeat_within_90_days` is set to `1` if user placed a second order within 90 days

In [18]:
```
profile_booking_data.set_index('Profile ID', inplace = True)
profile_booking_data['repeat_within_90_days'] = 0
profile_booking_data.loc[repeat_users, 'repeat_within_90_days'] = 1

train_df = profile_booking_data.copy()
train_df = train_df.reset_index().drop(columns = ['Profile ID'])
train_df.head()
```

Out[18]:

	Transaction_ID	Date_of_Booking	Date_of_Service_Requested	Source	Slot of Booking (Hour of the Day)	repeat_within_90_days
0	BBCHH	2018-05-20	2018-05-21	D	14	0
1	DYDMF	2018-11-10	2018-11-13	C	11	0
2	EZYSA	2018-04-12	2018-04-16	B	8	0
3	YRKFO	2018-03-02	2018-03-05	A	13	0
4	JSFWY	2018-01-06	2018-01-07	C	14	0

1. Calculating Gap

   `Gap` is difference between `Date_of_Service_Requested` and `Date_of_Booking`

In [19]:
```
num_of_orders = df[['Date_of_Booking', 'Profile ID']].groupby('Profile ID')[
'Date_of_Booking'].size()
new_users = num_of_orders[num_of_orders == 1].index

new_user_data = df.loc[df['Profile ID'].isin(num_of_orders[num_of_orders ==
1].index)]
test_data_orig = new_user_data.loc[(new_user_data['Date_of_Booking'].dt.mont
h == 11) & (new_user_data['Date_of_Booking'].dt.year == 2018)]

test_df = test_data_orig.copy()
test_df['repeat_within_90_days'] = [2] * test_df.shape[0]
test_df.drop(columns = ['Profile ID'], inplace = True)
test_df.head()
```

Out[19]:

	Transaction_ID	Date_of_Booking	Date_of_Service_Requested	Source	Slot of Booking (Hour of the Day)	repeat_within_90_day
2	DYDMF	2018-11-10	2018-11-13	C	11	
29	QIYRA	2018-11-27	2018-11-27	B	8	
124	KGYWP	2018-11-11	2018-11-12	C	14	
158	QFPHG	2018-11-04	2018-11-06	D	6	
173	UNMBP	2018-11-22	2018-11-23	D	10	

In [20]:
```
all_df = pd.concat([train_df, test_df], axis = 0)
```

Feature Engineering

- `Gap` is difference between `Date_of_Service_Requested` and `Date_of_Booking`
- `Date_of_Service_Requested_dayofweek` is the day-of-week for `Date_of_Service_Requested`
- `Date_of_Booking_dayofweek` is the day-of-week for `Date_of_Booking`

In [21]:
```
all_df['Gap'] = (all_df['Date_of_Service_Requested'] - all_df['Date_of_Booki
ng']).dt.days
all_df['Date_of_Service_Requested_dayofweek'] = all_df['Date_of_Service_Requ
ested'].dt.day_name()
all_df['Date_of_Booking_dayofweek'] = all_df['Date_of_Booking'].dt.day_name(
)
```

1. Processing the training data

In [22]:
```
all_df.tail()
```

Out[22]:

	Transaction_ID	Date_of_Booking	Date_of_Service_Requested	Source	Slot of Booking (Hour of the Day)	repeat_within_90_d
30727	MKCNH	2018-11-23	2018-11-24	B	14	
30854	SUAQY	2018-11-09	2018-11-09	B	14	
30858	MPLRD	2018-11-19	2018-11-23	D	14	
30859	AYBAL	2018-11-30	2018-12-02	C	6	
30915	BPNYR	2018-11-13	2018-11-17	C	11	

One-Hot-Encoding the categorical variables

```
Source,
Slot of Booking (Hour of the Day),
Date_of_Service_Requested_dayofweek,
Date_of_Booking_dayofweek,
Gap
```

Gap has been added as a categorical variable as it improves accuracy

In [23]:
```
train_test_df = pd.get_dummies(data = all_df, columns = ['Source', 'Slot of
Booking (Hour of the Day)', 'Date_of_Service_Requested_dayofweek', 'Date_of_
Booking_dayofweek', 'Gap'], prefix = ['Source_', 'Slot of Booking (Hour of t
he Day)_', 'Date_of_Service_Requested_dayofweek_', 'Date_of_Booking_dayofwee
k_', 'Gap_'], drop_first = True)
train_test_df.head()
```

Out[23]:

	Transaction_ID	Date_of_Booking	Date_of_Service_Requested	repeat_within_90_days	Source__B	Sourc
0	BBCHH	2018-05-20	2018-05-21	0	0	
1	DYDMF	2018-11-10	2018-11-13	0	0	
2	EZYSA	2018-04-12	2018-04-16	0	1	
3	YRKFO	2018-03-02	2018-03-05	0	0	
4	JSFWY	2018-01-06	2018-01-07	0	0	

Splitting the `train_test_df` into `training_df` and `testing_df`

In [24]:
```
training_df = train_test_df[train_test_df['repeat_within_90_days'] != 2]
testing_df = train_test_df[train_test_df['repeat_within_90_days'] == 2]

x = datetime.datetime.strptime("2018-September", "%Y-%B")
training_df = training_df[training_df['Date_of_Booking'] < x]

training_df.drop(columns = ['Transaction_ID', 'Date_of_Booking', 'Date_of_Se
rvice_Requested'], inplace = True)
testing_df.drop(columns = ['Transaction_ID', 'Date_of_Booking', 'Date_of_Ser
vice_Requested'], inplace = True)
```

Building the model

Getting the `dependent` and `independent` variables

In [25]:
```
X = training_df.iloc[:, 1:].values
y = training_df.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
dom_state=42)
```

There is a need of upsampling the minority class which is unbalanced in the dataset (22.77%)

```
SMOTE synthesises new minority instances between existing (real) minority instances
.
```

Upsampling the minority instances

In [26]:
```
sm = SMOTE(random_state=2)
X_train, y_train = sm.fit_sample(X_train, y_train.ravel())
```

Building the `Logistic Regression` Base Model without any parameters

In [27]:
```
model = LogisticRegression()
model.fit(X_train, y_train)

print(accuracy_score(model.predict(X_test), y_test))
```

```
0.7682387619749448
```

Optimizing the Logistic Regression by finding the right set of parameters using GridSearchCV

```
param_grid = {
    'penalty' : ['l1', 'l2'],
    'tol' : np.geomspace(0.01, 0.0000001, 20),
    'C' : np.arange(0.5, 5, 0.2),
    'fit_intercept' : [True, False]
}
```

In [29]:
```
lc = LogisticRegression(n_jobs = -1)

param_grid = {
    'penalty' : ['l1', 'l2'],
    'tol' : np.geomspace(0.01, 0.0000001, 20),
    'C' : np.arange(0.5, 5, 0.2),
    'fit_intercept' : [True, False]
}

clf = GridSearchCV(lc, param_grid, cv=5, n_jobs=-1, verbose = 2)
clf.fit(X_train, y_train)

print("Best Possible Score in the randomized dataset:", clf.best_score_)
print("Best Parameters :", clf.best_params_)
```

```
Fitting 5 folds for each of 1840 candidates, totalling 9200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  38 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done 516 tasks      | elapsed:   13.0s
[Parallel(n_jobs=-1)]: Done 1328 tasks      | elapsed:   31.8s
[Parallel(n_jobs=-1)]: Done 2460 tasks      | elapsed:   57.6s
[Parallel(n_jobs=-1)]: Done 3920 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 5700 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done 7808 tasks      | elapsed:  3.0min

Best Possible Score in the randomized dataset: 0.6764107203437747
Best Parameters : {'C': 2.8999999999999995, 'fit_intercept': True, 'penalty':
'l1', 'tol': 0.0008858667904100832}

[Parallel(n_jobs=-1)]: Done 9200 out of 9200 | elapsed:  3.6min finished
```

Additional CrossValidation using `RepeatedKFold`

In [30]:
```
kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)

X = training_df.iloc[:, 1:].values
y = training_df.iloc[:, 0].values

sm = SMOTE(random_state=2)
X, y = sm.fit_sample(X, y)

scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    lc = LogisticRegression(C = 2.01, fit_intercept = True, penalty = 'l1')
    lc.fit(X_train, y_train)

    scores.append(accuracy_score(lc.predict(X_test), y_test))

print("\nAverage accuracy_score :", np.mean(scores).round(5) * 100)
```

```
Average accuracy_score : 64.899
```

Making Predictions

Predicting the values for test-set

In [31]:
```
X = training_df.iloc[:, 1:].values
y = training_df.iloc[:, 0].values

sm = SMOTE(random_state=2)
X, y = sm.fit_sample(X, y)

lc = LogisticRegression(C = 2.01, fit_intercept = True, penalty = 'l1')
lc.fit(X, y)

testing_X = testing_df.iloc[:, 1:].values
testing_y = lc.predict(testing_X)
```

In [34]:
```
testing_y_dist = pd.Series(testing_y).value_counts()
print("Predicted 90 day rate :", testing_y_dist[1] * 100 / testing_y_dist.sum())
```

```
Predicted 90 day rate : 25.254237288135593
```

Solution 5 : Plot the distribution of users by frequency of their 90-day repeat

1. Group by Profile ID and finding the number of times new users who placed their second order before 90 days of placing their first Order

```
df[['Profile ID', 'Date_of_Booking']].groupby(['Profile ID'])['Date_of_Booking'].apply(lambda x: ((x - min(x)).dt.days < 90).sum())
```

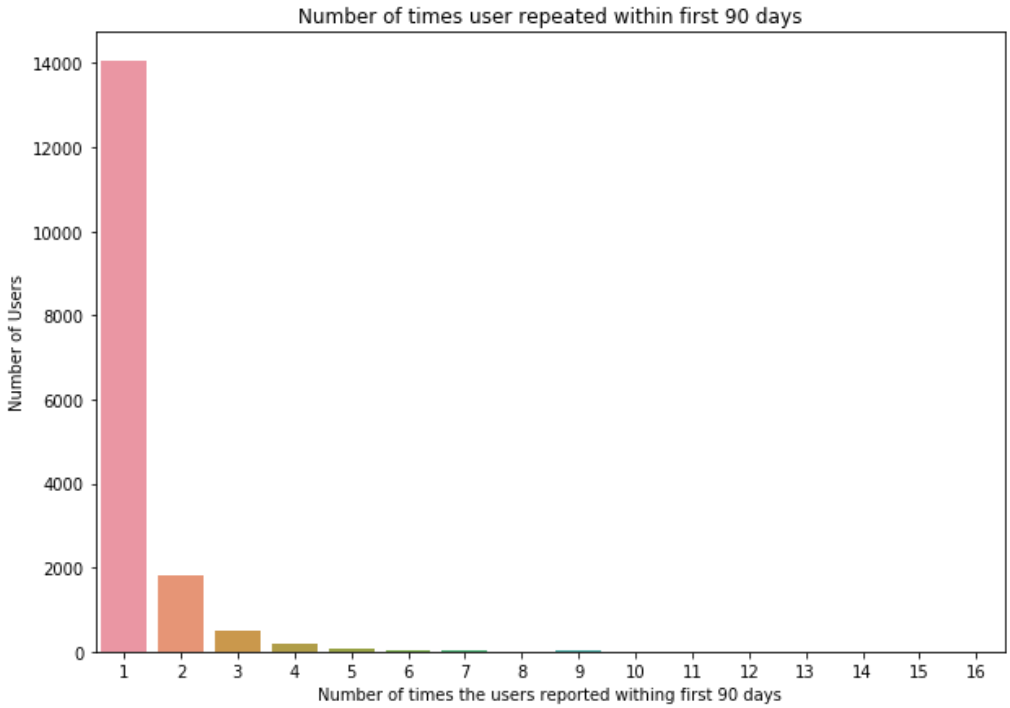
In [35]:
```
number_of_used_within_90days = df[['Profile ID', 'Date_of_Booking']].groupby(['Profile ID'])['Date_of_Booking'].apply(lambda x: ((x - min(x)).dt.days < 90).sum())
number_of_used_within_90days = number_of_used_within_90days.to_frame().rename(columns = {'Date_of_Booking' : 'Number of uses within 90 days'})
number_of_used_within_90days.head()
```

Out[35]:

Profile ID	Number of uses within 90 days
1	1
2	1
3	1
4	1
5	1

2. Plotting Using Seaborn

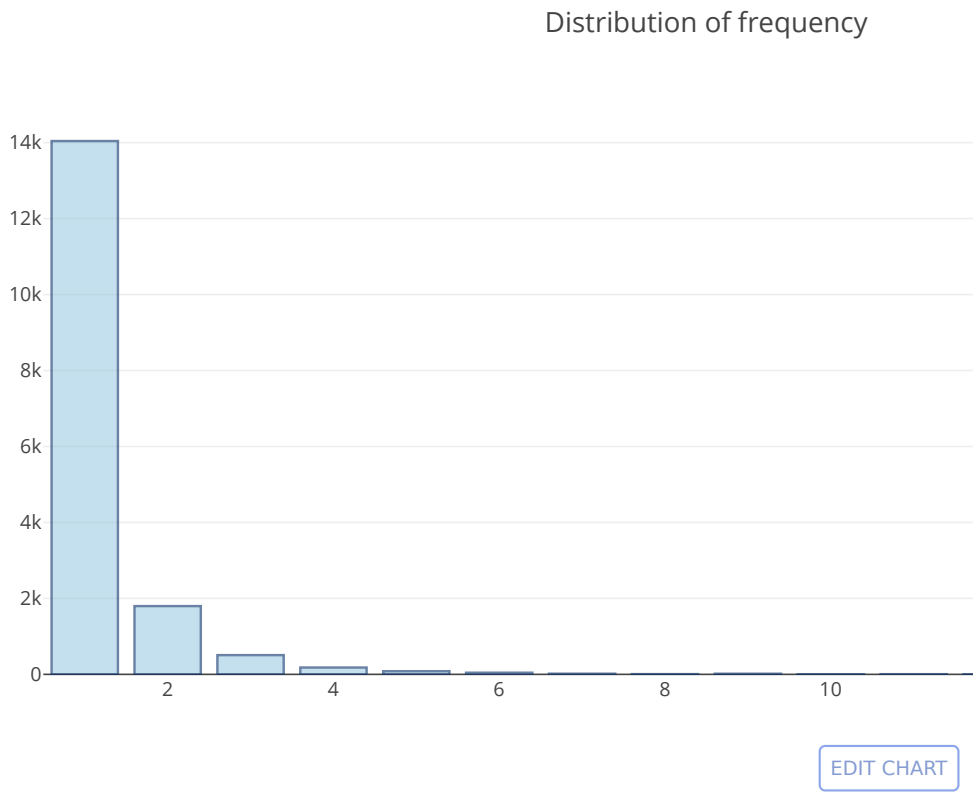
In [36]:
```
freq = number_of_used_within_90days['Number of uses within 90 days']

ax = plt.figure(figsize = (10, 7))
cplot = sns.barplot(
    x = freq.value_counts().sort_index().index,
    y = freq.value_counts().sort_index().values
)

cplot.set_xticklabels(freq.index)
cplot.set_title('Number of times user repeated within first 90 days')

cplot.set(xlabel='Number of times the users reported withing first 90 days',
          ylabel='Number of Users')

plt.show()
```



3. Plotting Using Plotly for more interactivity

In [37]:
```
trace0 = plotly.graph_objs.Bar(
    x=freq.value_counts().sort_index().index,
    y=freq.value_counts().sort_index().values,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='Distribution of frequency',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='Distribution_of_frequency')
```

Out[37]:



Alternative Solution to Problem 2 and 3

In [38]:
```
users = {}
for i, row in df.iterrows():
    if row['Profile ID'] not in users:
        users[row['Profile ID']] = [row['Date_of_Booking']]
    else:
        users[row['Profile ID']] += [row['Date_of_Booking']]

for i in users:
    users[i].sort()

c=0
tc = 0
for i in users:
    if users[i][0].month == 12 and users[i][0].year == 2017:
        tc += 1
        if len(users[i]) > 1 and (users[i][1] - users[i][0]).days <= 30:
            c+= 1
print("Solution 2 :", c/tc * 100)

c=0
tc = 0
for i in users:
    if users[i][0].month in [1, 2, 3]:
        tc += 1
        if len(users[i]) > 1 and (users[i][1] - users[i][0]).days <= 90:
            c+=1
print("Solution 3:", c/tc * 100)
```

```
Solution 2 : 16.831683168316832
Solution 3: 20.74629892516731
```

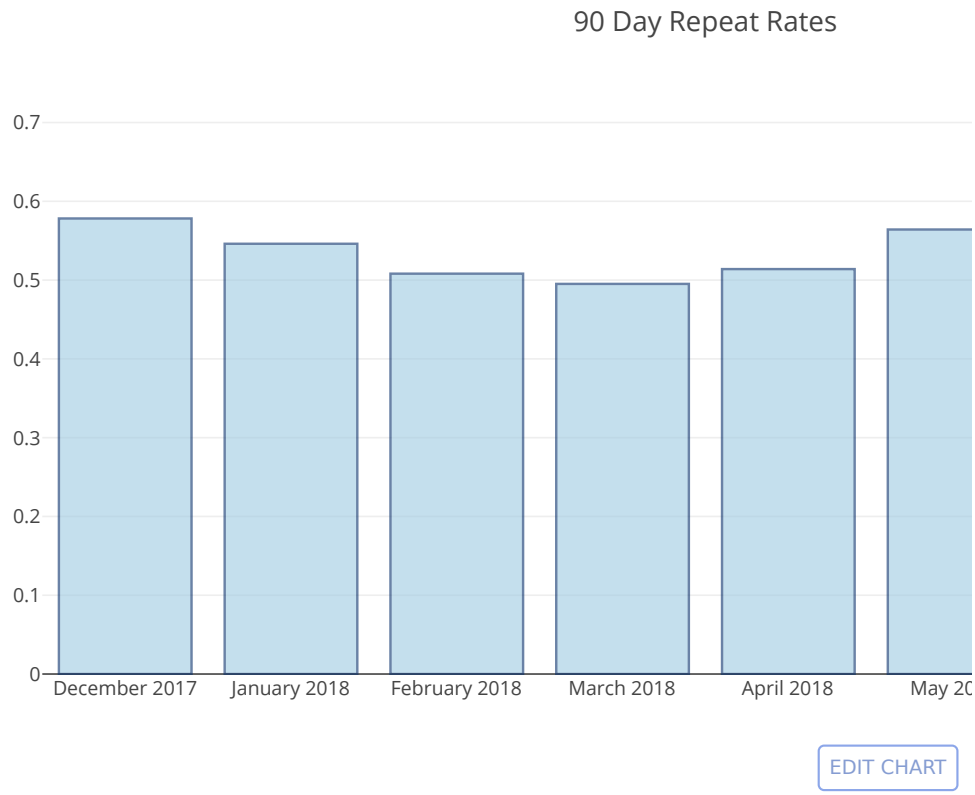
Some Useful Insights

Distribution of 90 Day Repeat Rate (month-wise)

In [39]:
```
formatted_months = []
day_repeat_rates = []

for month in [12] + list(np.arange(1, 8)):

    if month == 12:year = 2017
    else: year = 2018

    repeat_rate_mon, repeat_rate_year = [month], [year]
    rep_rate_interval = 90

    first_booking_data = df.groupby(['Profile ID'])['Date_of_Booking'].min()
.to_frame().reset_index()

    dec_users_id = first_booking_data[(first_booking_data['Date_of_Booking']
.dt.month.isin(repeat_rate_mon)) & (first_booking_data['Date_of_Booking'].dt
.year.isin(repeat_rate_year))]['Profile ID'].values
    dec_users_data = df.loc[df['Profile ID'].isin(dec_users_id)]
    dec_users_data.sort_values(['Profile ID', 'Date_of_Booking']).head()

    freq_dec_users = dec_users_data.groupby(['Profile ID']).size()
    repeat_dec_users = list(freq_dec_users[freq_dec_users > 1].index)
    dec_users_data = df.loc[df['Profile ID'].isin(repeat_dec_users)]

    a = dec_users_data.groupby(['Profile ID'])['Date_of_Booking'].nsmallest(
2).groupby(level = 'Profile ID')

    day_repeat_rates.append(((a.last() - a.first()).dt.days <= rep_rate_inte
rval).sum() / len(repeat_dec_users))
    formatted_months.append(time.strftime("%B", time.strptime(str(month), '%
m')) + f" {year}")
    print(f"90 Day Repeat Rate for {formatted_months[-1]} = {day_repeat_rate
s[-1]}")


trace0 = plotly.graph_objs.Bar(
    x=formatted_months,
    y=day_repeat_rates,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='90 Day Repeat Rates',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='90 Day Repeat Rates')
```

```
90 Day Repeat Rate for December 2017 = 0.5782060785767235
90 Day Repeat Rate for January 2018 = 0.5461077844311377
90 Day Repeat Rate for February 2018 = 0.5081699346405228
90 Day Repeat Rate for March 2018 = 0.4951644100580271
90 Day Repeat Rate for April 2018 = 0.5139664804469274
90 Day Repeat Rate for May 2018 = 0.5641891891891891
90 Day Repeat Rate for June 2018 = 0.5943775100401606
90 Day Repeat Rate for July 2018 = 0.676923076923077
```

Out[39]:



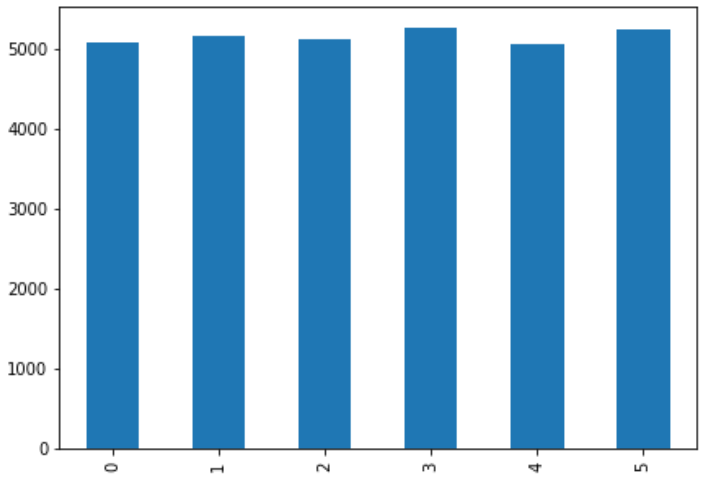
Insight

- 90 Day repeat rates showed a mild decline during the Spring season of 2018

Gap between the `Date of Booking` and `Date of Service`

In [40]:
```
ax = plt.figure(figsize = (7, 5))
(df['Date_of_Service_Requested'] - df['Date_of_Booking']).dt.days.value_coun
ts().sort_index().plot(kind = 'bar')
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7f076095ec88>

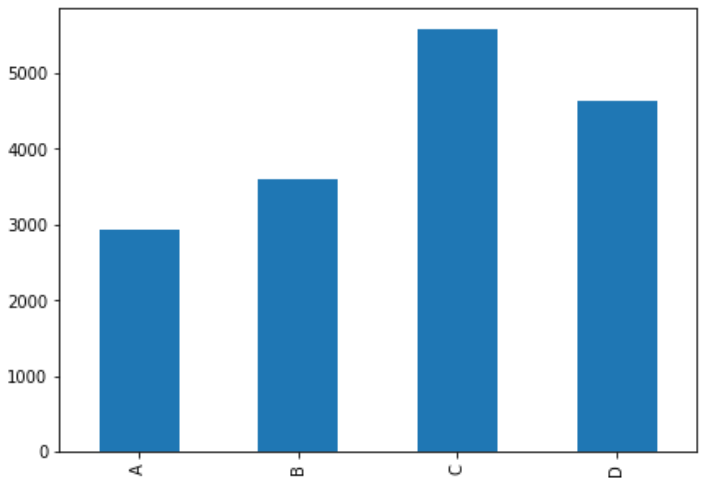


Insight
Gaps are more or less uniform. Nothing special here.

Most Preferred Source

In [41]:
```
ax = plt.figure(figsize = (7, 5))
df[['Profile ID', 'Source']].groupby('Profile ID')['Source'].apply(lambda x
: x.value_counts().index[0]).value_counts().sort_index().plot(kind = 'bar')
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f075e72d8d0>



In [42]:
```
df[['Profile ID', 'Source']].groupby('Profile ID')['Source'].apply(lambda x
: x.value_counts().index[0]).value_counts().sort_index()
```

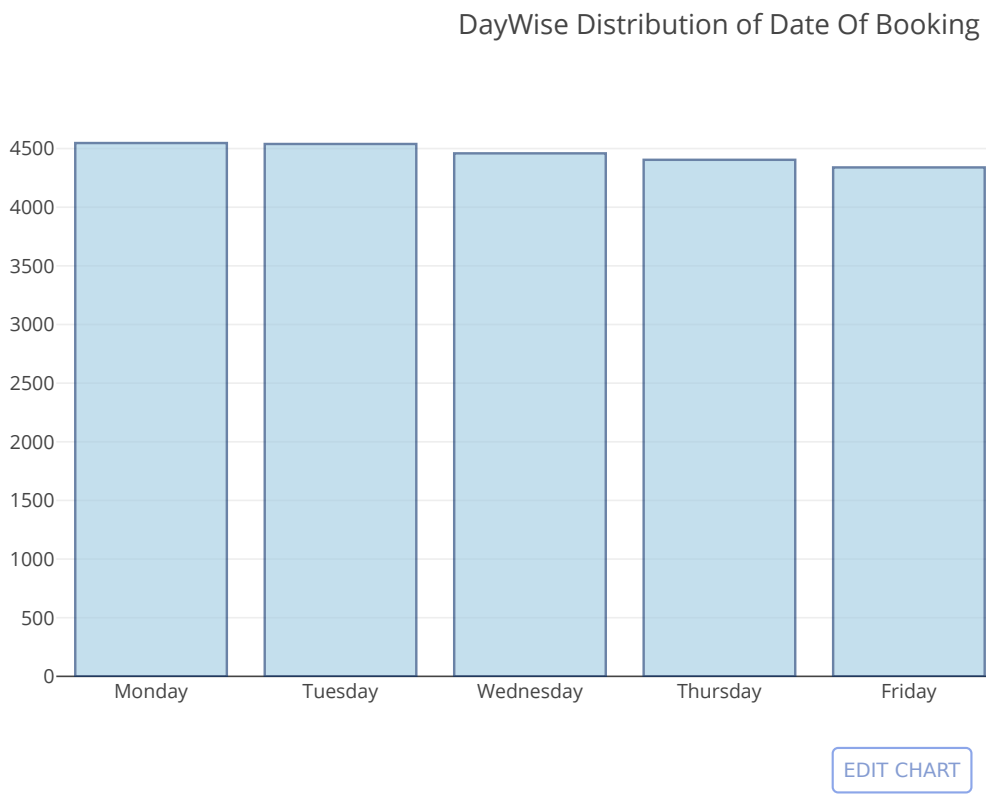
Out[42]:
```
A    2920
B    3600
C    5568
D    4623
Name: Source, dtype: int64
```

> **INSIGHT**
> Source C was Major Source for 38.84% of users

Distribution Of `Date_of_Booking` wrt day

In [43]:
```
booking_date_distribution = df['Date_of_Booking'].dt.dayofweek.value_counts(
)

trace0 = plotly.graph_objs.Bar(
    x=list(calendar.day_name),
    y=booking_date_distribution.values,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='DayWise Distribution of Date Of Booking',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='DayWise Distribution of Date Of Booking')
```

Out[43]:

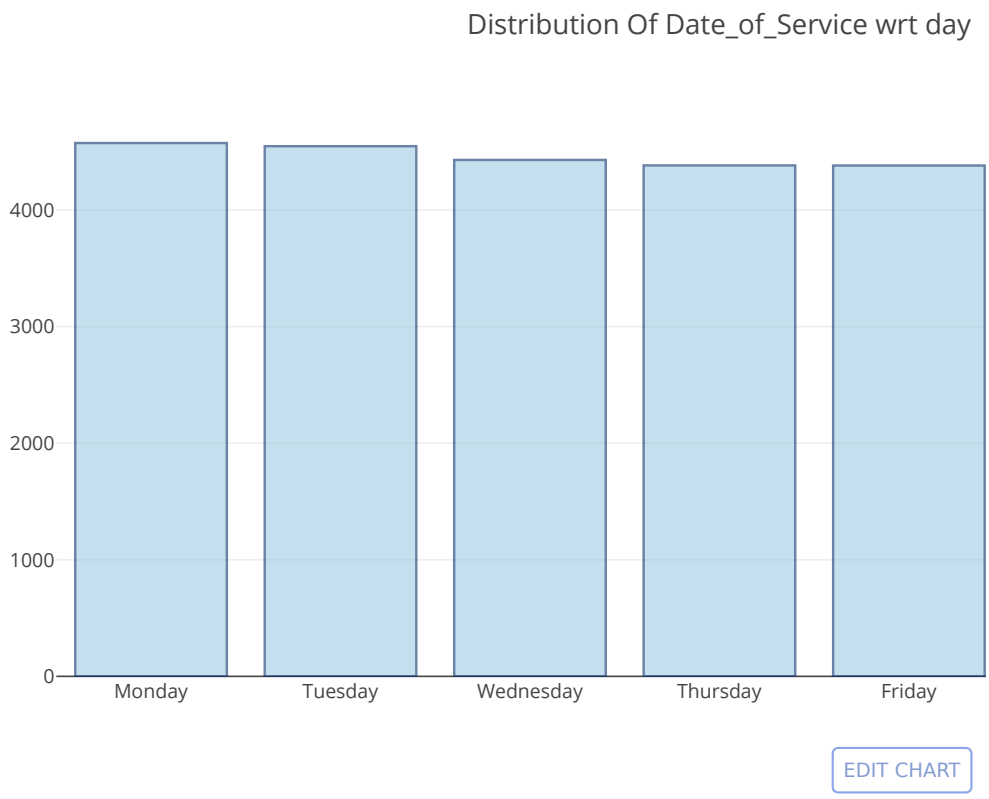


INSIGHT
Distribution of Orders on all weekdays is a Uniform Distribution

Distribution Of `Date_of_Service` wrt day

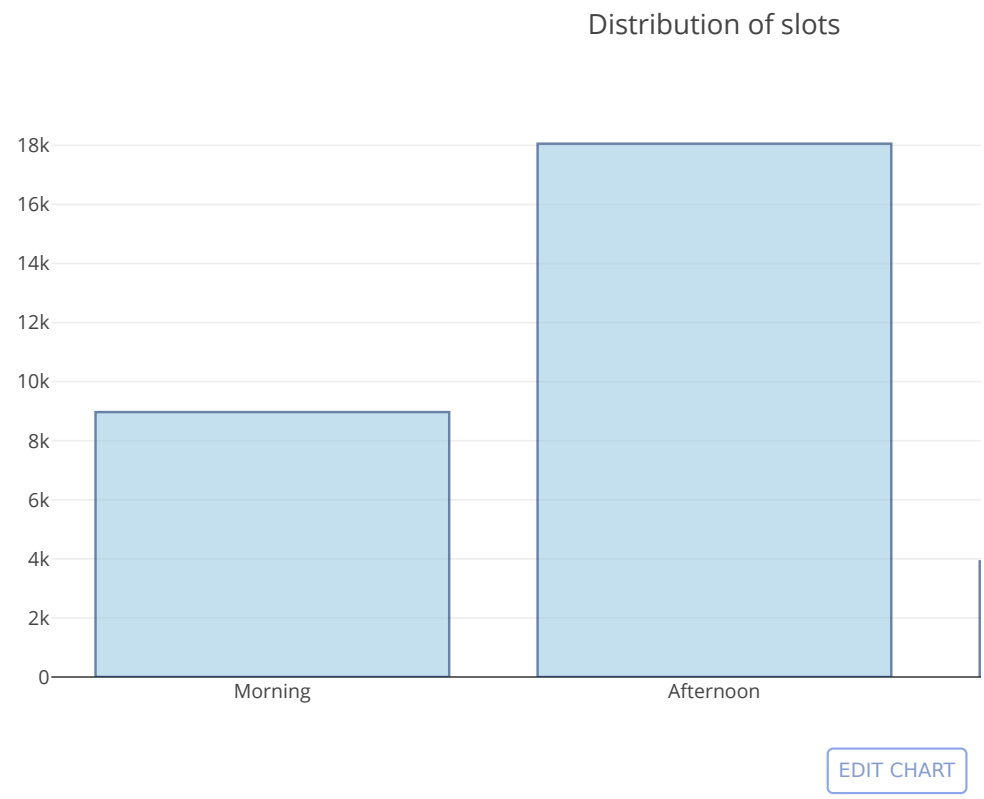
In [44]:
```
service_date_distribution = df['Date_of_Service_Requested'].dt.dayofweek.val
ue_counts()

trace0 = plotly.graph_objs.Bar(
    x=list(calendar.day_name),
    y=service_date_distribution.values,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='Distribution Of Date_of_Service wrt day',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='Distribution Of Date_of_Service wrt day')
```

Out[44]:



In [45]:
```
df['Morning']   = 0       # From 6 to 12
df['Afternoon'] = 0       # From 12 - 16
df['Evening']   = 0       # From 16+

df.loc[df['Slot of Booking (Hour of the Day)'].between(6, 12, inclusive = True), 'Morning'] = 1
df.loc[df['Slot of Booking (Hour of the Day)'].between(13, 16, inclusive = True), 'Afternoon'] = 1
df.loc[df['Slot of Booking (Hour of the Day)'] >= 17, 'Evening'] = 1

trace0 = plotly.graph_objs.Bar(
    x=['Morning', 'Afternoon', 'Evening'],
    y= [df['Morning'].sum(), df['Afternoon'].sum(), df['Evening'].sum()],
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='Distribution of slots',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='Distribution_of_slots1')
```

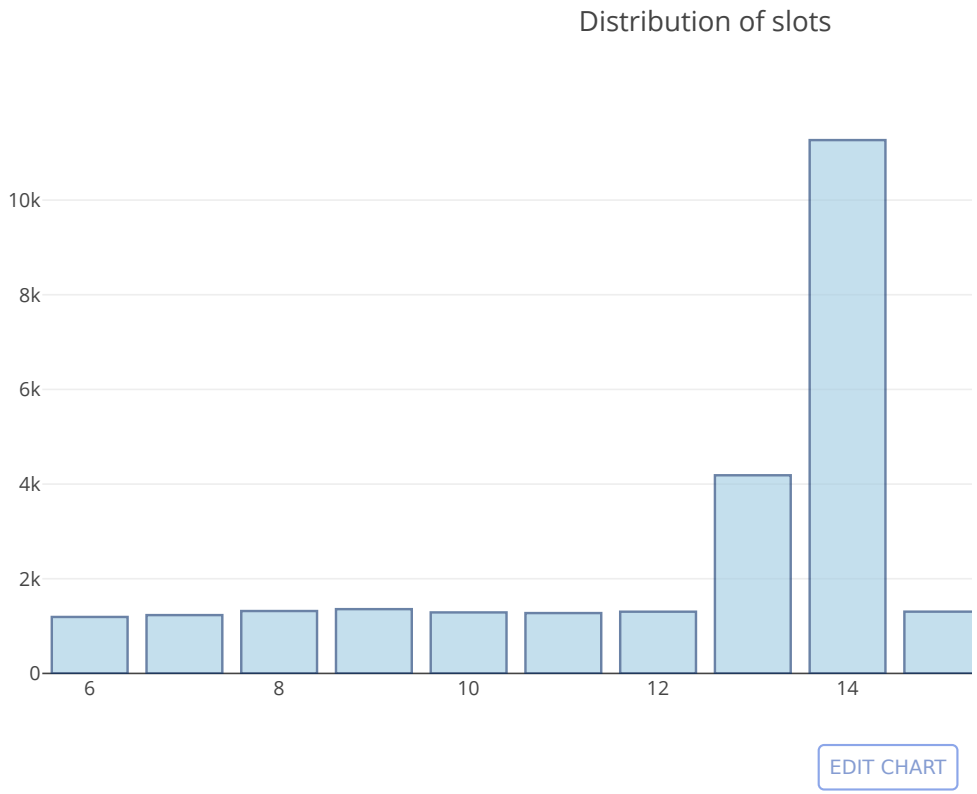
Out[45]:



INSIGHT
Roughly, 58.83% users chose their slots in Afternoon

In [46]:
```
freq_slots = df['Slot of Booking (Hour of the Day)'].value_counts().sort_ind
ex()

trace0 = plotly.graph_objs.Bar(
    x=freq_slots.index,
    y= freq_slots.values,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = plotly.graph_objs.Layout(
    title='Distribution of slots',
)

fig = plotly.graph_objs.Figure(data=data, layout=layout)
plotly.plotly.iplot(fig, filename='Distribution-of-slots')
```

Out[46]:



INSIGHT Roughly 38.83% of users chose the slot of 2PM

Closing Note

I would like to express my appreciation for considering me to be a part of the internship technical round.

I hope you like this small sampling of my work.
I **SINCERELY** want to work with a company as good as UrbanClap
I look forward to create many such documents and more with your team

Please get in touch just in case if you think I missed anyhting at the below contact details

- Phone - `+91-9871966592 (WhatsApp Active)`
- EMail - `revantgupta2@gmail.com`
- LinkedIN - `https://www.linkedin.com/in/revantg/`
- Twitter - `https://twitter.com/revant_g`