Team 18 - Uber and Lyft Price Prediction

1. Problem Statement

In this project, we analyze tabular data from Uber using a variety of machine learning methods and tools, including numpy, pandas, and matplotlib.

We examine several columns in the table, try to tie them to one another, and identify a relationship between those two. We look for and analyze the important variables, such as the date and the month, that enable the Uber Company to improve its operations by concentrating on those services and making the necessary adjustments.

2. Importing data

In [6]:
```
import pandas as pd

uberdata = pd.read_csv('rideshare_kaggle.csv')
```

3.EDA

Exploratory data analysis is the crucial process of doing preliminary analyses on data in order to find patterns, identify anomalies, test hypotheses, and double-check assumptions with the aid of summary statistics and graphical representations. Understanding the data first and attempting to glean as many insights from it as possible is a smart strategy. Understanding the data at hand is the foundation of EDA.

In [7]:
```
uberdata.head()
```

Out[7]:

	id	timestamp	hour	day	month	datetime	timezone	source
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	1.544953e+09	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square
1	4bd23055-6827-41c6-b23b-3c491f24e74d	1.543284e+09	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square
2	981a3613-77af-4620-a42a-0c0866077d1e	1.543367e+09	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	1.543554e+09	4	30	11	2018-11-30 04:53:02	America/New_York	Haymarket Square
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	1.543463e+09	3	29	11	2018-11-29 03:49:20	America/New_York	Haymarket Square

5 rows × 57 columns

In [8]:
```
uberdata.shape
```

Out[8]:
```
(693071, 57)
```

In [9]:
```
uberdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 57 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   id                             693071 non-null  object 
 1   timestamp                      693071 non-null  float64
 2   hour                           693071 non-null  int64  
 3   day                            693071 non-null  int64  
 4   month                          693071 non-null  int64  
 5   datetime                       693071 non-null  object 
 6   timezone                       693071 non-null  object 
 7   source                         693071 non-null  object 
 8   destination                    693071 non-null  object 
 9   cab_type                       693071 non-null  object 
 10  product_id                     693071 non-null  object 
 11  name                           693071 non-null  object 
 12  price                          637976 non-null  float64
 13  distance                       693071 non-null  float64
 14  surge_multiplier               693071 non-null  float64
 15  latitude                       693071 non-null  float64
 16  longitude                      693071 non-null  float64
 17  temperature                    693071 non-null  float64
 18  apparentTemperature            693071 non-null  float64
 19  short_summary                  693071 non-null  object 
 20  long_summary                   693071 non-null  object 
 21  precipIntensity                693071 non-null  float64
 22  precipProbability              693071 non-null  float64
 23  humidity                       693071 non-null  float64
 24  windSpeed                      693071 non-null  float64
 25  windGust                       693071 non-null  float64
 26  windGustTime                   693071 non-null  int64  
 27  visibility                     693071 non-null  float64
 28  temperatureHigh                693071 non-null  float64
 29  temperatureHighTime            693071 non-null  int64  
 30  temperatureLow                 693071 non-null  float64
 31  temperatureLowTime             693071 non-null  int64  
 32  apparentTemperatureHigh        693071 non-null  float64
 33  apparentTemperatureHighTime    693071 non-null  int64  
 34  apparentTemperatureLow         693071 non-null  float64
 35  apparentTemperatureLowTime     693071 non-null  int64  
 36  icon                           693071 non-null  object 
 37  dewPoint                       693071 non-null  float64
 38  pressure                       693071 non-null  float64
 39  windBearing                    693071 non-null  int64  
 40  cloudCover                     693071 non-null  float64
 41  uvIndex                        693071 non-null  int64  
 42  visibility.1                   693071 non-null  float64
 43  ozone                          693071 non-null  float64
 44  sunriseTime                    693071 non-null  int64  
 45  sunsetTime                     693071 non-null  int64  
 46  moonPhase                      693071 non-null  float64
 47  precipIntensityMax             693071 non-null  float64
 48  uvIndexTime                    693071 non-null  int64  
 49  temperatureMin                 693071 non-null  float64
 50  temperatureMinTime             693071 non-null  int64  
 51  temperatureMax                 693071 non-null  float64
```

```
 52  temperatureMaxTime           693071 non-null  int64
 53  apparentTemperatureMin       693071 non-null  float64
 54  apparentTemperatureMinTime   693071 non-null  int64
 55  apparentTemperatureMax       693071 non-null  float64
 56  apparentTemperatureMaxTime   693071 non-null  int64
dtypes: float64(29), int64(17), object(11)
memory usage: 301.4+ MB
```

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

Out[10]:

	timestamp	hour	day	month	price	dis
count	6.930710e+05	693071.000000	693071.000000	693071.000000	637976.000000	693071.0
mean	1.544046e+09	11.619137	17.794365	11.586684	16.545125	2.1
std	6.891925e+05	6.948114	9.982286	0.492429	9.324359	1.1
min	1.543204e+09	0.000000	1.000000	11.000000	2.500000	0.0
25%	1.543444e+09	6.000000	13.000000	11.000000	9.000000	1.2
50%	1.543737e+09	12.000000	17.000000	12.000000	13.500000	2.1
75%	1.544828e+09	18.000000	28.000000	12.000000	22.500000	2.9
max	1.545161e+09	23.000000	30.000000	12.000000	97.500000	7.8

8 rows × 46 columns

In [11]:
```
#Finding the null Values
uberdata.isnull().sum()
```

Out[11]:

id	0
timestamp	0
hour	0
day	0
month	0
datetime	0
timezone	0
source	0
destination	0
cab_type	0
product_id	0
name	0
price	55095
distance	0
surge_multiplier	0
latitude	0
longitude	0
temperature	0
apparentTemperature	0
short_summary	0
long_summary	0
precipIntensity	0
precipProbability	0
humidity	0
windSpeed	0
windGust	0
windGustTime	0
visibility	0
temperatureHigh	0
temperatureHighTime	0
temperatureLow	0
temperatureLowTime	0
apparentTemperatureHigh	0
apparentTemperatureHighTime	0
apparentTemperatureLow	0
apparentTemperatureLowTime	0
icon	0
dewPoint	0
pressure	0
windBearing	0
cloudCover	0
uvIndex	0
visibility.1	0
ozone	0
sunriseTime	0
sunsetTime	0
moonPhase	0
precipIntensityMax	0
uvIndexTime	0
temperatureMin	0
temperatureMinTime	0
temperatureMax	0
temperatureMaxTime	0
apparentTemperatureMin	0
apparentTemperatureMinTime	0
apparentTemperatureMax	0

```
apparentTemperatureMaxTime         0
dtype: int64
```

4. Feature Engineering

In essence, all machine learning algorithms generate outputs using some form of input data. The features contained in these input data often take the form of structured columns. In order for algorithms to function successfully, features must have a particular attribute. Feature engineering becomes necessary in this situation.

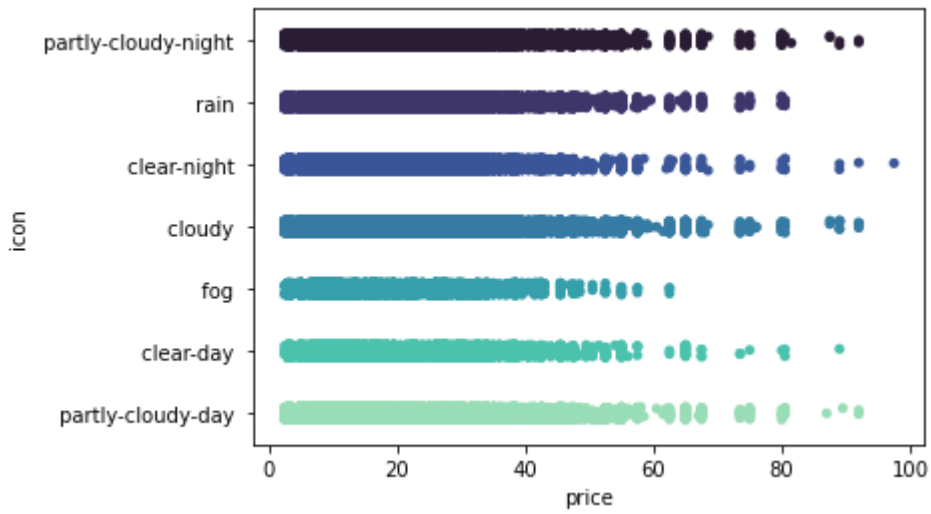
The key objectives of feature engineering are Creating an appropriate input dataset that complies with the demands of the machine learning algorithm and Improving how well machine learning models perform.

Visualization

In [12]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import seaborn as sns
import pandas as pd
```

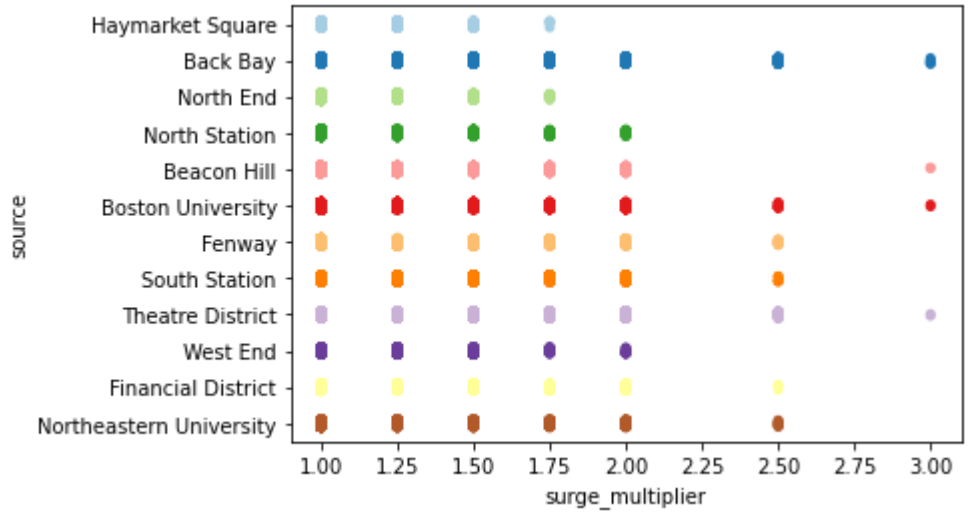
In [15]:
```
#finding the price range based on weather condition
sns.stripplot(data=uberdata, x='price', y='icon',palette='mako')
```

Out[15]: <AxesSubplot:xlabel='price', ylabel='icon'>



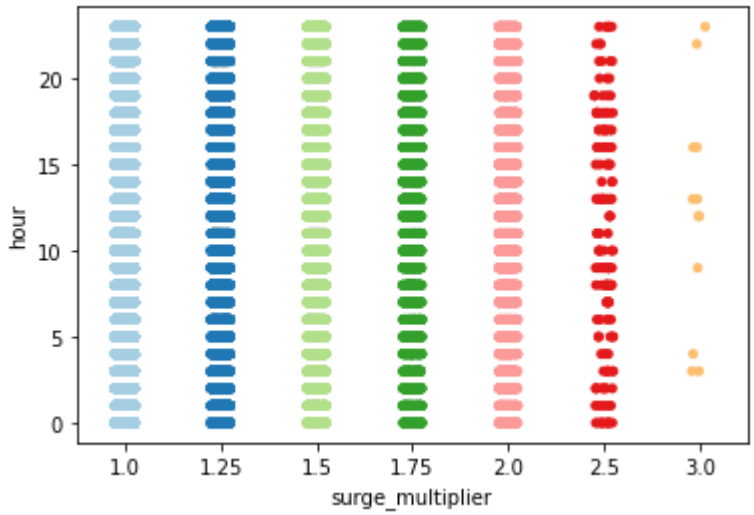
In [19]:
```
#Strip plot based on Source of destination
sns.stripplot(data=uberdata, x='surge_multiplier', y='source',palette='Pai
```

Out[19]: <AxesSubplot:xlabel='surge_multiplier', ylabel='source'>



In [20]:
```
#Strip plot based on hour vs price
sns.stripplot(data=uberdata, x='surge_multiplier', y='hour',palette='Paire
```

Out[20]: <AxesSubplot:xlabel='surge_multiplier', ylabel='hour'>



In [21]:
```
uberdata['timestamp'].head()
```

Out[21]:
```
0    1.544953e+09
1    1.543284e+09
2    1.543367e+09
3    1.543554e+09
4    1.543463e+09
Name: timestamp, dtype: float64
```

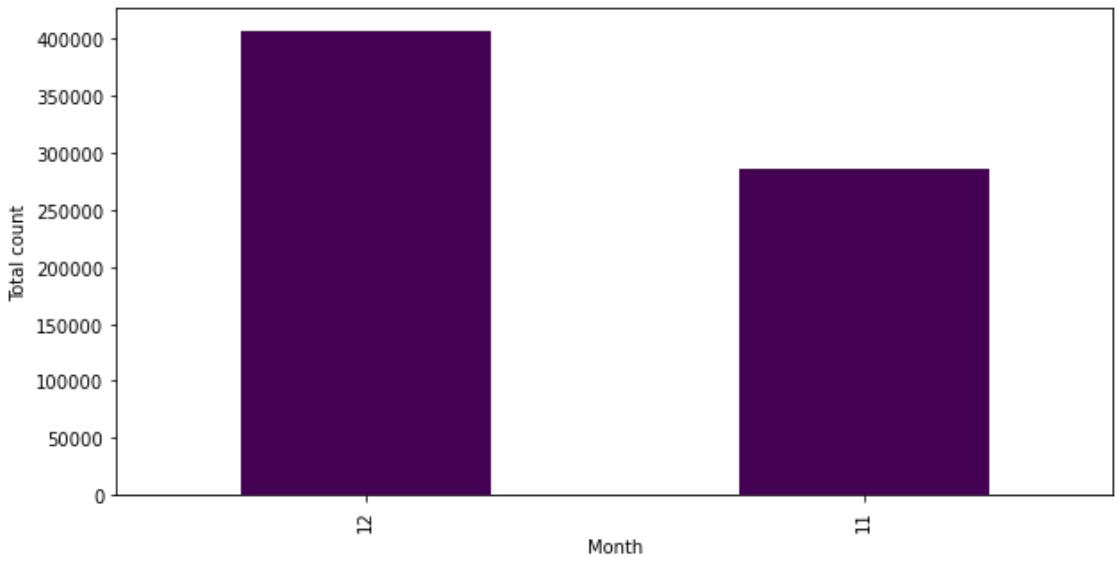
In [23]:
```
from datetime import datetime
timest1 = 1544952608
timest2 = 1543284024
timest3 = 1543818483
timest4 = 1543594384
timest5 = 1544728504
dt1 = datetime.fromtimestamp(timest1)
dt2 = datetime.fromtimestamp(timest2)
dt3 = datetime.fromtimestamp(timest3)
dt4 = datetime.fromtimestamp(timest4)
dt5 = datetime.fromtimestamp(timest5)

print("date_object =", dt1)
print("date_object =", dt2)
print("date_object =", dt3)
print("date_object =", dt4)
print("date_object =", dt5)
```

```
date_object = 2018-12-16 04:30:08
date_object = 2018-11-26 21:00:24
date_object = 2018-12-03 01:28:03
date_object = 2018-11-30 11:13:04
date_object = 2018-12-13 14:15:04
```

We learn that our data is from the year 2018 and only covers the months of November and December

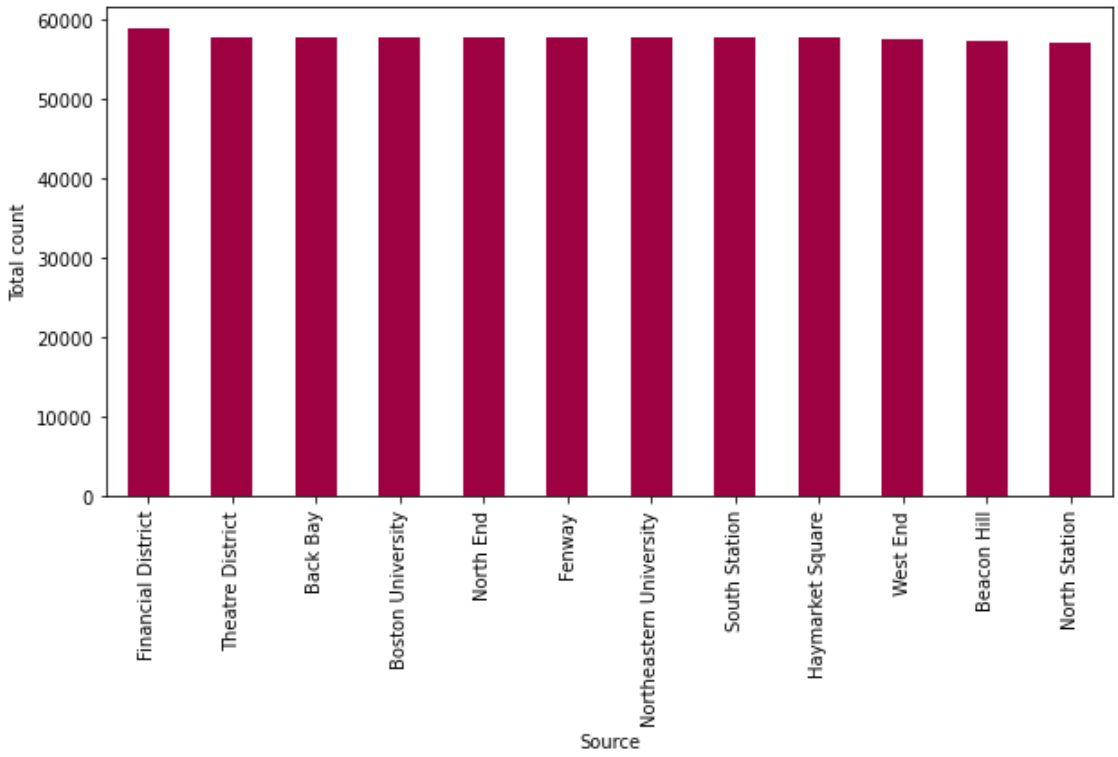
In [29]:
```
#Bar plot to find the count of rides in November and December
uberdata['month'].value_counts().plot(kind='bar', figsize=(10,5), cm='viri
plt.xlabel("Month")
plt.ylabel("Total count")
```

Out[29]:
```
Text(0, 0.5, 'Total count')
```



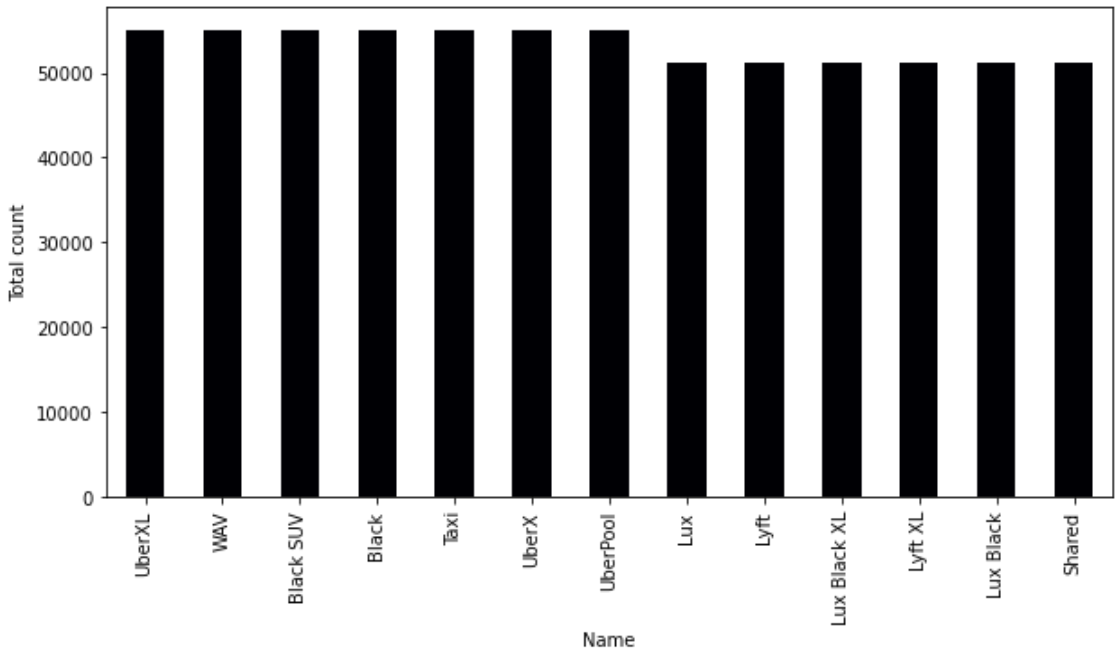
In [36]:
```
#Bar plot to find the rides based on source of the destinations
uberdata['source'].value_counts().plot(kind='bar', figsize=(10,5), cmap='S
plt.xlabel("Source")
plt.ylabel("Total count")
```

Out[36]: Text(0, 0.5, 'Total count')



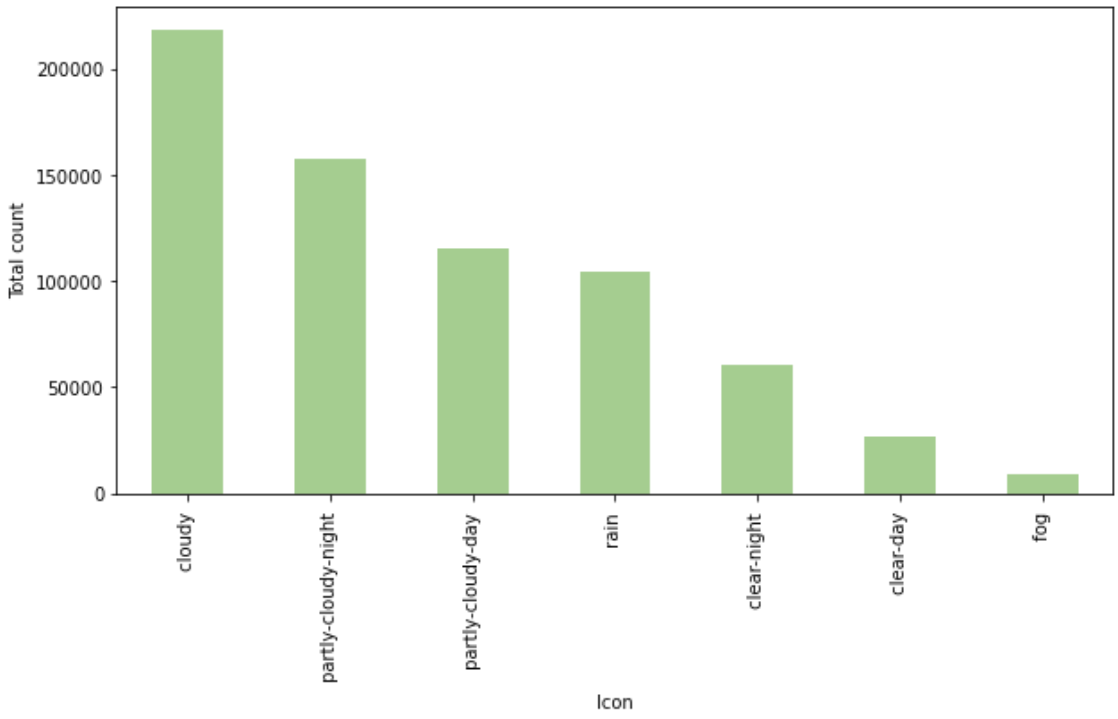
In [35]:
```
#Bar plot to find the rides based on car type
uberdata['name'].value_counts().plot(kind='bar', figsize=(10,5), cmap='mag
plt.xlabel("Name")
plt.ylabel("Total count")
```

Out[35]: Text(0, 0.5, 'Total count')



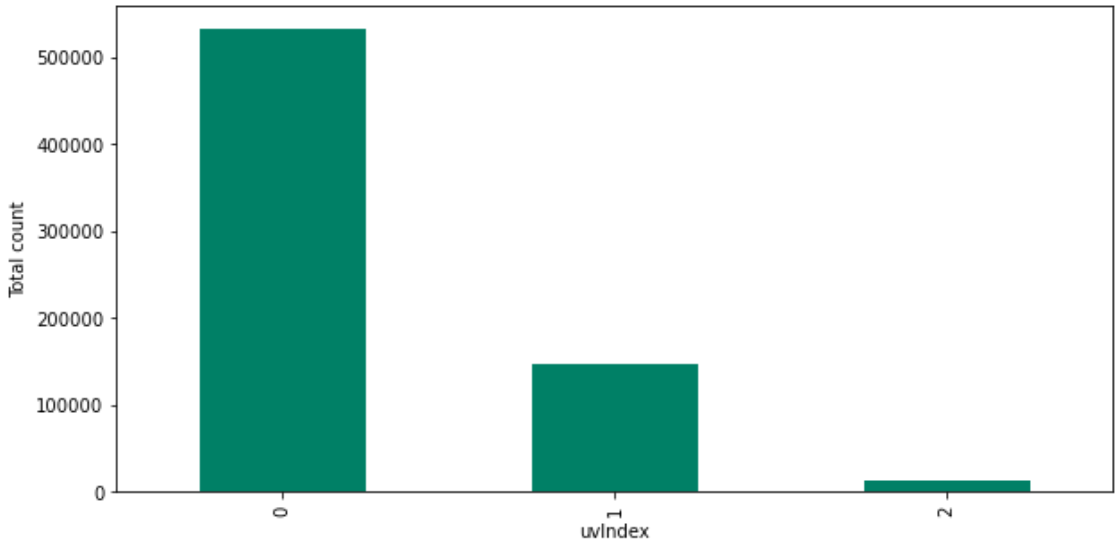
In [41]:
```
# based on the icon
uberdata['icon'].value_counts().plot(kind='bar', figsize=(10,5), cmap='cre
plt.xlabel("Icon")
plt.ylabel("Total count")
```

Out[41]: Text(0, 0.5, 'Total count')



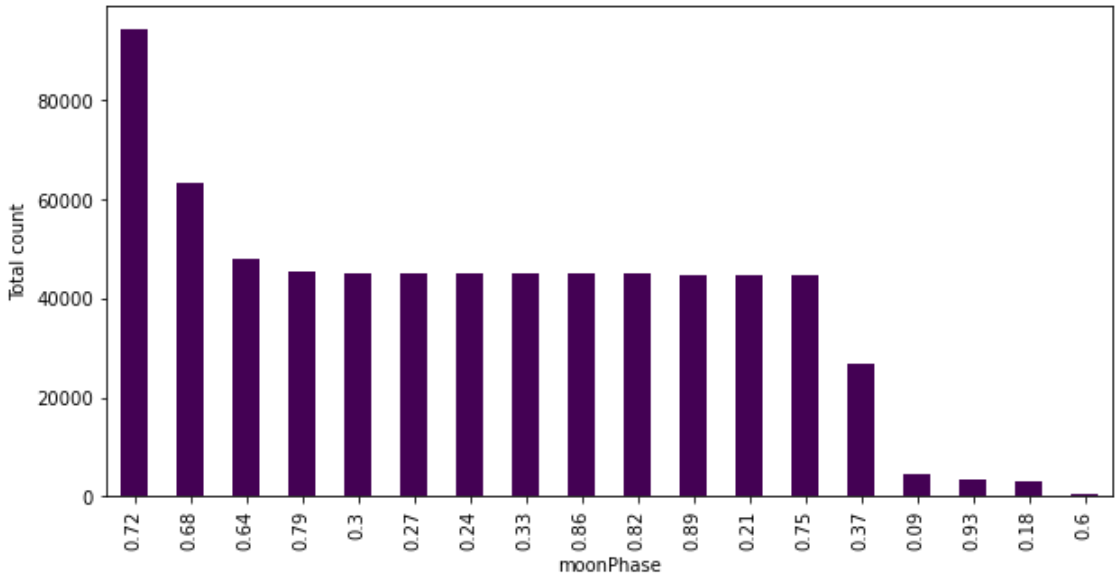
In [42]:
```
uberdata['uvIndex'].value_counts().plot(kind='bar', figsize=(10,5), cmap='
plt.xlabel("uvIndex")
plt.ylabel("Total count")
```

Out[42]: Text(0, 0.5, 'Total count')



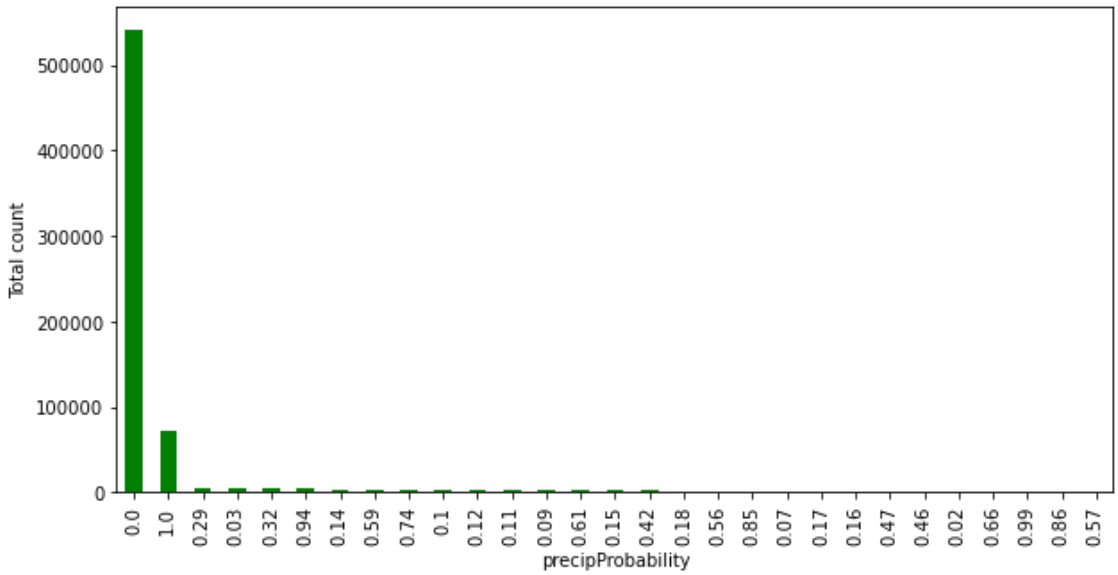
In [43]:
```
#Total Count based on Moonphase
uberdata['moonPhase'].value_counts().plot(kind='bar', figsize=(10,5), cmap
plt.xlabel("moonPhase")
plt.ylabel("Total count")
```

Out[43]: Text(0, 0.5, 'Total count')



In [44]:
```
#graph price probability vs total count
plt.xlabel("precipProbability")
plt.ylabel("Total count")
uberdata['precipProbability'].value_counts().plot(kind='bar', figsize=(10,
```

Out[44]: <AxesSubplot:xlabel='precipProbability', ylabel='Total count'>



Label Encoding

In [45]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
```

In [46]:
```
uberdata.dtypes
```

Out[46]:

id	object
timestamp	float64
hour	int64
day	int64
month	int64
datetime	object
timezone	object
source	object
destination	object
cab_type	object
product_id	object
name	object
price	float64
distance	float64
surge_multiplier	float64
latitude	float64
longitude	float64
temperature	float64
apparentTemperature	float64
short_summary	object
long_summary	object
precipIntensity	float64
precipProbability	float64
humidity	float64
windSpeed	float64
windGust	float64
windGustTime	int64
visibility	float64
temperatureHigh	float64
temperatureHighTime	int64
temperatureLow	float64
temperatureLowTime	int64
apparentTemperatureHigh	float64
apparentTemperatureHighTime	int64
apparentTemperatureLow	float64
apparentTemperatureLowTime	int64
icon	object
dewPoint	float64
pressure	float64
windBearing	int64
cloudCover	float64
uvIndex	int64
visibility.1	float64
ozone	float64
sunriseTime	int64
sunsetTime	int64
moonPhase	float64
precipIntensityMax	float64
uvIndexTime	int64
temperatureMin	float64
temperatureMinTime	int64
temperatureMax	float64
temperatureMaxTime	int64
apparentTemperatureMin	float64
apparentTemperatureMinTime	int64
apparentTemperatureMax	float64

```
apparentTemperatureMaxTime       int64
dtype: object
```

In [48]:
```
uberdata['id']= label_encoder.fit_transform(uberdata['id'])
uberdata['datetime']= label_encoder.fit_transform(uberdata['datetime'])
uberdata['timezone']= label_encoder.fit_transform(uberdata['timezone'])
uberdata['destination']= label_encoder.fit_transform(uberdata['destination
uberdata['product_id']= label_encoder.fit_transform(uberdata['product_id']
uberdata['short_summary']= label_encoder.fit_transform(uberdata['short_sum
uberdata['long_summary']= label_encoder.fit_transform(uberdata['long_summa
```

In [49]:
```
uberdata['cab_type'].unique()
```

Out[49]:
```
array(['Lyft', 'Uber'], dtype=object)
```

In [50]:
```
uberdata['cab_type']= label_encoder.fit_transform(uberdata['cab_type'])

print("Class mapping of cab_type: ")
for i, item in enumerate(label_encoder.classes_):
    print(item, "-->", i)
```

```
Class mapping of cab_type:
Lyft --> 0
Uber --> 1
```

In [51]:
```
uberdata['name']= label_encoder.fit_transform(uberdata['name'])

print("Class mapping of Name: ")
for i, item in enumerate(label_encoder.classes_):
    print(item, "-->", i)
```

```
Class mapping of Name:
Black --> 0
Black SUV --> 1
Lux --> 2
Lux Black --> 3
Lux Black XL --> 4
Lyft --> 5
Lyft XL --> 6
Shared --> 7
Taxi --> 8
UberPool --> 9
UberX --> 10
UberXL --> 11
WAV --> 12
```

In [52]:
```
uberdata['source']= label_encoder.fit_transform(uberdata['source'])

print("Class mapping of Source: ")
for i, item in enumerate(label_encoder.classes_):
    print(item, "-->", i)
```

```
Class mapping of Source:
Back Bay --> 0
Beacon Hill --> 1
Boston University --> 2
Fenway --> 3
Financial District --> 4
Haymarket Square --> 5
North End --> 6
North Station --> 7
Northeastern University --> 8
South Station --> 9
Theatre District --> 10
West End --> 11
```

In [53]:
```
uberdata['icon']= label_encoder.fit_transform(uberdata['icon'])

print("Class mapping of Icon: ")
for i, item in enumerate(label_encoder.classes_):
    print(item, "-->", i)
```

```
Class mapping of Icon:
 clear-day  --> 0
 clear-night  --> 1
 cloudy  --> 2
 fog  --> 3
 partly-cloudy-day  --> 4
 partly-cloudy-night  --> 5
 rain  --> 6
```

In [54]:
```
uberdata.dtypes
```

Out[54]:

id	int32
timestamp	float64
hour	int64
day	int64
month	int64
datetime	int32
timezone	int32
source	int32
destination	int32
cab_type	int32
product_id	int32
name	int32
price	float64
distance	float64
surge_multiplier	float64
latitude	float64
longitude	float64
temperature	float64
apparentTemperature	float64
short_summary	int32
long_summary	int32
precipIntensity	float64
precipProbability	float64
humidity	float64
windSpeed	float64
windGust	float64
windGustTime	int64
visibility	float64
temperatureHigh	float64
temperatureHighTime	int64
temperatureLow	float64
temperatureLowTime	int64
apparentTemperatureHigh	float64
apparentTemperatureHighTime	int64
apparentTemperatureLow	float64
apparentTemperatureLowTime	int64
icon	int32
dewPoint	float64
pressure	float64
windBearing	int64
cloudCover	float64
uvIndex	int64
visibility.1	float64
ozone	float64
sunriseTime	int64
sunsetTime	int64
moonPhase	float64
precipIntensityMax	float64
uvIndexTime	int64
temperatureMin	float64
temperatureMinTime	int64
temperatureMax	float64
temperatureMaxTime	int64
apparentTemperatureMin	float64
apparentTemperatureMinTime	int64
apparentTemperatureMax	float64

```
apparentTemperatureMaxTime       int64
dtype: object
```

In [55]:
```
uberdata.head()
```

Out[55]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_t
0	179271	1.544953e+09	9	16	12	25351	0	5	7	
1	205021	1.543284e+09	2	27	11	961	0	5	7	
2	411506	1.543367e+09	1	28	11	2534	0	5	7	
3	527263	1.543554e+09	4	30	11	6988	0	5	7	
4	606526	1.543463e+09	3	29	11	4400	0	5	7	

5 rows × 57 columns

Filling Null Values

In [56]:
```
uberdata.isnull().sum()
```

Out[56]:

id	0
timestamp	0
hour	0
day	0
month	0
datetime	0
timezone	0
source	0
destination	0
cab_type	0
product_id	0
name	0
price	55095
distance	0
surge_multiplier	0
latitude	0
longitude	0
temperature	0
apparentTemperature	0
short_summary	0
long_summary	0
precipIntensity	0
precipProbability	0
humidity	0
windSpeed	0
windGust	0
windGustTime	0
visibility	0
temperatureHigh	0
temperatureHighTime	0
temperatureLow	0
temperatureLowTime	0
apparentTemperatureHigh	0
apparentTemperatureHighTime	0
apparentTemperatureLow	0
apparentTemperatureLowTime	0
icon	0
dewPoint	0
pressure	0
windBearing	0
cloudCover	0
uvIndex	0
visibility.1	0
ozone	0
sunriseTime	0
sunsetTime	0
moonPhase	0
precipIntensityMax	0
uvIndexTime	0
temperatureMin	0
temperatureMinTime	0
temperatureMax	0
temperatureMaxTime	0
apparentTemperatureMin	0
apparentTemperatureMinTime	0
apparentTemperatureMax	0

```
apparentTemperatureMaxTime         0
dtype: int64
```

In [57]:
```
uberdata['price'].median()
```

Out[57]:
```
13.5
```

In [58]:
```
uberdata["price"].fillna(10.5, inplace = True)
```

In [59]:
```
uberdata.isnull().sum()
```

Out[59]:

id	0
timestamp	0
hour	0
day	0
month	0
datetime	0
timezone	0
source	0
destination	0
cab_type	0
product_id	0
name	0
price	0
distance	0
surge_multiplier	0
latitude	0
longitude	0
temperature	0
apparentTemperature	0
short_summary	0
long_summary	0
precipIntensity	0
precipProbability	0
humidity	0
windSpeed	0
windGust	0
windGustTime	0
visibility	0
temperatureHigh	0
temperatureHighTime	0
temperatureLow	0
temperatureLowTime	0
apparentTemperatureHigh	0
apparentTemperatureHighTime	0
apparentTemperatureLow	0
apparentTemperatureLowTime	0
icon	0
dewPoint	0
pressure	0
windBearing	0
cloudCover	0
uvIndex	0
visibility.1	0
ozone	0
sunriseTime	0
sunsetTime	0
moonPhase	0
precipIntensityMax	0
uvIndexTime	0
temperatureMin	0
temperatureMinTime	0
temperatureMax	0
temperatureMaxTime	0
apparentTemperatureMin	0
apparentTemperatureMinTime	0
apparentTemperatureMax	0

```
apparentTemperatureMaxTime     0
dtype: int64
```

In [60]:
```
uberdata['price'].dtype
```

Out[60]:
```
dtype('float64')
```

In [61]:
```
uberdata['price'] = uberdata['price'].astype(int)
```

In [62]:
```
uberdata['price'].head()
```

Out[62]:
```
0     5
1    11
2     7
3    26
4     9
Name: price, dtype: int32
```

RFE (Recursive Feature Elimination)

In [63]:
```
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [64]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [65]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [66]:
```
from sklearn.feature_selection import RFE
```

In [67]:
```
X = uberdata.drop('price', axis = 1)
y = uberdata['price']
```

In [68]:
```
X.head()
```

Out[68]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_t
0	179271	1.544953e+09	9	16	12	25351	0	5	7	
1	205021	1.543284e+09	2	27	11	961	0	5	7	
2	411506	1.543367e+09	1	28	11	2534	0	5	7	
3	527263	1.543554e+09	4	30	11	6988	0	5	7	
4	606526	1.543463e+09	3	29	11	4400	0	5	7	

5 rows × 56 columns

In [69]:
```
y.head()
```

Out[69]:
```
0     5
1    11
2     7
3    26
4     9
Name: price, dtype: int32
```

In [70]:
```
X.shape
```

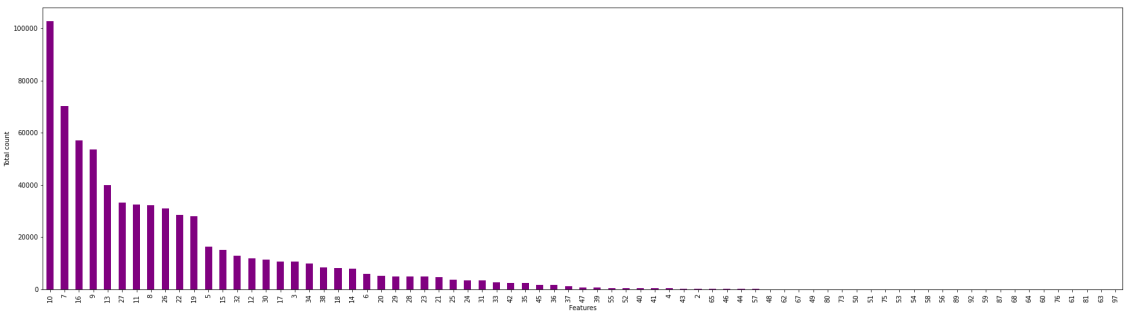
Out[70]:
```
(693071, 56)
```

In [71]:
```
y.shape
```

Out[71]:
```
(693071,)
```

In [72]:
```
y.value_counts().plot(kind='bar',figsize=(30,8),color='purple')
plt.xlabel("Features")
plt.ylabel("Total count")
```

Out[72]:
```
Text(0, 0.5, 'Total count')
```



Training accuracy in 56 features

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [74]:
```
X_train.shape
```

Out[74]: (554456, 56)

In [75]:
```
X_test.shape
```

Out[75]: (138615, 56)

In [76]:
```
y_train.shape
```

Out[76]: (554456,)

In [77]:
```
y_test.shape
```

Out[77]: (138615,)

In [78]:
```
#Creating model
reg = LinearRegression()
#Fitting training data
reg = reg.fit(X_train, y_train)
```

In [79]:
```
reg.score(X_train, y_train)
```

Out[79]: 0.5210613019979404

Training accuracy in 40 features

In [80]:
```
rfe = RFE(reg, 40, verbose=1)
rfe = rfe.fit(X, y)
```

```
C:\Users\Pratik\anaconda3\lib\site-packages\sklearn\utils\validation.py:7
0: FutureWarning: Pass n_features_to_select=40 as keyword args. From vers
ion 1.0 (renaming of 0.25) passing these as positional arguments will res
ult in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

Fitting estimator with 56 features.
Fitting estimator with 55 features.
Fitting estimator with 54 features.
Fitting estimator with 53 features.
Fitting estimator with 52 features.
Fitting estimator with 51 features.
Fitting estimator with 50 features.
Fitting estimator with 49 features.
Fitting estimator with 48 features.
Fitting estimator with 47 features.
Fitting estimator with 46 features.
Fitting estimator with 45 features.
Fitting estimator with 44 features.
Fitting estimator with 43 features.
Fitting estimator with 42 features.
Fitting estimator with 41 features.
```

In [81]:
```
rfe.support_
```

Out[81]:
```
array([False, False,  True,  True,  True, False, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True, False,  True,
        True, False,  True, False,  True, False,  True, False,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True, False,  True, False,  True, False,
        True, False])
```

In [104]:
```
XXfinal = X[X.columns[rfe.support_]]
```

In [105]:
```
#Final Dateset after preprocessing
XXfinal.head()
```

Out[105]:

	source	destination	cab_type	product_id	name	distance	surge_multiplier	latitude	lor
0	5	7	0	8	7	0.44	1.0	42.2148	·
1	5	7	0	12	2	0.44	1.0	42.2148	·
2	5	7	0	7	5	0.44	1.0	42.2148	·
3	5	7	0	10	4	0.44	1.0	42.2148	·
4	5	7	0	11	6	0.44	1.0	42.2148	·

5 rows × 25 columns

In [106]:
```
#Splitting the final data set into test and train
X_train, X_test, y_train, y_test = train_test_split(XXfinal, y, test_size
```

In [107]:
```
X_train.shape
```

Out[107]: (485149, 25)

In [108]:
```
#Creating model
reg1 = LinearRegression()
#Fitting training data
reg1 = reg1.fit(X_train, y_train)
```

In [109]:
```
reg1.score(X_train, y_train)
```

Out[109]: 0.5208176452576193

Training accuracy in 15 features

In [110]:
```
rfe = RFE(reg, 15, verbose=1)
rfe = rfe.fit(X, y)
```

```
C:\Users\Pratik\anaconda3\lib\site-packages\sklearn\utils\validation.py:7
0: FutureWarning: Pass n_features_to_select=15 as keyword args. From vers
ion 1.0 (renaming of 0.25) passing these as positional arguments will res
ult in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

Fitting estimator with 56 features.
Fitting estimator with 55 features.
Fitting estimator with 54 features.
Fitting estimator with 53 features.
Fitting estimator with 52 features.
Fitting estimator with 51 features.
Fitting estimator with 50 features.
Fitting estimator with 49 features.
Fitting estimator with 48 features.
Fitting estimator with 47 features.
Fitting estimator with 46 features.
Fitting estimator with 45 features.
Fitting estimator with 44 features.
Fitting estimator with 43 features.
Fitting estimator with 42 features.
Fitting estimator with 41 features.
Fitting estimator with 40 features.
Fitting estimator with 39 features.
Fitting estimator with 38 features.
Fitting estimator with 37 features.
Fitting estimator with 36 features.
Fitting estimator with 35 features.
Fitting estimator with 34 features.
Fitting estimator with 33 features.
Fitting estimator with 32 features.
Fitting estimator with 31 features.
Fitting estimator with 30 features.
Fitting estimator with 29 features.
Fitting estimator with 28 features.
Fitting estimator with 27 features.
Fitting estimator with 26 features.
Fitting estimator with 25 features.
Fitting estimator with 24 features.
Fitting estimator with 23 features.
Fitting estimator with 22 features.
Fitting estimator with 21 features.
Fitting estimator with 20 features.
Fitting estimator with 19 features.
Fitting estimator with 18 features.
Fitting estimator with 17 features.
Fitting estimator with 16 features.
```

In [111]:
```
XXfinal = X[X.columns[rfe.support_]]
```

In [112]:
```
XXfinal.head()
```

Out[112]:

	source	cab_type	product_id	name	distance	surge_multiplier	latitude	longitude	prec
0	5	0	8	7	0.44	1.0	42.2148	-71.033	
1	5	0	12	2	0.44	1.0	42.2148	-71.033	
2	5	0	7	5	0.44	1.0	42.2148	-71.033	
3	5	0	10	4	0.44	1.0	42.2148	-71.033	
4	5	0	11	6	0.44	1.0	42.2148	-71.033	

In [113]:
```
X_train, X_test, y_train, y_test = train_test_split(XXfinal, y, test_size
```

In [114]:
```
X_train.shape
```

Out[114]:
```
(485149, 15)
```

In [115]:
```
#Creating model
reg1 = LinearRegression()
#Fitting training data
reg1 = reg1.fit(X_train, y_train)
```

In [116]:
```
reg1.score(X_train, y_train)
```

Out[116]:
```
0.5207944456440701
```

Training accuracy in 25 features

In [117]:
```
rfe = RFE(reg, 25, verbose=1)
rfe = rfe.fit(X, y)
```

```
C:\Users\Pratik\anaconda3\lib\site-packages\sklearn\utils\validation.py:7
0: FutureWarning: Pass n_features_to_select=25 as keyword args. From vers
ion 1.0 (renaming of 0.25) passing these as positional arguments will res
ult in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

Fitting estimator with 56 features.
Fitting estimator with 55 features.
Fitting estimator with 54 features.
Fitting estimator with 53 features.
Fitting estimator with 52 features.
Fitting estimator with 51 features.
Fitting estimator with 50 features.
Fitting estimator with 49 features.
Fitting estimator with 48 features.
Fitting estimator with 47 features.
Fitting estimator with 46 features.
Fitting estimator with 45 features.
Fitting estimator with 44 features.
Fitting estimator with 43 features.
Fitting estimator with 42 features.
Fitting estimator with 41 features.
Fitting estimator with 40 features.
Fitting estimator with 39 features.
Fitting estimator with 38 features.
Fitting estimator with 37 features.
Fitting estimator with 36 features.
Fitting estimator with 35 features.
Fitting estimator with 34 features.
Fitting estimator with 33 features.
Fitting estimator with 32 features.
Fitting estimator with 31 features.
Fitting estimator with 30 features.
Fitting estimator with 29 features.
Fitting estimator with 28 features.
Fitting estimator with 27 features.
Fitting estimator with 26 features.
```

In [118]:
```
XXfinal = X[X.columns[rfe.support_]]
```

In [119]:
```
XXfinal.head()
```

Out[119]:

	source	destination	cab_type	product_id	name	distance	surge_multiplier	latitude	lor
0	5	7	0	8	7	0.44	1.0	42.2148	·
1	5	7	0	12	2	0.44	1.0	42.2148	·
2	5	7	0	7	5	0.44	1.0	42.2148	·
3	5	7	0	10	4	0.44	1.0	42.2148	·
4	5	7	0	11	6	0.44	1.0	42.2148	·

5 rows × 25 columns

In [120]:
```
X_train, X_test, y_train, y_test = train_test_split(XX, y, test_size = 0.3
```

In [121]:
```
X_train.shape
```

Out[121]:
```
(485149, 25)
```

In [122]:
```
#Creating model
reg1 = LinearRegression()
#Fitting training data
reg1 = reg1.fit(X_train, y_train)
#Y prediction
Y_pred = reg1.predict(X_test)
```

In [123]:
```
reg1.score(X_train, y_train)
```

Out[123]:
```
0.5206802050605615
```

- As a result of determining the accuracy for k = 56, 40, 25, and 15,
- As a result, we noted that the Linear Regression Model's training accuracy is at its highest when k = 25.

25 Columns After RFE

In [124]:
```
XXfinal.columns
```

Out[124]:
```
Index(['source', 'destination', 'cab_type', 'product_id', 'name', 'distan
ce',
       'surge_multiplier', 'latitude', 'longitude', 'temperature',
       'apparentTemperature', 'precipIntensity', 'precipProbability',
       'humidity', 'windSpeed', 'windGust', 'temperatureHigh',
       'apparentTemperatureHigh', 'icon', 'dewPoint', 'cloudCover', 'uvIn
dex',
       'precipIntensityMax', 'temperatureMax', 'apparentTemperatureMax'],
      dtype='object')
```

In [125]:
```
XXfinal.shape
```

Out[125]: (693071, 25)

In [126]:
```
XXfinal.head()
```

Out[126]:

	source	destination	cab_type	product_id	name	distance	surge_multiplier	latitude	lor
0	5	7	0	8	7	0.44	1.0	42.2148	·
1	5	7	0	12	2	0.44	1.0	42.2148	·
2	5	7	0	7	5	0.44	1.0	42.2148	·
3	5	7	0	10	4	0.44	1.0	42.2148	·
4	5	7	0	11	6	0.44	1.0	42.2148	·

5 rows × 25 columns

Droping Unwanted Features

In [127]:
```
featuresdrop = ['latitude', 'longitude', 'apparentTemperature',
        'precipIntensity', 'humidity', 'windSpeed', 'windGust',
       'temperatureHigh', 'apparentTemperatureHigh', 'dewPoint','precipInt
       'temperatureMax', 'apparentTemperatureMax', 'distance', 'cloudCover
newuberdata = XXfinal.drop(featuresdrop, axis=1)
```

In [128]:
```
newuberdata.head()
```

Out[128]:

	source	destination	cab_type	product_id	name	surge_multiplier	temperature	precipPr
0	5	7	0	8	7	1.0	42.34	
1	5	7	0	12	2	1.0	43.58	
2	5	7	0	7	5	1.0	38.33	
3	5	7	0	10	4	1.0	34.38	
4	5	7	0	11	6	1.0	37.44	

Binning

In [129]:
```
surge_multiplier_mapping = {1.: 0, 1.25: 1, 1.5: 2, 1.75: 3, 2.:4}
newuberdata['surge_multiplier'] = newuberdata['surge_multiplier'].map(surg
```

Final Dataset

In [130]:
```
newuberdata.head()
```

Out[130]:

	source	destination	cab_type	product_id	name	surge_multiplier	temperature	precipPr
0	5	7	0	8	7	0.0	42.34	
1	5	7	0	12	2	0.0	43.58	
2	5	7	0	7	5	0.0	38.33	
3	5	7	0	10	4	0.0	34.38	
4	5	7	0	11	6	0.0	37.44	

In [132]:
```
newuberdata.fillna(0, inplace=True)
```

In [133]:
```
y.head()
```

Out[133]:
```
0     5
1    11
2     7
3    26
4     9
Name: price, dtype: int32
```

5. Modeling

In [134]:
```
newuberdata.shape
```

Out[134]: (693071, 10)

In [135]:
```
y.shape
```

Out[135]: (693071,)

In [137]:
```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
xx_train, xx_test, yy_train, yy_test = train_test_split(newuberdata, y, te
```

In [138]:
```
xx_train.shape
```

Out[138]: (554456, 10)

In [139]:
```
xx_test.shape
```

Out[139]: (138615, 10)

In [140]:
```
yy_train.shape
```

Out[140]: (554456,)

In [141]:
```
yy_test.shape
```

Out[141]: (138615,)

In [142]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [143]:
```
newuberdata.fillna(X_train.mean(), inplace=True)
```

5.1 Linear regression

In [144]:
```
from scipy.stats import loguniform
from pandas import read_csv
from sklearn.linear_model import Ridge
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import RandomizedSearchCV
```

In [145]:
```
linear = LinearRegression(fit_intercept=True, normalize=False,copy_X=True,

linear.fit(xx_train, yy_train)
print('linear_score : ',linear.score(xx_test, yy_test))
```

```
linear_score :  0.41973282820736557
```

5.2 Decision Tree

In [146]:
```
decision = DecisionTreeRegressor(random_state = 0)
decision.fit(xx_train , yy_train)
print('Decision_tree_score :',decision.score(xx_test, yy_test))
```

```
Decision_tree_score : 0.9350893669242692
```

5.3 Random Forest

In [148]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {  'bootstrap': [True], 'max_depth': [5, 10, None], 'max_feat
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15]}

rfr = RandomForestRegressor(random_state = 1)

g_search = GridSearchCV(estimator = rfr, param_grid = param_grid,

                        cv = 3, n_jobs = 1, verbose = 0, return_train_sc
g_search.fit(xx_train, yy_train)

print(g_search.best_params_)
```

```
{'bootstrap': True, 'max_depth': None, 'max_features': 'auto', 'n_estimat
ors': 15}
```

In [149]:
```
random = RandomForestRegressor(n_estimators = 100, random_state = 0)
random.fit(xx_train , yy_train)
print('Random_forest_score :',random.score(xx_test, yy_test))
```

```
Random_forest_score : 0.9464565853684954
```

5.4 Gradient Boosting Regressor

In [150]:
```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_depth = 5
clf.fit(xx_train, yy_train)
```

Out[150]:
```
GradientBoostingRegressor(max_depth=5, n_estimators=400)
```

In [151]:
```
print('Grdient_Boosting_Regressor_score :',clf.score(xx_test, yy_test))
```

```
Grdient_Boosting_Regressor_score : 0.9569891521098086
```

K fold Crossvalidation

In [152]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
cv=ShuffleSplit(n_splits=5,test_size=0.2,random_state=0)
cross_val_score(LinearRegression(),xx_test,yy_test,cv=cv)
```

Out[152]:
```
array([0.41678743, 0.42065507, 0.41595018, 0.41507048, 0.41455486])
```

6. Testing

Linear regression

In [153]:
```
linear.coef_
```

Out[153]:
```
array([-4.45245258e-02, -1.15930115e-01,  6.73353262e+00,  6.11308643e-01,
       -1.58791992e+00,  5.42235989e+00, -1.32565999e-03, -1.87255254e-02,
        2.67006215e-03,  1.77189404e-02])
```

In [154]:
```
prediction = linear.predict(xx_test)
prediction
```

Out[154]:
```
array([ 7.95285918, 14.29355318, 14.35673938, ...,  8.75857276,
       18.07121757, 22.08682731])
```

In [155]:
```
prediction=  prediction.astype(int)
```

```
plt.scatter(yy_test,prediction)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

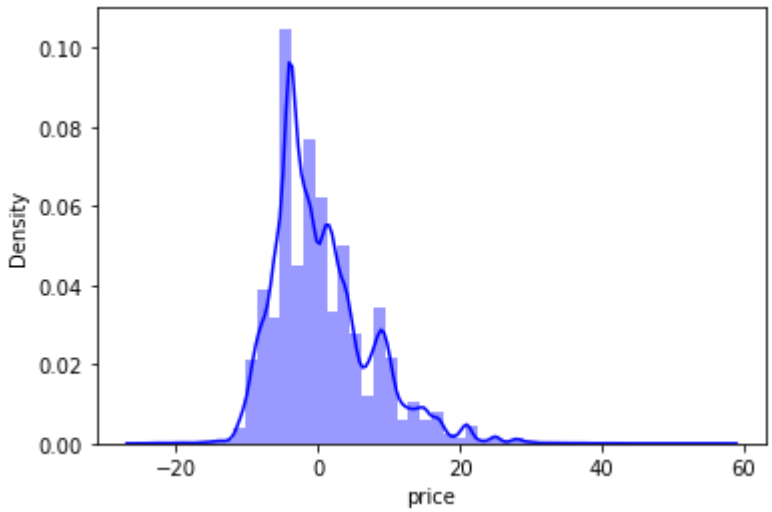
In [156]:
```
from sklearn import metrics
print('MAE :'," ", metrics.mean_absolute_error(yy_test,prediction))
print('MSE :'," ", metrics.mean_squared_error(yy_test,prediction))
print('RMAE :'," ", np.sqrt(metrics.mean_squared_error(yy_test,prediction)))
```

```
MAE :   5.291519676802655
MSE :   48.02777477185009
RMAE :   6.930207411892525
```

In [170]:
```
sns.distplot(yy_test - prediction,bins=50,color='blue')
```

```
C:\Users\Pratik\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a fi
gure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[170]: <AxesSubplot:xlabel='price', ylabel='Density'>

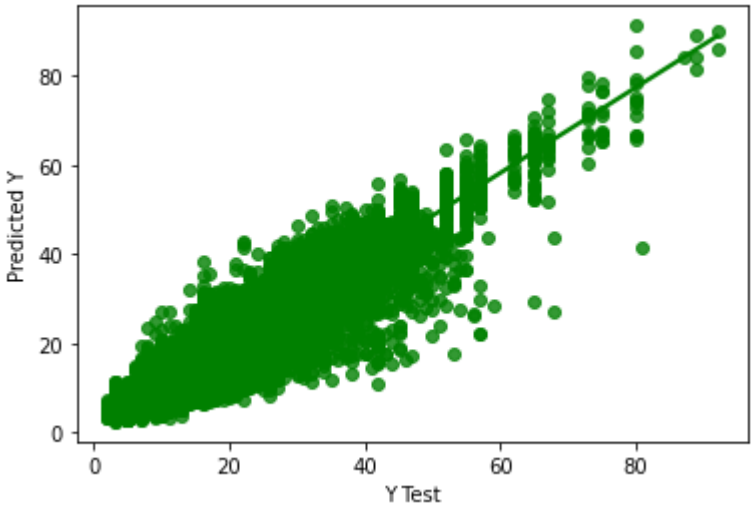


Random Forest

In [166]:
```
predictions = random.predict(xx_test)
```

In [169]:
```
sns.regplot(yy_test,predictions,color='green')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[169]: Text(0, 0.5, 'Predicted Y')



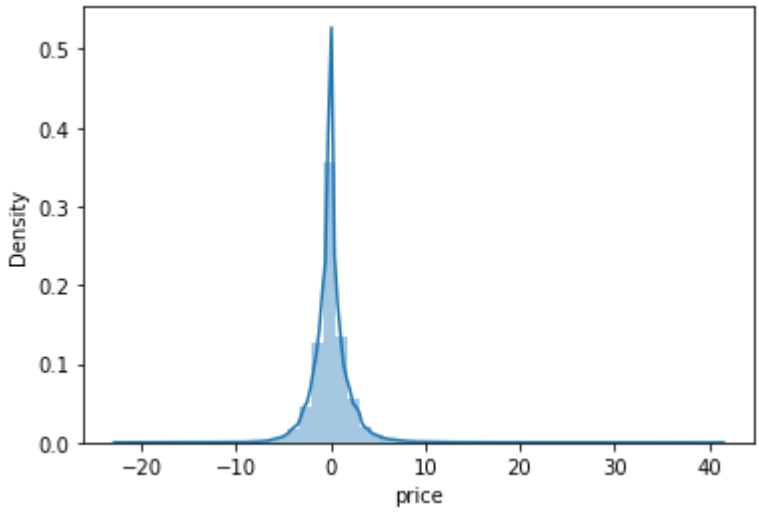
In [171]:
```
from sklearn import metrics
print('MAE :'," ", metrics.mean_absolute_error(yy_test,predictions))
print('MSE :'," ", metrics.mean_squared_error(yy_test,predictions))
print('RMAE :'," ", np.sqrt(metrics.mean_squared_error(yy_test,predictions
```

```
MAE :   1.2568165927706472
MSE :   4.406179756198409
RMAE :   2.099090221071598
```

In [172]:
```
sns.distplot(yy_test - predictions,bins=50)
```

```
C:\Users\Pratik\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a fi
gure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[172]: <AxesSubplot:xlabel='price', ylabel='Density'>



Price prediction function

In [174]:
```
newuberdata.head()
```

Out[174]:

	source	destination	cab_type	product_id	name	surge_multiplier	temperature	precipPr
0	5	7	0	8	7	0.0	42.34	
1	5	7	0	12	2	0.0	43.58	
2	5	7	0	7	5	0.0	38.33	
3	5	7	0	10	4	0.0	34.38	
4	5	7	0	11	6	0.0	37.44	

In [175]:
```
def predict_price(name,source,surge_multiplier,icon):
    loc_index = np.where(newuberdata.columns==name)[0]

    x = np.zeros(len(newuberdata.columns))
    x[0] = source
    x[1] = surge_multiplier
    x[2] = icon
    if loc_index >= 0:
        x[loc_index] = 1

    return random.predict([x])[0]
```

In [176]:
```
pre= random.predict(xx_test)
```

Follow these instructions before predicting the price:

- **For cab_name**: Black SUV --> 0 , Lux --> 1 , Shared --> 2 , Taxi --> 3 , UberPool --> 4 , UberX --> 5
- **For Source**: Back Bay --> 0 , Beacon Hill --> 1 , Boston University --> 2 , Fenway --> 3 , Financial District --> 4 , Haymarket Square --> 5 , North End --> 6 , North Station --> 7 , Northeastern University --> 8 , South Station --> 9 , Theatre District --> 10 , West End --> 11
- **For Surge_multiplier** : Enter Surge Multiplier value from 0 to 4
- **for Icon**: clear-day --> 0 , clear-night --> 1 , cloudy --> 2 , fog --> 3 , partly-cloudy-day --> 4 , partly-cloudy-night --> 5 , rain --> 6

predict_price(cab_name , source , surge_multiplier , icon)

In [177]:
```
predict_price(1 , 3, 2, 0)
```

```
C:\Users\Pratik\AppData\Local\Temp/ipykernel_33156/1257577895.py:8: Depre
cationWarning: The truth value of an empty array is ambiguous. Returning
False, but in future this will result in an error. Use `array.size > 0` t
o check that an array is not empty.
  if loc_index >= 0:
```

Out[177]:
```
34.481666666666676
```

Result Metrics

In [178]:
```
print("*****************************************")
print("Linear Regression Metrics")

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
print('MAE :'," ", metrics.mean_absolute_error(yy_test,prediction))
print('MSE :'," ", metrics.mean_squared_error(yy_test,prediction))
print('RMAE :'," ", np.sqrt(metrics.mean_squared_error(yy_test,prediction)

print("*****************************************")
print("Decision tree Metrics")

print('MAE :'," ", metrics.mean_absolute_error(yy_test,predictions))
print('MSE :'," ", metrics.mean_squared_error(yy_test,predictions))
print('RMAE :'," ", np.sqrt(metrics.mean_squared_error(yy_test,predictions


print("*****************************************")
```

```
*****************************************
Linear Regression Metrics
MAE :   5.291519676802655
MSE :   48.02777477185009
RMAE :   6.930207411892525
*****************************************
Decision tree Metrics
MAE :   1.2568165927706472
MSE :   4.406179756198409
RMAE :   2.099090221071598
*****************************************
```