Project 01: Customer Service Requests Analysis

1. Import a 311 NYC service request.

1: View and add the dataset

In [ ]:
```
#Import required library
import pandas as pd
import numpy as np
```

In [ ]:
```
#Manually load the customer service request dataset
csr_data = pd.read_csv("311_Service_Requests_from_2010_to_Present.csv")
```

2: Analyze the dataset

In [ ]:
```
#View the initial few records of the dataset
csr_data.head()
```

In [ ]:
```
#View the bottom few records of the dataset
csr_data.tail()
```

In [ ]:
```
#Check the total number of elements in the dataset
csr_data.size
```

In [ ]:
```
# view the type of the dataset
type(csr_data)
```

In [ ]:
```
#Check the number of observations (rows) and attributes (columns) in the dataset
csr_data.shape
```

In [ ]:
```
#View the names of each of the attributes
csr_data.columns
```

In [ ]:
```
# view the information of the dataser
csr_data.info()
```

In [ ]:
```
# Check for duplicates values
csr_data.duplicated().sum()
```

In [ ]:
```
# Check for Null values
csr_data.isna().sum()
```

2. Read or convert the columns 'Created Date' and Closed Date' to datetime datatype and create a new column 'Request_Closing_Time' as the time elapsed between request creation and request closing.

In [ ]:
```
# import datetime module
from datetime import datetime, timedelta
```

In [ ]:
```
# import dataset to inster new column Request CLosing Time
csr_data1 = pd.read_csv("311_Service_Requests_from_2010_to_Present.csv")
```

In [ ]:
```
# View top tows
csr_data1.head()
```

In [ ]:
```
# converting the columns Created Date and Closed Date to Datetime
csr_data1["Created Date"] = pd.to_datetime(csr_data1["Created Date"])
csr_data1["Closed Date"] = pd.to_datetime(csr_data1["Closed Date"])
```

In [ ]:
```
# create a column and calculate Request closing time by performing time difference
rct= (csr_data1["Created Date"] - csr_data1["Closed Date"])
```

In [ ]:
```
# insert the newly created column in the dataset
csr_data1.insert(3, "Request_Closing_Time", rct, True)
```

In [ ]:
```
# view the new wrangled dataset and verify if the column is added
csr_data1.head()
```

3. Provide major insights/patterns that you can offer in a visual format (graphs or tables); at least 4 major conclusions that you can come up with after generic data mining.

view and analyze the dataset

In [ ]:
```
# import required libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [ ]:
```
# view the top data to analyze which columns to access for insights
csr_data.head()
```

In [ ]:
```
# see the column names
csr_data.columns
```

Provide insights for 4 major conclusions

1. Provide insights for Complaint types using countplot
2. Have a look at the status of tickets with bar plot
3. Location type Breakdown with pie chart to figure out majority of complaints from location types and top 5 categories
4. Number of complaints registered from different cities

a. Provide insights for Complaint types using countplot

In [ ]:
```
# plot a horizontal count plot using seaborn
plt.figure(figsize=(10,10))                                    # gives figure size
fig= sns.countplot(y = "Complaint Type", data = csr_data)      # plot the graph
plt.title("Distribution of Complaint type", fontsize = 20)     # gives title
plt.xlabel("Number of Complaints", fontsize = 20)              # gives label for X axis
plt.ylabel("Complaint Type", fontsize = 20)                    # gives label for y axis
fig.text(x = 70000, y = 22, s = "plot by Srikant Kamatagi", fontsize = 20, color ='grey
plt.grid(alpha = 0.4)                                          # displays the grid
plt.show()
```

In [ ]:
```
# view the count of each type of complaints
csr_data["Complaint Type"].value_counts()
```

b. Have a look at the status of tickets

In [ ]:
```
# view the status of compliants
csr_data["Status"].value_counts()
```

In [ ]:
```
# plot a vertical bar graph
plt.figure(figsize=(15,10))
bar = csr_data['Status'].value_counts().plot(kind='bar',alpha=0.8,figsize=(7,7), grid =
plt.xlabel("Status", fontsize = 15)                    # gives label for X axis
plt.ylabel("Number of Complaints", fontsize = 15)      # gives label for y axis
plt.title("Status of the complaints", fontsize = 20)
plt.legend()
plt.show()
```

c. Location type Breakdown with pie chart to figure out majority of complaints from location types and top 5 categories

In [ ]:
```
# view the number of complaints registered for different location type
csr_data["Location Type"].value_counts()
```

In [ ]:
```
# plot a pie chart
plt.figure(figsize=(15,10))
pie = csr_data['Location Type'].value_counts().head(5).plot(kind='pie', autopct = "%0.2
```

```
plt.title("Breakdown of complaint types", fontsize = 20)
plt.show()
```

d. number of complaints registered from different cities

In [ ]:
```
# plot a horizontal countplot and change the color palette
plt.figure(figsize=(15,20))                                    # gives figure size
fig= sns.countplot(y = "City", data = csr_data, palette="Paired")    # plot the graph
plt.title("Number of Complaints in different Cities", fontsize = 25)   # gives title
plt.xlabel("Number of Complaints", fontsize = 20)              # gives label for X axis
plt.ylabel("Name of City", fontsize = 20)                    # gives label for y axis
fig.text(x = 100000, y = 50, s = "plot by Srikant Kamatagi", fontsize = 20,
         color ='grey', ha ='right', va ='bottom',alpha = 0.3)   # prints the name of t
plt.grid(alpha = 0.4)    # displays the grid
plt.show()
```

In [ ]:
```
# view top 10 cities from where the complaints were registered
csr_data["City"].value_counts().head(10)
```

4.Order the complaint types based on the average ‘Request_Closing_Time’, grouping them for different locations.

In [ ]:
```
# View the earlier wrangled dataset
csr_data1.head()
```

In [ ]:
```
# view the columns
csr_data1.columns
```

In [ ]:
```
# ordering groupby Location based on Average Request_closing_time
csr_data1.groupby(["Complaint Type", "Location"])["Request_Closing_Time"].mean()
```

In [ ]:
```
# groupby Location Type based on Average Request_closing_time
csr_data1.groupby(["Complaint Type", "Location Type"])["Request_Closing_Time"].mean()
```

In [ ]:
```
# groupby City based on Average Request_closing_time
csr_data1.groupby(["Complaint Type", "City"])["Request_Closing_Time"].mean()
```

In [ ]:
```
# groupby Borough based on Average Request_closing_time
csr_data1.groupby(["Complaint Type","Borough"])["Request_Closing_Time"].mean()
```

5.Perform a statistical test for the following:

Note : Please note: For the below statements you need to state the Null and Alternate and then provide a statistical test to accept or reject the Null Hypothesis along with the corresponding ‘p-value’.

1. Whether the average response time across complaint types is similar or not (overall)

2. Are the type of complaint or service requested and location related?

In [ ]:
```
# import statistical package from scipy
from scipy import stats
```

In [ ]:
```
# view the data
csr_data1.head()
```

1. Whether the average response time across complaint types is similar or not (overall)

In [ ]:
```
# analyze the relationship between multiple variables i,e average closing time and comp
data = pd.crosstab(csr_data1["Request_Closing_Time"].mean(), csr_data1["Complaint Type"
data
```

In [ ]:
```
help(stats.chi2_contingency)
```

In [ ]:
```
# use chi square test to analyse the relation between both the variables.
chi,pval,dof,exp=stats.chi2_contingency(data.values) # data.values
print("Chisquare", chi)
print("plvaue", pval)
print("Degrees of freedom", dof)
print("Expected", exp)
```

In [ ]:
```
# provide a statistical test to accept or reject the Null Hypothesis
if pval < 0.05:
    print("Alternate hypo--- Relation exists")
else:
    print("Null hypo-- no relation exists")
```

2. Are the type of complaint or service requested and location related?

In [ ]:
```
# analyze the relationship between multiple variables i,e Location and complaint type
data1 = pd.crosstab(csr_data1["Location"], csr_data1["Complaint Type"])
data1
```

In [ ]:
```
# use chi square test to analyse the relation between both the variables.
chi,pval,dof,exp=stats.chi2_contingency(data1.values) # data.values
print("Chisquare", chi)
print("plvaue", pval)
print("Degrees of freedom", dof)
print("Expected", exp)
```

In [ ]:
```
# provide a statistical test to accept or reject the Null Hypothesis
if pval < 0.05:
```

```
    print("Alternate hypo--- Relation exists")
else:
    print("Null hypo-- no relation exists")
```

In [ ]:
```
# view the columns
csr_data1.columns
```

In [ ]:
```
# analyze the relationship between multiple variables i,e Location Type and complaint t
data2 = pd.crosstab(csr_data1["Location Type"], csr_data1["Complaint Type"])
data2
```

In [ ]:
```
# use chi square test to analyse the relation between both the variables.
chi,pval,dof,exp=stats.chi2_contingency(data2.values) # data.values
print("Chisquare", chi)
print("plvaue", pval)
print("Degrees of freedom", dof)
print("Expected", exp)
```

In [ ]:
```
# provide a statistical test to accept or reject the Null Hypothesis
if pval < 0.05:
    print("Alternate hypo--- Relation exists")
else:
    print("Null hypo-- no relation exists")
```

In [ ]:
```
# analyze the relationship between multiple variables i,e City and complaint type
data3 = pd.crosstab(csr_data1["City"], csr_data1["Complaint Type"])
data3
```

In [ ]:
```
# use chi square test to analyse the relation between both the variables.
chi,pval,dof,exp=stats.chi2_contingency(data3.values) # data.values
print("Chisquare", chi)
print("plvaue", pval)
print("Degrees of freedom", dof)
print("Expected", exp)
```

In [ ]:
```
# provide a statistical test to accept or reject the Null Hypothesis
if pval < 0.05:
    print("Alternate hypo--- Relation exists")
else:
    print("Null hypo-- no relation exists")
```