```
# import libararies
from scipy.stats import kruskal
import numpy as np
from scipy.stats import ttest_1samp
import pandas as pd
import matplotlib.pyplot as plt

# plot inline
# %matplotlib inline

# import the dataset
df = pd.read_csv(
    '../Datasets/311_Service_Requests_from_2010_to_Present.csv', low_memory=False)

# print the first 5 rows
df.head()

# print the last 5 rows
df.tail()

# Visualize the dataset
df.plot()

# Priting the columns
print(df.columns)

# Printing the shape of the dataset
df.shape

# Print dataset null values
df.isna()

# Draw a frequency plot to show the number of null values in each column of the DataFrame
percentage_of_missing_values = (df.isna().sum(axis=0)/df.shape[0])*100
plt.figure(figsize=(10, 6))
plt.bar(percentage_of_missing_values.index, percentage_of_missing_values)
plt.xlabel('Columns')
plt.ylabel('Percentage of missing values')
plt.title('Percentage of missing values in each column')
plt.xticks(rotation=90)
plt.show()

# Drop rows woth missing values in closed date column
df1 = df.dropna(subset=['Closed Date'], axis=0)

# Convert Timeline columns to datetime, coerce errors to NaT (Not a Time)
df1['Closed Date'] = pd.to_datetime(df1['Closed Date'], errors='coerce')
df1['Created Date'] = pd.to_datetime(df1['Created Date'], errors='coerce')

# Set your criteria for valid date range (adjust as needed)
min_date = pd.to_datetime('1900-01-01')  # replace with your minimum valid date
max_date = pd.to_datetime('2023-12-31')  # replace with your maximum valid date

# Filter rows based on the date range
```

```
df1 = df1[(df1['Closed Date'] >= min_date) &
        (df1['Closed Date'] <= max_date)]

df1 = df1[(df1['Created Date'] >= min_date) &
        (df1['Created Date'] <= max_date)]

# Calculate the time elapsed in closed and creation date
time_elapsed = df1['Closed Date'] - df1['Created Date']

# Convert the calculated time elapsed into seconds
time_elapsed_in_seconds = time_elapsed.dt.total_seconds()

# View the descriptive statiscts for the new created column
time_elapsed_in_seconds.describe()

# Check the number of null values in the Complaint_Type and City columns
df1[['Complaint Type', 'City']].isna().sum(axis=0)

# Impute the NA value with 'Unknown City'
df1.loc[df1['City'].isna(), 'City'] = 'Unknown City'

# Draw a frequency plot for the complaints in each city
plt.figure(figsize=(10, 6))
df1['City'].value_counts().plot(kind='bar')
plt.xlabel('City')
plt.ylabel('Number of complaints')
plt.title('Number of complaints in each city')
plt.xticks(rotation=90)
plt.show()

# Create a scatter and hexbin plot of the concentration of complaints across Brooklyn
broolyn_df = df1[df1['City'] == 'BROOKLYN']
plt.figure(figsize=(10, 6))
plt.scatter(broolyn_df['Latitude'], broolyn_df['Latitude'])
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Scatter plot of the concentration of complaints across Brooklyn')
plt.show()

plt.figure(figsize=(10, 6))
plt.hexbin(broolyn_df['Longitude'], broolyn_df['Latitude'], gridsize=100)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Hexbin plot of the concentration of complaints across Brooklyn')
plt.show()

# Plot a bar graph to show the types of complaints
plt.figure(figsize=(10, 6))
df1['Complaint Type'].value_counts().plot(kind='bar')
plt.xlabel('Complaint Type')
plt.ylabel('Number of complaints')
plt.title('Number of complaints in each type')
plt.xticks(rotation=90)
plt.show()
```

```
# Check the frequency of Complaint type in Newyork city
newyork_df = df1[df1['City'] == 'NEW YORK']
percentage_of_complaint_type_newyork = (
    newyork_df['Complaint Type'].value_counts()/df.shape[0])*100
percentage_of_complaint_type_newyork

# Find the top 10 complaint types in Newyork city
top_10_complaint_types_newyork = percentage_of_complaint_type_newyork.head(10)
top_10_complaint_types_newyork

# Display the various types of complaints in each city
df1.groupby('City')['Complaint Type'].value_counts()

# Create a DataFrame, df_new, which contains cities as columns and complaint types in rows
df_new = df1.groupby('Complaint Type')['City'].value_counts().unstack()
df_new.head()


# Visualize the major types of complaints in each city
complaint_counts = df.groupby(['City', 'Complaint Type']).size()
complaint_count = complaint_counts.reset_index(name='Count')
major_complaints = complaint_count.loc[complaint_count.groupby('City')[
    'Count'].idxmax()]
print(major_complaints)
major_complaints.plot(kind='bar', figsize=(20, 6))
plt.xlabel('Complaint Type')
plt.ylabel('Number of complaints')
plt.title('Number of complaints in each type')
plt.xticks(rotation=90)
plt.show()

# Draw another chart that shows the types of complaints in  each city in a single chart, where different colors show the different types of complaints
df_new.plot(kind='bar', figsize=(20, 6))
plt.xlabel('Complaint Type')
plt.ylabel('Number of complaints')
plt.title('Number of complaints in each type')
plt.xticks(rotation=90)
plt.show()

# Create a DataFrame with the average closing time for each complaint type
df1['Response Time'] = time_elapsed_in_seconds / (60 * 60)  # Convert to hours
# Create a DataFrame with the average closing time for each complaint type and location
avg_closing_time = df1.groupby(['Complaint Type', 'Location'])[
    'Response Time'].mean().reset_index()
# Sort the DataFrame by average closing time
sorted_df = avg_closing_time.sort_values(by='Response Time')
# Display the sorted DataFrame
sorted_df

# Group by 'Complaint Type' and calculate average response time
request_closing_time = df1.groupby('Complaint Type')[
    'Response Time'].mean().sort_values()
```

```
# Display the average response time for each complaint type
print(request_closing_time)

# Plot a horizontal bar chart to visualize the average response time for each complaint type
plt.figure(figsize=(10, 6))
request_closing_time.plot(kind='barh', color='skyblue')
plt.title('Average Request Closing Time by Complaint Type')
plt.xlabel('Average Request Closing Time (hours)')
plt.ylabel('Complaint Type')
plt.show()

# Identify the significant variables by performing statistical analysis using p-values
response_time_mean = np.mean(df1['Response Time'])
print(response_time_mean)
tset, pval = ttest_1samp(df1['Response Time'], 30)
print("p-values", pval)
if pval < 0.05:    # alpha value is 0.05 or 5%
   print(" we are rejecting null hypothesis")
else:
   print("we are accepting null hypothesis")

# Perform Kruskal-Wallis H test
data = {
   'Request_Closing_Time': df1['Response Time'],
   'Complaint_Type': df1['Complaint Type']
}
df_subset = pd.DataFrame(data)
statistic, p_value = kruskal(*[group['Request_Closing_Time']
                             for name, group in df_subset.groupby('Complaint_Type')])
# Display the results
print(f"Kruskal-Wallis H statistic: {statistic}")
print(f"P-value: {p_value}")
# Make a decision based on the null hypothesis
alpha = 0.05
if p_value < alpha:
   print("Reject the null hypothesis: One or more sample distributions are not equal.")
else:
   print("Fail to reject the null hypothesis: All sample distributions are equal.")
```