```python
# Importing required libraries-------------------

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import calendar

# Reading data------------------------------------

data = pd.read_csv('311_Service_Requests_from_2010_to_Present.csv')

# Looking for features (columns) ------------------

data.columns

# Checking the data types and entries (if Nan)------------------------

data.info()

# Setting the option to view all columns of the dataset and view the data for any
4 random entries---------

pd.set_option('display.max_columns',None)
data.sample(4)

# Dropping columns and saving it in another data frame----------------------

data_mod = data.drop(columns=['Unique Key'],axis=1)

# Looking the column information in new data frame--------------------------

data_mod.columns

# To evaluate distinguishable outcomes belong to this feature--------------------
-

pd.unique(data['Agency'])

# Frequency of the different outcomes -------------------------

data['Agency'].value_counts()
```

```
data_mod = data_mod.drop(columns=['Agency'],axis=1)
pd.unique(data['Agency Name'])
data['Agency Name'].value_counts()
data['Complaint Type'].value_counts().head(5)
data.Descriptor.value_counts().head(5)
data['Location Type'].value_counts().head(4)
data['Incident Zip'].value_counts().head(4)
data['Incident Address'].value_counts().head(4)
data['Street Name'].value_counts().head(5)
data['Cross Street 1'].value_counts().head(4)
data['Cross Street 2'].value_counts().head(4)
data['Intersection Street 1'].value_counts().head(4)
data['Intersection Street 2'].value_counts().head(4)
data['Address Type'].value_counts().head(4)
data['City'].value_counts().head(4)
data.Landmark.value_counts().head(5)
data['Facility Type'].value_counts()
data.Status.value_counts()
data['Due Date'].value_counts().head(4)
data['Resolution Description'].value_counts().head(4)
data['School Name'].value_counts()
data['School Number'].value_counts()
data['School Region'].value_counts()
data['School Not Found'].value_counts()
data['School Code'].value_counts()
data['School Phone Number'].value_counts()
data['School Address'].value_counts()
data['School City'].value_counts()
data['School State'].value_counts()
data['School Zip'].value_counts()
data['School Not Found'].value_counts()
data['School or Citywide Complaint'].value_counts()
data.columns
data_mod = data_mod.drop(columns=['School Name', 'School Number', 'School
Region', 'School Code',
                              'School Phone Number', 'School Address', 'School
City', 'School State',
                                 'School Zip', 'School Not Found', 'School or
Citywide Complaint'],axis=1)
data['Vehicle Type'].value_counts()
data['Taxi Company Borough'].value_counts()
data['Taxi Pick Up Location'].value_counts()
data_mod = data_mod.drop(columns=['Vehicle Type','Taxi Company Borough','Taxi
Pick Up Location'],axis=1)
data_mod.columns
```

```
data['Bridge Highway Name'].value_counts().head(5)
data['Bridge Highway Direction'].value_counts().head(4)
data['Road Ramp'].value_counts()
data['Bridge Highway Segment'].value_counts().head(6)
data['Garage Lot Name'].value_counts()
data['Ferry Direction'].value_counts()
data['Ferry Terminal Name'].value_counts()
data_mod = data_mod.drop(columns=['Garage Lot Name','Ferry Direction','Ferry
Terminal Name'],axis=1)
data_mod.columns
data['Latitude'].value_counts().head(5)
data['Longitude'].value_counts().head(5)
data['Location'].value_counts().head(4)
data_mod.sample(10)
data_mod.columns


# Final info.(features) after cleaning  ------------------

data_mod.info()
data_mod['Closed Date'] = pd.to_datetime(data_mod['Closed Date'])
data_mod['Created Date'] = pd.to_datetime(data_mod['Created Date'])

data_mod['Request_Closing_Time'] = data_mod['Closed Date'] - data_mod['Created
Date']

#data_mod = data_mod[(data_mod.Request_Closing_Time)>=0]

data_mod.info()
data_mod.sample(4)
data_mod.columns

# Measuring the frequency (occurence) of the different complaint-----------------
----------

data_complaint = data['Complaint Type'].value_counts()
data_complaint = data_complaint.to_frame()
data_complaint = data_complaint.rename(columns={'Complaint Type':'Counts'})
data_complaint

# Evaluate the above in percentage ----------------------------

data_complaint['Percentage'] =
np.around((data_complaint.Counts/data_complaint.Counts.sum())*100,decimals=2)
data_complaint
```

```
# Keeping the major complaint types ---------------

data_complaint = data_complaint[data_complaint.Percentage>1.0]
data_complaint = data_complaint.reset_index()
data_complaint = data_complaint.rename(columns={'index':'Complaint Type'})
data_complaint

# Visualization of the above evaluated dataset ---------------------

plt.figure(figsize=(12,6))
com_type = sns.barplot(x=data_complaint['Complaint
Type'],y=data_complaint.Percentage,data=data_complaint)
com_type.set_xticklabels(com_type.get_xticklabels(), rotation=30, ha="right")
plt.title('Proportion of different complaint type (major)')
plt.show()
plt.tight_layout()


# Applying the above procedure for Descriptor-------------------

data_descriptor = np.around(((data_mod['Descriptor'].value_counts()*100) /
data_mod['Descriptor'].value_counts().sum()),
                            decimals=2)
data_descriptor = data_descriptor.to_frame()
data_descriptor = data_descriptor.rename(columns={'Descriptor':'Percentage'})
data_descriptor['Descriptor'] = data_descriptor.index
cols = data_descriptor.columns.tolist()
cols = cols[-1:]+cols[:-1]
data_descriptor = data_descriptor[cols]
data_descriptor = data_descriptor[(data_descriptor.Percentage) >= 2.0]
data_descriptor = data_descriptor.reset_index()
data_descriptor = data_descriptor.drop(columns=['index'],axis=1)
data_descriptor

# Applying the above procedure for Location Type-------------------

data_location_type = np.around(((data_mod['Location Type'].value_counts()*100) /
data_mod['Location Type'].value_counts().sum()),
                            decimals=2)
data_location_type = data_location_type.to_frame()
data_location_type = data_location_type.rename(columns={'Location
Type':'Percentage'})
data_location_type['Location Type'] = data_location_type.index
cols = data_location_type.columns.tolist()
cols = cols[-1:]+cols[:-1]
data_location_type = data_location_type[cols]
data_location_type = data_location_type[(data_location_type.Percentage) >= 0.1]
data_location_type = data_location_type.reset_index()
data_location_type = data_location_type.drop(columns=['index'],axis=1)
data_location_type

# Applying the above procedure for City-------------------

data_city = np.around(((data_mod['City'].value_counts()*100) /
data_mod['City'].value_counts().sum()),
                            decimals=2)
data_city = data_city.to_frame()
data_city = data_city.rename(columns={'City':'Percentage'})
data_city['City'] = data_city.index
cols = data_city.columns.tolist()
cols = cols[-1:]+cols[:-1]
data_city = data_city[cols]
data_city = data_city[(data_city.Percentage) >= 1.0]
data_city = data_city.reset_index()
data_city = data_city.drop(columns=['index'],axis=1)
data_city

# Applying the above procedure for Address Type-------------------

data_address_type = np.around(((data_mod['Address Type'].value_counts()*100) /
data_mod['Address Type'].value_counts().sum()),
                            decimals=2)
data_address_type = data_address_type.to_frame()
data_address_type = data_address_type.rename(columns={'Address
Type':'Percentage'})
data_address_type['Address Type'] = data_address_type.index
cols = data_address_type.columns.tolist()
cols = cols[-1:]+cols[:-1]
data_address_type = data_address_type[cols]
#data_address_type = data_address_type[(data_address_type.Percentage) >= 1.0]
data_address_type = data_address_type.reset_index()
data_address_type = data_address_type.drop(columns=['index'],axis=1)
data_address_type


fig, ax = plt.subplots(2, 2, figsize=(12, 10))

#sns.set_theme(style="whitegrid")
#plt.suptitle("Proportion of different outcomes for few interesting features.")
```

```
descriptor =
sns.barplot(ax=ax[0,0],x=data_descriptor.Descriptor,y=data_descriptor.Percentage,
)
descriptor.set_xticklabels(descriptor.get_xticklabels(), rotation=30, ha="right")

location_type = sns.barplot(ax=ax[0,1],x=data_location_type['Location
Type'],y=data_location_type.Percentage,)
location_type.set_xticklabels(location_type.get_xticklabels(), rotation=30,
ha="right")

city = sns.barplot(ax=ax[1,0],x=data_city['City'],y=data_city.Percentage,)
city.set_xticklabels(city.get_xticklabels(), rotation=30, ha="right")

address = sns.barplot(ax=ax[1,1],x=data_address_type['Address
Type'],y=data_address_type.Percentage,)
address.set_xticklabels(address.get_xticklabels(), rotation=30, ha="right")


#plt.subplots_adjust(left=None, bottom=None, right=None, top=0.0, wspace=None,
hspace=None)
plt.tight_layout()


data_place_CType_RCTime = data_mod[['City','Complaint
Type','Request_Closing_Time']]
data_place_CType_RCTime.dropna(subset = ['City','Complaint
Type','Request_Closing_Time'], inplace = True)
data_place_CType_RCTime['DeltaT(in_hr.)'] = np.around(
(data_place_CType_RCTime['Request_Closing_Time'].astype(np.int64)/
                                                      (pow(10,9)*3600) ),
decimals=2)
neg_time = data_place_CType_RCTime[data_place_CType_RCTime['DeltaT(in_hr.)'] <
0].sum()
print('The no negative time difference (Created Time > Clossing Time, which is
not possible) = \n',neg_time)
#data_place_CType_RCTime['DeltaT(in sec)/Avg.'] =
np.around((data_place_CType_RCTime['DeltaT(in sec)']/Avarage_time),decimals=1)
data_place_CType_RCTime.head(6)


Avarage_time =
np.around((data_place_CType_RCTime['DeltaT(in_hr.)'].mean()),decimals=2)
print('Avarage time gap between logging the complaint and problem solved =
',Avarage_time, 'hour')
```

```
Central_val =
np.around((data_place_CType_RCTime['DeltaT(in_hr.)'].median()),decimals=2)
print('Central value of the distribution = ',Central_val, 'hour')
Most_occoor =
np.around((data_place_CType_RCTime['DeltaT(in_hr.)'].mode()),decimals=2)
print('Most occered value = ',Most_occoor, 'hour')
stand_dev =
np.around((data_place_CType_RCTime['DeltaT(in_hr.)'].std()),decimals=2)
print('Deviation is = ',stand_dev)


conditions = [data_place_CType_RCTime['DeltaT(in_hr.)'] <= 0.5,
             (0.50 < data_place_CType_RCTime['DeltaT(in_hr.)']) &
(data_place_CType_RCTime['DeltaT(in_hr.)'] <= 1.00),
             (1.00 < data_place_CType_RCTime['DeltaT(in_hr.)']) &
(data_place_CType_RCTime['DeltaT(in_hr.)'] <= 2.00),
             (2.00 < data_place_CType_RCTime['DeltaT(in_hr.)']) &
(data_place_CType_RCTime['DeltaT(in_hr.)'] <= 6.00),
             (6.00 < data_place_CType_RCTime['DeltaT(in_hr.)']) &
(data_place_CType_RCTime['DeltaT(in_hr.)'] <= 10.00),
             (10.00 < data_place_CType_RCTime['DeltaT(in_hr.)'])]

choices = ['Super fast','Very fast','Fast','Normal','Slow','Super Slow']

data_place_CType_RCTime['Solution Status'] = np.select(conditions,choices)


data_place_CType_RCTime.head(6)


data_place_CType_RCTime['Solution Status'].value_counts()



data_place_CType_RCTime['Solution Status'].value_counts().plot(kind='bar')
plt.xlabel('Time Status')
plt.ylabel('Counts')
plt.title('Proportion of the fastness of different Solution status')
plt.show()
plt.tight_layout()
data_mod['Created Date'].head(5)


# Creating a data frame Contain Days and Months of Complaint date ---------------
--------------
```

```
Year_Month_Day = pd.to_datetime(data_mod['Created Date'].dt.date)
Month_Day = pd.DataFrame()
Month_Day['Date'] = pd.to_datetime(Year_Month_Day.dt.date)
Month_Day['Month'] = Year_Month_Day.dt.month
Month_Day['Day'] = Year_Month_Day.dt.day
Month_Day['Month Name'] = Month_Day['Month'].apply(lambda x:
calendar.month_abbr[x])
Month_Day['Day No'] = Month_Day['Date'].dt.weekday
Month_Day['Day Name'] = Month_Day['Day
No'].map({0:'Monday',1:'Tuesday',2:'Wednesday',3:'Thursday',4:'Friday',
                                                  5:'Saturday',6:'Sunday'})
Month_Day.sample(20)

Month_plot = Month_Day['Month Name'].value_counts()
Month_plot = Month_plot.to_frame()
Month_plot = Month_plot.rename(columns={'Month Name':'Counts'})
Month_plot

Day_plot = Month_Day['Day Name'].value_counts()
Day_plot = Day_plot.to_frame()
Day_plot = Day_plot.rename(columns={'Day Name':'Counts'})
Day_plot


fig, axes = plt.subplots(1,2, figsize=(14,8))

axes[0].pie(Month_plot['Counts'], labels = Month_plot.index,autopct='%1.1f%%')
axes[0].set_title('Complain logged in different months of the year')

axes[1].pie(Day_plot['Counts'], labels = Day_plot.index,autopct='%1.1f%%')
axes[1].set_title('Complain logged in different days of the year')

plt.tight_layout()


Month_Day_grouped = Month_Day.groupby(['Month Name','Day
Name'],as_index=False)['Day No'].count()
Month_Day_grouped_final = Month_Day_grouped.rename(columns={'Day No':'Counts'})
Month_Day_grouped_final.head(15)


Month_Day[( (Month_Day['Month Name'] == 'Apr') & (Month_Day['Day Name'] ==
'Monday') )].count()

plt.figure(figsize=(20,8))
```

```
month_day_plot = sns.barplot(x=Month_Day_grouped_final['Month Name'],
y=Month_Day_grouped_final['Counts'],
                             hue=Month_Day_grouped_final['Day Name'],
data=Month_Day_grouped_final)
month_day_plot.set_xticklabels(month_day_plot.get_xticklabels(), rotation=30,
ha="right")
plt.title('Distribution of daily complain in different months of the year')
plt.show()
plt.tight_layout()


Month_Day_grouped[Month_Day_grouped['Month Name'] == 'Mar']

data_mod['Status'].value_counts().plot(kind='barh')
plt.xlabel('Status')
plt.ylabel('Counts')
plt.title('Proportion of different Solution status')
plt.show()
plt.tight_layout()

Complaint_City_AvgTime_grouped =
data_place_CType_RCTime.groupby(['City','Complaint
Type']).agg({'DeltaT(in_hr.)':'mean'})
Complaint_City_AvgTime_grouped = Complaint_City_AvgTime_grouped.rename(
    columns={'DeltaT(in_hr.)':'Avg. Time(Given City, Complaint Type)'})
Complaint_City_AvgTime_grouped = Complaint_City_AvgTime_grouped.transform('Avg.
Time(Given City, Complaint Type)')
Complaint_City_AvgTime_grouped = Complaint_City_AvgTime_grouped.to_frame()
Complaint_City_AvgTime_grouped = Complaint_City_AvgTime_grouped.sort_values(
    ['City','Avg. Time(Given City, Complaint Type)'])

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
Complaint_City_AvgTime_grouped

import scipy.stats as stat
# Average response time across complaint types --------------------

Complaint_AvgTime = data_place_CType_RCTime.groupby(['Complaint
Type']).agg({'DeltaT(in_hr.)':'mean'})
Complaint_AvgTime = pd.DataFrame(Complaint_AvgTime)
Complaint_AvgTime =
Complaint_AvgTime.sort_values(['DeltaT(in_hr.)']).reset_index()
Complaint_AvgTime
```

```
Tmean_without = float(Complaint_AvgTime[Complaint_AvgTime['Complaint
Type']!='Animal in a Park'].mean())
print("Without complaint type 'Animal in a Park' ----- ",Tmean_without)
Tmean_with = float(Complaint_AvgTime['DeltaT(in_hr.)'].mean())
print("With complaint type 'Animal in a Park' ----- ",Tmean_with)

ttest_with, pval_with = stat.ttest_1samp(Complaint_AvgTime['DeltaT(in_hr.)'],
Tmean_with)
print('T-statistic is =',ttest_with)
print('p value is =',np.around(pval_with))

if (pval_with<0.05):
    print('Null hypothesis is rejected since p value ({}) is less than
0.05'.format(np.around(pval_with,decimals=2)))
else:
    print('Null hypothesis is accepted since p value ({}) is greater than
0.05'.format(np.around(pval_with,decimals=2)))

Complaint_AvgTime_without = Complaint_AvgTime.drop([len(Complaint_AvgTime)-
1],axis=0)
Complaint_AvgTime_without

ttest_without, pval_without =
stat.ttest_1samp(Complaint_AvgTime_without['DeltaT(in_hr.)'], Tmean_without)
print('T-statistic is =',ttest_without)
print('p value is =',np.around(pval_without,decimals=8))

if (pval_without<0.05):
    print('Null hypothesis is rejected since p value ({}) is less than
0.05'.format(np.around(pval_without,decimals=2)))
else:
    print('Null hypothesis is accepted since p value ({}) is greater than
0.05'.format(np.around(pval_without,decimals=2)))

#### (b) 2-sample T-test

sample1 = Complaint_AvgTime.sample(frac=.5)
sample1

sample2 = Complaint_AvgTime.drop(sample1.index)
sample2
```

```
print('Mean of 1st sample
=',np.around(float(sample1['DeltaT(in_hr.)'].mean()),decimals=2))
print('Standard dev. of 1st sample
=',np.around(float(sample1['DeltaT(in_hr.)'].std()),decimals=2))
print('Mean of 2nd sample
=',np.around(float(sample2['DeltaT(in_hr.)'].mean()),decimals=2))
print('Standard dev. of 2nd sample
=',np.around(float(sample2['DeltaT(in_hr.)'].std()),decimals=2))


ttest_2sp, p_val =
stat.ttest_ind(sample1['DeltaT(in_hr.)'],sample2['DeltaT(in_hr.)'])
print('T-statistic is =',ttest_2sp)
print('p value is =',np.around(p_val,decimals=2))



if (p_val<0.05):
    print('Null hypothesis is rejected since p value ({}) is less than
0.05'.format(np.around(p_val,decimals=2)))
else:
    print('Null hypothesis is accepted since p value ({}) is greater than
0.05'.format(np.around(p_val,decimals=2)))

sample1_anova = Complaint_AvgTime.sample(frac=1/3)
sample1_anova

rest_data = Complaint_AvgTime.drop(sample1_anova.index)
rest_data

sample2_anova = rest_data.sample(frac=1/2)
sample2_anova

sample3_anova = rest_data.drop(sample2_anova.index)
sample3_anova

print('Mean of 1st sample
=',np.around(float(sample1_anova['DeltaT(in_hr.)'].mean()),decimals=2))
print('Standard dev. of 1st sample
=',np.around(float(sample1_anova['DeltaT(in_hr.)'].std()),decimals=2))
print('Mean of 2nd sample
=',np.around(float(sample2_anova['DeltaT(in_hr.)'].mean()),decimals=2))
print('Standard dev. of 2nd sample
=',np.around(float(sample2_anova['DeltaT(in_hr.)'].std()),decimals=2))
```

```
print('Mean of 3rd sample
=',np.around(float(sample3_anova['DeltaT(in_hr.)'].mean()),decimals=2))
print('Standard dev. of 3rd sample
=',np.around(float(sample3_anova['DeltaT(in_hr.)'].std()),decimals=2))

f_val,p_val = stat.shapiro(sample1_anova['DeltaT(in_hr.)'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))

f_val,p_val = stat.shapiro(sample2_anova['DeltaT(in_hr.)'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))

f_val,p_val = stat.shapiro(sample3_anova['DeltaT(in_hr.)'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))

f_val,p_val =
stat.levene(sample1_anova['DeltaT(in_hr.)'],sample2_anova['DeltaT(in_hr.)'],sampl
e3_anova['DeltaT(in_hr.)'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))

f_val,p_val =
stat.f_oneway(sample1_anova['DeltaT(in_hr.)'],sample2_anova['DeltaT(in_hr.)'],sam
ple3_anova['DeltaT(in_hr.)'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))

if (p_val<0.05):
    print('Null hypothesis is rejected since p value ({}) is less than
0.05'.format(np.around(p_val,decimals=2)))
else:
    print('Null hypothesis is accepted since p value ({}) is greater than
0.05'.format(np.around(p_val,decimals=2)))

t_val,p_val =
stat.ttest_ind(sample1_anova['DeltaT(in_hr.)'],sample2_anova['DeltaT(in_hr.)'])
print('T-statistic for sample 1 and 2 is =',t_val)
print('p value is =',np.around(p_val,decimals=2))

t_val,p_val =
stat.ttest_ind(sample1_anova['DeltaT(in_hr.)'],sample3_anova['DeltaT(in_hr.)'])
print('T-statistic for sample 1 and 3 is =',t_val)
print('p value is =',np.around(p_val,decimals=2))
```

```python
t_val,p_val =
stat.ttest_ind(sample2_anova['DeltaT(in_hr.)'],sample3_anova['DeltaT(in_hr.)'])
print('T-statistic for sample 2 and 3 is =',t_val)
print('p value is =',np.around(p_val,decimals=2))

print('Null data in Complaint Type =',data_mod['Complaint Type'].isnull().sum())
print('Null data in City =',data_mod['City'].isnull().sum())


df_cc = data_mod[['Complaint Type','City']]
df_cc = df_cc.dropna()
#df_cc.isnull().sum()
#df_cc

City_Complaint = pd.crosstab(data_mod['Complaint
Type'],data_mod['City'],margins=True, margins_name='Total')
#City_Complaint = pd.crosstab(df_cc['Complaint Type'],df_cc['City'])
City_Complaint.head(6)


print("For 'ARVERNE' and 'ASTORIA' pair -------")
f_val,p_val = stat.f_oneway(City_Complaint['ARVERNE'],City_Complaint['ASTORIA'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))


print("For 'ARVERNE' and 'BROOKLYN' pair -------")
f_val,p_val = stat.f_oneway(City_Complaint['ARVERNE'],City_Complaint['BROOKLYN'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))


print("For 'HOLLIS' and 'JAMAICA' pair -------")
f_val,p_val = stat.f_oneway(City_Complaint['HOLLIS'],City_Complaint['JAMAICA'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))


print("For 'MASPETH' and 'QUEENS' pair -------")
f_val,p_val = stat.f_oneway(City_Complaint['MASPETH'],City_Complaint['QUEENS'])
print('F-statistic is =',f_val)
print('p value is =',np.around(p_val,decimals=2))


chai2, p_val, df, exp_frq = stat.chi2_contingency(City_Complaint)
```

```
print('Chai square value =',chai2)
print('p-value is =',p_val)
if (p_val<0.05):
    print('Null hypothesis is rejected since p value ({}) is less than 
0.05'.format(np.around(p_val,decimals=2)))
else:
    print('Null hypothesis is accepted since p value ({}) is greater than 
0.05'.format(np.around(p_val,decimals=2)))
```