Police Overtime

```
print("Welcome to Police Overtime Team E! 🚓🚓🚓")
```

```
Welcome to Police Overtime Team E! 🚓🚓🚓
```

```
# import necessary libraries
import numpy as np                 # manipulating arrays
import scipy.stats                 # mathematical algorithms
import csv                         # opening csvs
import pandas as pd                # manipulating tabular data
import matplotlib.pyplot as plt    # regression plot
import math

import os
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
# os.chdir("/content/drive/MyDrive/datasets")
os.chdir("/content/drive/.shortcut-targets-by-id/1nymOaRxWTT19KO9aepkx
eFG9N_NJswKf/datasets")
```

```
Mounted at /content/drive
```

```
pwd
```

```
{"type":"string"}
```

I. Imports

Import Internal Affairs Officers

```
internal_affairs_officers =
pd.read_csv("internal_affairs_officers.csv")
```

Import Overtime Data

```
# # Read datas of Overtime Details of BPD (2012-2022)

overtime_data_list = []
overtime_2013 = pd.read_csv("overtime_dataset/details-2013.csv")
overtime_data_list.append(overtime_2013)
overtime_2014 = pd.read_csv("overtime_dataset/details-2014.csv")
overtime_data_list.append(overtime_2014)
overtime_2015 = pd.read_csv("overtime_dataset/details-2015.csv")
overtime_data_list.append(overtime_2015)
overtime_2016 = pd.read_csv("overtime_dataset/details-2016.csv")
```

```
overtime_data_list.append(overtime_2016)
overtime_2017 = pd.read_csv("overtime_dataset/details-2017.csv")
overtime_data_list.append(overtime_2017)
overtime_2018 = pd.read_csv("overtime_dataset/details-2018.csv")
overtime_data_list.append(overtime_2018)
overtime_2019 = pd.read_csv("overtime_dataset/details-2019.csv")
overtime_data_list.append(overtime_2019)
overtime_2020 = pd.read_csv("overtime_dataset/details-2020.csv")
overtime_data_list.append(overtime_2020)
overtime_2021 = pd.read_csv("overtime_dataset/details-2021.csv")
overtime_data_list.append(overtime_2021)
overtime_2022 = pd.read_csv("overtime_dataset/details-2022.csv")
overtime_data_list.append(overtime_2022)
```

Import Court Overtime Data

```
# Read datas of Court Overtime Details of BPD (2012-2022)
court_overtime_data_list = []

co_2012 = pd.read_csv("court_dataset/Court_Overtime_2012.csv")
court_overtime_data_list.append(co_2012)
co_2013 = pd.read_csv("court_dataset/Court_Overtime_2013.csv")
court_overtime_data_list.append(co_2013)
co_2014 = pd.read_csv("court_dataset/Court_Overtime_2014.csv")
court_overtime_data_list.append(co_2014)
co_2015 = pd.read_csv("court_dataset/Court_Overtime_2015.csv")
court_overtime_data_list.append(co_2015)
co_2016 = pd.read_csv("court_dataset/Court_Overtime_2016.csv")
court_overtime_data_list.append(co_2016)
co_2017 = pd.read_csv("court_dataset/Court_Overtime_2017.csv")
court_overtime_data_list.append(co_2017)
co_2018 = pd.read_csv("court_dataset/Court_Overtime_2018.csv")
court_overtime_data_list.append(co_2018)
co_2019 = pd.read_csv("court_dataset/Court_Overtime_2019.csv")
court_overtime_data_list.append(co_2019)
co_2020 = pd.read_csv("court_dataset/Court_Overtime_2020.csv")
court_overtime_data_list.append(co_2020)
co_2021 = pd.read_csv("court_dataset/Court_Overtime_2021.csv")
court_overtime_data_list.append(co_2021)
co_2022 = pd.read_csv("court_dataset/Court_Overtime_2022.csv")
court_overtime_data_list.append(co_2022)
```

Import Suffolk Brady List Data 2020

```
suffolk_brady_2020 = pd.read_excel("suffolk_brady_list_2020.xlsx")
```

Import Campaign Contribution Data

```
# # Read datas of Campaign Contributions (2010-2020)

campaign_contribution_data = []

all_bpd_contributions =
pd.read_csv("campaign_contribution_dataset/all_bpd_contributions.csv")
campaign_contribution_data.append(all_bpd_contributions)
all_cc_contributions =
pd.read_csv("campaign_contribution_dataset/all_cc_contributions.csv")
campaign_contribution_data.append(all_cc_contributions)
all_non_bpd_contributions =
pd.read_csv("campaign_contribution_dataset/all_non_bpd_contributions.c
sv")
campaign_contribution_data.append(all_non_bpd_contributions)
all_non_police_contributions =
pd.read_csv("campaign_contribution_dataset/all_non_police_contribution
s.csv")
campaign_contribution_data.append(all_non_police_contributions)
all_police_contributions =
pd.read_csv("campaign_contribution_dataset/all_police_contributions.cs
v")
campaign_contribution_data.append(all_police_contributions)
```

Import Earnings Data

```
# # Read datas of BPD earning (2011 - 2022)
# # Should not be touched now!

# BPD dataset
earning_data_list = []
e_2011 = pd.read_csv("bpd_dataset/earning-2011.csv")
earning_data_list.append(e_2011)
e_2012 = pd.read_csv("bpd_dataset/earning-2012.csv")
earning_data_list.append(e_2012)
e_2013 = pd.read_csv("bpd_dataset/earning-2013.csv")
earning_data_list.append(e_2013)
e_2014 = pd.read_csv("bpd_dataset/earning-2014.csv")
earning_data_list.append(e_2014)
e_2015 = pd.read_csv("bpd_dataset/earning-2015.csv")
earning_data_list.append(e_2015)
e_2016 = pd.read_csv("bpd_dataset/earning-2016.csv")
earning_data_list.append(e_2016)
e_2017 = pd.read_csv("bpd_dataset/earning-2017.csv")
earning_data_list.append(e_2017)
e_2018 = pd.read_csv("bpd_dataset/earning-2018.csv")
earning_data_list.append(e_2018)
e_2019 = pd.read_csv("bpd_dataset/earning-2019.csv")
earning_data_list.append(e_2019)
```

```
e_2020 = pd.read_csv("bpd_dataset/earning-2020.csv")
earning_data_list.append(e_2020)
e_2021 = pd.read_csv("bpd_dataset/earning-2021.csv")
earning_data_list.append(e_2021)
e_2022 = pd.read_csv("bpd_dataset/earning-2022.csv")
earning_data_list.append(e_2022)

# List of years for ploting
year = []
for i in range(11, 23):
    year += [2000 + i]


# non BPD datasets
earning_data_list_nonpd = []
ne_2011 = pd.read_csv("non_bpd_dataset/earning-2011.csv")
earning_data_list_nonpd.append(ne_2011)
ne_2012 = pd.read_csv("non_bpd_dataset/earning-2012.csv")
earning_data_list_nonpd.append(ne_2012)
ne_2013 = pd.read_csv("non_bpd_dataset/earning-2013.csv")
earning_data_list_nonpd.append(ne_2013)
ne_2014 = pd.read_csv("non_bpd_dataset/earning-2014.csv")
earning_data_list_nonpd.append(ne_2014)
ne_2015 = pd.read_csv("non_bpd_dataset/earning-2015.csv")
earning_data_list_nonpd.append(ne_2015)
ne_2016 = pd.read_csv("non_bpd_dataset/earning-2016.csv")
earning_data_list_nonpd.append(ne_2016)
ne_2017 = pd.read_csv("non_bpd_dataset/earning-2017.csv")
earning_data_list_nonpd.append(ne_2017)
ne_2018 = pd.read_csv("non_bpd_dataset/earning-2018.csv")
earning_data_list_nonpd.append(ne_2018)
ne_2019 = pd.read_csv("non_bpd_dataset/earning-2019.csv")
earning_data_list_nonpd.append(ne_2019)
ne_2020 = pd.read_csv("non_bpd_dataset/earning-2020.csv")
earning_data_list_nonpd.append(ne_2020)
ne_2021 = pd.read_csv("non_bpd_dataset/earning-2021.csv")
earning_data_list_nonpd.append(ne_2021)
ne_2022 = pd.read_csv("non_bpd_dataset/earning-2022.csv")
earning_data_list_nonpd.append(ne_2022)
```

Import Officers Data

```
officers = pd.read_csv("officers.csv")
```

Import Crime Incident Reports Data

```
# crime incidents preprocessing
crime_data_list = []
```

```
crime_2015 = pd.read_csv("crime_incidents/crime_incident_2015.csv",
low_memory=False)
crime_data_list.append(crime_2015)
crime_2016 = pd.read_csv("crime_incidents/crime_incident_2016.csv",
low_memory=False)
crime_data_list.append(crime_2016)
crime_2017 = pd.read_csv("crime_incidents/crime_incident_2017.csv",
low_memory=False)
crime_data_list.append(crime_2017)
crime_2018 = pd.read_csv("crime_incidents/crime_incident_2018.csv",
low_memory=False)
crime_data_list.append(crime_2018)
crime_2019 = pd.read_csv("crime_incidents/crime_incident_2019.csv",
low_memory=False)
crime_data_list.append(crime_2019)
crime_2020 = pd.read_csv("crime_incidents/crime_incident_2020.csv",
low_memory=False)
crime_data_list.append(crime_2020)
crime_2021 = pd.read_csv("crime_incidents/crime_incident_2021.csv",
low_memory=False)
crime_data_list.append(crime_2021)
crime_2022 = pd.read_csv("crime_incidents/crime_incident_2022.csv",
low_memory=False)
crime_data_list.append(crime_2022)
```

Import Field Contact Data

```
field_contact_list = []
field_2015 = pd.read_csv("field_activity_dataset/New_RMS/FieldContact-
2015.csv", low_memory=False)
field_contact_list.append(field_2015)
field_2016 = pd.read_csv("field_activity_dataset/New_RMS/FieldContact-
2016.csv", low_memory=False)
field_contact_list.append(field_2016)
field_2017 = pd.read_csv("field_activity_dataset/New_RMS/FieldContact-
2017.csv", low_memory=False)
field_contact_list.append(field_2017)
field_2018 = pd.read_csv("field_activity_dataset/New_RMS/FieldContact-
2018.csv", low_memory=False)
field_contact_list.append(field_2018)
field_2019 = pd.read_csv("field_activity_dataset/Mark43/FieldContact-
2019.csv", low_memory=False)
field_contact_list.append(field_2019)
field_2020 = pd.read_csv("field_activity_dataset/Mark43/FieldContact-
2020.csv", low_memory=False)
field_contact_list.append(field_2020)
field_2021 = pd.read_csv("field_activity_dataset/Mark43/FieldContact-
2021.csv", low_memory=False)
field_contact_list.append(field_2021)
```

```
field_2022 = pd.read_csv("field_activity_dataset/Mark43/FieldContact-
2022.csv", low_memory=False)
field_contact_list.append(field_2022)
```

Import BPD Personal Data

```
bpd_personnel = pd.read_excel("BPD_personnel_PRR_9_4_2020.xls")
```

II. Data Preprocessing

Earnings Data Preprocessing

```
# Pre-process data, change from str to float

# def convert_data(value):
#     if not isinstance(value, float):
#       if '-' in value or 'NaN' in value:
#           return 0.0
#       else:
#           return float(value.replace("$", "").replace(",",
"").replace("(","").replace(")",""))
#     else:
#       return 0.0



def contains_alphabetic(input_string):
    return any(char.isalpha() for char in input_string)

def convert_data(value):
    if value == None:
      return 0.0
    elif not isinstance(value, float):
      if '-' in value:
          return np.nan
      elif '(' in value:
         return -1 * float(value.replace("$", "").replace(",",
"").replace("(", "").replace(")", ""))
      elif not contains_alphabetic(value):
          return float(value.replace("$", "").replace(",", ""))
      else:
          return 0.0

# def convert_to_float(monetary_value):
#     # Remove currency symbols and commas, and convert parentheses to
negative numbers
#     if isinstance(monetary_value, str):
```

```
#         clean_value = monetary_value.replace('$', '').replace(',',
'')
#         if '(' in clean_value and ')' in clean_value:
#             clean_value = clean_value.replace('(', '').replace(')',
'')
#             return float(clean_value) * -1
#         elif '  -  ' in clean_value:
#             return 0.0
#         else:
#             return float(clean_value)
#     else:
#         return monetary_value

for data in earning_data_list:
  # change all name to uppercase to be consistent with the Overtime
dataset
  data['NAME'] = data['NAME'].str.upper()
  for column in data.columns[3:11]:
      data[column] = data[column].apply(convert_data)

for data in earning_data_list_nonpd:
  for column in data.columns[3:11]:
        data[column] = data[column].apply(convert_data)

# #  Sample display for 2013
# e_2016.head()
# standardize the column names across all datasets
# define a list of standardized column names
std_col_names = ['NAME', 'DEPT_NAME', 'TITLE', 'REGULAR', 'RETRO',
'OTHER', 'OVERTIME', 'INJURED', 'DETAIL', 'EDUCATION','TOTAL_EARNING',
'POSTAL']
df = None
for df in earning_data_list:
  df.columns = std_col_names
```

Overtime Data Preprocessing

```
# Pre-process data, change from str to float

# standardize the column names across all datasets
# define a list of standardized column names


col_names = ["JOB_NO", "EMPLOYEE_ID", "EMPLOYEE", "RANK", "LOCATION",
"XSTREET", "DATE", "START_TIME", "END_TIME", "HOURS_WORKED",
"HOURS_PAID", "TYPE", "CUSTOMER_NO", "CUSTOMER", "CUST_ADDRESS",
"CUST_ADDRESS_1", "CUST_ADDRESS_3", "CITY", "STATE", "ZIP"]
df = None
```

```
for df in overtime_data_list:
  df.columns = col_names

for data in overtime_data_list:
    for column in ['JOB_NO', "EMPLOYEE_ID", "RANK", "START_TIME",
"END_TIME", "HOURS_WORKED", "HOURS_PAID", "CUSTOMER_NO"]:
      if data[column].dtype == 'object':
          data[column] = pd.to_numeric(data[column],
errors='coerce').fillna(0).astype(int)

print(overtime_data_list[0].head())
```

```
   JOB_NO  EMPLOYEE_ID           EMPLOYEE  RANK         LOCATION
XSTREET  \
0   11490        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV
NaN
1   11528        12011  BAUSEMER,DANIEL P     9  COMMONWEALTH AV
NaN
2   11528        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV
NaN
3   11500        11165     ARAICA,HENRY A     9        TALBOT AV
NaN
4   11500        86212       STEELE,MEL A     9         RIVER ST
NaN

                  DATE  START_TIME  END_TIME  HOURS_WORKED  HOURS_PAID
TYPE  \
0  2013-11-13 00:00:00           0       530           5.5           8
Z
1  2013-11-15 00:00:00           0       530           5.5           8
Z
2  2013-11-15 00:00:00           0       530           5.5           8
Z
3  2013-11-15 00:00:00         830      1400           5.5           8
Z
4  2013-11-15 00:00:00         830      1430           6.0           8
Z

   CUSTOMER_NO CUSTOMER    CUST_ADDRESS  CUST_ADDRESS_1
CUST_ADDRESS_3  \
0         1103  VERIZON  649 SUMMER ST.             NaN
NaN
1         1103  VERIZON  649 SUMMER ST.             NaN
NaN
2         1103  VERIZON  649 SUMMER ST.             NaN
NaN
3         1103  VERIZON  649 SUMMER ST.             NaN
NaN
4         1103  VERIZON  649 SUMMER ST.             NaN
```

```
NaN

     CITY STATE    ZIP
0  BOSTON    MA  02210
1  BOSTON    MA  02210
2  BOSTON    MA  02210
3  BOSTON    MA  02210
4  BOSTON    MA  02210
```

Court Overtime Data Preprocessing

```
col_names = ["ID", "NAME", "RANK", "ASSIGNED_DESC", "CHARGED_DESC",
"OTDATE", "OTCODE", "DESCRIPTION", "STARTTIME", "ENDTIME", "WRKDHRS",
"OTHOURS"]

for df in court_overtime_data_list:
    df.columns = col_names

for data in court_overtime_data_list:
    for column in ['ID', "STARTTIME", "ENDTIME", "WRKDHRS",
"OTHOURS"]:
        if data[column].dtype == 'object':
            data[column] = pd.to_numeric(data[column],
errors='coerce').fillna(0).astype(float)

print(court_overtime_data_list[0].head())
```

```
       ID                NAME RANK ASSIGNED_DESC CHARGED_DESC
OTDATE  \
0  103591  Bissonnette,Philip  Ptl   DISTRICT 03  DISTRICT 03
01/04/12
1  103782     Rooney,Kevin D.  Ptl   DISTRICT 03  DISTRICT 03
01/03/12
2   11045        Ruiz,Jose A.  Ptl   DISTRICT 03  DISTRICT 03
01/03/12
3    9726     Doherty,Henry J  Ptl   DISTRICT 03  DISTRICT 03
01/04/12
4   11395     Boylan,Edward J  Ptl   DISTRICT 11  DISTRICT 11
01/03/12

   OTCODE         DESCRIPTION  STARTTIME  ENDTIME  WRKDHRS  OTHOURS
0     280         COURT:TRIAL        900      915     0.25      4.0
1     280         COURT:TRIAL        915      930     0.25      4.0
2     283  COURT:MOTIONS HRG.        830     1000     1.50      4.0
3     280         COURT:TRIAL        830      915     0.75      4.0
4     280         COURT:TRIAL        830     1000     1.50      4.0
```

Campaign Contribution Data Preprocessing

```
for data in campaign_contribution_data:
    print(data.columns)
    for column in ['Amount', 'Datetime']:
      if column == 'Datetime':
          data['Datetime'] = pd.to_datetime(data['Datetime'])
      elif data[column].dtype == 'object':
          data[column] = pd.to_numeric(data[column],
errors='coerce').fillna(0).astype(int)
```

```
Index(['Address', 'Amount', 'CPF ID', 'City', 'Contributor', 'Date',
       'Datetime', 'Employer', 'Occupation', 'Principal Officer',
'Recipient',
       'Record Type Description', 'Record Type ID', 'Source
Description',
       'State', 'Tender Type Description', 'Tender Type ID', 'UUID',
'Zip'],
      dtype='object')
Index(['Date', 'Contributor', 'Address', 'City', 'State', 'Zip',
'Occupation',
       'Employer', 'Principal Officer', 'Amount', 'CPF ID',
'Recipient',
       'Tender Type ID', 'Tender Type Description', 'Record Type ID',
       'Record Type Description', 'Source Description', 'Datetime',
'UUID'],
      dtype='object')
Index(['Address', 'Amount', 'CPF ID', 'City', 'Contributor', 'Date',
       'Datetime', 'Employer', 'Occupation', 'Principal Officer',
'Recipient',
       'Record Type Description', 'Record Type ID', 'Source
Description',
       'State', 'Tender Type Description', 'Tender Type ID', 'UUID',
'Zip'],
      dtype='object')
Index(['Date', 'Contributor', 'Address', 'City', 'State', 'Zip',
'Occupation',
       'Employer', 'Principal Officer', 'Amount', 'CPF ID',
'Recipient',
       'Tender Type ID', 'Tender Type Description', 'Record Type ID',
       'Record Type Description', 'Source Description', 'Datetime',
'UUID'],
      dtype='object')
Index(['Address', 'Amount', 'CPF ID', 'City', 'Contributor', 'Date',
       'Datetime', 'Employer', 'Occupation', 'Principal Officer',
'Recipient',
       'Record Type Description', 'Record Type ID', 'Source
Description',
       'State', 'Tender Type Description', 'Tender Type ID', 'UUID',
'Zip'],
      dtype='object')
```

Internal Affairs Officers Data Preprocessing

```
col_names = ["ia_number", "case_number", "incident_type",
"received_date", "occurred_date", "summary", "name", "title", "badge",
"allegation", "finding", "finding_date", "action_taken",
"officer_active", "officer_employee_id", "officer_name",
"officer_organization", "officer_title", "officer_doa",
"officer_badge", "officer_zip_code", "officer_city", "officer_state",
"officer_neighborhood", "officer_regular", "officer_retro",
"officer_other", "officer_overtime", "officer_injured",
"officer_detail", "officer_quinn", "officer_total", "officer_rank",
"officer_ia_score", "officer_ia_sustained_conduct_unbecoming",
"officer_ia_sustained_neg_duty",
"officer_ia_sustained_respectful_treatment",
"officer_ia_sustained_self_identification",
"officer_ia_sustained_use_of_force", "officer_ia_sustained_details",
"officer_ia_sustained_cases", "officer_ia_sustained_allegations",
"officer_ia_cases", "officer_ia_allegations",
"officer_field_contacts_count", "officer_incidents_count",
"officer_complaints_count", "officer_swats_count",
"officer_details_count", "officer_citations_count",
"officer_articles_officers_count", "officer_retirement_date",
"officer_retirement_amount", "officer_lead_added",
"officer_lead_entry", "officer_url", "received_year", "YEAR"]

# Converting monetary columns to numeric
monetary_columns = ['officer_regular', 'officer_retro',
'officer_other', 'officer_overtime', 'officer_injured',
'officer_detail', 'officer_quinn', 'officer_total',
'officer_retirement_amount']
for column in monetary_columns:
    if internal_affairs_officers[column].dtype == 'object':
        internal_affairs_officers[column] =
pd.to_numeric(internal_affairs_officers[column],
errors='coerce').fillna(0).astype(int)

print(internal_affairs_officers.head())
```

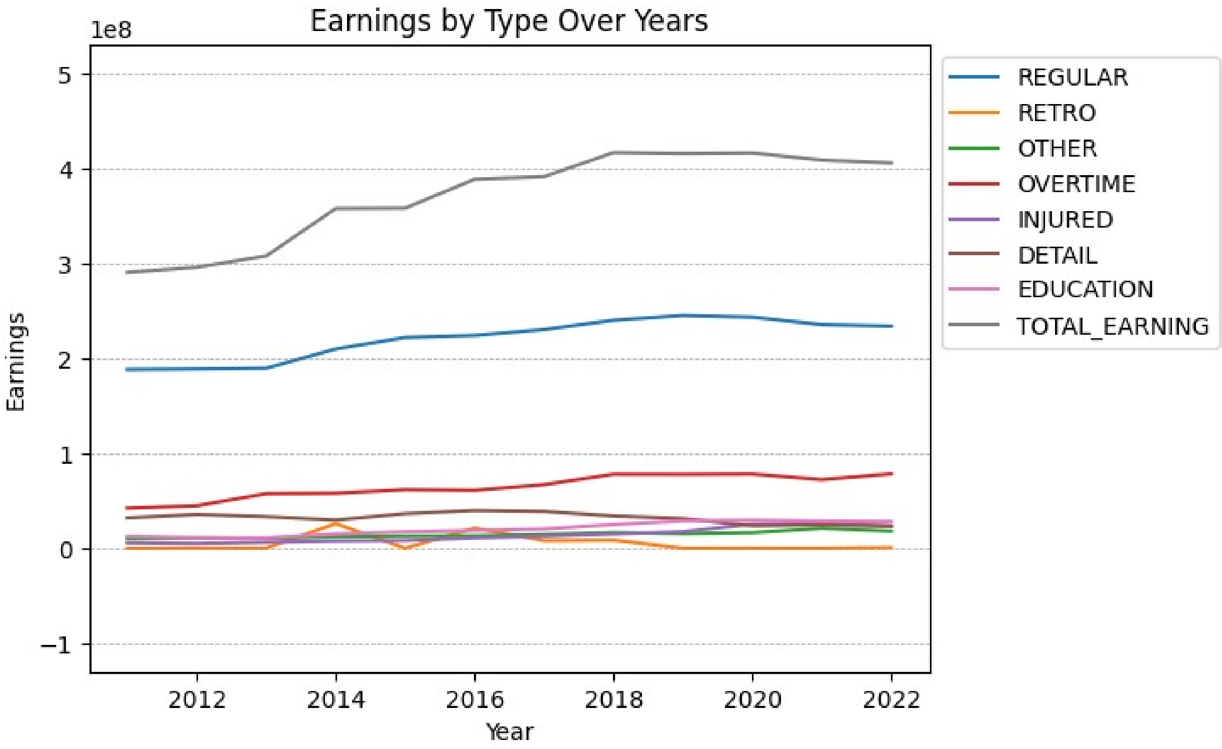
```
      ia_number  case_number           incident_type received_date  \
0  IAD2012-0198          NaN       Citizen complaint    2012-06-04
1  IAD2016-0326          NaN  Internal investigation    2016-08-25
2  IAD2012-0198          NaN       Citizen complaint    2012-06-04
3  IAD2016-0326          NaN  Internal investigation    2016-08-25
4  IAD2016-0328          NaN  Internal investigation    2016-08-28

  occurred_date summary                name           title badge  \
0           NaN     NaN             Unknown  Police Officer   NaN
```

```
1            NaN     NaN  Kenneally,John F.  Police Officer  1696
2            NaN     NaN            Unknown  Police Officer   NaN
3            NaN     NaN  Kenneally,John F.  Police Officer  1696
4            NaN     NaN  Sandefur,Roland D  Police Officer  4667

                               allegation  ...
officer_complaints_count  \
0         Respectful Treatment (2 counts)  ...
NaN
1             Neg.Duty/Unreasonable Judge  ...
4.0
2                            Use of Force  ...
NaN
3  Uniform & Equipment-Care & Maintenance  ...
4.0
4                       Directives/Orders  ...
8.0

  officer_swats_count officer_details_count officer_citations_count  \
0                 NaN                   NaN                     NaN
1                 0.0                 132.0                   197.0
2                 NaN                   NaN                     NaN
3                 0.0                 132.0                   197.0
4                 0.0                  24.0                    20.0

   officer_articles_officers_count officer_retirement_date  \
0                              NaN                     NaN
1                              5.0                     NaN
2                              NaN                     NaN
3                              5.0                     NaN
4                              9.0                     NaN

  officer_retirement_amount officer_lead_added officer_lead_entry  \
0                       NaN                NaN                NaN
1                       NaN                NaN                NaN
2                       NaN                NaN                NaN
3                       NaN                NaN                NaN
4                       NaN                NaN                NaN

                                         officer_url
0                                                NaN
1  https://www.wokewindows.org/officers/12021-joh...
2                                                NaN
3  https://www.wokewindows.org/officers/12021-joh...
4  https://www.wokewindows.org/officers/11360-rol...

[5 rows x 56 columns]
```

III. Exploratory Data Analysis (EDA)

Total Earnings for each type over years

```
# types = 2D list, each element is a list for a singel year, and each
list contains all the column name
types = [[] for _ in range(len(earning_data_list))]
# values = 2D list, each element is a list for a single year, and each
list contains all column's value sum
values = [[] for _ in range(len(earning_data_list))]

for j in range(len(earning_data_list)):
    for i in range(8):
        types[j].append(earning_data_list[j].columns[i+3])
        values[j].append(earning_data_list[j][types[j][i]].sum())

years = [i for i in range(2011, 2023)]

values_each_year = [[] for _ in range(len(values[0]))]

for i in range(len(values[0])):
  for j in range(len(values)):
    values_each_year[i].append(values[j][i])

for i in range(len(types[0])):  # Assuming each year has the same
types
    plt.plot(years, values_each_year[i], label=types[0][i])

# Adding title and labels
plt.title('Earnings by Type Over Years')
plt.xlabel('Year')
# Get current y-axis ticks
y_values = plt.gca().get_yticks()

# Add horizontal line for each y tick value
for y in y_values:
    plt.axhline(y=y, color='gray', linestyle='--', linewidth=0.5,
alpha=0.7)

plt.ylabel('Earnings')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Show the plot
plt.show()
```



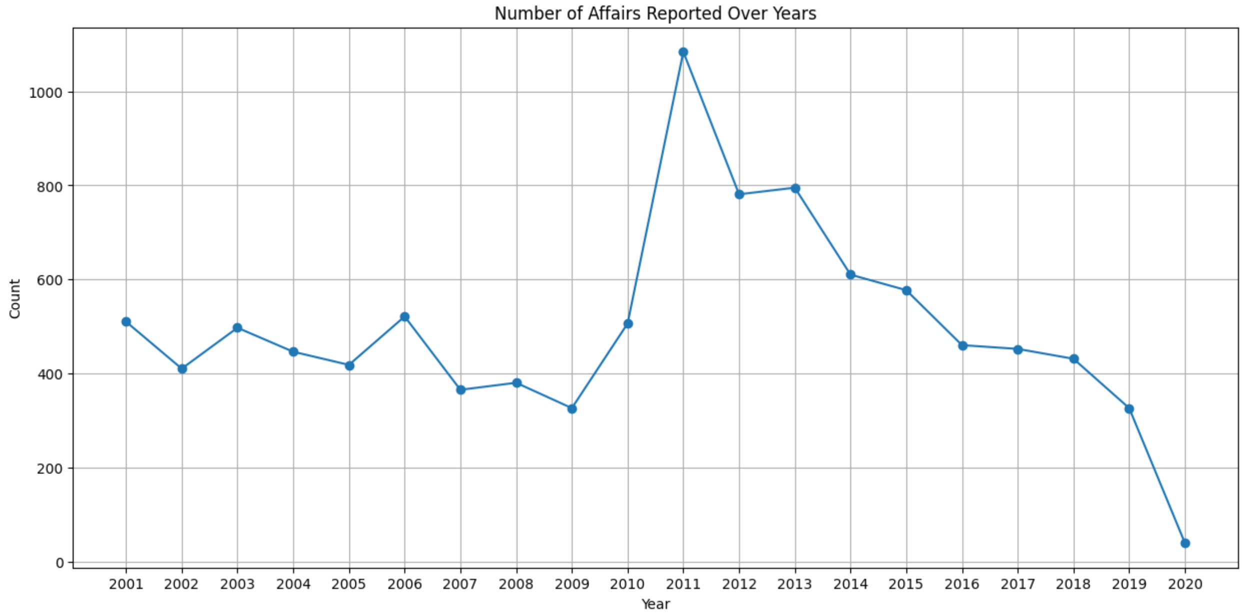
Number of internal affairs over year

```
import copy
# Create a deep copy of internal_affairs_officers
affairs = copy.deepcopy(internal_affairs_officers)

affairs.head(5)

      ia_number  case_number           incident_type received_date  \
0  IAD2012-0198          NaN       Citizen complaint    2012-06-04
1  IAD2016-0326          NaN  Internal investigation    2016-08-25
2  IAD2012-0198          NaN       Citizen complaint    2012-06-04
3  IAD2016-0326          NaN  Internal investigation    2016-08-25
4  IAD2016-0328          NaN  Internal investigation    2016-08-28

  occurred_date summary               name          title badge  \
0           NaN     NaN            Unknown  Police Officer   NaN
1           NaN     NaN  Kenneally,John F.  Police Officer  1696
2           NaN     NaN            Unknown  Police Officer   NaN
3           NaN     NaN  Kenneally,John F.  Police Officer  1696
4           NaN     NaN  Sandefur,Roland D  Police Officer  4667

                               allegation  ...
officer_complaints_count  \
0          Respectful Treatment (2 counts)  ...
NaN
1              Neg.Duty/Unreasonable Judge  ...
```

```
4.0
2                            Use of Force  ...
NaN
3  Uniform & Equipment-Care & Maintenance  ...
4.0
4                       Directives/Orders  ...
8.0

  officer_swats_count officer_details_count officer_citations_count  \
0                 NaN                   NaN                     NaN
1                 0.0                 132.0                   197.0
2                 NaN                   NaN                     NaN
3                 0.0                 132.0                   197.0
4                 0.0                  24.0                    20.0

   officer_articles_officers_count officer_retirement_date  \
0                              NaN                     NaN
1                              5.0                     NaN
2                              NaN                     NaN
3                              5.0                     NaN
4                              9.0                     NaN

  officer_retirement_amount officer_lead_added officer_lead_entry  \
0                       NaN                NaN                NaN
1                       NaN                NaN                NaN
2                       NaN                NaN                NaN
3                       NaN                NaN                NaN
4                       NaN                NaN                NaN

                                         officer_url
0                                                NaN
1  https://www.wokewindows.org/officers/12021-joh...
2                                                NaN
3  https://www.wokewindows.org/officers/12021-joh...
4  https://www.wokewindows.org/officers/11360-rol...

[5 rows x 56 columns]
```

```python
# Drop UNKNOWN and NaN for name column
affairs = affairs.dropna(subset=['name'])
affairs = affairs[affairs['name'] != "UNKNOWN"]
affairs["name"] = affairs["name"].str.upper().str.strip()
affairs.rename(columns={'name': 'EMPLOYEE'}, inplace=True)

# Extract the year and update the 'received_date' column
affairs['received_date'] = pd.to_datetime(affairs['received_date'])
affairs['year'] = affairs['received_date'].dt.year
count_overyear = affairs.groupby('year').size().sort_index()
```

```
# Plot the results with dots and lines between them
plt.figure(figsize=(15, 7))
plt.plot(count_overyear.index, count_overyear.values, '-o')  # Line
with dots

# Set x-axis ticks to show every distinct year as integer values
plt.xticks(ticks=count_overyear.index,
labels=count_overyear.index.astype(int))

# Set the plot labels and title
plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Number of Affairs Reported Over Years')
plt.grid(True)

# Show the plot
plt.show()
```

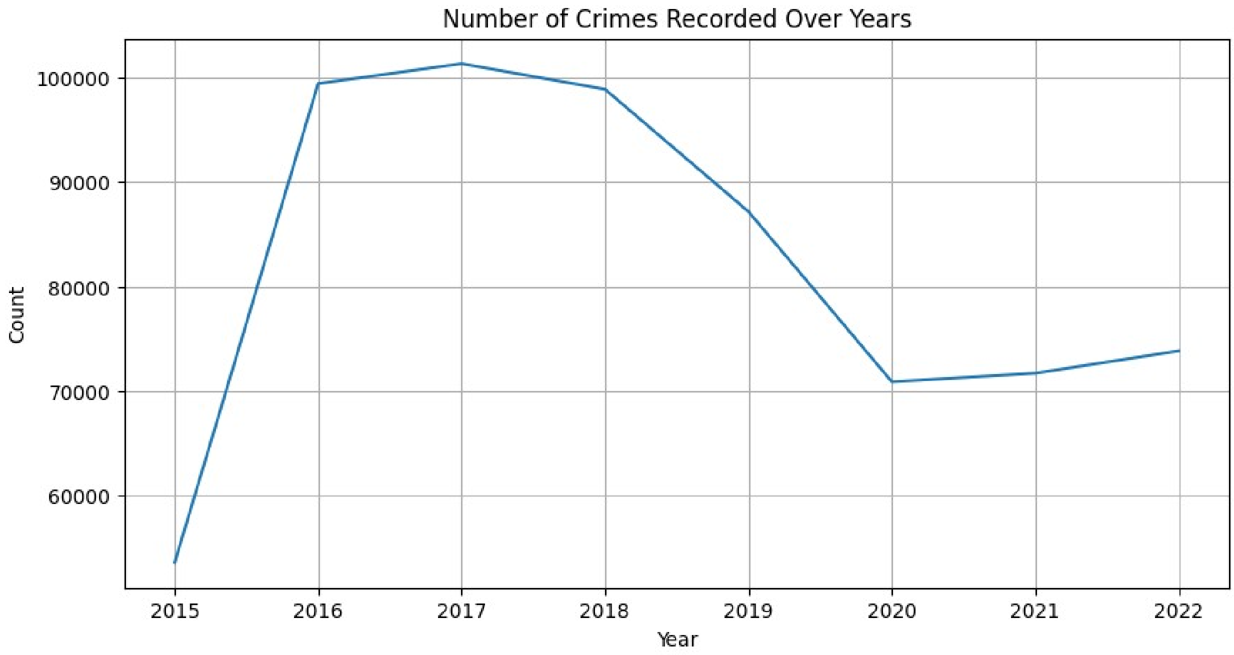


Number of Crime Recorded over years

```
crime_data_list[0].head(5)

  INCIDENT_NUMBER  OFFENSE_CODE  OFFENSE_CODE_GROUP  \
0      I172040657          2629          Harassment
1      I182061268          3201       Property Lost
2      I162013546          3201       Property Lost
3      I152051083          3115  Investigate Person
4      I152059178          2647               Other

          OFFENSE_DESCRIPTION DISTRICT REPORTING_AREA SHOOTING  \
0                  HARASSMENT      C11            397      NaN
```

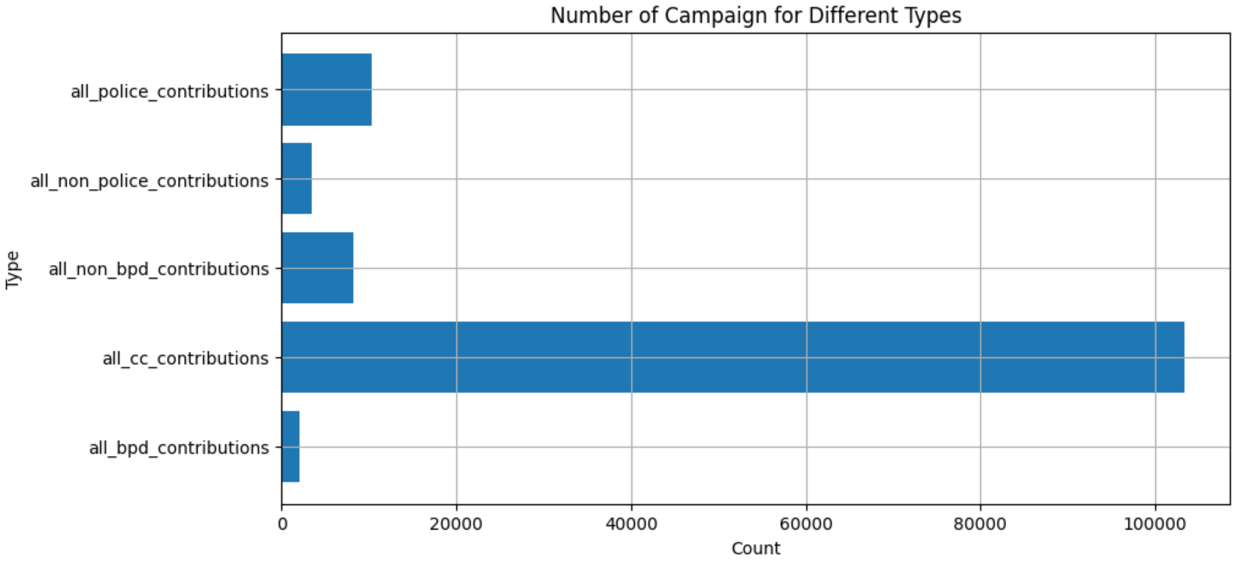
```
1            PROPERTY - LOST      NaN                     NaN
2            PROPERTY - LOST       B3           433       NaN
3         INVESTIGATE PERSON       A7            20       NaN
4  THREATS TO DO BODILY HARM      C11           359       NaN

      OCCURRED_ON_DATE  YEAR  MONTH DAY_OF_WEEK  HOUR    UCR_PART  \
0  2015-06-15 00:00:00  2015      6      Monday     0    Part Two
1  2015-06-15 00:00:00  2015      6      Monday     0  Part Three
2  2015-06-15 00:00:00  2015      6      Monday     0  Part Three
3  2015-06-15 00:00:00  2015      6      Monday     0  Part Three
4  2015-06-15 00:00:00  2015      6      Monday     0    Part Two

          STREET        Lat       Long                     Location
0   MELBOURNE ST  42.291093 -71.065945  (42.29109287, -71.06594539)
1        BERNARD        NaN        NaN                          NaN
2     NORFOLK ST  42.283634 -71.082813  (42.28363434, -71.08281320)
3       PARIS ST  42.377023 -71.032247  (42.37702319, -71.03224730)
4  WASHINGTON ST  42.293606 -71.071887  (42.29360585, -71.07188650)
```

```
crime_year = [i for i in range(2015, 2023)]
num_crime_overyear = []

for i in range(len(crime_year)):
  num_crime_overyear.append(len(crime_data_list[i]))

plt.figure(figsize=(10, 5))  # Set the figure size (optional)
plt.plot(crime_year, num_crime_overyear)  # Create a scatter plot
plt.title('Number of Crimes Recorded Over Years')  # Add a title
plt.xlabel('Year')  # Label the x-axis
plt.ylabel('Count')  # Label the y-axis
plt.grid(True)  # Show a grid for easier reading of the plot
plt.show()  # Display the plot
```



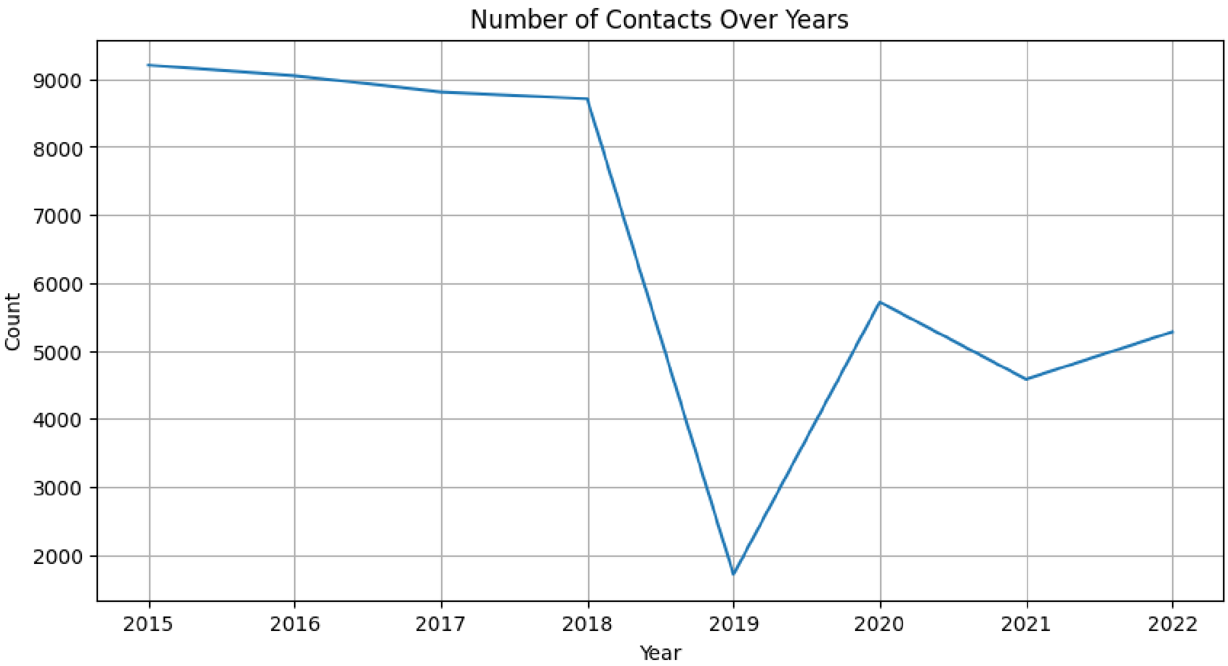
Number of Campaign for Different Types

```
num_campaign = []
type_campaign = ['all_bpd_contributions', 'all_cc_contributions',
'all_non_bpd_contributions', 'all_non_police_contributions',
'all_police_contributions']

for df in campaign_contribution_data:
  num_campaign.append(len(df))

plt.figure(figsize=(10, 5))  # Set the figure size
plt.barh(type_campaign, num_campaign)  # Create a horizontal bar chart
plt.ylabel('Type')  # Correct the label for the y-axis
plt.xlabel('Count')  # Correct the label for the x-axis
plt.title('Number of Campaign for Different Types')  # Add a title
plt.grid(True)  # Show a grid
plt.show()  # Display the plot
```



Number of Contacts over years

```
field_contact_overyear = []
field_year = [i for i in range(2015, 2023)]

for i in range(len(field_year)):
  field_contact_overyear.append(len(field_contact_list[i]))

plt.figure(figsize=(10, 5))  # Set the figure size
plt.plot(field_year, field_contact_overyear)  # Create a horizontal bar chart
plt.ylabel('Count')  # Correct the label for the y-axis
plt.xlabel('Year')  # Correct the label for the x-axis
plt.title('Number of Contacts Over Years')  # Add a title
plt.grid(True)  # Show a grid
plt.show()  # Display the plot
```

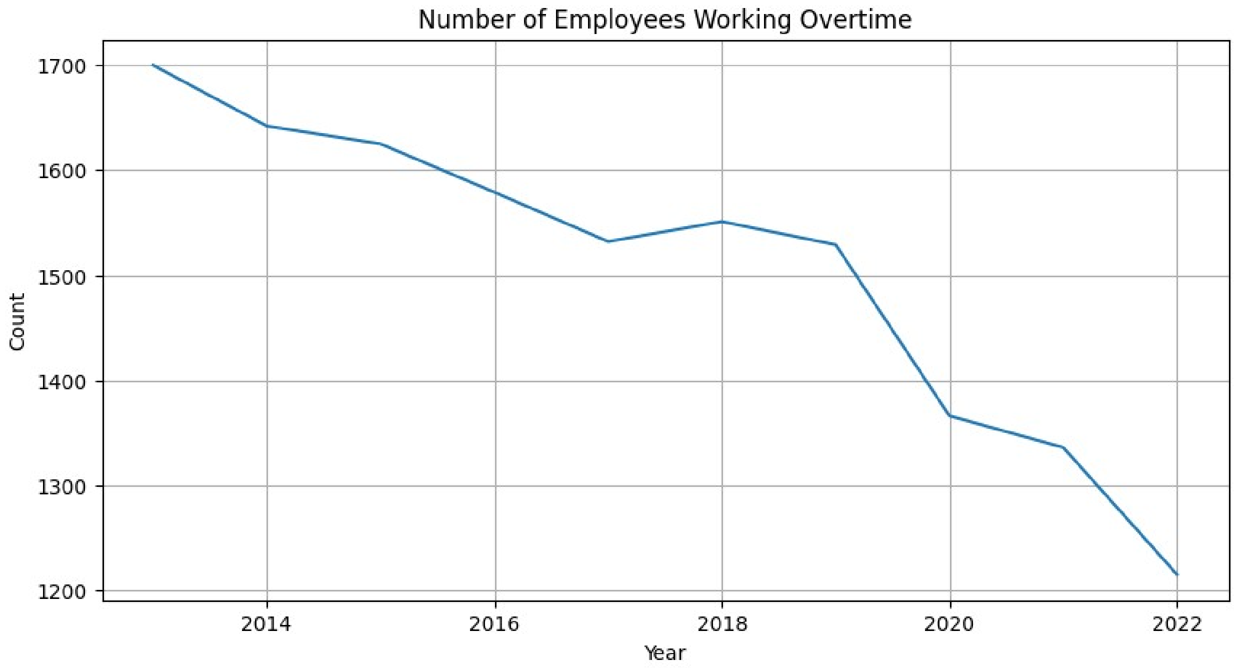


Number of employees worked overtime over years

```
overtime_data_list[0].head(5)

   JOB_NO  EMPLOYEE_ID           EMPLOYEE  RANK         LOCATION
XSTREET  \
0   11490        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV
NaN
1   11528        12011  BAUSEMER,DANIEL P     9  COMMONWEALTH AV
NaN
2   11528        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV
NaN
3   11500        11165     ARAICA,HENRY A     9        TALBOT AV
NaN
4   11500        86212       STEELE,MEL A     9         RIVER ST
NaN

                  DATE  START_TIME  END_TIME  HOURS_WORKED  HOURS_PAID
TYPE  \
0  2013-11-13 00:00:00           0       530           5.5           8
Z
1  2013-11-15 00:00:00           0       530           5.5           8
Z
2  2013-11-15 00:00:00           0       530           5.5           8
Z
3  2013-11-15 00:00:00         830      1400           5.5           8
Z
4  2013-11-15 00:00:00         830      1430           6.0           8
Z
```

```
   CUSTOMER_NO CUSTOMER    CUST_ADDRESS  CUST_ADDRESS_1
CUST_ADDRESS_3  \
0          1103  VERIZON  649 SUMMER ST.             NaN
NaN
1          1103  VERIZON  649 SUMMER ST.             NaN
NaN
2          1103  VERIZON  649 SUMMER ST.             NaN
NaN
3          1103  VERIZON  649 SUMMER ST.             NaN
NaN
4          1103  VERIZON  649 SUMMER ST.             NaN
NaN

     CITY STATE    ZIP
0  BOSTON    MA  02210
1  BOSTON    MA  02210
2  BOSTON    MA  02210
3  BOSTON    MA  02210
4  BOSTON    MA  02210
```

```python
num_distinct_emp = []
overtime_year = [i for i in range(2013, 2023)]

for i in range(len(overtime_year)):
  num_distinct_emp.append(len(overtime_data_list[i]
['EMPLOYEE'].unique()))

plt.figure(figsize=(10, 5))  # Set the figure size
plt.plot(overtime_year, num_distinct_emp)  # Create a horizontal bar
chart
plt.ylabel('Count')  # Correct the label for the y-axis
plt.xlabel('Year')  # Correct the label for the x-axis
plt.title('Number of Employees Working Overtime')  # Add a title
plt.grid(True)  # Show a grid
plt.show()  # Display the plot
```



Number of distinct job types, number of distinct job locations, race distributions

```
bpd_personnel.head(5)
```

```
  Job Data with Academy Date    2149   Unnamed: 2           Unnamed: 3
\
0                      LN,FN      ID  Empl Record             Eff Date

1           Santry,Patrick B  002277            0  2019-07-06 00:00:00

2           Santry,Michael S  006987            0  2019-07-06 00:00:00

3           Guilford,Richard  007442            0  2019-07-06 00:00:00

4           Ajemian,Gerald F  007546            0  2019-07-06 00:00:00


  Unnamed: 4 Unnamed: 5  Unnamed: 6 Unnamed: 7 Unnamed: 8  \
0   Sequence       Last  First Name     Middle     Prefix
1          0     Santry     Patrick          B        NaN
2          0     Santry     Michael          S        NaN
3          0   Guilford     Richard        NaN        NaN
4          0    Ajemian      Gerald          F        NaN

            Unnamed: 9  ... Unnamed: 41 Unnamed: 42 Unnamed: 43
Unnamed: 44  \
0           Start Date  ...    Chng Amt         Pct   Annual Rt
Monthly Rt
```

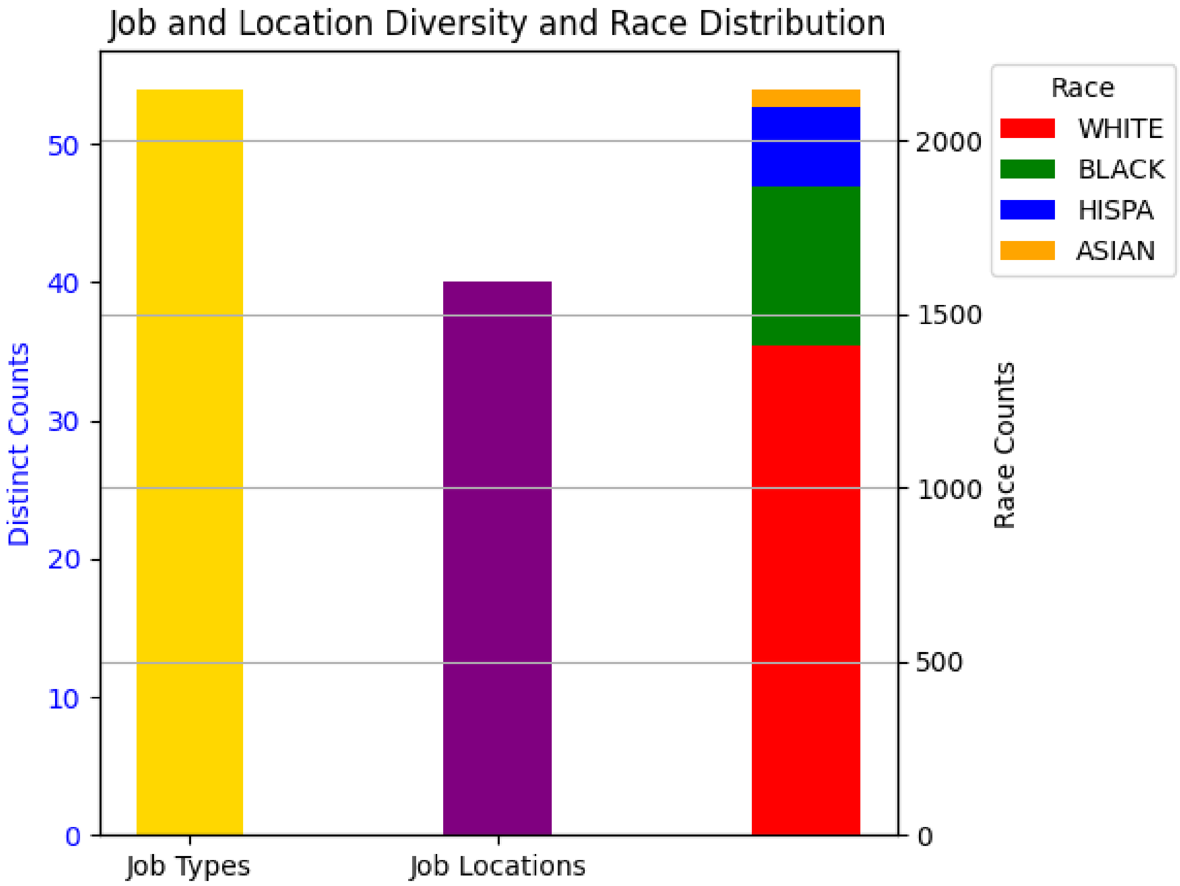
```
1  1975-02-19 00:00:00  ...   40.184572          2  106569.556
8880.796
2  1976-07-14 00:00:00  ...   47.195206      2.629   95790.874
7982.573
3  1978-01-12 00:00:00  ...   47.195206      2.893   87270.674
7272.556
4  1989-11-27 00:00:00  ...   47.195206      2.893   87270.674
7272.556

  Unnamed: 45                   Unnamed: 46          Unnamed: 47
Unnamed: 48  \
0   Hrly Rate                     Job Title                As Of
TskProfID
1   51.235363              Police Sergeant  2020-09-03 00:00:00
TSKPP36131
2   46.053305  Police Offc/Auto Invest 4$10  2020-09-03 00:00:00
TSKPP45130
3   41.957055                Police Officer  2020-09-03 00:00:00
TSKPP36131
4   41.957055                Police Officer  2020-09-03 00:00:00
TSKPP36131

                    Unnamed: 49 Unnamed: 50
0            Task Profile Descr  Ethnic Grp
1  Medically Incapacitated Unit       WHITE
2                   District 13       WHITE
3  Medically Incapacitated Unit       BLACK
4  Medically Incapacitated Unit       WHITE

[5 rows x 51 columns]
```

```
bpd_personnel_copy = bpd_personnel.drop(bpd_personnel.index[0])

num_diff_job =
len(bpd_personnel_copy[bpd_personnel_copy.columns[19]].unique())
num_diff_loc =
len(bpd_personnel_copy[bpd_personnel_copy.columns[25]].unique())
num_diff_ethnic = bpd_personnel_copy[bpd_personnel_copy.columns[-
1]].value_counts()
print("number of different jobs: ", num_diff_job)
print("number of different locations: " ,num_diff_loc)
print("race and number: ", num_diff_ethnic)
```

```
number of different jobs:  54
number of different locations:  40
race and number:  WHITE    1409
BLACK     459
HISPA     230
ASIAN      51
Name: Unnamed: 50, dtype: int64
```

```
race_distribution = {
    'WHITE': 1409,
    'BLACK': 459,
    'HISPA': 230,
    'ASIAN': 51
}

# Create a figure and a set of subplots
fig, ax1 = plt.subplots()

# Set the bar width
bar_width = 0.35

# Set positions of the bars
bar_positions = [1, 2]  # Adjust positions as needed

# Bar values for the left y-axis
bar_values = [num_diff_job, num_diff_loc]

# Plot the first two bars
ax1.bar(bar_positions, bar_values, bar_width, color=['gold',
'purple'])

# Labeling the left y-axis
ax1.set_ylabel('Distinct Counts', color='b')
ax1.set_xticks(bar_positions + [0.3])  # Adjusting position for the
tick
ax1.set_xticklabels(['Job Types', 'Job Locations', 'Race'])
ax1.tick_params(axis='y', labelcolor='b')

# Create the right y-axis for the race values
ax2 = ax1.twinx()

# Each part of the bar should represent a different race
bottom = 0
colors = ['red', 'green', 'blue', 'orange']
race_bar_position = 3  # Adjust position as needed

for race, count in race_distribution.items():
    ax2.bar(race_bar_position, count, bar_width, bottom=bottom,
color=colors.pop(0), label=race)
    bottom += count

# Labeling the right y-axis
ax2.set_ylabel('Race Counts', color='k')
ax2.tick_params(axis='y', labelcolor='k')

# Add a legend and title
ax2.legend(title='Race', bbox_to_anchor=(1.1, 1), loc='upper left')
plt.title('Job and Location Diversity and Race Distribution')
```

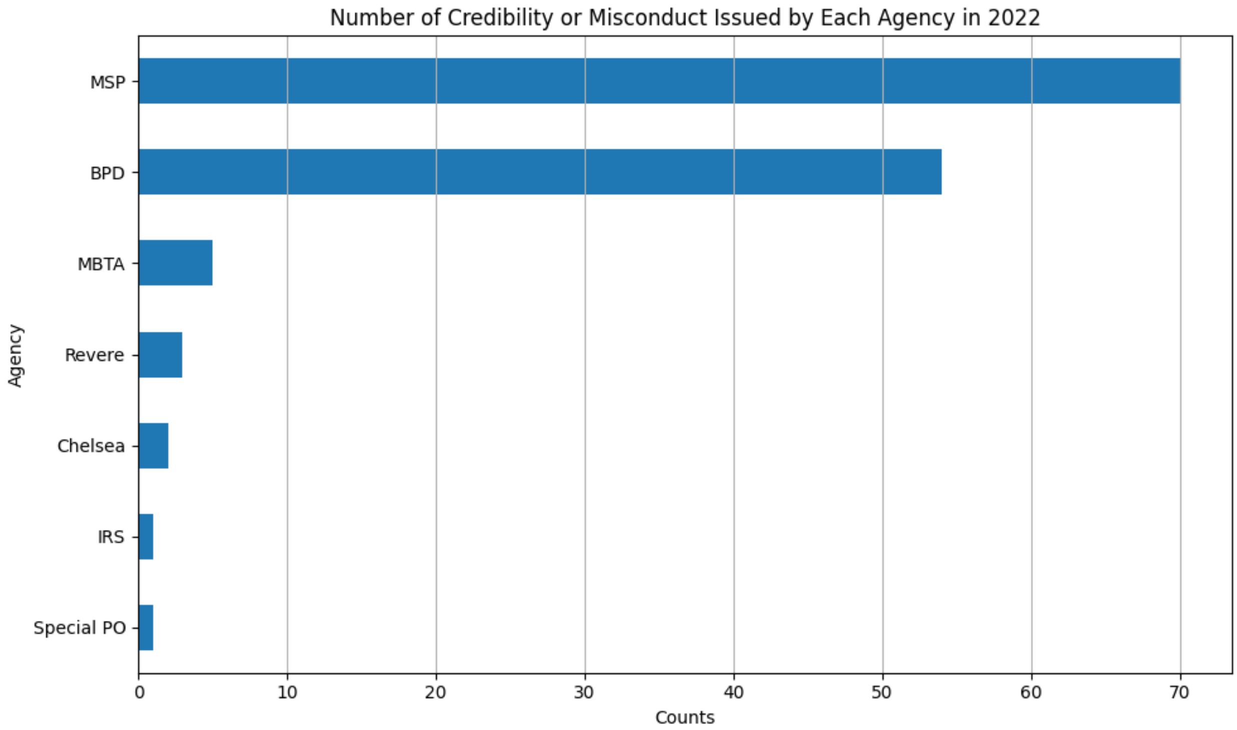
```
# Show the plot with a tight layout
plt.grid(True, axis='y')
plt.tight_layout()
plt.show()
```



Number of Credibility or Misconduct Issued by Each Agency in 2022

```
agency_dict_num = suffolk_brady_2020['AGENCY'].value_counts()
print(agency_dict_num)

MSP              70
BPD              54
MBTA              5
Revere            3
Chelsea           2
IRS               1
Special PO        1
Name: AGENCY, dtype: int64
```

```
# Create a bar chart
plt.figure(figsize=(10, 6))  # Set the figure size
agency_dict_num.plot(kind='barh')  # Plot a bar chart

# Invert the y-axis to have the highest count at the top
plt.gca().invert_yaxis()

# Set the labels and title
plt.xlabel('Counts')
plt.ylabel('Agency')
plt.title('Number of Credibility or Misconduct Issued by Each Agency
in 2022')

# Show grid lines for the x-axis
plt.grid(axis='x')

# Show the plot with a tight layout to ensure everything fits
plt.tight_layout()
plt.show()
```



```
agency_dict_num.item

<bound method IndexOpsMixin.item of MSP           70
BPD           54
MBTA           5
Revere         3
Chelsea        2
```

```
IRS              1
Special PO       1
Name: AGENCY, dtype: int64>
```

```
suffolk_brady_2020.head(8)
```

```
                 NAME           DATE ADDED   AGENCY  \
0        ADAMS, John  2020-09-25 00:00:00      MSP
1      AMARO, Carlos  2014-05-23 00:00:00   Revere
2  ANDERSON, Susan J.  2020-09-25 00:00:00      MSP
3     ANDRADE, David  2020-09-25 00:00:00      MSP
4        ARONE, JOHN  2020-09-25 00:00:00      MSP
5      ATKINS, James  2020-09-25 00:00:00  Chelsea
6      AUGUSTA, Mark  2020-09-25 00:00:00      MSP
7   BARTLETT, Dorston  2020-09-25 00:00:00      BPD

                        STATUS  \
0                  Disciplined
1  Resigned (on previous LEAD)
2                       Public
3                     Indicted
4                       Public
5                   Conviction
6                  Disciplined
7                     Indicted

                    INFORMATION REGARDING LEAD ENTRY
0  Time & attendance/overtime investigation. Bost...
1  Larceny: Theft during execution of search warr...
2  Norfolk County District Attorney Brady/Giglio ...
3  Larceny, public employee standards of conduct ...
4  Middlesex County District Attorney Brady/Gigli...
5          Larceny. SCDAO investigation/prosecution.
6  Time & attendance/overtime investigation. Bost...
7  ABDW, False Police Report. Retired. SCDAO inve...
```

IV. Base Question Answers

1. Identifying instances of financial excess in BPD spending

Statistics Analysis

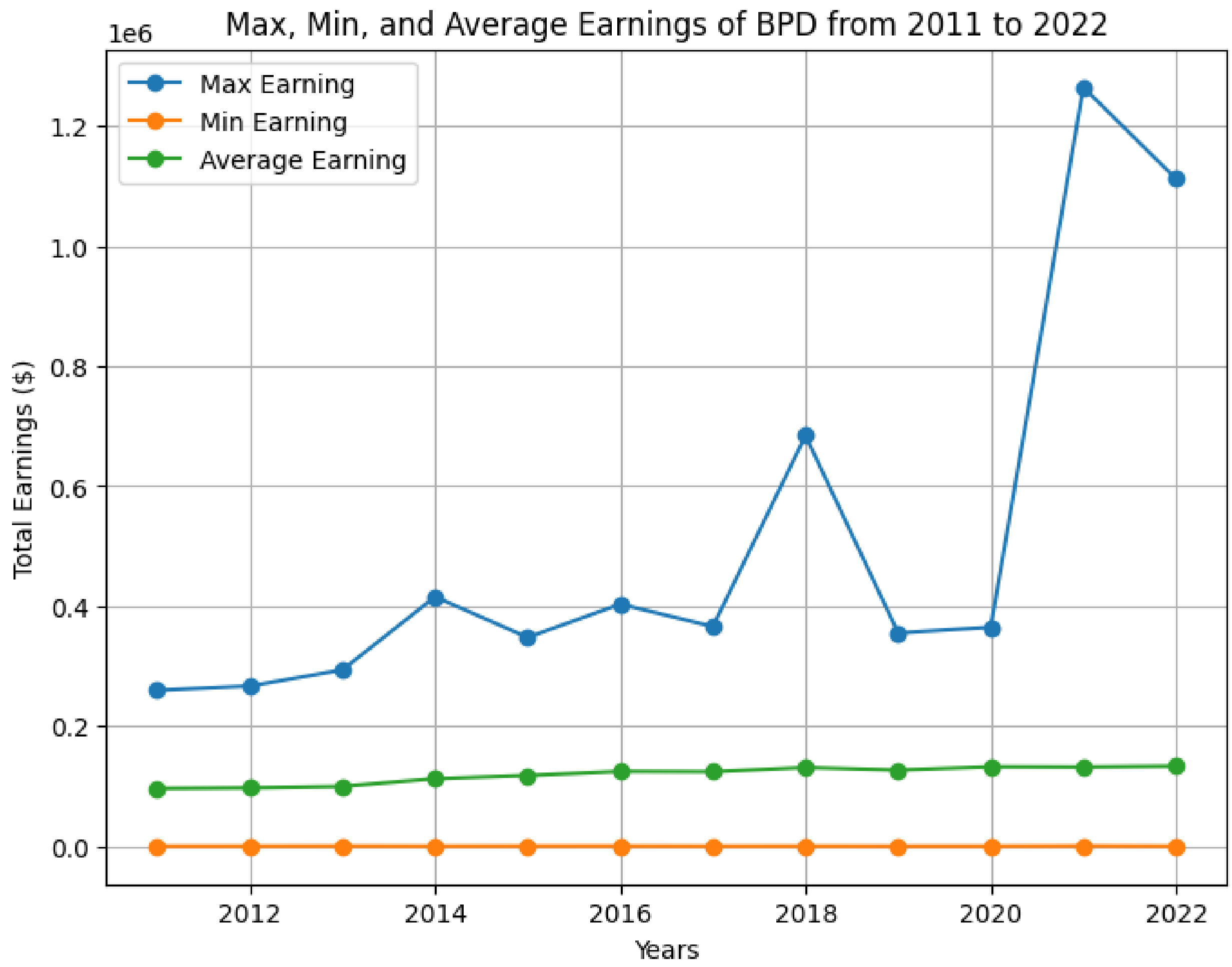
Goal:

- Determine average total earning of a police officer
- How average total earning of officers changed over years from 2011-2022
- Analyze police total earning statistics in the year of 2011 (beginning of dataset) and 2022 (end of dataset)

```
# Calculate max, min and average total earning from 2011-2022
year = [x for x in range(2011, 2023)]
max_earn_by_years = []
min_earn_by_years = []
avg_earn_by_years = []

for data in earning_data_list:
  max_earn_by_years += [data['TOTAL_EARNING'].max()]
  min_earn_by_years += [data['TOTAL_EARNING'].min()]
  avg_earn_by_years += [data['TOTAL_EARNING'].mean()]

# create a panda dataframe for the statistics
stats_data = {
    'Max Earning' : max_earn_by_years,
    'Min Earning' : min_earn_by_years,
    'Average Earning' : avg_earn_by_years
}
stats_df = pd.DataFrame(stats_data, index=year)
print("Table showing Max, Min and Average Total Earning of an officer 
over years.")
print(stats_df)
print("We can see that there is a great difference in max and min 
total earning of police officers.")
```

```
Table showing Max, Min and Average Total Earning of an officer over 
years.
      Max Earning  Min Earning  Average Earning
2011    259914.04        11.70     96421.474132
2012    266971.82        58.52     97515.361269
2013    293892.24       187.69     99771.862159
2014    415709.53         9.36    112589.650642
2015    348096.80       223.02    118041.488626
2016    403408.61       238.85    124787.164775
2017    366232.65         3.50    124254.563280
2018    684410.90       105.90    131321.462320
2019    355538.70         2.50    127094.346316
2020    365001.16        25.00    132487.610436
2021   1264843.63       400.00    132114.566694
2022   1112348.25        23.68    133494.427569
We can see that there is a great difference in max and min total 
earning of police officers.
```

```
# create a line graph illustrating the total earning statistics over 
years
plt.figure(figsize=(8, 6))
plt.plot(year, max_earn_by_years, label='Max Earning', marker='o')
plt.plot(year, min_earn_by_years, label='Min Earning', marker='o')
plt.plot(year, avg_earn_by_years, label='Average Earning', marker='o')

# Add labels and title
plt.xlabel('Years')
plt.ylabel('Total Earnings ($)')
plt.title('Max, Min, and Average Earnings of BPD from 2011 to 2022')

# Add legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



- From the graph above, we can see that the average earnings per officer grew gradually over the years.

- However, the max total earnings per officer experienced a dramatic increase from 2020 to 2022.

- Notice an abnormal increase in police earnings between 2020-2022, we looked into it and found out that the officer was actually awarded $2 million in a gender discrimination lawsuit by the Federal Jury.

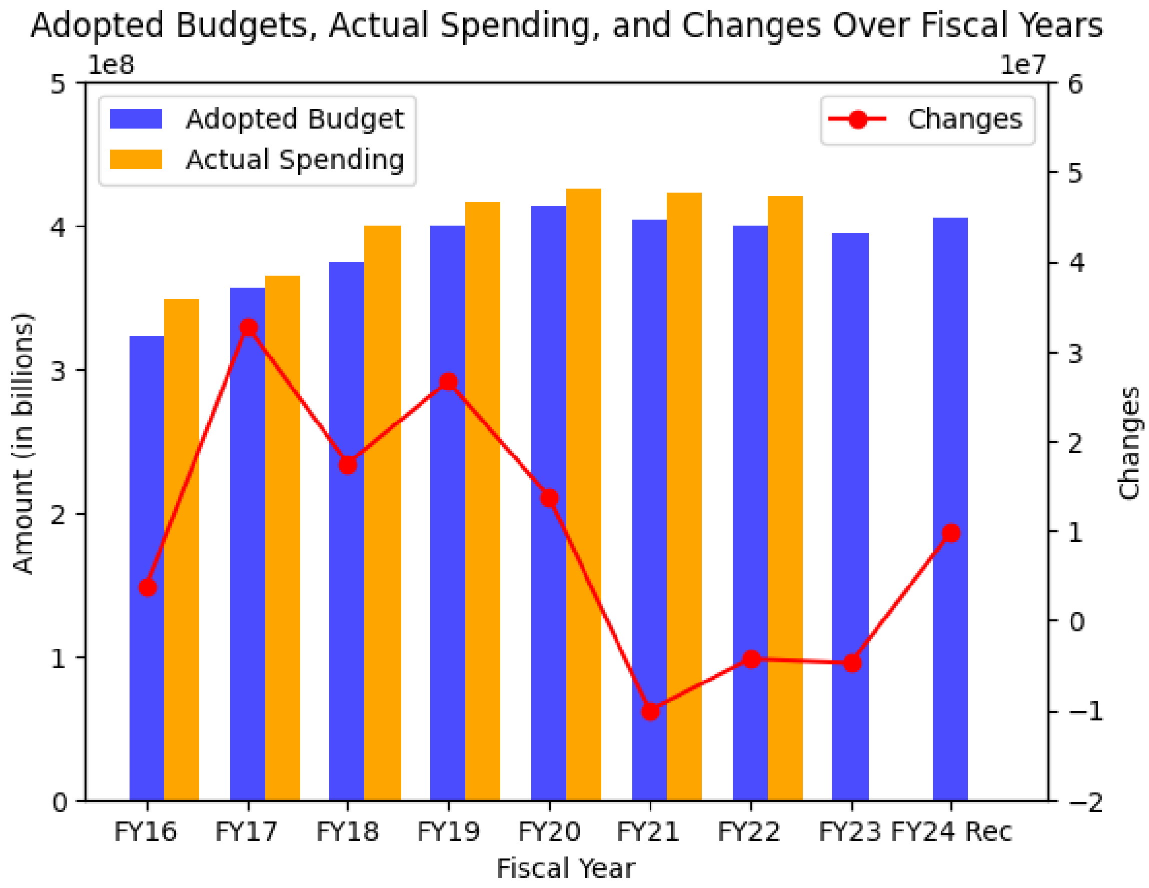
- Jury Awards Millions to BPD

- More info on the case

How have BPD budged changed year-over-year?

- The data is obtained from https://data.aclum.org/2023/05/05/analyzing-fy24-boston-police-department-budget-recommendation/

```
import matplotlib.pyplot as plt
import numpy as np

# Data
years = ['FY16', 'FY17', 'FY18', 'FY19', 'FY20', 'FY21', 'FY22',
'FY23', 'FY24 Rec']
adopted_budgets = [323509388, 356341193, 373814105, 400425675,
414237376, 404182025, 399871217, 395094796, 404973192]
actual_spending = [348945220, 364594820, 399924493, 416762368,
425553508, 422917498, 420411579, 0, 0]
changes = [3809307, 32831805, 17472912, 26611570, 13811701, -10055351,
-4310808, -4776421, 9878396]

# Create a figure and axis
fig, ax1 = plt.subplots()

# Plotting the adopted budgets and actual spending as side-by-side
bars
bar_width = 0.35
index = np.arange(len(years))
bar1 = ax1.bar(index, adopted_budgets, bar_width, label='Adopted
Budget', color='b', alpha=0.7)
bar2 = ax1.bar(index + bar_width, actual_spending, bar_width,
label='Actual Spending', color='orange')

# Creating the line graph for changes
ax2 = ax1.twinx()
line = ax2.plot(years, changes, label='Changes', color='red',
marker='o')

# Adding labels and title
ax1.set_xlabel('Fiscal Year')
ax1.set_ylabel('Amount (in billions)')
ax2.set_ylabel('Changes')
plt.title('Adopted Budgets, Actual Spending, and Changes Over Fiscal
Years')

# Adding legend
bars = [bar1, bar2]
labels = [bar.get_label() for bar in bars]
lines = line
labels += [line[0].get_label()]

ax1.legend(bars, labels, loc='upper left')
ax2.legend(lines, [line[0].get_label()], loc='upper right')

# Set y-axis limit
ax1.set_ylim(0, 500000000)
ax2.set_ylim(-20000000, 60000000)

# Display the plot
plt.show()
```



Observations:

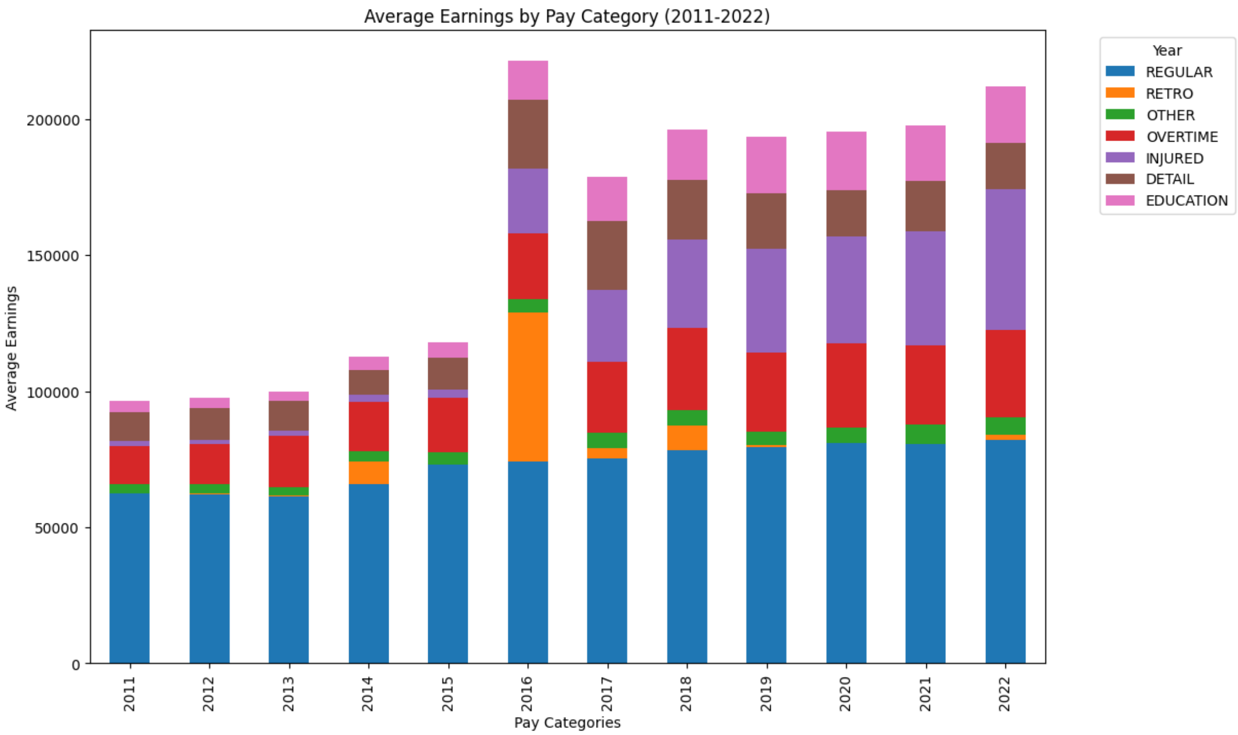
- The Boston Police Department's (BPD) total adopted budget has generally increased over the fiscal years from FY16 to FY24 Rec.

- FY17 saw a substantial increase of over 32 million in the budget compared to the previous fiscal year (FY16).
- FY21 experienced a notable decrease of over 10 million from the previous fiscal year (FY20), representing a budget reduction.
- The overall trend indicates some variability in the budget, with both increases and decreases occurring in different fiscal years

How have BPD paychecks changed year-over-year?

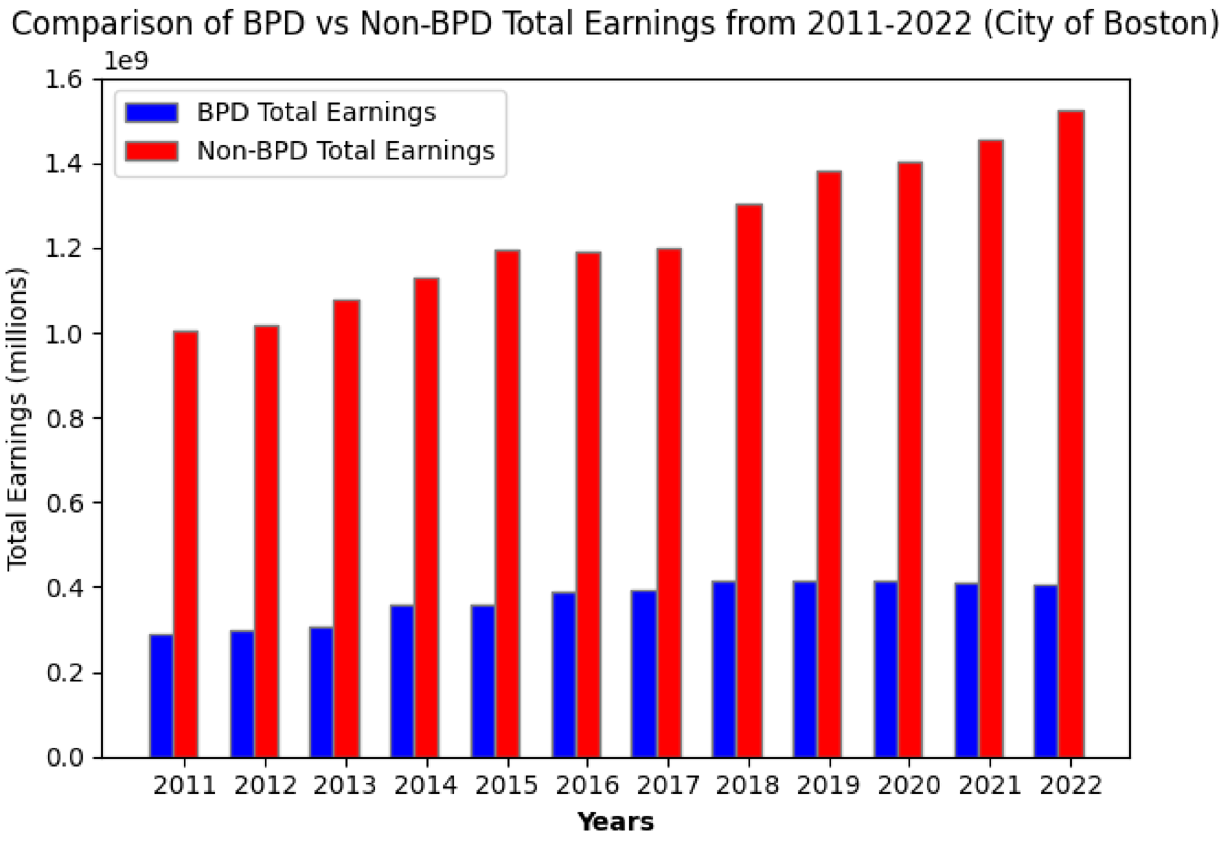
- Both the average amount, as compared with non-BPD Boston city employees, and the breakdown (regular pay v. overtime pay, etc.)?

```
# Categories to aggregate
categories = ['REGULAR', 'RETRO', 'OTHER', 'OVERTIME', 'INJURED',
'DETAIL', 'EDUCATION','TOTAL_EARNING']
year = [x for x in range(2011,2023)]
# Dictionary to store the average data for each category by year
avg_by_category = {category: [] for category in categories}

# Looping through each year's data
for data in earning_data_list:
    for category in categories:
        avg_by_category[category].append(data[category].mean())

# Creating a table using pandas
average_spending_df = pd.DataFrame(avg_by_category, index=year)
# print("Table showing average spending in each category over the
years.")
# print(average_spending_df)

# drop the TOTAL_EARNING column
stacked_df = average_spending_df.drop(['TOTAL_EARNING'], axis = 1)

# Plotting a stacked bar chart
stacked_df.plot(kind='bar', stacked=True, figsize=(12, 8))

# Adding labels and title
plt.xlabel('Pay Categories')
plt.ylabel('Average Earnings')
plt.title('Average Earnings by Pay Category (2011-2022)')

# Adding legend
plt.legend(title='Year', bbox_to_anchor=(1.05, 1), loc='upper left')

# Display the plot
plt.show()
```



Total Earnings Comparison

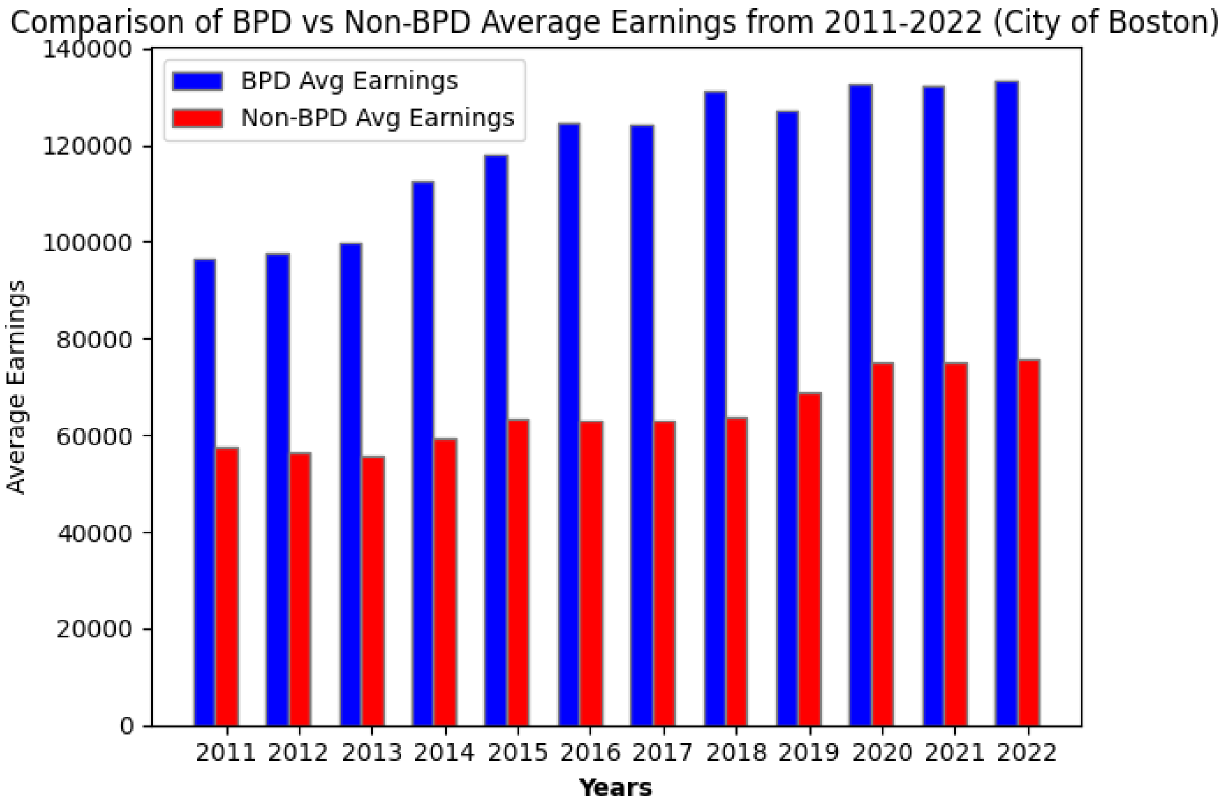
```
# Compute total earnings from 2011-2022 for BPD and non-BPD
bpd_total_earning = []
non_bpd_total_earning = []

for bpd_data, non_bpd_data in zip(earning_data_list,
earning_data_list_nonpd):
    # Convert the 'TOTAL_EARNING' columns to numeric, coercing any
errors
    bpd_data['TOTAL_EARNING'] =
pd.to_numeric(bpd_data['TOTAL_EARNING'], errors='coerce')
    non_bpd_data['TOTAL_EARNING'] =
pd.to_numeric(non_bpd_data['TOTAL_EARNING'], errors='coerce')

    # Append the sums to the respective lists
    bpd_total_earning.append(bpd_data['TOTAL_EARNING'].sum())
    non_bpd_total_earning.append(non_bpd_data['TOTAL_EARNING'].sum())

# Set the position and width for the bars
barWidth = 0.3
r1 = range(len(bpd_total_earning))
r2 = [x + barWidth for x in r1]

# Plot
plt.bar(r1, bpd_total_earning, width=barWidth, color='blue',
edgecolor='grey', label='BPD Total Earnings')
plt.bar(r2, non_bpd_total_earning, width=barWidth, color='red',
edgecolor='grey', label='Non-BPD Total Earnings')

year = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020,
2021, 2022]

# Add labels, title and legend
plt.xlabel('Years', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(bpd_total_earning))],
year)
plt.ylabel('Total Earnings (millions)')
plt.title('Comparison of BPD vs Non-BPD Total Earnings from 2011-2022
(City of Boston)')
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



Average Earnings Comparison

```
# Compute avearge earnings from 2011-2022 for BPD and non-BPD
bpd_average_earning = []
non_bpd_average_earning = []
```

```
for bpd_data, non_bpd_data in zip(earning_data_list,
earning_data_list_nonpd):
    # Convert the 'TOTAL_EARNING' columns to numeric, coercing any
errors
    bpd_data['TOTAL_EARNING'] =
pd.to_numeric(bpd_data['TOTAL_EARNING'], errors='coerce')
    non_bpd_data['TOTAL_EARNING'] =
pd.to_numeric(non_bpd_data['TOTAL_EARNING'], errors='coerce')

    # Append the sums to the respective lists
    bpd_average_earning.append(bpd_data['TOTAL_EARNING'].mean())

non_bpd_average_earning.append(non_bpd_data['TOTAL_EARNING'].mean())

# Set the position and width for the bars
barWidth = 0.3
r1 = range(len(bpd_average_earning))
r2 = [x + barWidth for x in r1]

# Plot
plt.bar(r1, bpd_average_earning, width=barWidth, color='blue',
edgecolor='grey', label='BPD Avg Earnings')
plt.bar(r2, non_bpd_average_earning, width=barWidth, color='red',
edgecolor='grey', label='Non-BPD Avg Earnings')

year = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020,
2021, 2022]

# Add labels, title and legend
plt.xlabel('Years', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(bpd_average_earning))],
year)
plt.ylabel('Average Earnings')
plt.title('Comparison of BPD vs Non-BPD Average Earnings from 2011-
2022 (City of Boston)')
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
```



Assumptions:

- Average of “Total Earning” can be used as a measurement for paychecks.
- Non-BPD population includes all jobs in the City of Boston (i.e:cashiers, teachers, etc.).

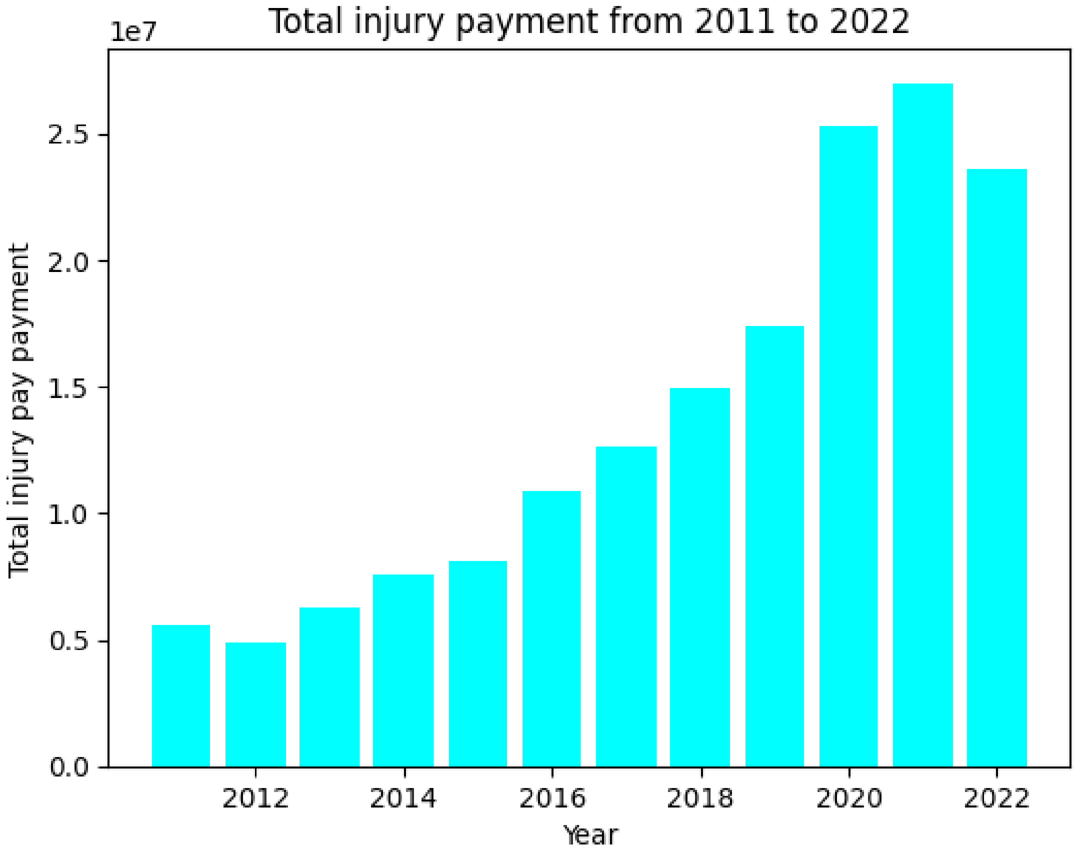
Observations:

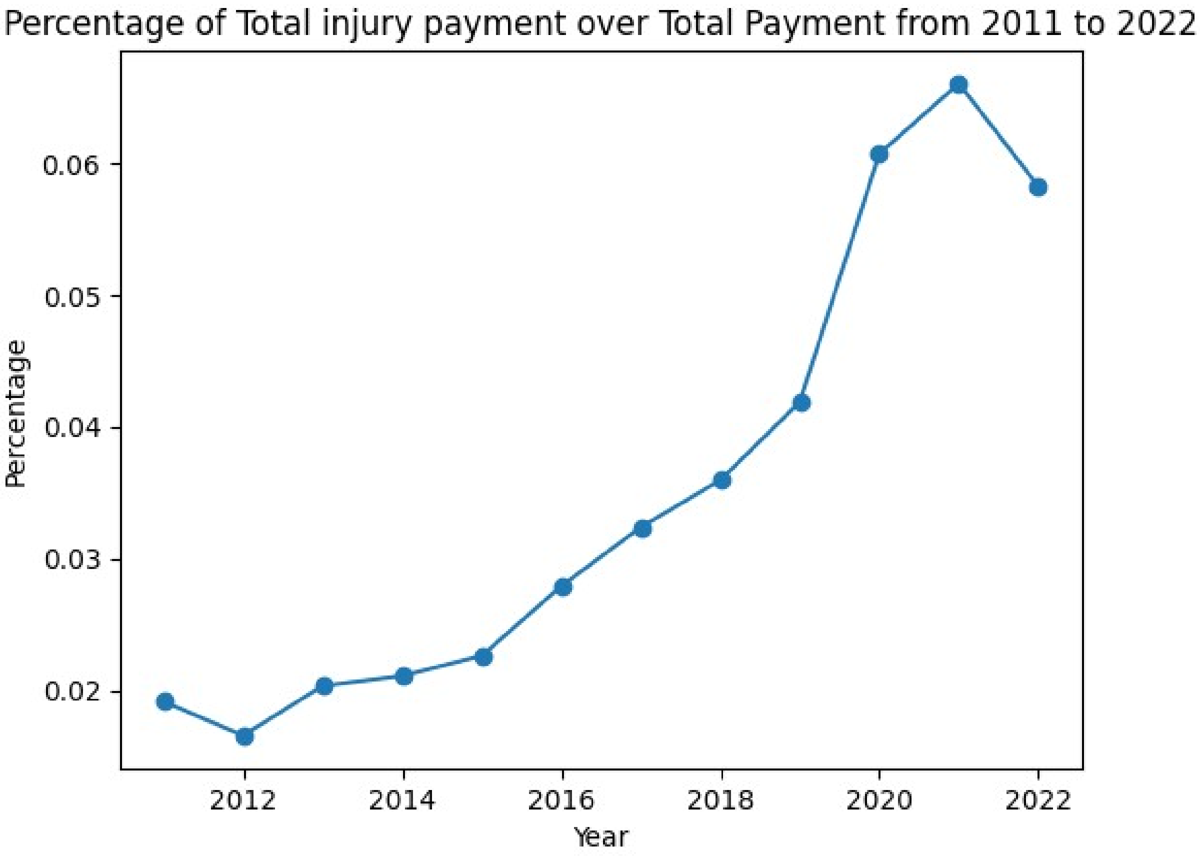
- Earnings for Boston Police Department (BPD) employees have been going up over time.
- Average salary for non-BPD city workers is approximately half that of BPD employees.
- Notice an abnormal increase in police earnings between 2020-2022, we looked into it and found out that the officer was actually awarded $2 million in a gender discrimination lawsuit by the Federal Jury.

How much BPD officer pay came from injury pay?

##What percentage of officers took injury pay in a given year? Can Wang

```
# total injury payment from 2011-2022
injury_pay = []
# total overall payment total from 2011-2022
overall_pay = []
# injury_pay/overall_pay
injury_payratio = []

for data in earning_data_list:
    injury_pay += [data['INJURED'].sum()]
    overall_pay += [data['TOTAL_EARNING'].sum()]
    injury_payratio += [data['INJURED'].sum() / 
data['TOTAL_EARNING'].sum()]

# Plot the total injury pay
plt.bar(year, injury_pay, color = 'cyan')

# Add labels and title
plt.xlabel("Year")
plt.ylabel("Total injury pay payment")
plt.title("Total injury payment from 2011 to 2022")

# Show the chart
plt.show()

# Plot the total injury payment ratio
plt.plot(year, injury_payratio, marker='o', linestyle='-')

# Add labels and title
plt.xlabel("Year")
plt.ylabel("Percentage")
plt.title("Percentage of Total injury payment over Total Payment from 
2011 to 2022")

# Show the chart
plt.show()
```

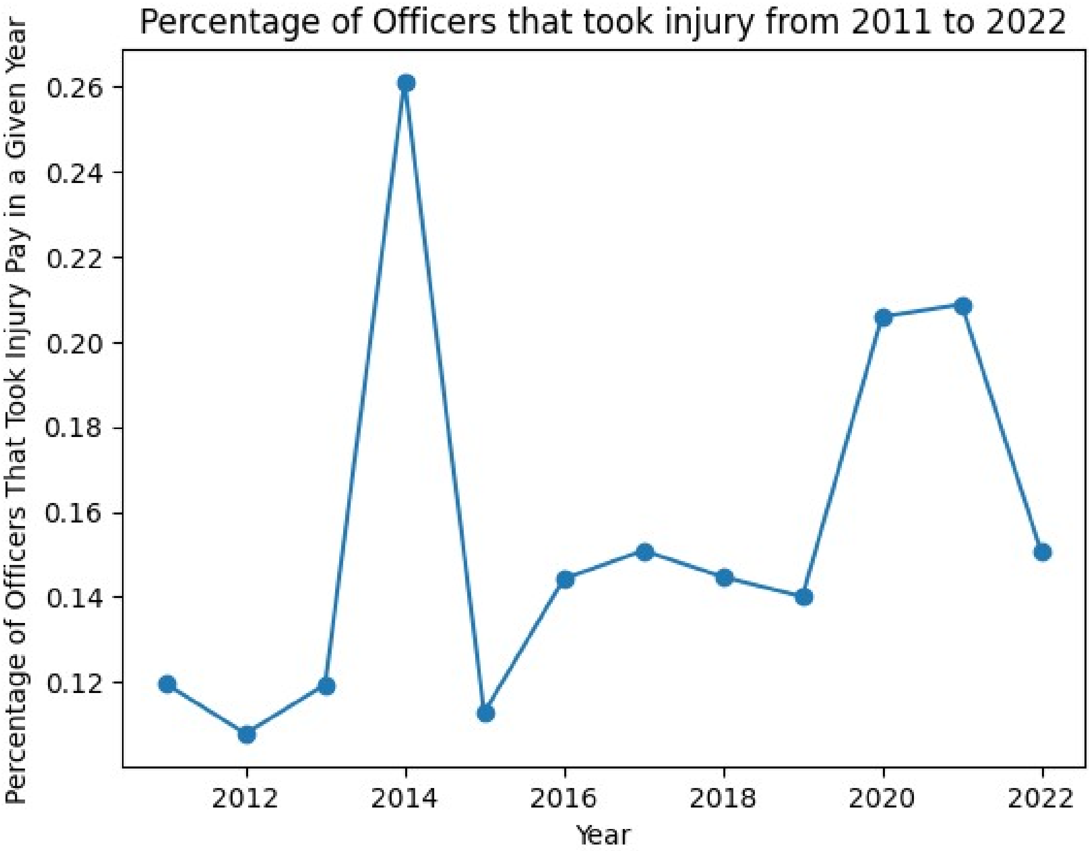

Total injury payment from 2011 to 2022
1e7
2.5
2.0
1.5
1.0
0.5
0.0
Total injury pay payment
2012
2014
2016
2018
2020
2022
Year



```
# for percentage of officers took injury pay in a given year
countnonzeros = []
#keep a track on how many exactly are not injured
nonzeroratio = []
#stores the ratio of injury versus the sample space
for data in earning_data_list:
  nonzerocount = data['INJURED'].fillna(0).ne(0).sum()
  #nonzerocount = (data['INJURED'] != 0 |
np.isnan(data['INJURED'])).sum()
  countnonzeros.append(nonzerocount)
  nonzeroratio.append(nonzerocount / len(data['INJURED']))


print(nonzeroratio)

# Plot the total injury payment ratio
plt.plot(year, nonzeroratio, marker='o', linestyle='-')

# Add labels and title
plt.xlabel("Year")
plt.ylabel("Percentage of Officers That Took Injury Pay in a Given
Year")
plt.title("Percentage of Officers that took injury from 2011 to 2022")
```

```
# Show the chart
plt.show()
```

```
[0.11948224361101892, 0.10777851021753461, 0.1192868719611021,
0.26117054751415986, 0.11272247857613711, 0.14418754014129737,
0.15084153699587172, 0.1446580523164198, 0.1400183430143687,
0.20591979630808402, 0.2087912087912088, 0.15065789473684211]
```



2. Characterizing wasteful BPD overtime practices

How do overtime hours paid compare to overtime hours worked?

##What does the discrepancy financially amount to, year after year?

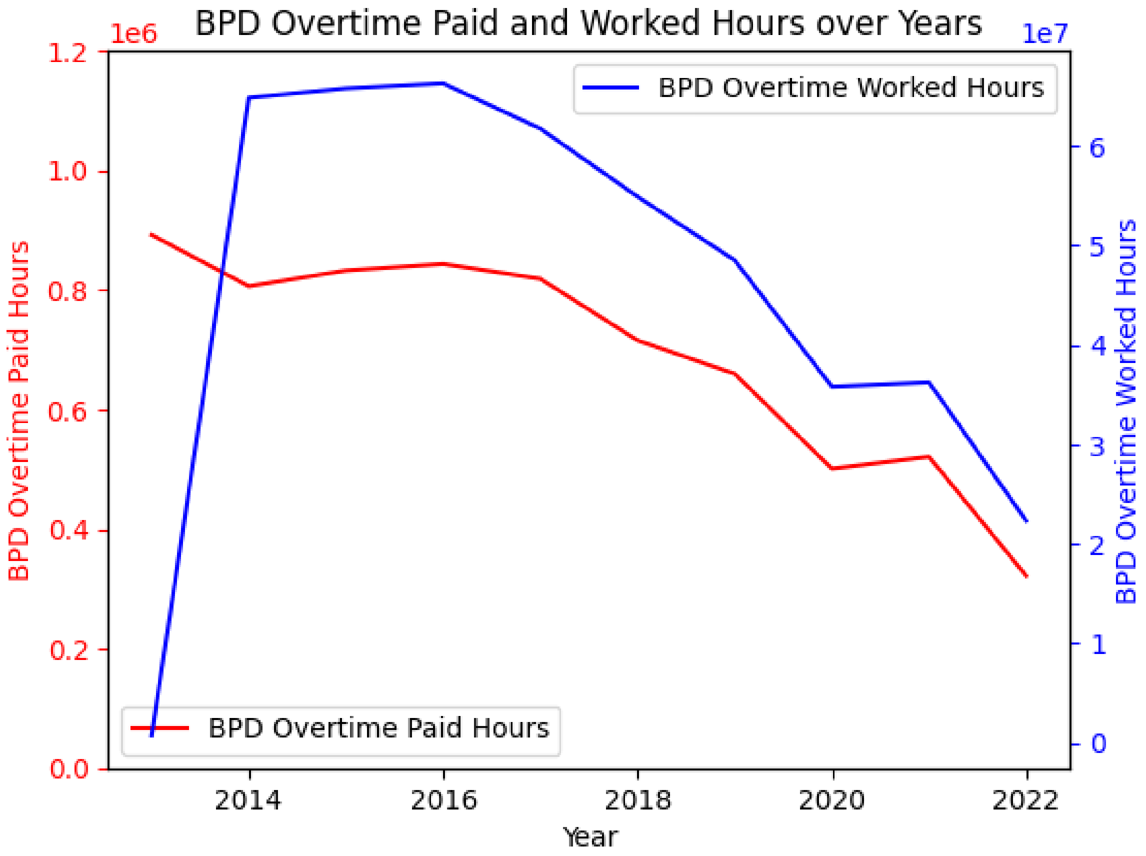
(Riva)

```python
hours_paid = []
hours_worked = []

for i in range(len(overtime_data_list)):
  hours_worked.append(overtime_data_list[i]['HOURS_WORKED'].sum())
  hours_paid.append(overtime_data_list[i]['HOURS_PAID'].sum())

print(hours_worked)
print(hours_paid)
```

```
[732020.02, 64867237, 65802530, 66309207, 61769880, 54925189,
48532187, 35816301, 36249622, 22317355]
[892118, 806287, 832427, 843448, 818716, 715582, 659747, 501070,
520939, 321542]
```

```python
# calculate the overtime hours and work hours from year 2013 - 2022
year = [x for x in range(2013,2023)]

# Create the first axis
fig, ax1 = plt.subplots()
# Plot the staffing data on the left axis
ax1.plot(year, hours_paid, color='red', label='BPD Overtime Paid Hours')
ax1.set_xlabel('Year')
ax1.set_ylabel('BPD Overtime Paid Hours', color='red')
ax1.tick_params('y', colors='red')
ax1.set_ylim([0, 1200000])


# Create the second axis sharing the same x-axis
ax2 = ax1.twinx()
# Plot the second data on the right axis
ax2.plot(year, hours_worked, color='blue', label='BPD Overtime Worked Hours')
ax2.set_ylabel('BPD Overtime Worked Hours', color='blue')
ax2.tick_params('y', colors='blue')

# Display the legend
ax2.legend(loc='upper right')
ax1.legend(loc='lower left')
plt.title("BPD Overtime Paid and Worked Hours over Years")
```

```
Text(0.5, 1.0, 'BPD Overtime Paid and Worked Hours over Years')
```



Observations: Plots above are for the same data, but with different scales. By the plot at the top, we can see that the pattern for the number of hours paid follows the number of hours worked. But if we see from the plot at the bottom, within the same scale for comparison, the number of hours worked is much smaller than the number of hours paid, indicating that there exists a waste of money in overtime expenditure to BPD.

Conclusion: If the BPD department wants to decrease the amount of waste expenditure, they can consider paying overtime money by using actual overtime worked hours as a counter.

Amount of overtime earnings paid per hour

(Riva)

```
# Get EMPLOYEE name and HOURS_PAID from overtime dataset
sum_hours_paid_per_employee = []

for i in range(len(overtime_data_list)):

sum_hours_paid_per_employee.append(overtime_data_list[i].groupby('EMPL
OYEE')['HOURS_PAID'].sum().reset_index())
    sum_hours_paid_per_employee[i]['EMPLOYEE'] =
sum_hours_paid_per_employee[i]['EMPLOYEE'].str.lower()
```

```
sum_hours_paid_per_employee[7].head(10)
```

```
              EMPLOYEE  HOURS_PAID
0        abreu,gabriel         561
1       abreu,moises j        1079
2       ace,richard k.        1188
3    acevedo,rafael w.         236
4   acloque,jean moise          74
5        acosta,jose l         232
6    adams,christopher         308
7  adams,christopher p          58
8       adams,daniel j         992
9        ahern,john b.          88
```

```
# Get EMPLOYEE name and OVERTIME earnings from earnings dataset
sum_overtime_earnings = []

for i in range(len(overtime_data_list)):

sum_overtime_earnings.append(earning_data_list[i+2].groupby(earning_da
ta_list[i+2].columns[0])
[earning_data_list[i+2].columns[6]].sum().reset_index())
    sum_overtime_earnings[i].rename(columns={'NAME': 'EMPLOYEE'},
inplace=True)
    sum_overtime_earnings[i]['EMPLOYEE'] = sum_overtime_earnings[i]
['EMPLOYEE'].str.lower()

sum_overtime_earnings[7].head(10)
```

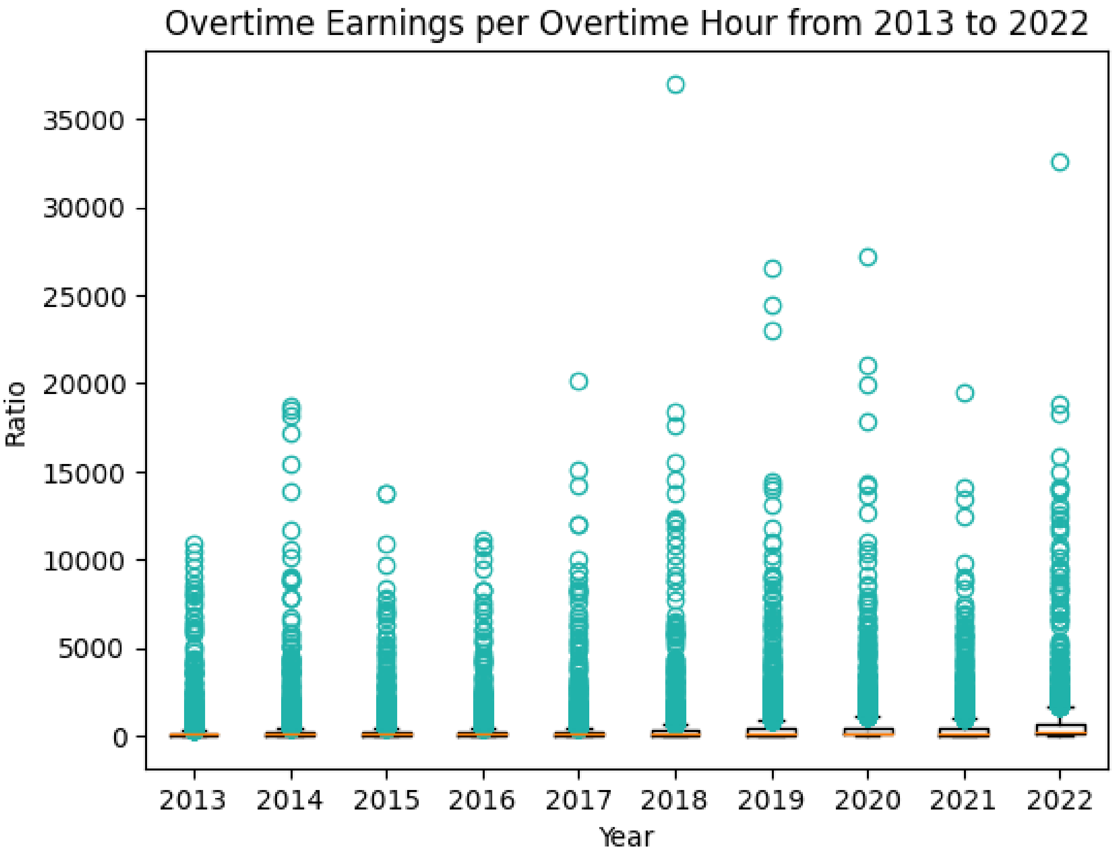
```
                  EMPLOYEE  OVERTIME
0          abasciano,joseph  16595.52
1      abdul-aziz,ramadani      0.00
2                abel,keny      0.00
3  abrahamson,patrick olaf  12940.29
4    abreu,carlos de jesus  15676.01
5              abreu,cesar  43322.09
6            abreu,gabriel  32298.83
7           abreu,moises j  20042.67
8           ace,richard k.   7281.42
9          acevedo,dora luz      0.00
```

```
# Merge two dataset by same EMPLOYEE name
overtime_hours_paid = []

for i in range(len(overtime_data_list)):
    merged_dataset = pd.merge(sum_hours_paid_per_employee[i],
sum_overtime_earnings[i], on='EMPLOYEE', how='inner')
    overtime_hours_paid.append(merged_dataset[['EMPLOYEE',
'HOURS_PAID', 'OVERTIME']])
```

```
overtime_hours_paid[3].head(10)
```

```
                  EMPLOYEE  HOURS_PAID   OVERTIME
0      abdul-aziz,ramadani        1265   25411.32
1  abrahamson,patrick olaf          77    4804.80
2              abreu,cesar         822   50193.20
3           abreu,moises j         859    7710.63
4           ace,richard k.        1447    3886.74
5        acevedo,rafael w.        1132   18296.84
6       acloque,jean moise          49   75193.68
7            acosta,carina          66    7470.55
8            acosta,jose l         466  122251.81
9        adams,christopher           8    1055.86
```

```
# Get overtime earnings / overtime hours paid
overtime_work_counted = []
overtime_paid = []

for i in range(len(overtime_data_list)):
    overtime_work_counted.append(overtime_hours_paid[i]
['HOURS_PAID'].tolist())
    overtime_paid.append(overtime_hours_paid[i]['OVERTIME'].tolist())

ratio = []

for i in range(len(overtime_data_list)):
    ratio_sub = []
    for j in range(len(overtime_paid[i])):
        ratio_sub.append(overtime_paid[i][j]/overtime_work_counted[i]
[j])
    ratio.append(ratio_sub)

import matplotlib.pyplot as plt

# Generate a simple range for the x-axis
x = [i for i in range(2013, 2023)]

# Create a plot
plt.boxplot(ratio, labels=x, flierprops=dict(marker='o',
markeredgecolor='lightseagreen'))

# Add titles and labels
plt.title('Overtime Earnings per Overtime Hour from 2013 to 2022')
plt.xlabel('Year')
plt.ylabel('Ratio')

# Show the plot
plt.show()
```



By years, the overtime earnings per hour stabalized around 0 to 15000. But number of outliers increased as time passes.

```
overtime_data_list[0].head(3)

   JOB_NO  EMPLOYEE_ID           EMPLOYEE  RANK         LOCATION  
XSTREET  \
0   11490        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV  
NaN   
1   11528        12011  BAUSEMER,DANIEL P     9  COMMONWEALTH AV  
NaN   
2   11528        53805   MCCARTHY,DENIS K     9  COMMONWEALTH AV  
NaN   

                  DATE  START_TIME  END_TIME  HOURS_WORKED  HOURS_PAID  
TYPE  \
0  2013-11-13 00:00:00           0       530           5.5           8  
Z   
1  2013-11-15 00:00:00           0       530           5.5           8  
Z   
2  2013-11-15 00:00:00           0       530           5.5           8  
Z   
```

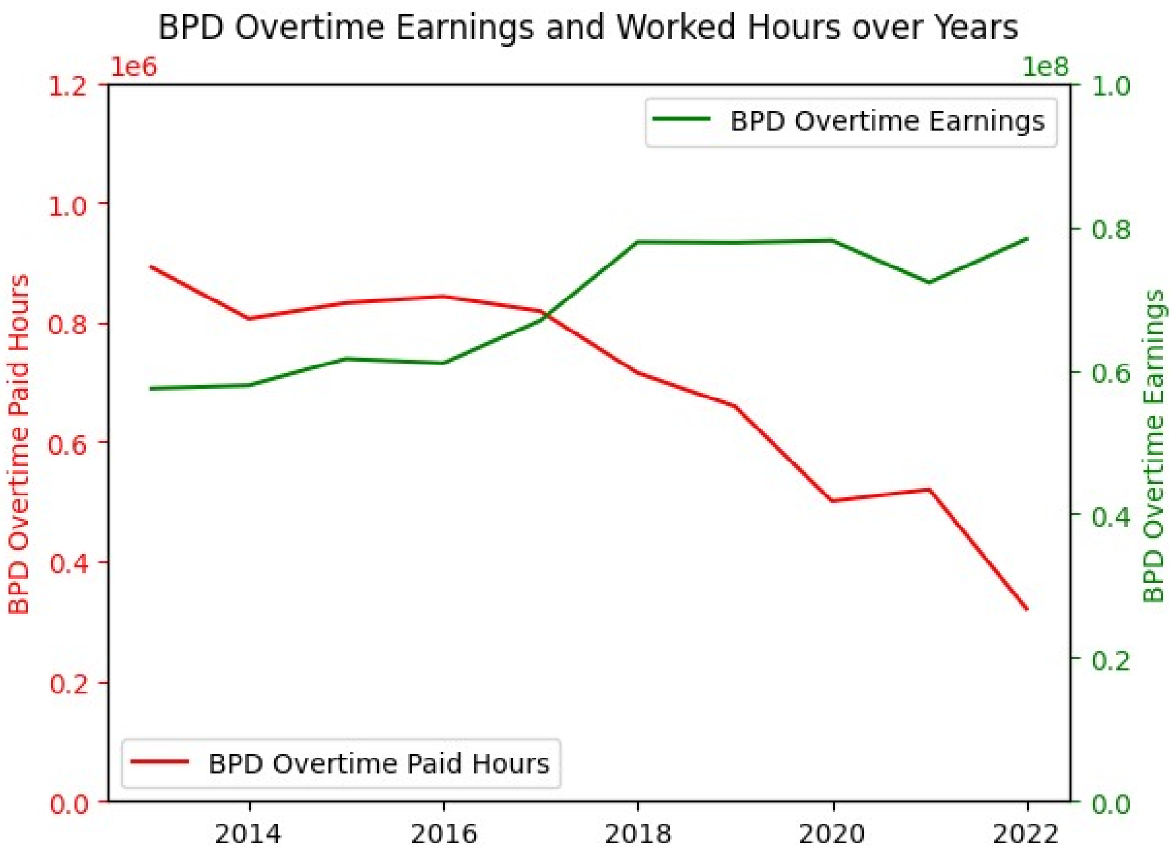
```
   CUSTOMER_NO CUSTOMER    CUST_ADDRESS  CUST_ADDRESS_1
CUST_ADDRESS_3  \
0         1103  VERIZON  649 SUMMER ST.             NaN
NaN
1         1103  VERIZON  649 SUMMER ST.             NaN
NaN
2         1103  VERIZON  649 SUMMER ST.             NaN
NaN

     CITY STATE    ZIP
0  BOSTON    MA  02210
1  BOSTON    MA  02210
2  BOSTON    MA  02210
```

```
overtime_paid_money = []
overtime_paid_hours = []

for i in range(len(overtime_data_list)):
      overtime_paid_money.append(earning_data_list[i+2]
[earning_data_list[i+2].columns[6]].sum())
      overtime_paid_hours.append(overtime_data_list[i]
[overtime_data_list[i].columns[10]].sum())

print(len(overtime_paid_money))
```

```
10
```

```
year = [i for i in range(2013, 2023)]
# Create the first axis
fig, ax1 = plt.subplots()
# Plot the overtime hour  on the left axis
ax1.plot(year, overtime_paid_hours, color='red', label='BPD Overtime
Paid Hours')
ax1.set_ylabel('BPD Overtime Paid Hours', color='red')
ax1.tick_params('y', colors='red')
# Set the y-axis range for overtime_paid_hours
ax1.set_ylim([0, 1200000])


# Create the second axis sharing the same x-axis
ax2 = ax1.twinx()
ax2.plot(year, overtime_paid_money, color='green', label='BPD Overtime
Earnings')
ax2.set_xlabel('Year')
ax2.set_ylabel('BPD Overtime Earnings', color='green')
ax2.tick_params('y', colors='green')
ax2.set_ylim([0, 100000000])

# Display the legend
ax1.legend(loc='lower left')
ax2.legend(loc='upper right')
```

```
plt.title("BPD Overtime Earnings and Worked Hours over Years")

# Show the plot
plt.show()
```



```
print(overtime_paid_money)

[57483767.629999995, 57914605.89, 61608537.989999995,
60998676.760000005, 66933649.86999999, 77855435.97999999,
77764302.50999999, 78057696.23, 72223009.96000001, 78265758.01]
```

Observations: As we can see from the plot, from 2013 to 2022, overtime earnings are increasing yearly. However the number of hours worked is decreasing. This explains the outliers, the huge amount of money paid per overtime hours, in the previous plot.

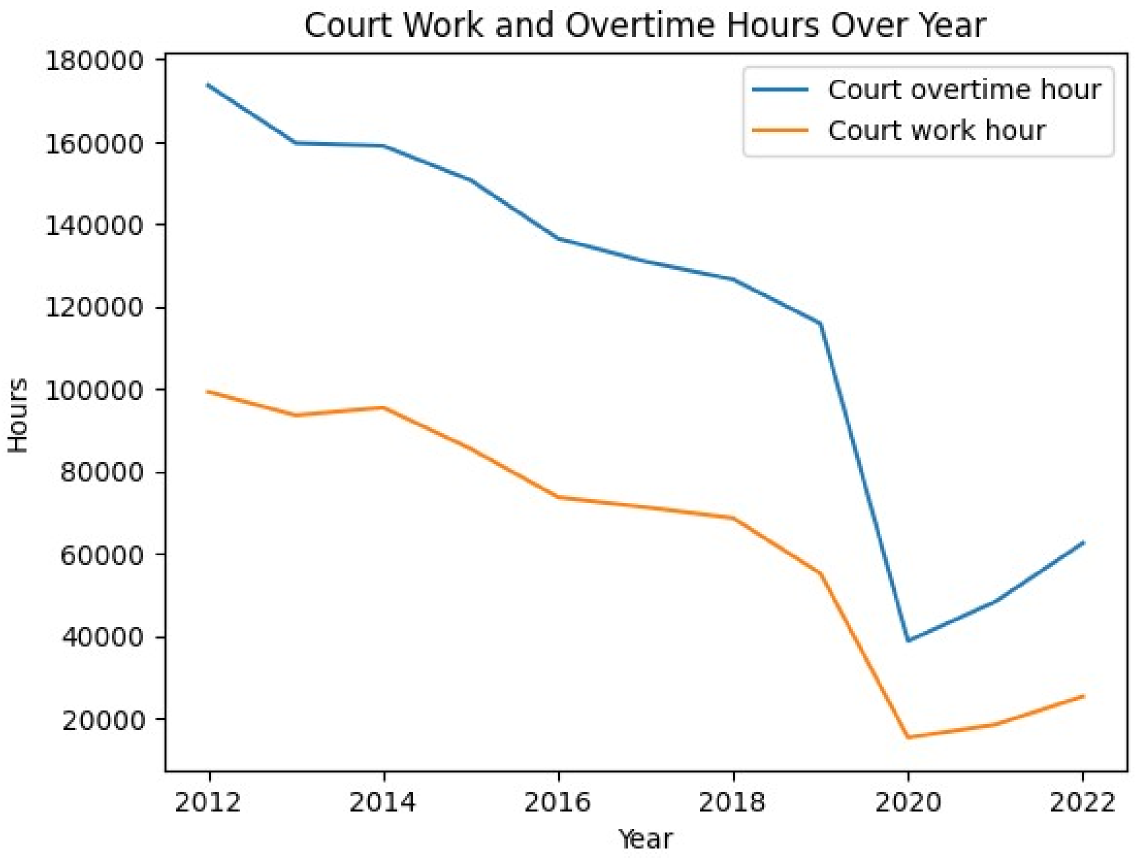
How has overtime for court appearances changed year-over-year?

(Truc Duong)

- Assumptions:

- We used the reported WRKHRS and OTHRS as a measurement for “appearances” in court

```
# calculate the overtime hours and work hours from year 2012 - 2022
overtime_hrs = [df['OTHOURS'].sum() for df in
court_overtime_data_list] #overtime pay
wrk_hrs = [df['WRKDHRS'].sum() for df in court_overtime_data_list] #
court overtime wrk
year = [x for x in range(2012,2023)]
# Create the line chart
plt.plot(year, overtime_hrs, label='Court overtime hour')
plt.plot(year, wrk_hrs, label='Court work hour')

# Add labels and title
plt.xlabel('Year')
plt.ylabel('Hours')
plt.title('Court Work and Overtime Hours Over Year')

# Add legend
plt.legend()

# Show the plot
plt.show()

print(overtime_hrs)
print(wrk_hrs)
```



```
[173592.5, 159650.25, 158954.75, 150605.25, 136450.25, 130883.75,
126520.25, 115869.0, 38814.5, 48371.25, 62557.5]
[99272.25, 93562.25, 95441.0, 85469.5, 73697.5, 71268.0, 68655.75,
55214.75, 15408.75, 18535.25, 25392.0]
```

Observations:

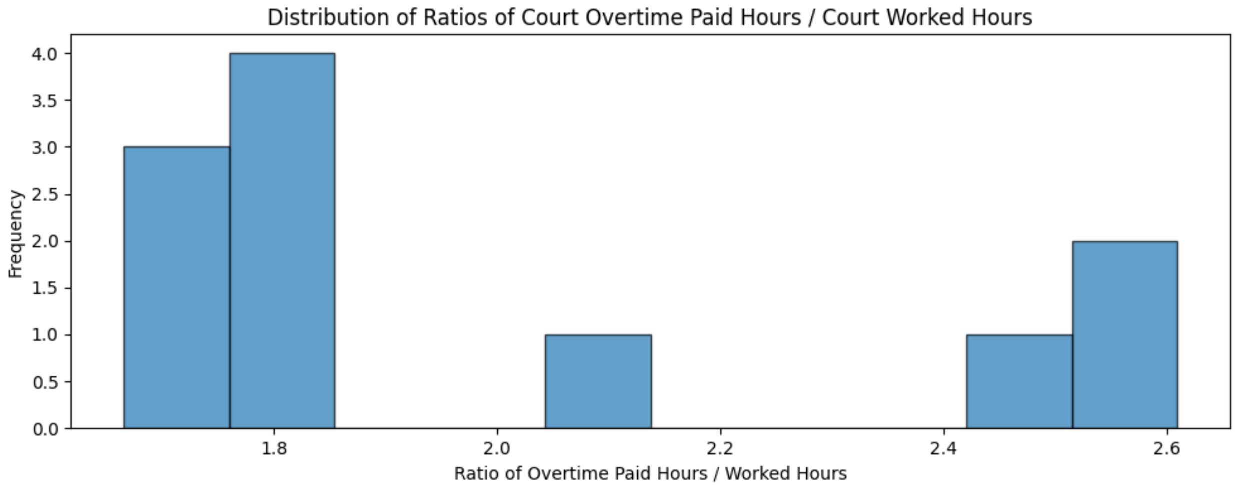
- In general, the total overtime hours consistently appeared to be twice the total work hours.
- The year 2012 recorded the highest reported court overtime and worked hours. Conversely, 2020 witnessed the lowest reported court overtime and worked hours, potentially influenced by the COVID-19 pandemic and a surge in remote jobs.

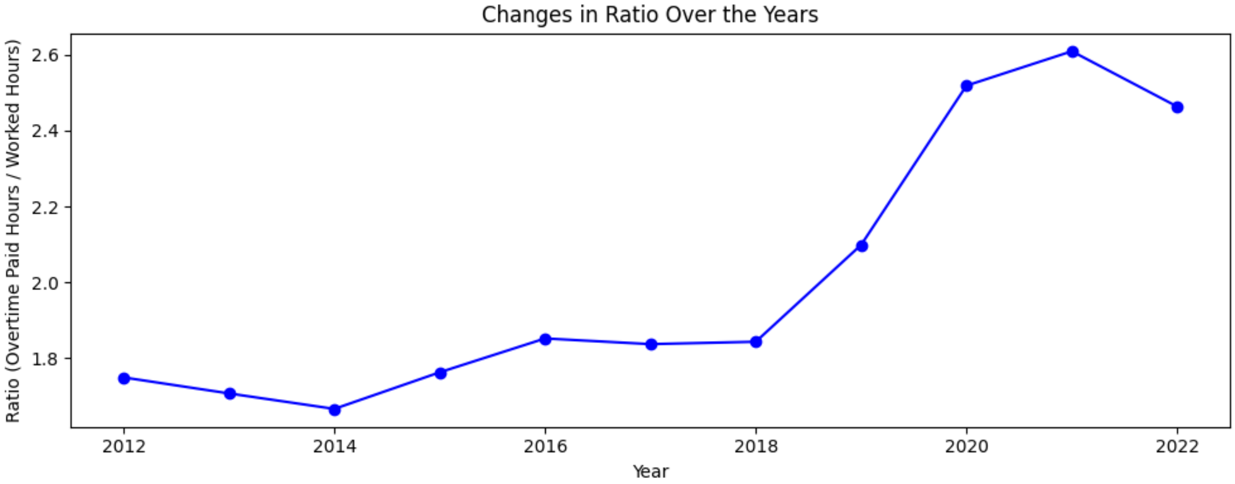
What is the distribution of ratios of overtime worked vs. overtime paid?

Are there any outliers?

(WRKDHRS vs. OTHOURS in the court OT database).

```python
# Calculate the ratio of overtime worked vs. overtime paid
overtime_hrs_arr = np.array(overtime_hrs)
wrk_hrs_arr = np.array(wrk_hrs)
ratio_overtime = overtime_hrs_arr / (wrk_hrs_arr)

# Calculate the ratio of overtime worked vs. overtime paid
ratio_overtime = np.array(ratio_overtime)

# Create a figure with subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=False)

# Plot the distribution of ratios using a histogram
ax1.hist(ratio_overtime, bins=10, edgecolor='black', alpha=0.7)
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Ratio of Overtime Paid Hours / Worked Hours')
ax1.set_title('Distribution of Ratios of Court Overtime Paid Hours /
Court Worked Hours')

# Plot the changes in ratio over the years
ax2.plot(year, ratio_overtime, marker='o', linestyle='-', color='b')
ax2.set_xlabel('Year')
ax2.set_ylabel('Ratio (Overtime Paid Hours / Worked Hours)')
ax2.set_title('Changes in Ratio Over the Years')

# Display the plots
plt.tight_layout()
plt.show()
```





Observations:

- The ratio of overtime paid hours to worked hours is approximately 1.7 for three years (2012, 2013 and 2014). This suggests a consistent level of overtime paid relative to the hours worked during these years.
- For four years, the ratio is around 1.8. This indicates a slightly higher proportion of overtime paid hours compared to the total worked hours.
- The years 2020, 2021 and 2022 experienced the ratios of overtime paid hours/ worked hours at about 2.5 times. However, if we look at the graph preceding this graph, we can see that the both overtime hours and worked hours were decreasing in these years.

3. Narratives around waste & misconduct by individual BPD officer

How much overlap is there between frequency overtime users and officers who have the highest salaries on the force?

(Truc Duong + Can Wang)

For each year from 2013 to 2022 we will find:

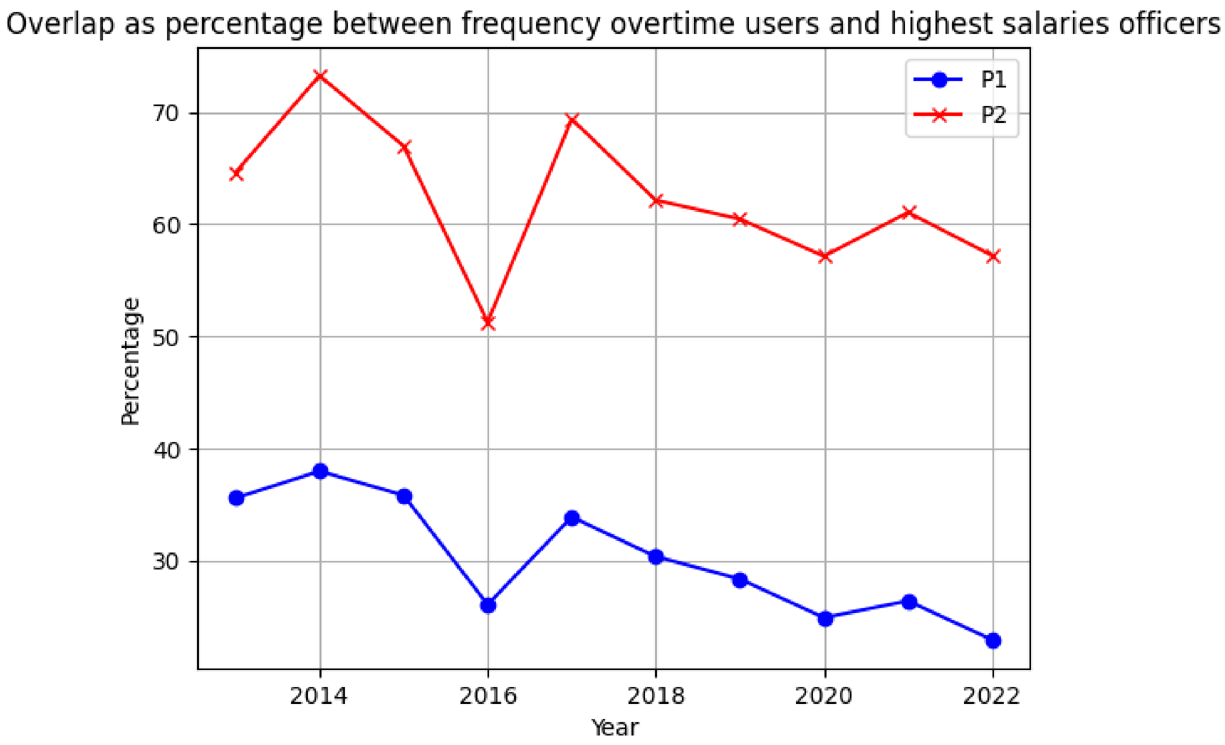
- Most frequent overtime users set = the top 20% officers who have highest overtime taking hours (using HOURS_PAID)
- Highest earning officer set = the top 20% highest earning officers
- We will find the overlap between these 2 set, and find its proportion compared to the union of the 2 sets.

Challenges:

- There is no EMPLOYEE_ID provided in the earning_data_list. Only the overtime_data_list has officers EMPLOYEE_ID
- Moreover, there are officers that don't take overtime, and thus, their EMPLOYEE_ID are not recorded in the overtime_dataset
- Since the eaning_data_list only use officer names. We will assume that officer names are unique

```
# function to find names of officers who are in top 20% of overtime
users
def find_top_20_overtime_names(df):
  # df is a year from overtime_data_list
  # Group by 'EMPLOYEE' name and sum the total hours worked for each
officer
  total_hours_per_officer = df.groupby('EMPLOYEE')['HOURS_PAID'].sum()

  # Sort the officers by total hours worked in descending order
  sorted_officers =
total_hours_per_officer.sort_values(ascending=False)

  # Calculate the top 20% threshold
  top_20_percent_threshold = sorted_officers.quantile(0.8)

  # Filter the officers who have worked more than the threshold
  top_20_overtime_officers = sorted_officers[sorted_officers >
top_20_percent_threshold]

  # Get the names of the top 20% officers
  top_20_percent_names =
df.loc[df['EMPLOYEE'].isin(top_20_overtime_officers.index),
'EMPLOYEE'].unique()

  return top_20_percent_names


# function to find names of officers who are in top 20% highest
earning
def find_top_20_earning_names(df):
  # df is a year from the earning_data_list
  # Group by 'NAME' and sum the total earnings for each officer
  total_earnings_per_officer = df.groupby('NAME')
['TOTAL_EARNING'].sum()

  # Sort the officers by total earnings in descending order
  sorted_earnings =
total_earnings_per_officer.sort_values(ascending=False)

  # Calculate the top 20% threshold
  top_20_percent_threshold = sorted_earnings.quantile(0.8)

  # Filter the officers who have earned more than the threshold
  top_20_percent_earnings = sorted_earnings[sorted_earnings >
top_20_percent_threshold]

  # Get the names of the top 20% officers
  top_20_percent_names = top_20_percent_earnings.index

  return top_20_percent_names

# find percentage of officers who are in top 20% overtime users and in
top 20% of highest income in the force
# from 2013 to 2022
# create a new copy of the earning data list, but only use from year
2013 - 2022
earning_data_list_2 = []
for data in earning_data_list[2:]:
  earning_data_list_2.append(data.copy())

overtime_year = [x for x in range(2013,2023)]

# let p1 = percentage of officers that are in top 20% of overtime user
given that they are in top 20% of highest income user
# let p2 = percentage of officers that are in top 20% of highest
income user given that they are in top 20% of overtime user
p1_list = []
p2_list = []

for i in range(len(overtime_data_list)):
  top_20_overtime_officers =
find_top_20_overtime_names(overtime_data_list[i])
```

```
  top_20_earning_officers =
find_top_20_earning_names(earning_data_list_2[i])
  # find the intersection of the 2 sets
  overlap_officers = np.intersect1d(top_20_overtime_officers,
top_20_earning_officers)
  # find the union of the 2 sets
  officers_in_either_set =
set(top_20_overtime_officers).union(set(top_20_earning_officers))
  # calculate percentage of officers that are in top 20% of overtime
user given that they are in top 20% of highest income user
  p1 = (len(overlap_officers) / len(top_20_earning_officers)) * 100
  p1_list.append(p1)
  # calculate percentage of officers that are in top 20% of highest
income user given that they are in top 20% of overtime user
  p2 = (len(overlap_officers) / len(top_20_overtime_officers)) * 100
  p2_list.append(p2)


plt.plot(overtime_year, p1_list, color='blue', marker='o', label='P1')
plt.plot(overtime_year, p2_list, color='red', marker='x', label='P2')
plt.xlabel("Year")
plt.ylabel("Percentage")
plt.title("Overlap as percentage between frequency overtime users and
highest salaries officers")
plt.grid(True)
plt.legend()
plt.show()
```



Explanation:

- P1 = Percentage of officers that are in top 20% of overtime user given that they are in top 20% of highest income
- P2 = Percentage of officers that are in top 20% of highest income given that they are in top 20% of overtime user

Observations:

- An officer who had high income was very likely (>50%) to take overtime frequently
- However, an officer who frequently took overtime didn't necessarily have high income

How much overlap is there between frequency overtime users and officers who are listed on the Suffolk County police watch list?

(Truc)

```
# suffolk_brady_2020.info()

# top_20_overtime_officers' is the list of top 20% frequent overtime
user officer names calculated above
# suffolk_brady_2020 is the DataFrame containing the Suffolk Brady
List 2020

# Standardize the 'NAME' columns in both datasets
```

```
top_20_overtime_officers = [name.upper().strip() for name in top_20_overtime_officers]
# Remove spaces after commas and standardize the capitalization in 'brady_list_data'
suffolk_brady_2020['NAME'] = suffolk_brady_2020['NAME'].str.replace(', ', ',').str.upper().str.strip()

# Find the overlapping officers
overlap_officers = set(top_20_overtime_officers).intersection(set(suffolk_brady_2020['NAME']))

# Print or use the overlapping officer names as needed
print("Number of Overlapping Officers:", len(overlap_officers))

Number of Overlapping Officers: 0
```

How much overlap is there between frequency overtime users and officers who have previously been disciplined for overtime abuse or other misconduct?

How much overlap is there between frequency overtime users and officers who have internal affairs complaint records?

(Truc)

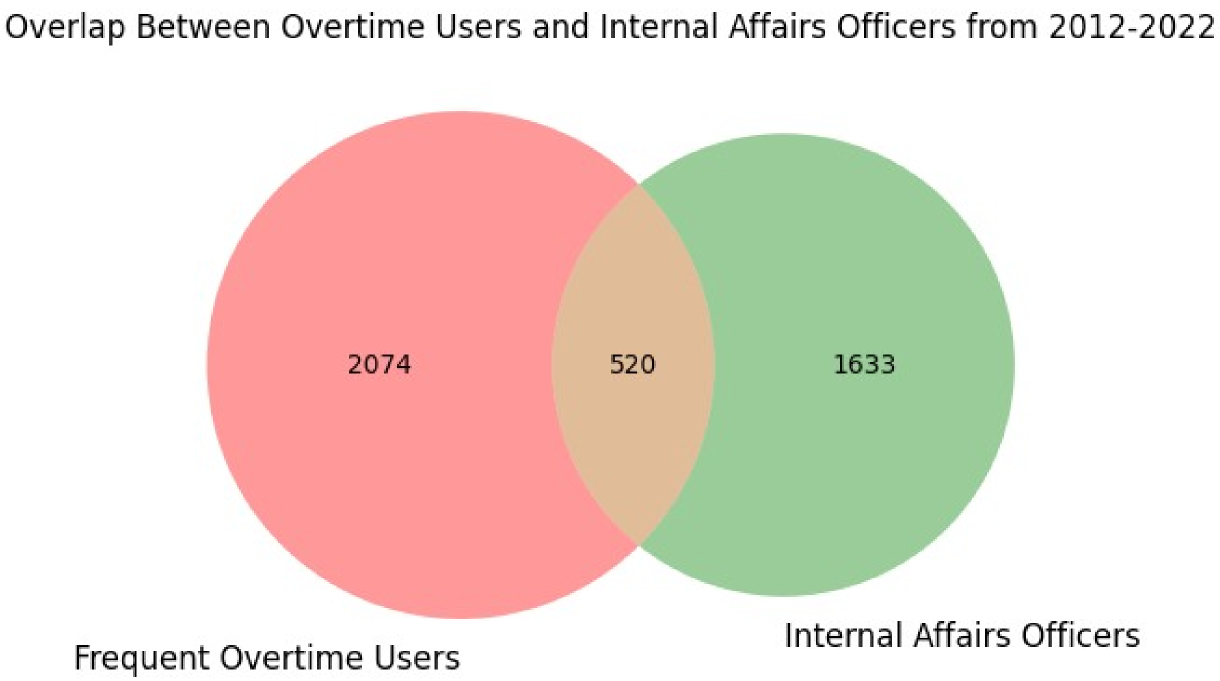
- Most frequent overtime users set = the top 20% officers who had the highest overtime taking hours (using HOURS_PAID)
- The names and the number of officers who had internal affair complaint records data was calculated from the internal affairs dataset.

```
# Combining the yearly datasets
combined_overtime = pd.concat(overtime_data_list, ignore_index=True)

# Standardizing the 'NAME' column in both datasets
combined_overtime['EMPLOYEE'] = combined_overtime['EMPLOYEE'].str.upper().str.strip()
internal_affairs_officers['name'] = internal_affairs_officers['name'].str.upper().str.strip()

# Group by 'NAME' and count the number of overtime entries in the combined dataset
overtime_frequency = combined_overtime.groupby('EMPLOYEE').size().reset_index(name='OT_COUNT')
```

```
# Identifying frequent overtime users (e.g., top quartile of officers
based on overtime count)
top_quartile_threshold = overtime_frequency['OT_COUNT'].quantile(0.80)
frequent_overtime_users =
overtime_frequency[overtime_frequency['OT_COUNT'] >=
top_quartile_threshold]

# Merging the datasets on 'NAME' to find overlap
overlap = pd.merge(frequent_overtime_users, internal_affairs_officers,
left_on='EMPLOYEE', right_on='name', how='left')

# Counting the number of unique overlapping officers
overlap_count = overlap['EMPLOYEE'].nunique()

# Outputting the result
print('Number of overlapping police officers between internal affairs
list, and overtime data:', overlap_count)
```

```
Number of overlapping police officers between internal affairs list,
and overtime data: 520
```

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn2

# Number of unique officers in each set
total_overtime_users = overtime_frequency['EMPLOYEE'].nunique()
total_internal_affairs_officers =
internal_affairs_officers['name'].nunique()

# Number of overlapping officers
overlap_count = overlap['EMPLOYEE'].nunique()

# Create the Venn diagram
venn_labels = {'100': f'{total_overtime_users}\nOvertime Users',
               '010': f'{total_internal_affairs_officers}\nInternal
Affairs',
               '110': f'{overlap_count}\nOverlap'}

venn2(subsets=(total_overtime_users - overlap_count,
               total_internal_affairs_officers - overlap_count,
               overlap_count),
      set_labels=('Frequent Overtime Users', 'Internal Affairs
Officers'))

# Display the plot
plt.title('Overlap Between Overtime Users and Internal Affairs
Officers from 2012-2022')
plt.show()
```


Overlap Between Overtime Users and Internal Affairs Officers from 2012-2022

Observations:

- The overlap represents a considerable portion of both frequent overtime users and officers with internal affairs complaint records. This suggests that a significant number of officers are simultaneously involved in both categories.
- The overlap may raise questions or concerns about the work behavior or conduct of these officers. It could indicate instances where officers who work extensive overtime also have internal affairs matters to address

4. Project Extension: BPD Staffing Analysis

A.Staffing vs Overtime Spending

(Truc)

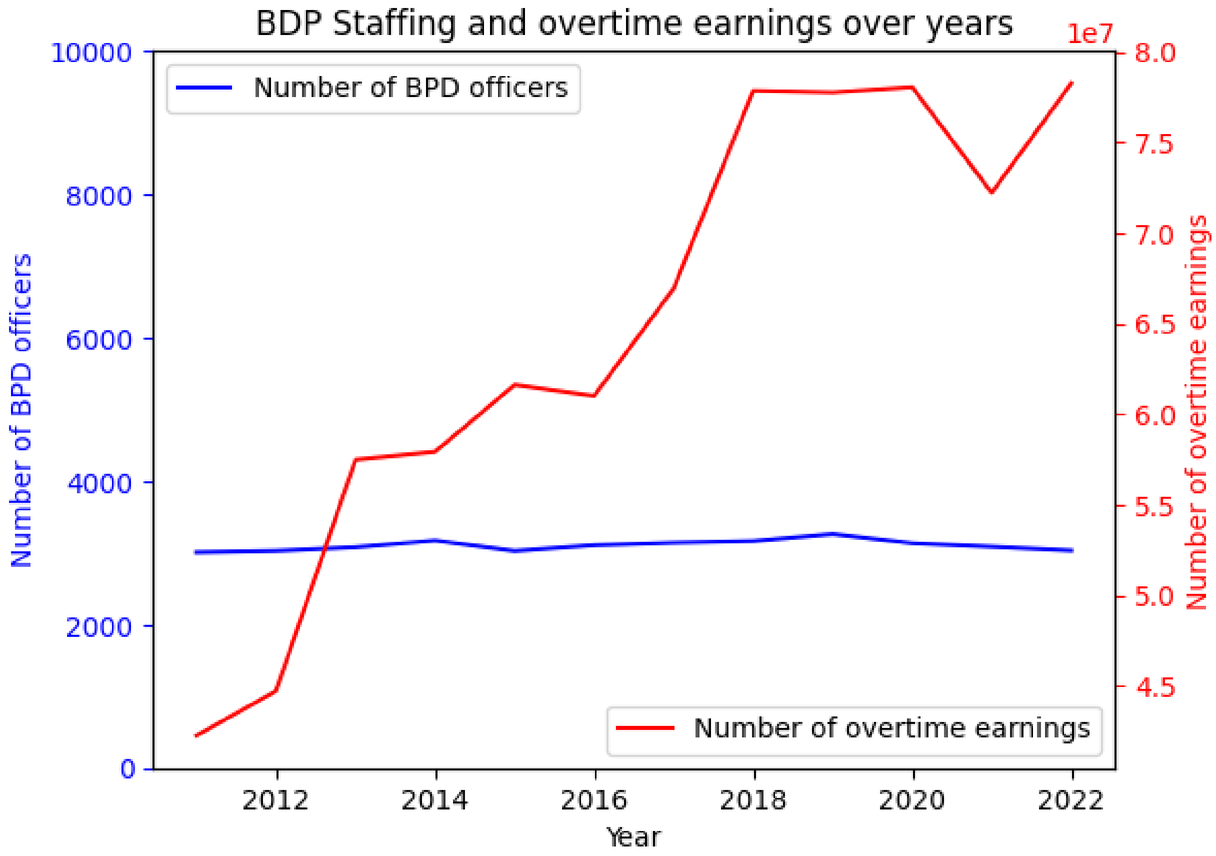
How does the staffing level within the BPD correlate with the frequency and magnitude of overtime expenditures?

```
earning_data_list[0].columns[6]

{"type":"string"}

overtime_earnings = []
for i in range(len(earning_data_list)):
  overtime_earnings.append(earning_data_list[i]
[earning_data_list[i].columns[6]].sum())

print(overtime_earnings)
```

```
[42237500.79000001, 44698730.70999999, 57483767.629999995,
57914605.89, 61608537.989999995, 60998676.760000005,
66933649.86999999, 77855435.97999999, 77764302.50999999, 78057696.23,
72223009.96000001, 78265758.01]
```

```
num_officers = []
staff_years = [x for x in range(2011, 2023)]

for bpd_data in earning_data_list:
  num_officers.append(bpd_data['NAME'].nunique()) # the number of
officers = staffing size

print("Number of Boston police officers over years:")
print(num_officers)
print("Number of overtime earnings over years:")
print(overtime_earnings)

# Create the first axis
fig, ax1 = plt.subplots()
# Plot the staffing data on the left axis
ax1.plot(staff_years, num_officers, color='blue', label='Number of BPD
officers')
ax1.set_xlabel('Year')
ax1.set_ylabel('Number of BPD officers', color='blue')
ax1.set_ylim(0, 10000)
ax1.tick_params('y', colors='blue')

# Create the second axis sharing the same x-axis
ax2 = ax1.twinx()
# Plot the second data on the right axis
ax2.plot(staff_years, overtime_earnings, color='red', label='Number of
overtime earnings')
ax2.set_ylabel('Number of overtime earnings', color='red')
ax2.tick_params('y', colors='red')

# Display the legend
ax1.legend(loc='upper left')
ax2.legend(loc='lower right')
plt.title("BDP Staffing and overtime earnings over years")
```

```
Number of Boston police officers over years:
[3010, 3030, 3080, 3173, 3029, 3108, 3143, 3166, 3263, 3136, 3087,
3035]
Number of overtime earnings over years:
[42237500.79000001, 44698730.70999999, 57483767.629999995,
57914605.89, 61608537.989999995, 60998676.760000005,
66933649.86999999, 77855435.97999999, 77764302.50999999, 78057696.23,
72223009.96000001, 78265758.01]

Text(0.5, 1.0, 'BDP Staffing and overtime earnings over years')
```



B. Staffing vs Crime Rates

Analyzing the relationship between the number of police officers and the number of crime incident reports over the years

1. Time Series Line Chart
2. Correlation Analysis: Calculate the correlation coefficient between the number of police officers and the number of crime incident reports
   - A positive correlation suggests that as the number of police officers increases, the number of reported incidents also increases. A negative correlation suggests the opposite.
3. Calculate the number of police officers and crime incident reports per capita (per 1,000 residents or another relevant metric)

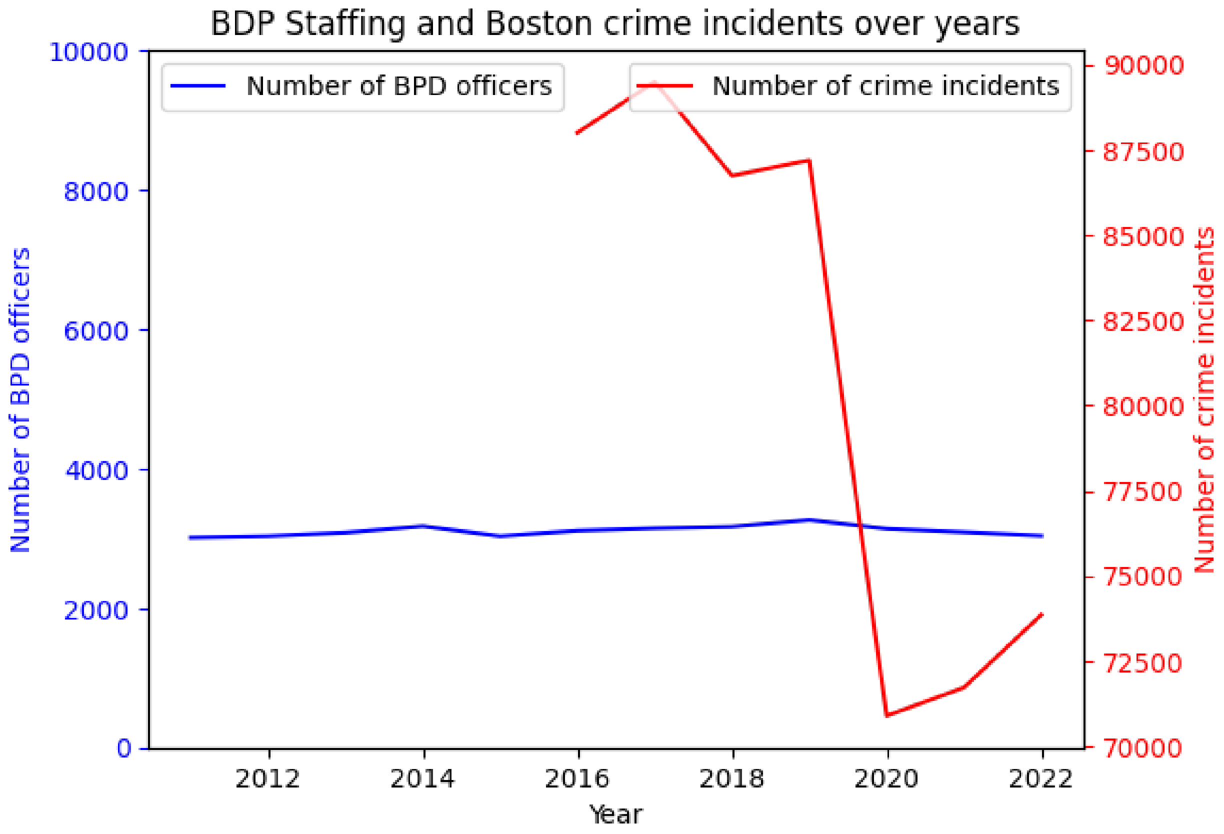
   - This normalization allows you to assess the efficiency of law enforcement efforts relative to population size

4. Break down crime incident reports into categories (e.g., violent crimes, property crimes) and analyze the trends in each category.
   - Use stacked bar charts or grouped bar charts to illustrate the distribution of crime categories.
   - Identify specific crime categories that may be more influenced by changes in police staffing.

How did BPD staffing and the number of crime reports change year-over-year

```python
num_officers = []
num_crimes = []
staff_years = [x for x in range(2011, 2023)]
crime_years = [x for x in range(2016, 2023)]

for bpd_data in earning_data_list:
  num_officers.append(bpd_data['NAME'].nunique()) # the number of officers = staffing size

for crime_df in crime_data_list[1:]:
  num_crimes.append(crime_df['INCIDENT_NUMBER'].nunique())

print("Number of Boston police officers over years:")
print(num_officers)
print("Number of Boston crime incidents over years:")
print(num_crimes)

# Create the first axis
fig, ax1 = plt.subplots()
# Plot the staffing data on the left axis
ax1.plot(staff_years, num_officers, color='blue', label='Number of BPD officers')
ax1.set_xlabel('Year')
ax1.set_ylabel('Number of BPD officers', color='blue')
ax1.set_ylim(0, 10000)
ax1.tick_params('y', colors='blue')

# Create the second axis sharing the same x-axis
ax2 = ax1.twinx()
# Plot the second data on the right axis
ax2.plot(crime_years, num_crimes, color='red', label='Number of crime incidents')
ax2.set_ylabel('Number of crime incidents', color='red')
ax2.tick_params('y', colors='red')

# Display the legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
```

```
plt.title("BDP Staffing and Boston crime incidents over years")


Number of Boston police officers over years:
[3010, 3030, 3080, 3173, 3029, 3108, 3143, 3166, 3263, 3136, 3087,
3035]
Number of Boston crime incidents over years:
[87994, 89486, 86734, 87184, 70894, 71721, 73852]

Text(0.5, 1.0, 'BDP Staffing and Boston crime incidents over years')
```



Observations:

- The number of police officers is generally stable around 3000 officers.
- There is a dramatic decrease in the number of reported crime incidents in 2020. This could be resulted from the Covid-19 pandemic.
- It's not immediately clear from the plot if there is a strong linear relationship between the number of police officers and the number of crime incidents
- Further statistical analysis, such as correlation coefficients or regression analysis, may be necessary to quantify the relationship between the number of police officers and crime incidents

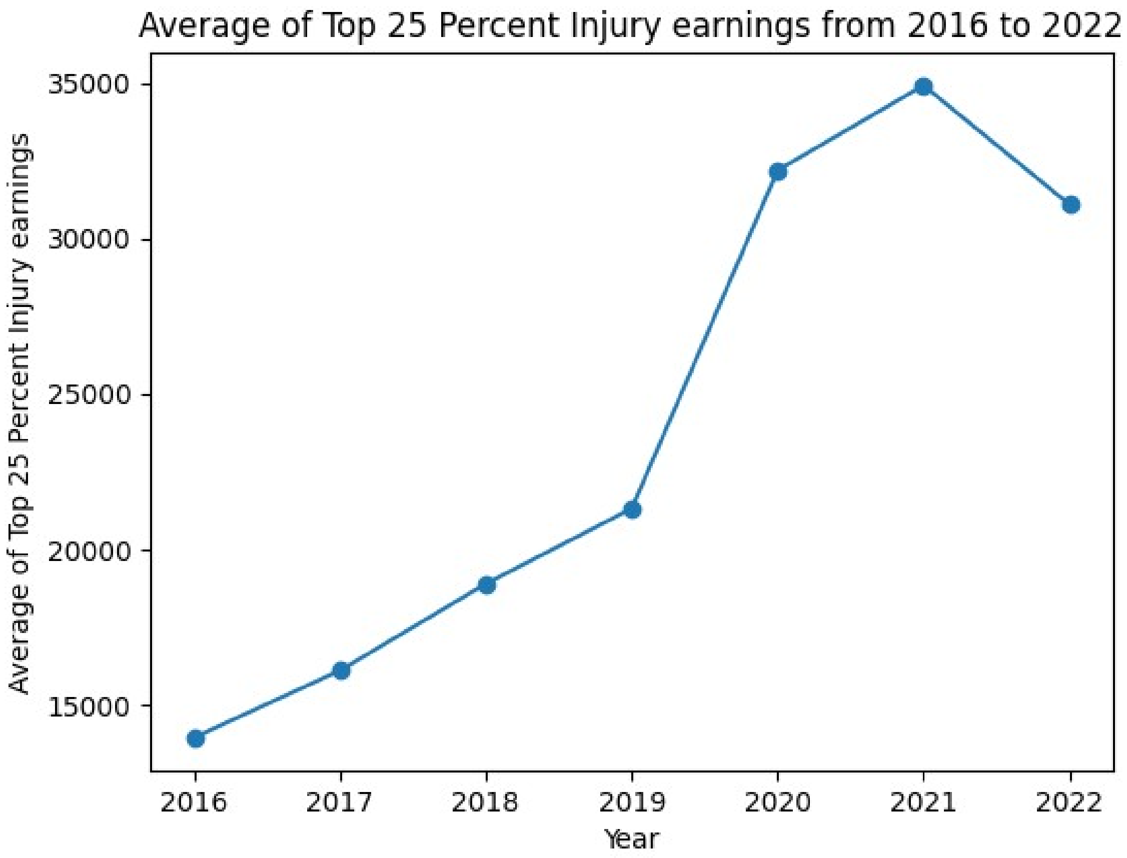
Year 2020 abnormal decrease in crime incident reports

We found that at year 2020, the number of crime indicies decresed significntly, but the percentage of total injury parymen incresed. We will dive further and try to find the reasons for it.

```
top_25_injury = []

for i in range(7):

  # Step 1: Sort 2020 injury earnigs by 'INJURED' in descending order
  sorted_2020_injury =
earning_data_list[i+5].sort_values(by=earning_data_list[i+5].columns[7
], ascending=False)

  # Step 2: Calculate the number of rows for the top 25%
  top_25_percent = int(len(sorted_2020_injury) * 0.25)

  # Step 3: Extract the top 25% rows
  top_25_percent_rows = sorted_2020_injury.iloc[:top_25_percent]

  top_25_injury.append(top_25_percent_rows)

top_25_injury[0].head(3)

                         NAME                     DEPT_NAME
TITLE  \
1443             JEAN,HARRY Y  Boston Police Department   Police
Detective
1557         KENNEDY,JOSEPH M  Boston Police Department   Police
Detective
1281  HARTGROVE,CHRISTOPHER A  Boston Police Department   Police
Detective

      REGULAR     RETRO     OTHER  OVERTIME    INJURED  DETAIL
EDUCATION  \
1443      NaN   5003.20    850.00       NaN  144961.78     NaN
NaN
1557      NaN  25064.54  11083.55       NaN  121223.12     NaN
12088.8
1281      NaN  64118.84  14175.06       NaN  119319.55     NaN
NaN

      TOTAL_EARNING POSTAL
1443      150464.67  02124
1557      169127.87  01906
1281      197613.45  02066

avg_top_25_percent = []

for i in range(len(top_25_injury)):
  avg_top_25_percent.append(top_25_injury[i]
[top_25_injury[i].columns[7]].sum() / len(top_25_injury[i]))

print(avg_top_25_percent[0])
```

```
13931.779408740362
```

```
import matplotlib.pyplot as plt

years_want = [i for i in range(2016, 2023)]

# Create a line plot
plt.plot(years_want, avg_top_25_percent, marker='o', linestyle='-')

# Adding labels and title
plt.xlabel('Year')  # Replace with your actual label
plt.ylabel('Average of Top 25 Percent Injury earnings')  # You can
customize this label
plt.title('Average of Top 25 Percent Injury earnings from 2016 to
2022')  # Replace with your actual title

# Show the plot
plt.show()
```

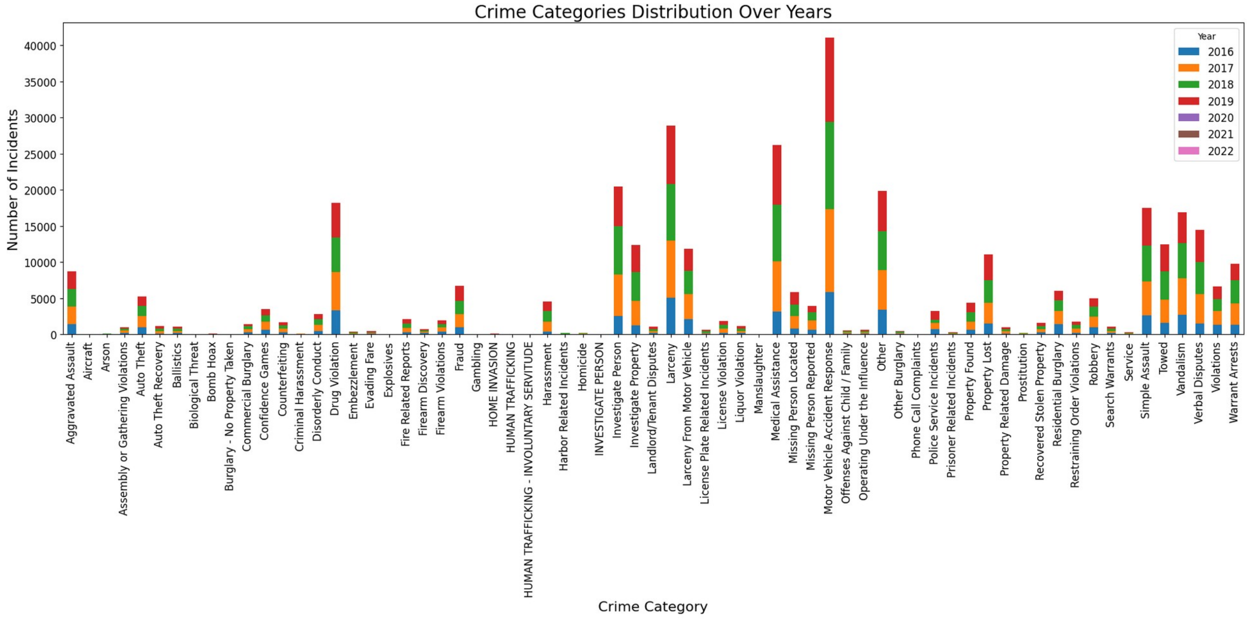


By the plot we can see that at year 2020, although the number of incidents decreased, average amount of injury earnings for the top 25% increased significntly. Thus, leading to the result of increasing in injury total expenditure.

Crime Incident Reports By Category over Years

Crime Incidents Analysis

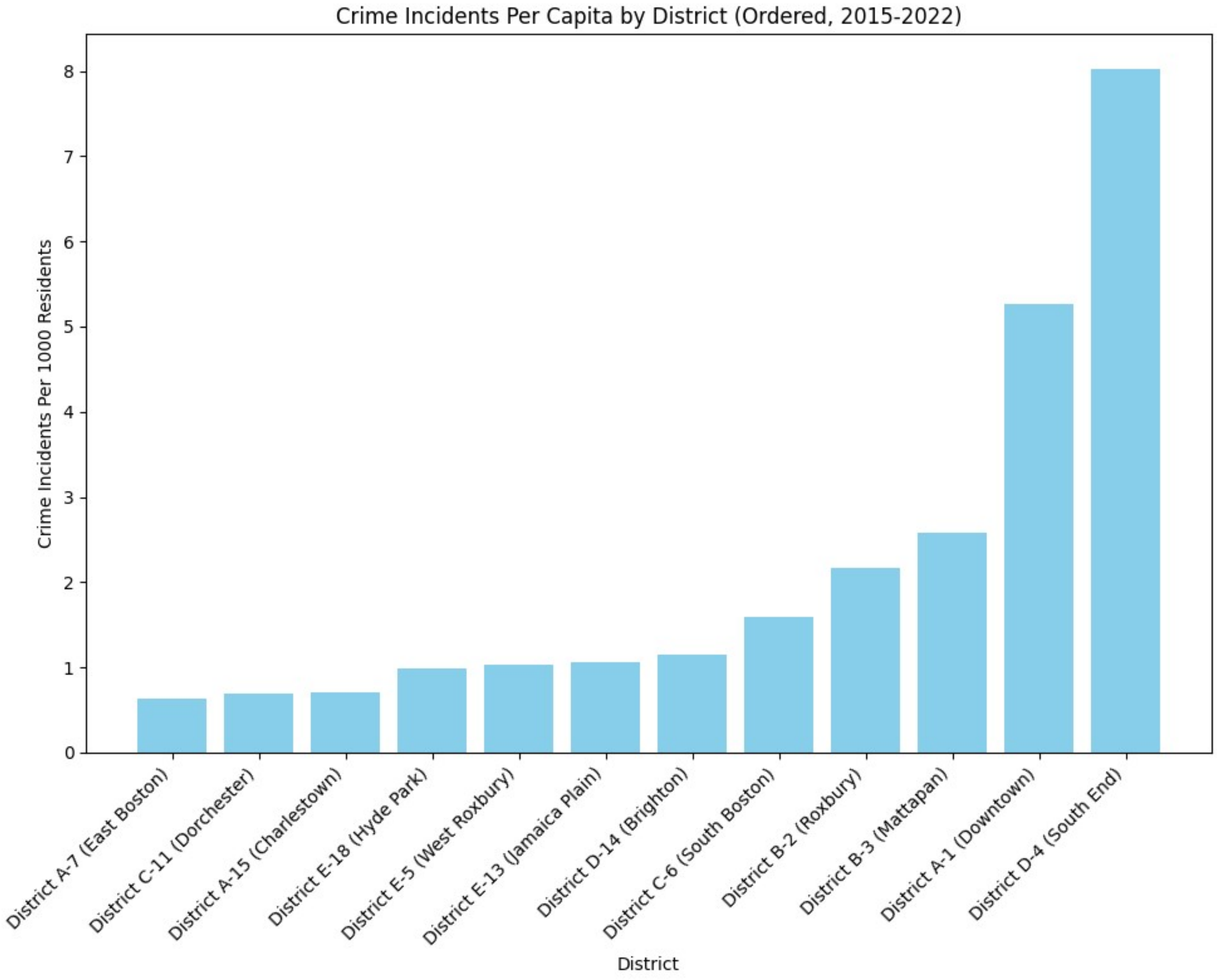
```
crime_categories_per_year = {}

# Categorizing crime incidents for each year and counting occurrences
for year, crime_df in zip(range(2016, 2023), crime_data_list):
    crime_categories_per_year[year] =
crime_df['OFFENSE_CODE_GROUP'].value_counts()

# Preparing data for the stacked bar chart
category_df = pd.DataFrame(crime_categories_per_year)

# Plotting the stacked bar chart with adjusted parameters for improved
legibility
category_df.plot(kind='bar', stacked=True, figsize=(20, 10))
plt.title('Crime Categories Distribution Over Years', fontsize=20)
plt.xlabel('Crime Category', fontsize=16)
plt.ylabel('Number of Incidents', fontsize=16)
plt.xticks(rotation=90, fontsize=12)  # Rotate labels to 90 degrees
for better visibility
plt.yticks(fontsize=12)
plt.legend(title='Year', fontsize=12)
plt.tight_layout()  # Adjust layout to make room for the rotated x-
axis labels
plt.show()
```

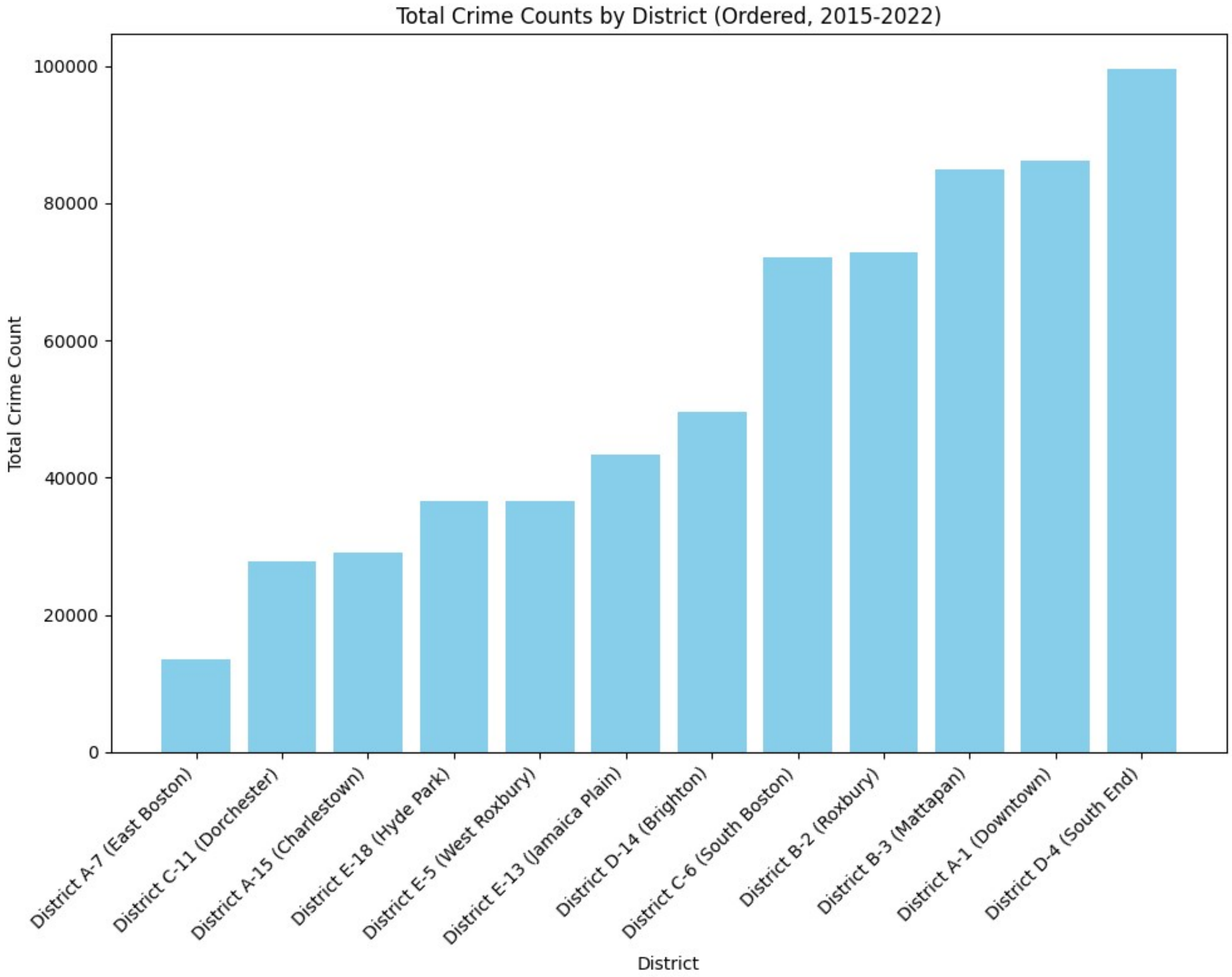


Crime Incident Reports Per Capita by District

```
combined_crime_data = pd.concat(crime_data_list)

# Counting the number of crimes in each district
crime_counts_by_district =
combined_crime_data['DISTRICT'].value_counts()

# Population data for each district
population_data = {
    'A1': 13827, 'A15': 19273, 'A7': 44295, 'B2': 45898, 'B3': 27900,
'C6': 31132,
    'C11': 124208, 'D4': 10575, 'D14': 37785, 'E5': 28283, 'E13':
34587, 'E18': 36883
}

# Remove any district codes from the crime data that are not in the
population data
crime_counts_by_district =
crime_counts_by_district[crime_counts_by_district.index.isin(populatio
n_data.keys())]

# Calculate the crime incidents per capita for each district
crimes_per_capita = {district: (crime_counts /
population_data[district])
                     for district, crime_counts in
crime_counts_by_district.items() if district in population_data}

# Sort the crimes_per_capita by value from lowest to highest
crimes_per_capita_sorted = dict(sorted(crimes_per_capita.items(),
key=lambda item: item[1]))

# Replace district codes with full names for clarity
district_full_names = {
    'A1': 'District A-1 (Downtown)', 'A15': 'District A-15
(Charlestown)', 'A7': 'District A-7 (East Boston)',
    'B2': 'District B-2 (Roxbury)', 'B3': 'District B-3 (Mattapan)',
'C6': 'District C-6 (South Boston)',
    'C11': 'District C-11 (Dorchester)', 'D4': 'District D-4 (South
End)', 'D14': 'District D-14 (Brighton)',
    'E5': 'District E-5 (West Roxbury)', 'E13': 'District E-13
(Jamaica Plain)', 'E18': 'District E-18 (Hyde Park)'
}

# Create a list for the sorted district names and their per capita
values
sorted_district_names = [district_full_names[district] for district in
crimes_per_capita_sorted.keys()]
sorted_crimes_per_capita = list(crimes_per_capita_sorted.values())

# Plotting the data
```

```python
plt.figure(figsize=(10, 8))
bars = plt.bar(sorted_district_names, sorted_crimes_per_capita, color='skyblue')
plt.title("Crime Incidents Per Capita by District (Ordered, 2015-2022)")
plt.xlabel("District")
plt.ylabel("Crime Incidents Per 1000 Residents")
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()

# Convert the Series to a DataFrame for easier plotting
crime_counts_df = crime_counts_by_district.reset_index()
crime_counts_df.columns = ['District', 'Crime Count']

# Sort the DataFrame by crime count in descending order
crime_counts_df = crime_counts_df.sort_values(by='Crime Count', ascending=True)

# Plotting the data
plt.figure(figsize=(10, 8))
bars = plt.bar(sorted_district_names, crime_counts_df['Crime Count'], color='skyblue')
plt.title("Total Crime Counts by District (Ordered, 2015-2022)")
plt.xlabel("District")
plt.ylabel("Total Crime Count")
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```


Crime Incidents Per Capita by District (Ordered, 2015-2022)
Crime Incidents Per 1000 Residents
0
1
2
3
4
5
6
7
8
District
District A-7 (East Boston)
District C-11 (Dorchester)
District A-15 (Charlestown)
District E-18 (Hyde Park)
District E-5 (West Roxbury)
District E-13 (Jamaica Plain)
District D-14 (Brighton)
District C-6 (South Boston)
District B-2 (Roxbury)
District B-3 (Mattapan)
District A-1 (Downtown)
District D-4 (South End)



Overtime Spending vs Staffing vs Crime Rates

Objective: Try to find out what is the relationship between the number of officers, the amount of overtime spending, and the crime rates by districts

Approach:

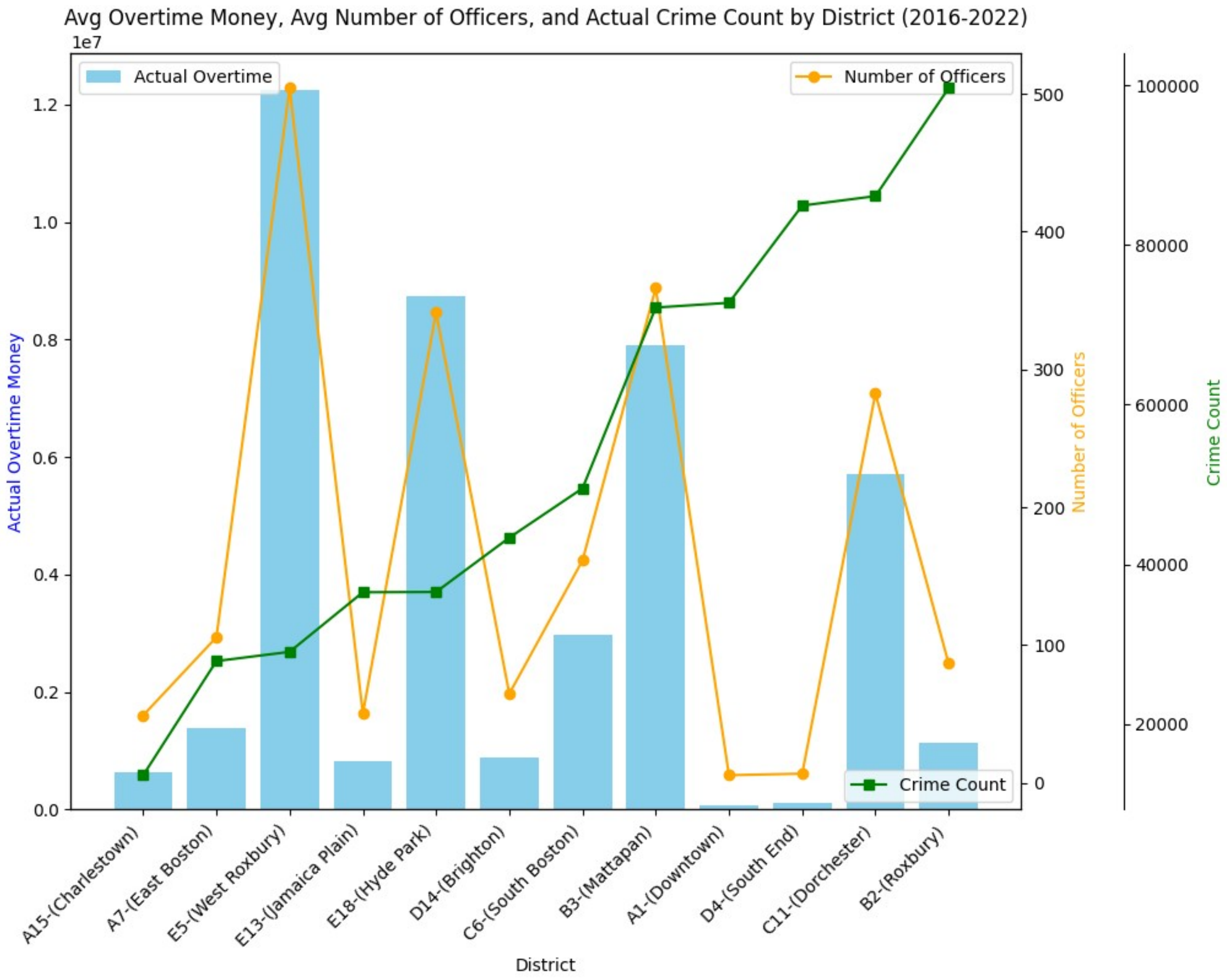
- To find the number of police officers by District, I used the zipcode in the earnings_data_list. Then I calculated the average number of officers in each district from year 2016 - 2022.
- The total overtime spending by District is calculate from the 'OVERTIME' field in earnings_data_list.
- The average crime rates is calculated using the Crime Incident reports (calculated above)

```
crime_counts_df.head(20)
```

```
   District  Crime Count
11      A15        13600
10       A7        27895
9        E5        29050
```

```
8       E13        36510
7       E18        36547
6       D14        43346
5        C6        49522
4        B3        72162
3        A1        72755
2        D4        84927
1       C11        86111
0        B2        99635

district_zipcodes = {
    'A15-(Charlestown)': [2129, '02129', '2129'],
    'A7-(East Boston)': [2128, '02128', '2128'],
    'E5-(West Roxbury)': [2132, 2131, '02132', '02131' ,'2132',
'2131'],
    'E13-(Jamaica Plain)': [2130, '02130', '2130'],
    'E18-(Hyde Park)': [2136, '02136', '2136'],
    'D14-(Brighton)': [2135, '02135', '2135'],
    'C6-(South Boston)': [2127, '02127', '2127'],
    'B3-(Mattapan)': [2124, 2126, '02124', '02126' , '2124', '2126'],
    'A1-(Downtown)': [2108, 2109, 2110, 2111, '02108', '02109',
'02110', '02111', '2108', '2109', '2110', '2111'],
    'D4-(South End)': [2116, '02116', '2116'],
    'C11-(Dorchester)': [2121, 2122, '02121', '02122', '2121',
'2122'],
    'B2-(Roxbury)': [2119, 2120, '02119', '02120', '2119', '2120'],
}

years = [x for x in range(2016, 2023)]
district_codes = list(district_zipcodes.keys())

# overtime money by districts over years from 2016 to 2023
overtime_by_districts = []
# number of officers by districts over years from 2016 to 2023
officers_by_districts = []
# Crime count by district
crime_count_data = pd.DataFrame({
    'District': ['A15-(Charlestown)', 'A7-(East Boston)', 'E5-(West
Roxbury)', 'E13-(Jamaica Plain)',\
                 'E18-(Hyde Park)','D14-(Brighton)', 'C6-(South
Boston)', 'B3-(Mattapan)',\
                 'A1-(Downtown)', 'D4-(South End)', 'C11-
(Dorchester)', 'B2-(Roxbury)'],
    'Crime Count': [13600, 27895, 29050, 36510, 36547, 43346, 49522,
72162, 72755, 84927, 86111, 99635]
})

for i in range(5,12):
  new_df = earning_data_list[i].copy()
  new_df['DISTRICT'] = new_df['POSTAL'].apply(lambda x: next((k for k,
v in district_zipcodes.items() if x in v), None))
  overtime_by_districts.append(new_df.groupby('DISTRICT')
['OVERTIME'].sum())
  officers_by_districts.append(new_df.groupby('DISTRICT')
['NAME'].nunique())

# Concatenate series into a DataFrame
df_overtime = pd.concat(overtime_by_districts, axis=1, keys=years)
df_officers = pd.concat(officers_by_districts, axis=1, keys=years)

# Transpose the DataFrame for year as the x-axis
df_overtime = df_overtime.transpose()
df_officers = df_officers.transpose()

# Calculate average overtime money by district
average_overtime_by_district = df_overtime.mean()
# Calculate average number of officers by district
average_officers_by_district = df_officers.mean()

# Merge dataframes based on the 'District' column
average_overtime_by_district_df = pd.DataFrame({'Avg Overtime':
average_overtime_by_district})
average_officers_by_district_df = pd.DataFrame({'Number of Officers':
average_officers_by_district})
merged_data = pd.merge(crime_count_data,
average_officers_by_district_df, left_on='District', right_index=True)
merged_data = pd.merge(merged_data, average_overtime_by_district_df,
left_on='District', right_index=True)

# Create a plot with three y-axes
fig, ax1 = plt.subplots(figsize=(10, 8))
plt.xticks(rotation=45, ha='right')

# Plot actual overtime as bars
ax1.bar(merged_data['District'], merged_data['Avg Overtime'],
color='skyblue', label='Actual Overtime')

# Create a second y-axis for the number of officers
ax2 = ax1.twinx()
ax2.plot(merged_data['District'], merged_data['Number of Officers'],
color='orange', marker='o', label='Number of Officers')

# Create a third y-axis for the crime count
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))
ax3.plot(merged_data['District'], merged_data['Crime Count'],
color='green', marker='s', label='Crime Count')

# Set labels and title
ax1.set_xlabel('District')
```

```
ax1.set_ylabel('Actual Overtime Money', color='blue')
ax2.set_ylabel('Number of Officers', color='orange')
ax3.set_ylabel('Crime Count', color='green')
plt.title('Avg Overtime Money, Avg Number of Officers, and Actual
Crime Count by District (2016-2022)')

# Show legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
ax3.legend(loc='lower right')

# Display the plot
plt.show()
```

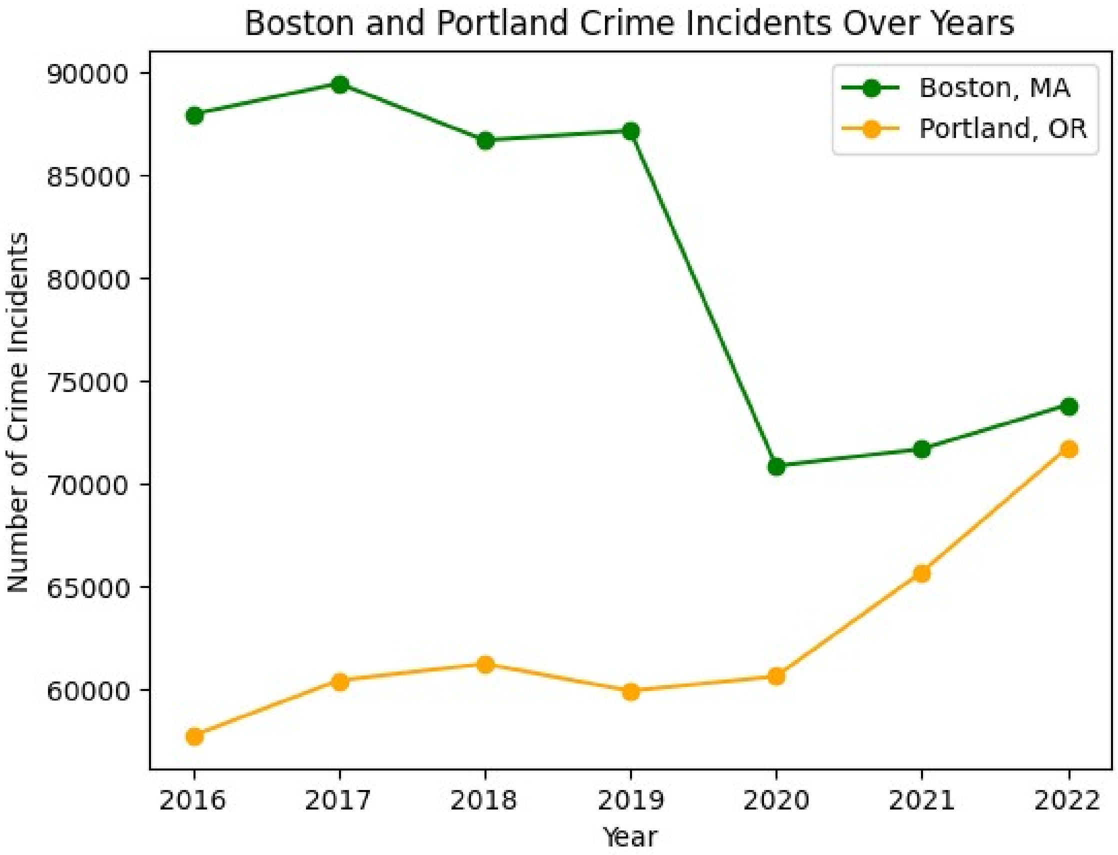


Observation:

- We can see that the amount of money spent on overtime experiences a very similar trend to the number of officers.
- From the graph: There are higher crime rates in districts with fewer number of officers. For instance: In Roxbury, the number of officers is small and there is a high crime rate

- However, in district with more number of officers, crime rate is still high (i.e: Mattapan)

```
# Portland crime data
num_crimes_portland = [57786, 60467, 61268, 59958, 60666, 65734,
71780]

# Create a figure for plotting
fig, ax = plt.subplots()

# Plot the BPD crime data
ax.plot(crime_years, num_crimes, color='green', marker='o',
label='Boston, MA')

# Plot the Portland crime data
ax.plot(crime_years, num_crimes_portland, color='orange', marker='o',
label='Portland, OR')

# Setting the labels and title
ax.set_xlabel('Year')
ax.set_ylabel('Number of Crime Incidents')
ax.set_title("Boston and Portland Crime Incidents Over Years")

# Display the legend
ax.legend(loc='upper right')

# Show the plot
plt.show()

print("Number of Boston crime incidents over years:")
print(num_crimes)
print("Number of Portland crime incidents over years:")
print(num_crimes_portland)
```



```
Number of Boston crime incidents over years:
[87994, 89486, 86734, 87184, 70894, 71721, 73852]
Number of Portland crime incidents over years:
[57786, 60467, 61268, 59958, 60666, 65734, 71780]
```

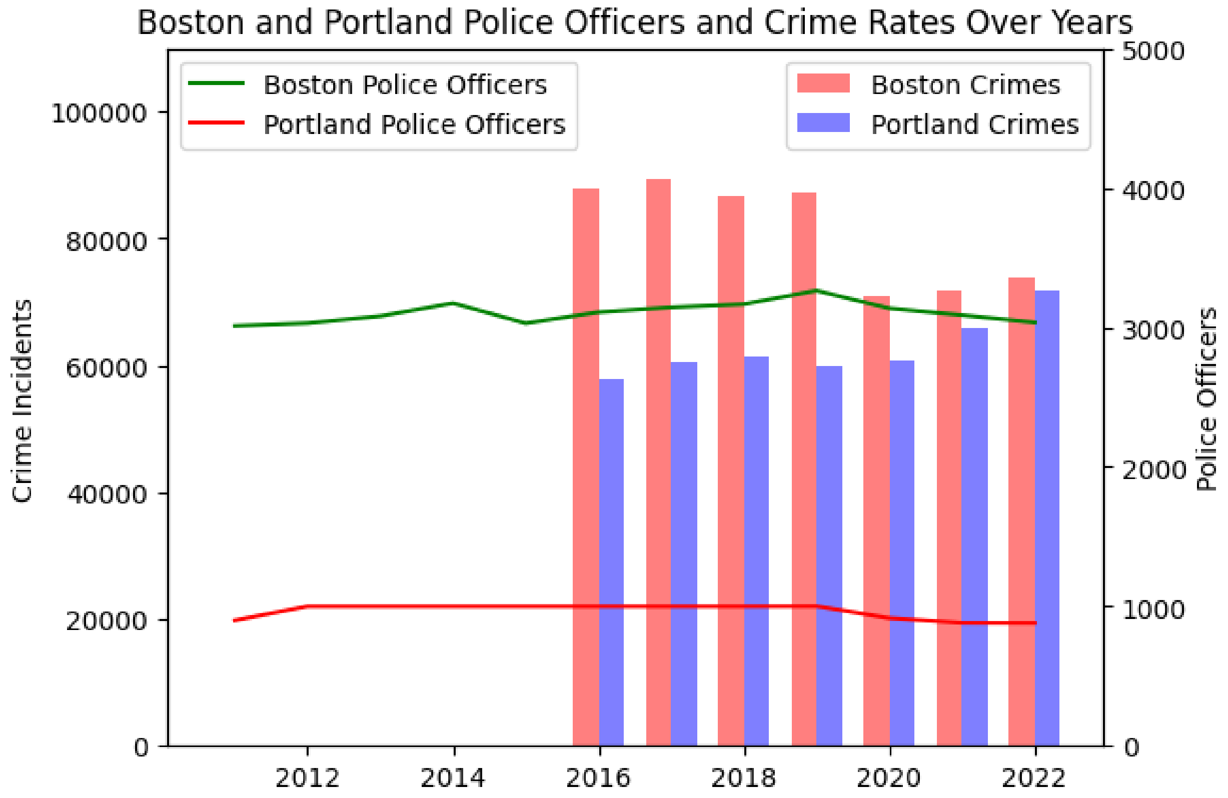
Observations:

- Portland, Oregon was selected due to its population being the most similar to Boston's at around 650k
- The data only represents all years from 2016-2022 due to not having access to data prior to these years
- Although Boston had almost double the crime incident reports in 2016, Portland has experienced an overall increase in reports while Boston experienced a sharp decrease
- Both the decline in Boston reports and the increase in Portland reports around 2020, it is safe to assume that the global pandemic played a key role

```
years = list(range(2011,2023))

num_officers_boston = []
# Boston staffing
for bpd_data in earning_data_list:
  num_officers_boston.append(bpd_data['NAME'].nunique()) # the number
of officers = staffing size
```

```
print("Number of Boston police officers over years:")
print(num_officers)

# Number of officers for Portland PD
num_officers_portland = [900, 1000, 1000, 1000, 1000, 1000, 1000,
1000, 1001, 916, 882, 881]
print("Number of Portland police officers over years:")
print(num_officers_portland)

# Crime data for Boston
num_crimes_boston = [0, 0, 0, 0, 0] + num_crimes #since we only have
crime data for 2016-2022

# Crime data for Portland
num_crimes_portland = [0, 0, 0, 0, 0, 57786, 60467, 61268, 59958,
60666, 65734, 71780] #since we only have crime data for 2016-2022

bar_width = 0.35

# Create a figure and axis
fig, ax1 = plt.subplots()

# Adjust the x coordinates for the second set of bars
years_boston = np.array(years) - bar_width / 2
years_portland = np.array(years) + bar_width / 2

ax1.bar(years_boston, num_crimes_boston, width=bar_width, alpha=0.5,
label='Boston Crimes', color='red')
ax1.bar(years_portland, num_crimes_portland, width=bar_width,
alpha=0.5, label='Portland Crimes', color='blue')

# Set the y-axis label for crime rates
ax1.set_ylabel('Crime Incidents', color='black')
ax1.tick_params('y', colors='black')

ax1.set_ylim(0,110000)

# Create a second y-axis for crime rates
ax2 = ax1.twinx()

# Plot police officers with a line graph
ax2.plot(years, num_officers_boston, label='Boston Police Officers',
color='green')
ax2.plot(years, num_officers_portland, label='Portland Police
Officers', color='red')

# Set the y-axis label
ax2.set_ylabel('Police Officers', color='black')
ax2.tick_params('y', colors='black')
```

```
ax2.set_ylim(0,5000)

# Set the x-axis label
plt.xlabel('Year')

# Add a legend
ax1.legend(loc='upper right')
ax2.legend(loc='upper left')

# Show the plot
plt.title('Boston and Portland Police Officers and Crime Rates Over
Years')
plt.show()

Number of Boston police officers over years:
[3010, 3030, 3080, 3173, 3029, 3108, 3143, 3166, 3263, 3136, 3087,
3035]
Number of Portland police officers over years:
[900, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1001, 916, 882, 881]
```



Observations:

- Police understaffing is currently a nationwide issue and doesn't just affect these two departments

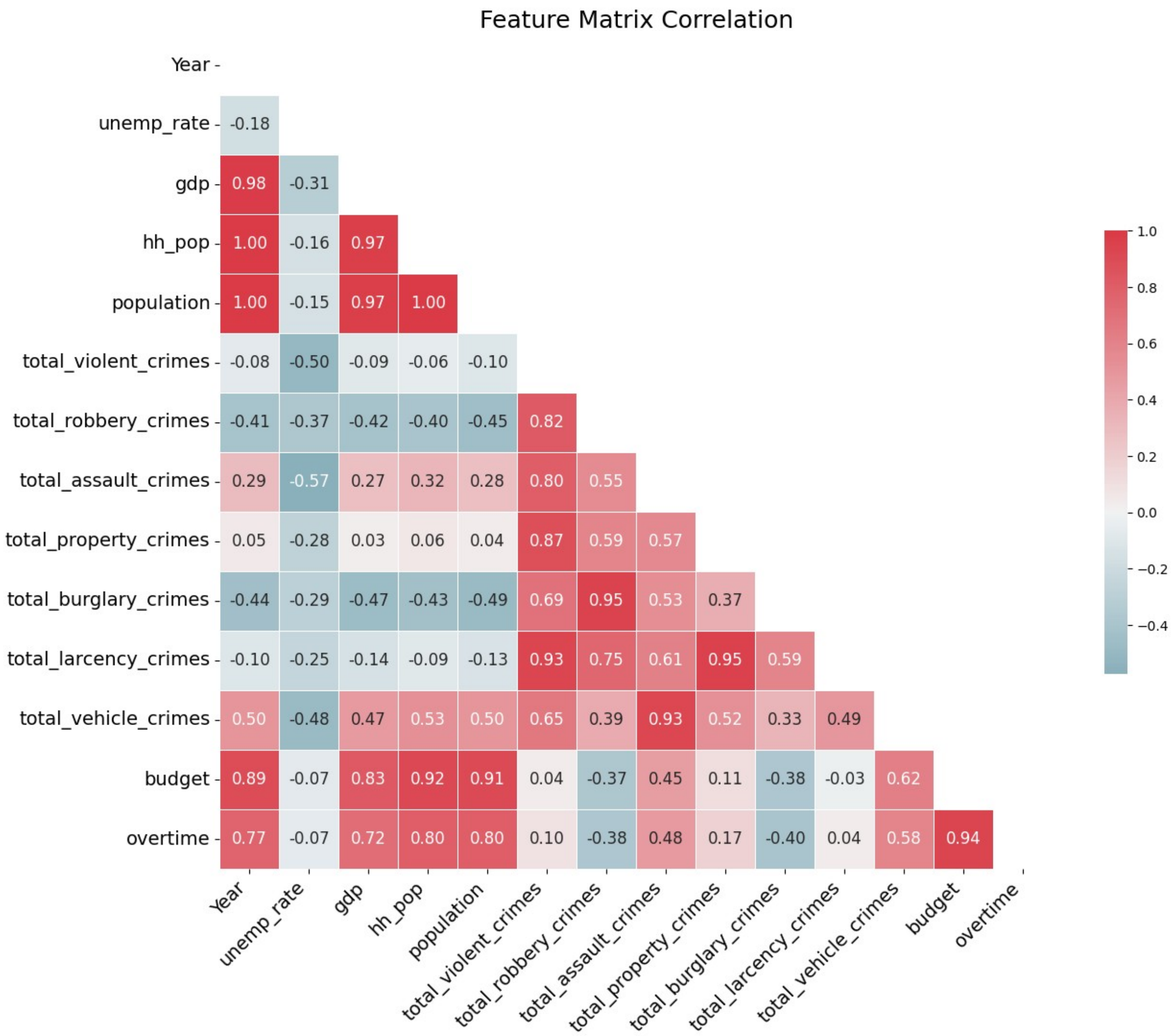
- Portland PD is regulary said to be offering double overtime as a response to the slow decrease in the number of officers each year
- Despite a decline in crime reports, the earnings of BPD officers have seen an upward trajectory while staffing levels have remained relatively unchanged
- This suggests that the increase in officer earnings may not be directly linked to heightened workload or understaffing
- This could potentially be due to increased revenue streams for both the BPD and the state with Massachusetts having the 2nd highest GDP per capita of any state
- Portland, which is vastly less staffed than Boston, is experiencing a decline in staffing but a rise in crime; a telltale sign of understaffing
- Since Boston's staffing is steady while crime is decreasing, it suggest that in the absolute worst case, Boston is adequately staffed

C. Prediction Model

```
train_df = pd.read_csv('model/traindataset.csv')
prediction_df = pd.read_csv('model/prediction_df.csv')

# Corr matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
# Load your data into a DataFrame
# df = pd.read_csv('your_dataset.csv')

# Calculate the correlation matrix
print(train_df.columns)
new_order_col = ['Year', 'unemp_rate', 'gdp', 'hh_pop', 'population',
       'total_violent_crimes', 'total_robbery_crimes',
'total_assault_crimes',
       'total_property_crimes', 'total_burglary_crimes',
       'total_larcency_crimes', 'total_vehicle_crimes', 'budget',
'overtime']
train_df = train_df[new_order_col]

plt.figure(figsize=(14, 12))

# Calculate the correlation matrix
corr = train_df.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Generate a custom diverging colormap with light red and blue
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
```

```
ax = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0,
                 square=True, linewidths=.5, cbar_kws={"shrink": .5},
annot=True, fmt=".2f", annot_kws={"size": 12})

# Set the title with increased font size
ax.set_title('Feature Matrix Correlation', fontsize=18)

# Rotate the x-axis labels for better visibility and increase font
size
plt.xticks(rotation=45, ha='right', fontsize=14)

# Rotate the y-axis labels for better visibility and increase font
size
plt.yticks(rotation=0, fontsize=14)

plt.show()
```

```
Index(['Year', 'unemp_rate', 'gdp', 'hh_pop', 'population',
       'total_violent_crimes', 'total_robbery_crimes',
'total_assault_crimes',
       'total_property_crimes', 'total_burglary_crimes',
       'total_larcency_crimes', 'total_vehicle_crimes', 'overtime',
'budget'],
      dtype='object')
```

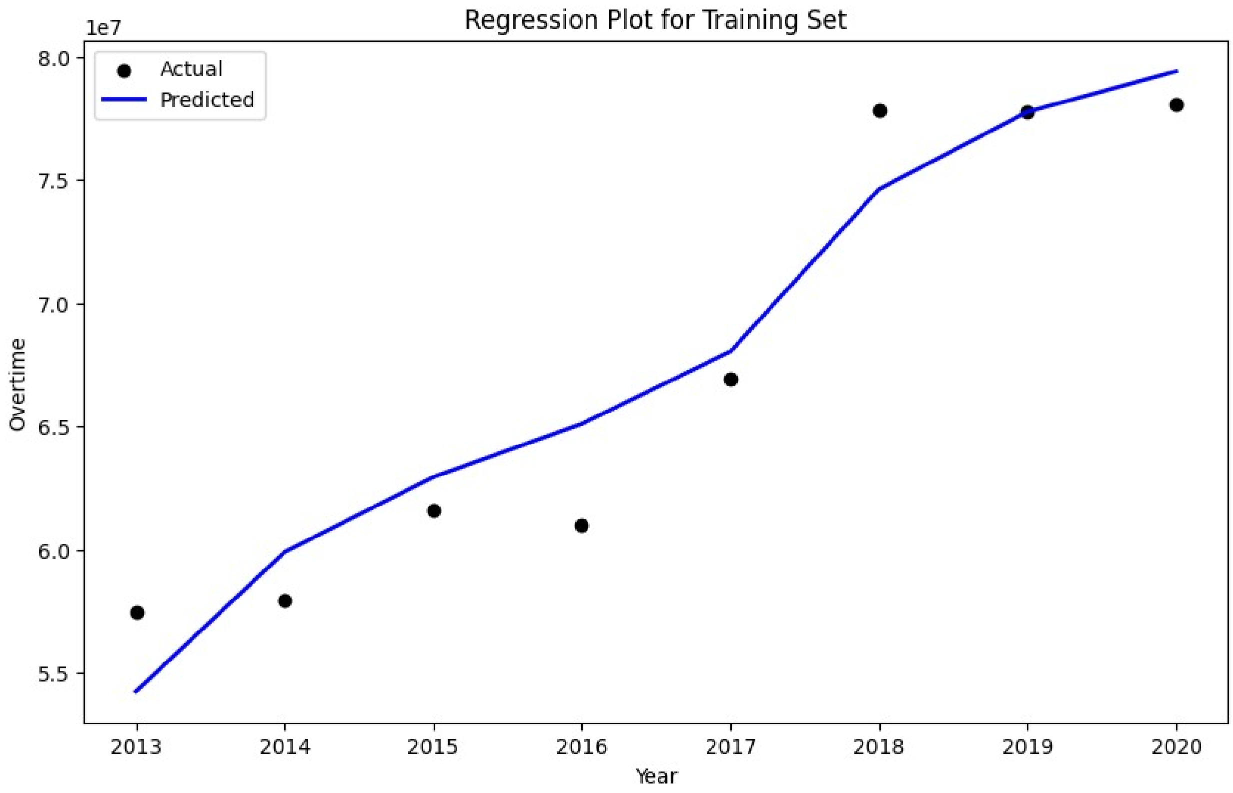


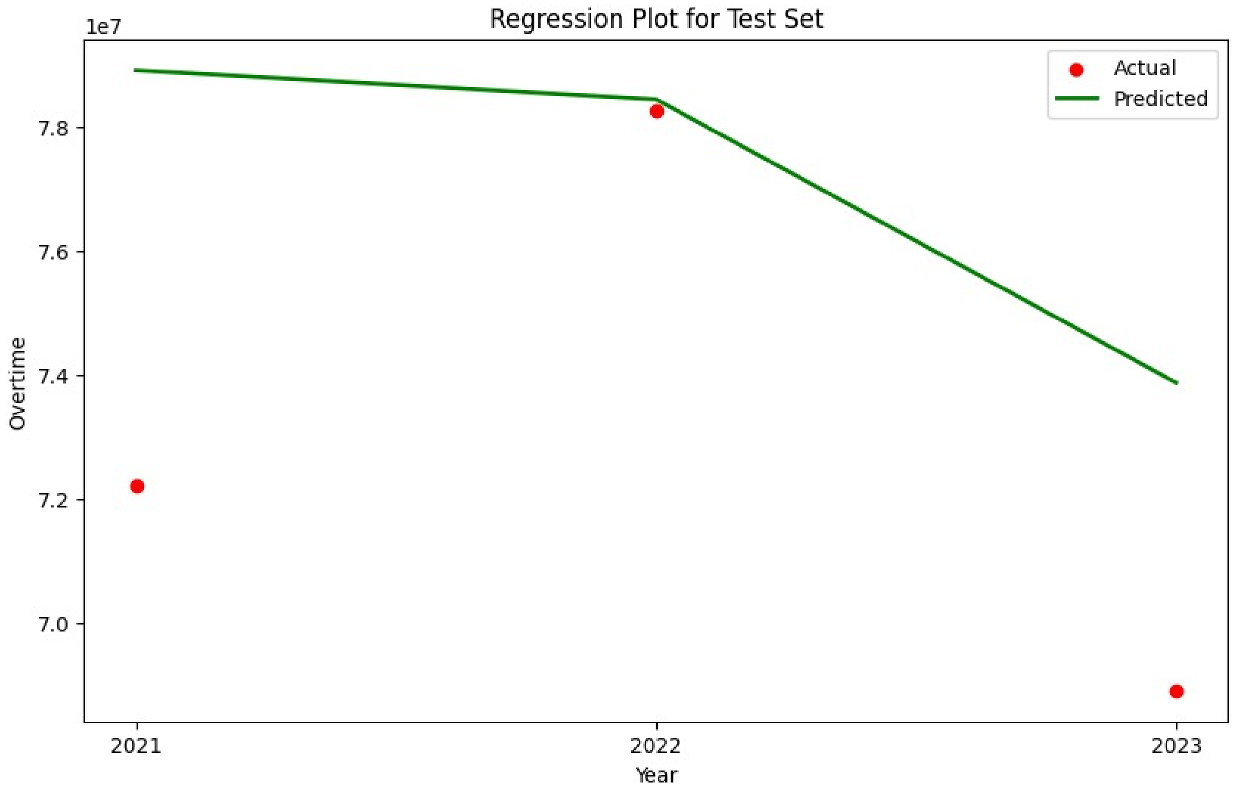
```
from sklearn.calibration import LinearSVC
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_percentage_error,
mean_squared_error
from sklearn.svm import LinearSVR
from xgboost import XGBRegressor


train_df.drop(['total_violent_crimes', 'total_robbery_crimes',
'total_assault_crimes','total_property_crimes','total_burglary_crimes'
,'total_larcency_crimes','total_vehicle_crimes', 'hh_pop'], axis=1,
inplace=True)

new_order_col = ['Year', 'unemp_rate', 'gdp', 'population', 'budget',
'overtime']
```

```
train_df = train_df[new_order_col]

#print(train_df)
y = train_df['overtime']

# X_train, X_test, Y_train, Y_test = train_test_split(
#         train_df.drop(['overtime'], axis=1),
#         y,
#         test_size=1/4.0,
#     )

X_train = train_df[:8]
X_test = train_df[8:]
Y_train = X_train['overtime']
Y_test = X_test['overtime']

X_train.drop(['overtime'], axis=1, inplace=True)
X_test.drop(['overtime'], axis=1, inplace=True)

# Step 3: Model selection
model = LinearSVR()
#model = RandomForestRegressor()


# Step 4: Tran the model
model.fit(X_train, Y_train)


# Step 5: Evaluate the model
Y_pred = model.predict(X_test)

Y_pred_train = model.predict(X_train)



print(f"Root Mean Squared Error Test:
{round(mean_squared_error(Y_test, Y_pred)**(1/2), 2)}")

print(f"Root Mean Squared Error Train:
{round(mean_squared_error(Y_train, Y_pred_train)**(1/2), 2)}")


print(f"Mean Absolute Percentage Error (MAPE) Test:
{round(mean_absolute_percentage_error(Y_test, Y_pred), 2)}%")

print(f"Mean Absolute Percentage Error (MAPE) Train:
{round(mean_absolute_percentage_error(Y_train, Y_pred_train), 2)}%")


# Plotting the regression plot for the training set with 'Year' as the
```

```
x-axis
plt.figure(figsize=(10, 6))
plt.scatter(X_train['Year'], Y_train, color='black', label='Actual')
plt.plot(X_train['Year'], Y_pred_train, color='blue',
label='Predicted', linewidth=2)
plt.title('Regression Plot for Training Set')
plt.xlabel('Year')
plt.ylabel('Overtime')
plt.legend()
plt.show()

# Plotting the regression plot for the test set with 'Year' as the x-
axis
plt.figure(figsize=(10, 6))
plt.scatter(X_test['Year'], Y_test, color='red', label='Actual')
plt.plot(X_test['Year'], Y_pred, color='green', label='Predicted',
linewidth=2)
plt.title('Regression Plot for Test Set')
plt.xlabel('Year')
plt.xticks([2021, 2022,2023])
plt.ylabel('Overtime')
plt.legend()
plt.show()
```

```
<ipython-input-173-ef41a1fd0692>:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X_train.drop(['overtime'], axis=1, inplace=True)
<ipython-input-173-ef41a1fd0692>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X_test.drop(['overtime'], axis=1, inplace=True)
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  warnings.warn(

Root Mean Squared Error Test: 4810203.02
Root Mean Squared Error Train: 2410246.88
Mean Absolute Percentage Error (MAPE) Test: 0.06%
Mean Absolute Percentage Error (MAPE) Train: 0.03%
```


Regression Plot for Training Set
1e7
8.0
7.5
7.0
6.5
6.0
5.5
Overtime
Actual
Predicted
2013
2014
2015
2016
2017
2018
2019
2020
Year


Regression Plot for Test Set
1e7
7.8
7.6
7.4
7.2
7.0
Overtime
Actual
Predicted
2021
2022
2023
Year

```
final_prediciton_2024 = model.predict(prediction_df)

print()

print("Our final prediction for Total Overtime Payment", f"$
{round(final_prediciton_2024[0], 2)}")
print()

print(f"Root Mean Squared Error Test:
{round(mean_squared_error(Y_test, Y_pred)**(1/2), 2)}")

print(f"Root Mean Squared Error Train:
{round(mean_squared_error(Y_train, Y_pred_train)**(1/2), 2)}")
print()
print(f"Mean Absolute Percentage Error (MAPE) Test:
{mean_absolute_percentage_error(Y_test, Y_pred)}%")

print(f"Mean Absolute Percentage Error (MAPE) Train:
{mean_absolute_percentage_error(Y_train, Y_pred_train)}%")
```

```
Our final prediction for Total Overtime Payment $75564625.34

Root Mean Squared Error Test: 4810203.02
Root Mean Squared Error Train: 2410246.88

Mean Absolute Percentage Error (MAPE) Test: 0.05564591171261419%
Mean Absolute Percentage Error (MAPE) Train: 0.03181940243074722%
```