# Import the pandas library as pd

import pandas as pd

# Read 'police.csv' into a DataFrame named ri

ri = pd.read\_csv('police.csv')

# Examine the head of the DataFrame

print(ri.head(5))

# Count the number of missing values in each column

print(ri.isnull().sum())

==

# Examine the shape of the DataFrame

print(ri.shape)

# Drop the 'county\_name' and 'state' columns

ri.drop(['county\_name', 'state'], axis='columns', inplace=True)

# Examine the shape of the DataFrame (again)

print(ri.shape)

==

# Count the number of missing values in each column

print(ri.isnull().sum())

# Drop all rows that are missing 'driver\_gender'

ri.dropna(subset=['driver\_gender'], inplace=True)

# Count the number of missing values in each column (again)

print(ri.isnull().sum())

# Examine the shape of the DataFrame

print(ri.shape)

==

# Examine the head of the 'is\_arrested' column

print(ri.is\_arrested.head())

# Change the data type of 'is\_arrested' to 'bool'

ri['is\_arrested'] = ri.is\_arrested.astype('bool')

# Check the data type of 'is\_arrested'

print(ri.is\_arrested.dtype)

==

# Concatenate 'stop\_date' and 'stop\_time' (separated by a space)

combined = ri.stop\_date.str.cat(ri.stop\_time, sep=' ')

# Convert 'combined' to datetime format

ri['stop\_datetime'] = pd.to\_datetime(combined)

# Examine the data types of the DataFrame

print(ri.dtypes)

==

# Set 'stop\_datetime' as the index

ri.set\_index('stop\_datetime', inplace=True)

# Examine the index

print(ri.index)

# Examine the columns

print(ri.columns)

==

# Count the unique values in 'violation'

print(ri.violation.value\_counts())

# Express the counts as proportions

print(ri.violation.value\_counts(normalize=True))

==

# Create a DataFrame of female drivers

female = ri[ri.driver\_gender=='F']

# Create a DataFrame of male drivers

male = ri[ri.driver\_gender=='M']

# Compute the violations by female drivers (as proportions)

print(female.violation.value\_counts(normalize=True))

# Compute the violations by male drivers (as proportions)

print(male.violation.value\_counts(normalize=True))

==

# Create a DataFrame of female drivers stopped for speeding

female\_and\_speeding = ri[(ri.driver\_gender=='F') & (ri.violation == 'Speeding')]

# Create a DataFrame of male drivers stopped for speeding

male\_and\_speeding = ri[(ri.driver\_gender=='M') & (ri.violation == 'Speeding')]

# Compute the stop outcomes for female drivers (as proportions)

print(female\_and\_speeding.stop\_outcome.value\_counts(normalize=True))

# Compute the stop outcomes for male drivers (as proportions)

print(male\_and\_speeding.stop\_outcome.value\_counts(normalize=True))

==

# Check the data type of 'search\_conducted'

print(ri.search\_conducted.dtype)

# Calculate the search rate by counting the values

print(ri.search\_conducted.value\_counts(normalize=True))

# Calculate the search rate by taking the mean

print(ri.search\_conducted.mean())

==

# Calculate the search rate for female drivers

print(ri[ri.driver\_gender=='F'].search\_conducted.mean())

# Calculate the search rate for male drivers

print(ri[ri.driver\_gender=='M'].search\_conducted.mean())

# Calculate the search rate for both groups simultaneously

print(ri.groupby('driver\_gender').search\_conducted.mean())

==

# Calculate the search rate for each combination of gender and violation

print(ri.groupby(['driver\_gender', 'violation']).search\_conducted.mean())

# Reverse the ordering to group by violation before gender

print(ri.groupby(['violation', 'driver\_gender']).search\_conducted.mean())

==

# Count the 'search\_type' values

print(ri.search\_type.value\_counts())

# Check if 'search\_type' contains the string 'Protective Frisk'

ri['frisk'] = ri.search\_type.str.contains('Protective Frisk', na=False)

# Check the data type of 'frisk'

print(ri.frisk.dtype)

# Take the sum of 'frisk'

print(ri.frisk.sum())

==

# Create a DataFrame of stops in which a search was conducted

searched = ri[ri.search\_conducted == True]

# Calculate the overall frisk rate by taking the mean of 'frisk'

print(searched.frisk.mean())

# Calculate the frisk rate for each gender

print(searched.groupby('driver\_gender').frisk.mean())

==

# Calculate the overall arrest rate

print(ri.is\_arrested.mean())

# Calculate the hourly arrest rate

print(ri.is\_arrested.groupby(ri.index.hour).mean())

# Save the hourly arrest rate

hourly\_arrest\_rate = ri.is\_arrested.groupby(ri.index.hour).mean()

==

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Create a line plot of 'hourly\_arrest\_rate'

hourly\_arrest\_rate.plot()

# Add the xlabel, ylabel, and title

plt.xlabel('Hour')

plt.ylabel('Arrest Rate')

plt.title('Arrest Rate by Time of Day')

# Display the plot

plt.show()

==

# Calculate the annual rate of drug-related stops

print(ri.drugs\_related\_stop.resample('A').mean())

# Save the annual rate of drug-related stops

annual\_drug\_rate = ri.drugs\_related\_stop.resample('A').mean()

# Create a line plot of 'annual\_drug\_rate'

annual\_drug\_rate.plot()

# Display the plot

plt.show()

==

# Calculate and save the annual search rate

annual\_search\_rate = ri.search\_conducted.resample('A').mean()

# Concatenate 'annual\_drug\_rate' and 'annual\_search\_rate'

annual = pd.concat([annual\_drug\_rate, annual\_search\_rate], axis='columns')

# Create subplots from 'annual'

annual.plot(subplots=True)

# Display the subplots

plt.show()

==

# Create a frequency table of districts and violations

print(pd.crosstab(ri.district, ri.violation))

# Save the frequency table as 'all\_zones'

all\_zones = pd.crosstab(ri.district, ri.violation)

# Select rows 'Zone K1' through 'Zone K3'

print(all\_zones.loc['Zone K1' : 'Zone K3'])

# Save the smaller table as 'k\_zones'

k\_zones = all\_zones.loc['Zone K1' : 'Zone K3']

==

# Create a bar plot of 'k\_zones'

k\_zones.plot(kind='bar')

# Display the plot

plt.show()

# Create a stacked bar plot of 'k\_zones'

k\_zones.plot(kind='bar', stacked=True)

# Display the plot

plt.show()

==

# Print the unique values in 'stop\_duration'

print(ri.stop\_duration.unique())

# Create a dictionary that maps strings to integers

mapping = {'0-15 Min':8, '16-30 Min':23, '30+ Min':45}

# Convert the 'stop\_duration' strings to integers using the 'mapping'

ri['stop\_minutes'] = ri.stop\_duration.map(mapping)

# Print the unique values in 'stop\_minutes'

print(ri.stop\_minutes.unique())

==

# Calculate the mean 'stop\_minutes' for each value in 'violation\_raw'

print(ri.stop\_minutes.groupby(ri.violation\_raw).mean())

# Save the resulting Series as 'stop\_length'

stop\_length = ri.stop\_minutes.groupby(ri.violation\_raw).mean()

# Sort 'stop\_length' by its values and create a horizontal bar plot

stop\_length.sort\_values().plot(kind='barh')

# Display the plot

plt.show()

==

# Read 'weather.csv' into a DataFrame named 'weather'

weather = pd.read\_csv('weather.csv')

# Describe the temperature columns

print(weather[['TMIN', 'TAVG', 'TMAX']].describe())

# Create a box plot of the temperature columns

weather[['TMIN', 'TAVG', 'TMAX']].plot(kind='box')

# Display the plot

plt.show()

==

# Create a 'TDIFF' column that represents temperature difference

weather['TDIFF'] = weather.TMAX - weather.TMIN

# Describe the 'TDIFF' column

print(weather.TDIFF.describe())

# Create a histogram with 20 bins to visualize 'TDIFF'

weather.TDIFF.plot(kind='hist', bins=20)

# Display the plot

plt.show()

==

# Copy 'WT01' through 'WT22' to a new DataFrame

WT = weather.loc[:, 'WT01':'WT22']

# Calculate the sum of each row in 'WT'

weather['bad\_conditions'] = WT.sum(axis='columns')

# Replace missing values in 'bad\_conditions' with '0'

weather['bad\_conditions'] = weather.bad\_conditions.fillna(0).astype('int')

# Create a histogram to visualize 'bad\_conditions'

weather.bad\_conditions.plot(kind='hist')

# Display the plot

plt.show()

==

# Count the unique values in 'bad\_conditions' and sort the index

print(weather.bad\_conditions.value\_counts().sort\_index())

# Create a dictionary that maps integers to strings

mapping = {0:'good', 1:'bad', 2:'bad', 3:'bad', 4:'bad', 5:'worse', 6:'worse', 7:'worse', 8:'worse', 9:'worse'}

# Convert the 'bad\_conditions' integers to strings using the 'mapping'

weather['rating'] = weather.bad\_conditions.map(mapping)

# Count the unique values in 'rating'

print(weather.rating.value\_counts())

==

# Create a list of weather ratings in logical order

cats = ['good', 'bad', 'worse']

# Change the data type of 'rating' to category

weather['rating'] = weather.rating.astype('category', ordered=True, categories=cats)

# Examine the head of 'rating'

print(weather.rating.head())

==

# Reset the index of 'ri'

ri.reset\_index(inplace=True)

# Examine the head of 'ri'

print(ri.head())

# Create a DataFrame from the 'DATE' and 'rating' columns

weather\_rating = weather[['DATE', 'rating']]

# Examine the head of 'weather\_rating'

print(weather\_rating.head())

==

# Examine the shape of 'ri'

print(ri.shape)

# Merge 'ri' and 'weather\_rating' using a left join

ri\_weather = pd.merge(left=ri, right=weather\_rating, left\_on='stop\_date', right\_on='DATE', how='left')

# Examine the shape of 'ri\_weather'

print(ri\_weather.shape)

# Set 'stop\_datetime' as the index of 'ri\_weather'

ri\_weather.set\_index('stop\_datetime', inplace=True)

==

# Calculate the overall arrest rate

print(ri\_weather.is\_arrested.mean())

# Calculate the arrest rate for each 'rating'

print(ri\_weather.groupby('rating').is\_arrested.mean())

# Calculate the arrest rate for each 'violation' and 'rating'

print(ri\_weather.groupby(['violation', 'rating']).is\_arrested.mean())

==

# Save the output of the groupby operation from the last exercise

arrest\_rate = ri\_weather.groupby(['violation', 'rating']).is\_arrested.mean()

# Print the 'arrest\_rate' Series

print(arrest\_rate)

# Print the arrest rate for moving violations in bad weather

print(arrest\_rate.loc['Moving violation','bad'])

# Print the arrest rates for speeding violations in all three weather conditions

print(arrest\_rate.loc['Speeding'])

==

# Unstack the 'arrest\_rate' Series into a DataFrame

print(arrest\_rate.unstack())

# Create the same DataFrame using a pivot table

print(ri\_weather.pivot\_table(index='violation', columns='driver\_gender', values='is\_arrested'))

==