All the Python code described below is available on Github and pasted below: https://github.com/tashafiq/CAJ-Investintech.

While researching my pitch, I noticed that previous coverage of the Landlord Tenant Board's wait times had focused on averages across application types, despite the LTB's open data breaking down wait times more granularly. I hypothesized that tenant wait times would be longer than those for landlords.

I entered and cleaned the data manually since I believed it would be faster than writing a Python script, considering there were multiple .csv structures. Given the large volume of application types, I chose only the two most-common types for analysis. I created a Flourish visual similar to the final result and found that tenants faced the greatest wait times.

We then had to decide how to describe these findings simply. I didn't want to average the tenant and landlord times since the data for individual application types was an average anyways; further averaging could introduce errors. We settled on using the maximum discrepancy from the most recent data: "in the last nine months of available data, average wait times for tenant cases were up to seven months longer than for those brought by landlords."

That proved to be the easy part. Sources said the LTB hadn't invested sufficient resources to address the backlog, but how could we put a number to that? First, I noticed that many online hearings were being scheduled each week. When the LTB didn't provide weekly numbers for these hearings, I realized that an activist-run Twitter account which was publicly sharing the links could provide the data. This was around the time when Elon Musk declared the Twitter API would be a paid-for service, so I decided to scrape the activist's inbox to get the data instead.

After meeting with the activist and obtaining his IMAP credentials, I modified some open-source Gmail scraping code—my first time using the 'email' and 'imaplib' packages—to count the Zoom links in the schedules received from the LTB. Each link was preceded by the words "Zoom link:" so I summed over that and put the count at the bottom of each email in a .txt file so I could double check manually. I filled in some missed weeks with links posted by a Facebook group, Landlords and Tenants of Ontario.

Next, I asked the LTB for their staffing numbers. They said the number of adjudicators fluctuated with mid-year hirings and could only provide a snapshot in time—clearly unreliable data. I noticed each adjudicator's contract term was listed in an annual report, so I wrote an Excel function (after struggling with Python) to validate whether a given week fell between their start and end data. Finally, we had a clear metric to back up my sources: the number of hearings per week skyrocketed while hiring stayed steady. More workload, same resources. A rushed process.

Some more experience would have sped this process up. I tried to use the 'datetime' Python package but found the date syntax confusing, so switched to an elaborate Excel system of linked sheets. The adjudicator contracts were in one sheet, the desired weeks in another, and a third sheet checked for matches between them. I didn’t know how to use pivot tables, so this seemed like the best approach.

In further investigations, I plan to use pivot tables for any complex Excel analysis. I’ve also recently learned that there are OSINT social media scraping tools out there which don’t require official API access. Any of these might have made it easier to scrape the Twitter and Facebook posts I used.

```
"""
Extract selected mails from your gmail account

Starter code:
https://github.com/bnsreenu/python_for_microscopists/tree/master/AMT_02_extra
ct_gmails_from_a_user

1. Make sure you enable IMAP in your gmail settings
(Log on to your Gmail account and go to Settings, See All Settings, and
select
 Forwarding and POP/IMAP tab. In the "IMAP access" section, select Enable
IMAP.)

2. If you have 2-factor authentication, gmail requires you to create an
application
specific password that you need to use.
Go to your Google account settings and click on 'Security'.
Scroll down to App Passwords under 2 step verification.
Select Mail under Select App. and Other under Select Device. (Give a name,
e.g., python)
The system gives you a password that you need to use to authenticate from
python.

"""

# Importing libraries
import imaplib
import email as email
import yaml as yaml  # To load saved login credentials from a yaml file #yml-
1.3#

with open("credentials.yml") as f:
    content = f.read()

# from credentials.yml import user name and password
my_credentials = yaml.load(content, Loader=yaml.FullLoader)

# Load the user name and passwd from yaml file
user, password = my_credentials["user"], my_credentials["password"]

# URL for IMAP connection
imap_url = 'imap.gmail.com'

# Connection with GMAIL using SSL
my_mail = imaplib.IMAP4_SSL(imap_url)
```

```
# Log in using your credentials
my_mail.login(user, password)

# Select the Inbox to fetch messages
my_mail.select('Inbox')

# Define Key and Value for email search
# For other keys (criteria):
https://gist.github.com/martinrusev/6121028#file-imap-search
key = 'FROM'
value = 'LTB@ontario.ca'
_, data = my_mail.search(None, key, value)  # Search for emails with specific
key and value

mail_id_list = data[0].split()  # IDs of all emails that we want to fetch

msgs = []  # empty list to capture all messages
# Iterate through messages and extract data into the msgs list
for num in mail_id_list:
    typ, data = my_mail.fetch(num, '(RFC822)')  # RFC822 returns whole
message (BODY fetches just body)
    msgs.append(data)

# Now we have all messages, but with a lot of details
# Let us extract the right text and print on the screen

# In a multipart e-mail, email.message.Message.get_payload() returns a
# list with one item for each part. The easiest way is to walk the message
# and get the payload on each part:
# https://stackoverflow.com/questions/1463074/how-can-i-get-an-email-
messages-text-content-using-python

# NOTE that a Message object consists of headers and payloads.

#Files to write to
file = open('data.txt', 'a')

for msg in msgs[::-1]:
    for response_part in msg:
        if type(response_part) is tuple:
            my_msg = email.message_from_bytes((response_part[1]))
            file.write("__________________________________________\n")
            file.write("subj:+" +str(my_msg['subject']) +'\n')
            file.write("from:+" +str(my_msg['from']) +'\n')
            file.write("date:+" +str(my_msg['date']) +'\n')
            for part in my_msg.walk():
                if part.get_content_type() == 'text/plain':
                    #We'll use the base64 package's decoder:
https://stackoverflow.com/questions/38970760/how-to-decode-a-mime-part-of-a-
message-and-get-a-unicode-string-in-python-2
                    bytes = part.get_payload(decode=True)
                    charset = part.get_content_charset('iso-8859-1')
                    chars = bytes.decode(charset, 'replace')
                    #Now we have the email body as a string, from which we
can pull our relevant data
                    body = chars.split()
                    hearing_count = 0
                    urgent_count = 0
                    adjourned_count = 0
                    for i in range(0,len(body)-1):
                        if body[i] == 'Zoom' and body[i+1] == 'Link:':
                            hearing_count += 1
                        if body[i] == 'Urgent' and body[i+1] == '-':
                            urgent_count += 1
                        if body[i] == 'Adjourned' and body[i+1] == 'Block':
                            adjourned_count = 0
                    " ".join(body)
                    file.write(str(body) + '\n')
                    file.write('\n')
                    file.write("total hearings this week:" +
str(hearing_count) + '\n')
                    file.write("urgent hearings:" + str(urgent_count) + '\n')
                    file.write("adjourned hearings:" + str(adjourned_count) +
'\n')
```