ISA 444: Business Forecasting

02: A Python Primer

Fadel M. Megahed, PhD

Professor
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
Automated Scheduler for Office Hours

Spring 2025

Quick Refresher of Last Class

✅ Describe the role of forecasting in business.

✅ Describe the key components of a **time-series** (**trend**, **seasonality**, **multiple seasonality**, and **cycles**).

✅ Explain the concept of data-generating process (DGP)

✅ Discuss limits of forecasting

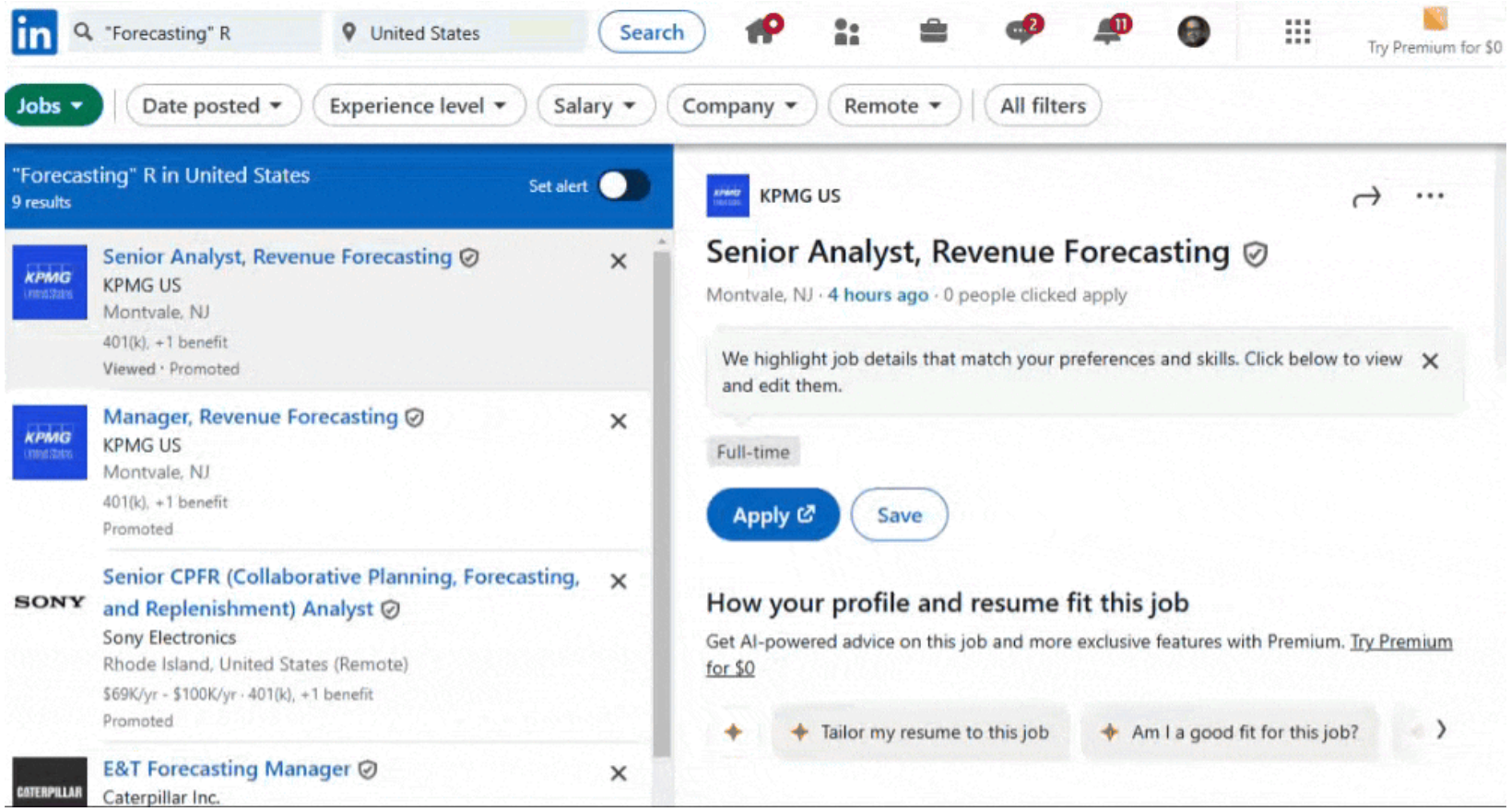
✅ Understand key forecasting terminology

Learning Objectives for Today's Class

- Setting up Python (Colab, Anaconda, and/or VS Code)
- Practice basic data reading (from CSVs and the Web)
- Use panda's datatime, indexing, and slicing capabilities
- (Optional) Discuss generative AI usage in Google Colab

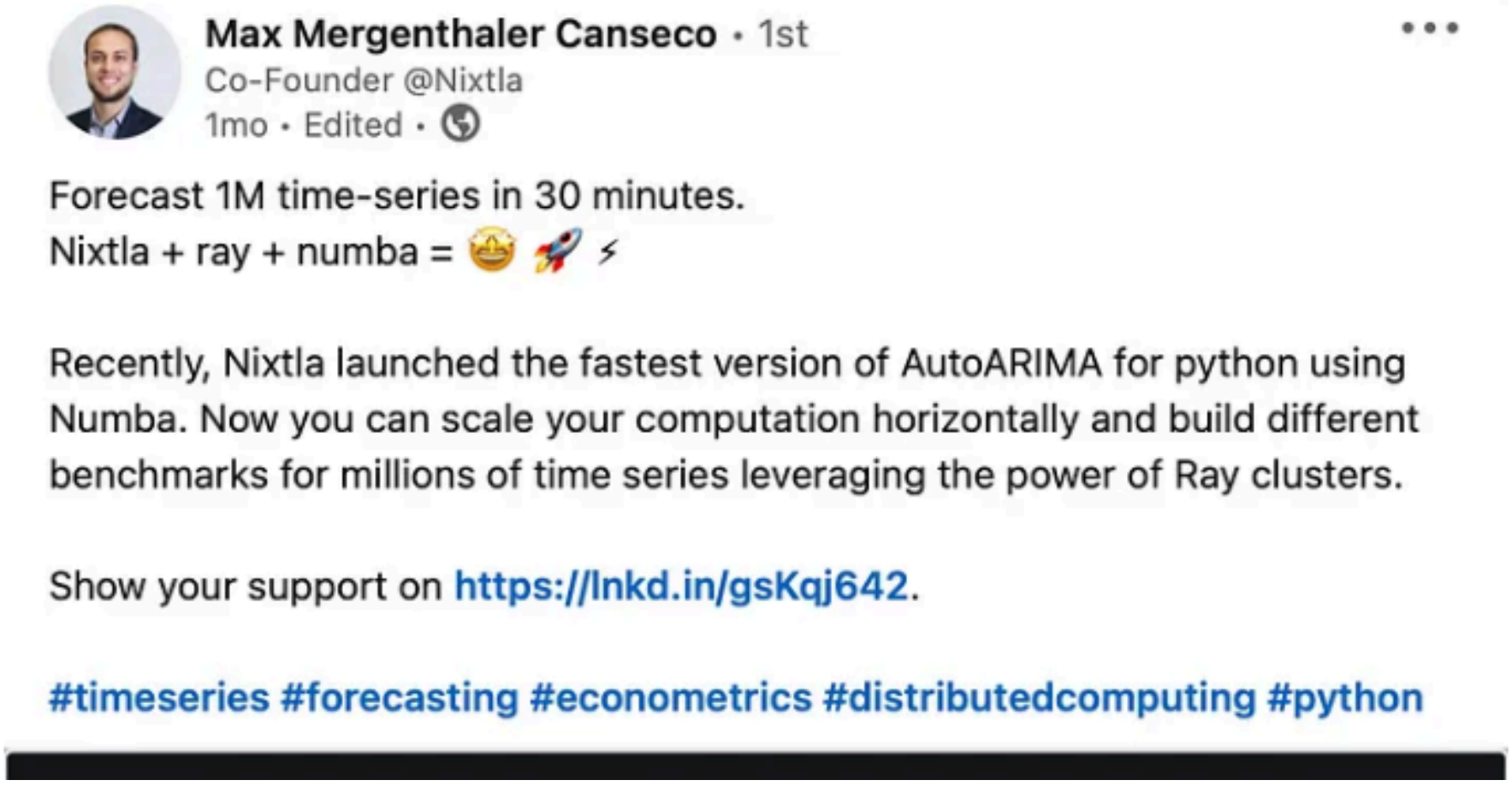
Setting Up Python

Why Python? Insights from a LinkedIn Job Search



Note: The above image is a screenshot from LinkedIn's job search results for "forecasting" jobs from **January 29, 2025**. While LinkedIn may not be the best source for job market trends, it does provide a snapshot of the demand for forecasting skills in the job market. **The results are clearly in favor of Python (112 jobs) over R (9 jobs)**. You can replicate the search by clicking on the following links: R Forecasting Jobs and Python Forecasting Jobs.

Why Python? Pedagogical Reasons



Max Mergenthaler Canseco · 1st
Co-Founder @Nixtla
1mo · Edited ·

Forecast 1M time-series in 30 minutes.
Nixtla + ray + numba = 🤩 🚀 ⚡

Recently, Nixtla launched the fastest version of AutoARIMA for python using Numba. Now you can scale your computation horizontally and build different benchmarks for millions of time series leveraging the power of Ray clusters.

Show your support on https://lnkd.in/gsKqj642.

#timeseries #forecasting #econometrics #distributedcomputing #python

Image Source: An opinionated article by Valeriy Manokhin Python vs R for time-series forecasting on Medium. While the article is opinionated, it does provide a good overview of the strengths and weaknesses of Python and R for time-series forecasting.

Why Python? Pedagogical Reasons

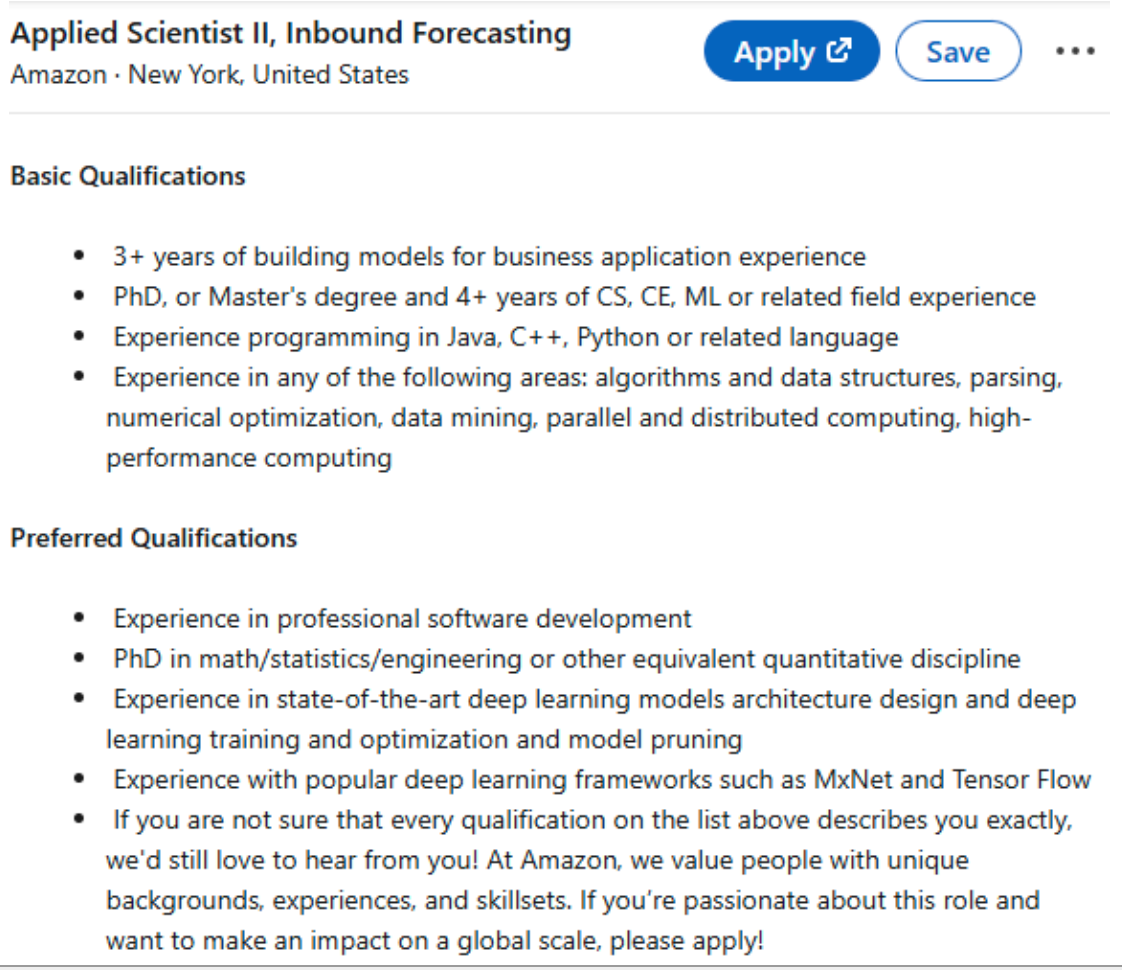
New cutting edge time series packages (especially machine learning) are almost exclusively released in Python. Most of modern time series forecasting books are being published with Python code. Python moved leaps and bounds during the last 5+ years in terms of developments for time series and forecasting.

R contains a lot of time-series functionality but is primarily focused on classical forecasting models. On the other side, most of the innovation for time series has predominantly been happening in the machine learning space.

Machine learning algorithms based on boosted trees such as LightGBM dominated most Kaggle forecasting competitions for over a decade.

Image Source: An opinionated article by Valeriy Manokhin Python vs R for time-series forecasting on Medium. While the article is opinionated, it does provide a good overview of the strengths and weaknesses of Python and R for time-series forecasting.

Why Python? Pedagogical Reasons



Applied Scientist II, Inbound Forecasting
Amazon · New York, United States

Apply | Save | •••

Basic Qualifications

- 3+ years of building models for business application experience
- PhD, or Master's degree and 4+ years of CS, CE, ML or related field experience
- Experience programming in Java, C++, Python or related language
- Experience in any of the following areas: algorithms and data structures, parsing, numerical optimization, data mining, parallel and distributed computing, high-performance computing

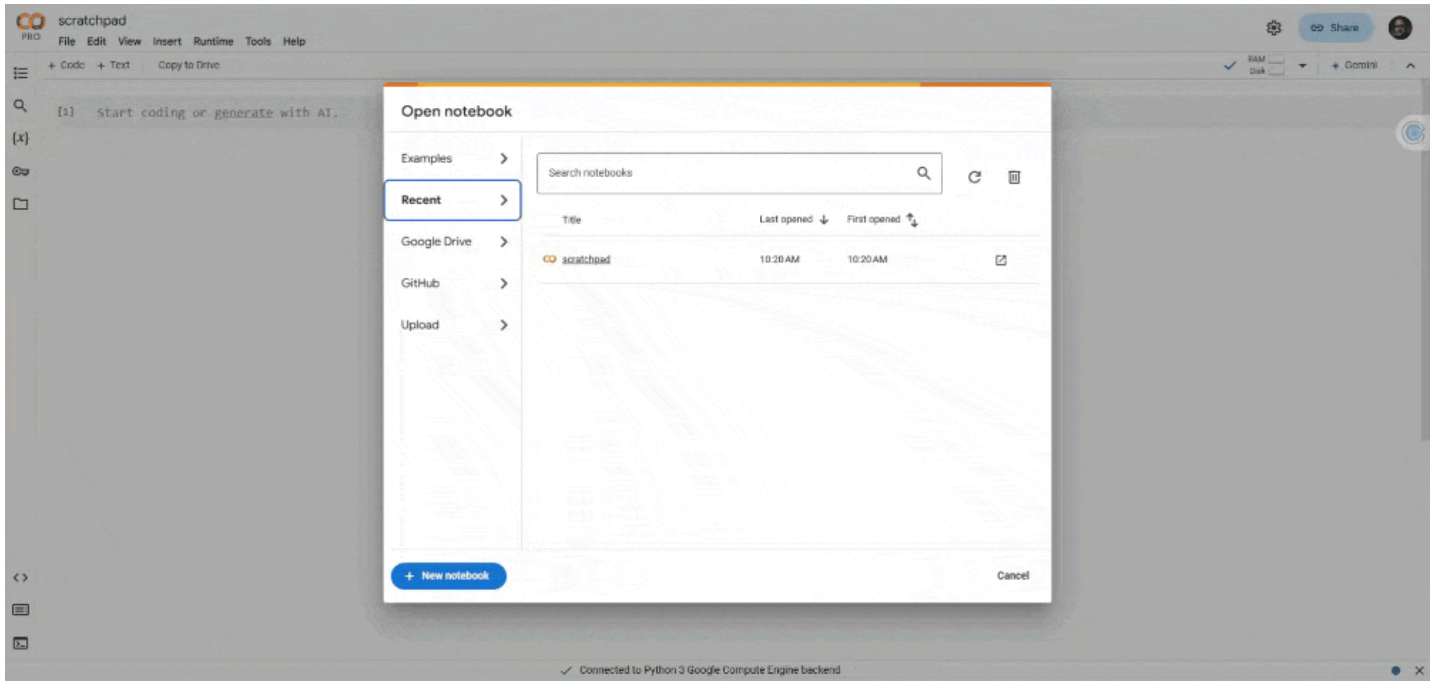
Preferred Qualifications

- Experience in professional software development
- PhD in math/statistics/engineering or other equivalent quantitative discipline
- Experience in state-of-the-art deep learning models architecture design and deep learning training and optimization and model pruning
- Experience with popular deep learning frameworks such as MxNet and Tensor Flow
- If you are not sure that every qualification on the list above describes you exactly, we'd still love to hear from you! At Amazon, we value people with unique backgrounds, experiences, and skillsets. If you're passionate about this role and want to make an impact on a global scale, please apply!

Image Source: Amazon's preferred qualifications for a mid-senior PhD preferred Applied Scientist II, Inbound Forecasting confirming the importance of Python and the transition to ML-DL in forecasting. So to graduate **#BeyondReady graduates, you need to be introduced to the state-of-the-art**, which is readily available in Python but not in R.

Python via Google Colab: Preferred Choice for our Class

- Google Colab is a free cloud service for running Python in a Jupyter notebook environment.
- It offers free access to GPUs and TPUs, useful in running deep learning models.
- To get started, you can create a new notebook by clicking on **New Notebook**.



Class Activity: Navigating Colab + Your First Python

02:30

Create a new notebook in Google Colab, **run the following code**, and observe the output.

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

healthexp = sns.load_dataset('healthexp') # Load the healthexp dataset

# Use the relplot function to plot
ax = sns.relplot(
  data=healthexp, kind='line',
  x='Year', y='Life_Expectancy', hue='Country',  style='Country', palette='Paired'
  )

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

In **separate cells**, try the following and **observe the output**:

- Now, remove `kind='line'` from the code and rerun the cell.
- Now, add the following function in a new line before the `plt.show()` function:
  `sns.move_legend(ax, loc = "lower center", bbox_to_anchor=(0.5, 0.85), ncol=3)`

Python via Anaconda: For Local Use (If you Want)

- Anaconda is a free and open-source distribution of Python for scientific computing.
- It comes with a package manager (`conda`) and a collection of pre-installed packages.
- To get started, you can **download Anaconda** from here.
- You can **create a new environment** by running the following command in the **terminal**:

```
conda create --name isa444 python=3.12
```

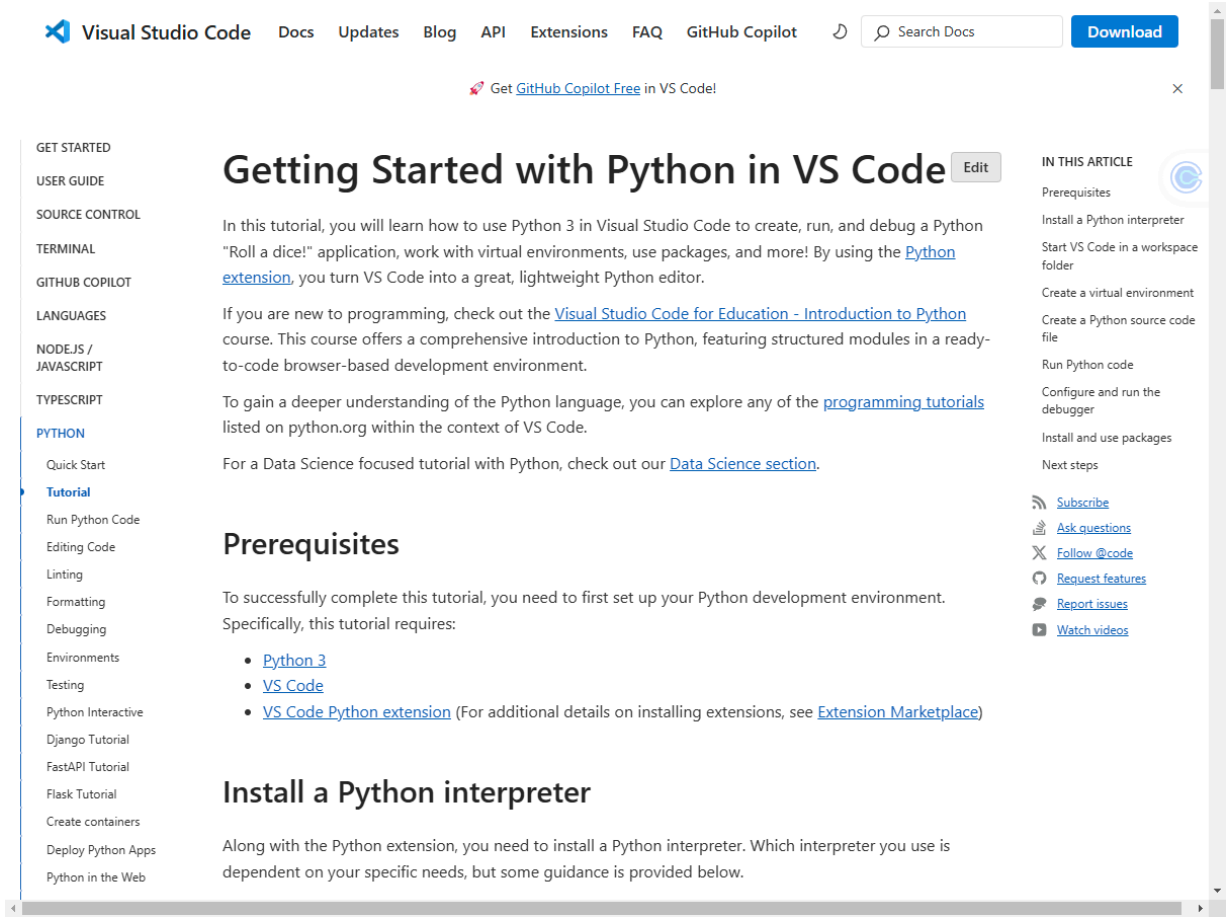
- To **activate** the environment, run the following command:

```
conda activate isa444
```

- Then, you can **install any required packages** (done once) by running:

```
conda install statsforecast mlforecast neuralforecast # and any other packages you need
```

Anaconda + VS Code: For Local Use (If you Want)

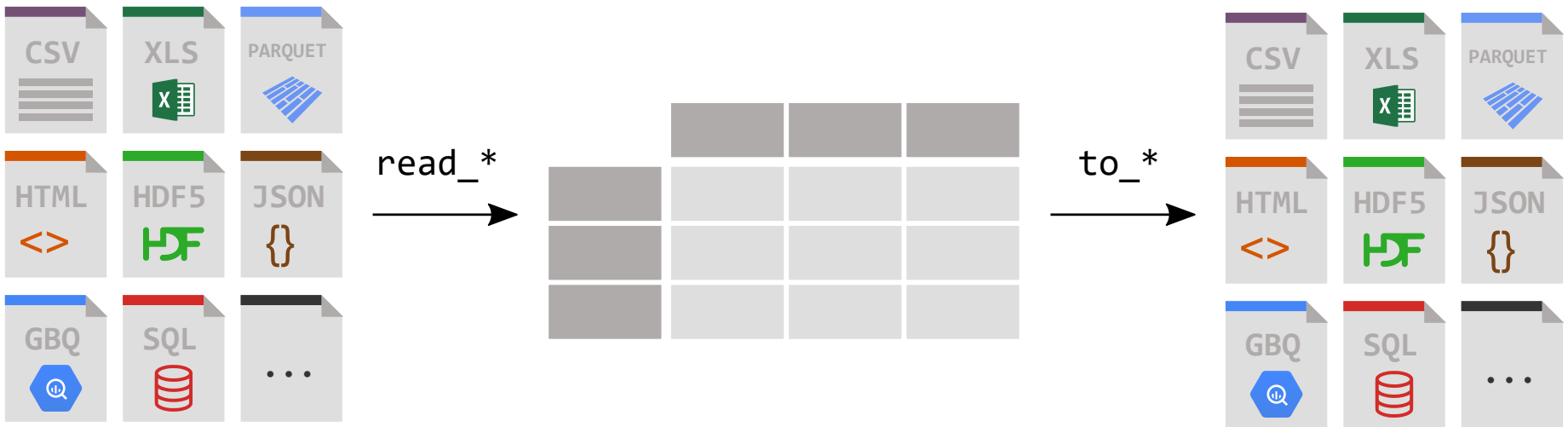


Please read through the Getting Started with Python in VS Code for more details.

Importing Data Using Pandas

What is Pandas?

- Along with polars, it is one of the most popular data analysis and manipulation libraries in Python, designed to make working with 'relational' or 'labeled' data both easy and intuitive.
- pandas supports the integration with many file formats or data sources out of the box (`csv`, `excel`, `sql`, `json`, parquet,...). Importing data from each of these data sources is provided by function with the prefix `read_*`. Similarly, the `to_*` methods are used to store data.



Source: The content and image are from the *Getting Started with Pandas* description in pandas.pydata.org. The image can be directly accessed here.

Things to Consider while Importing Data with Pandas

- **Check the file extension:** Your **choice of function** will depend on the file extension.
- **Check the file path:**
  - **Local file:**
    - If the file is in the **same directory** as your code, you can simply provide the **file name**.
    - **Otherwise**, you will need to provide the **path** to the file.
  - **Remote file:** You will need to provide the **URL to the file**.
- **Inspect the file:** to know which **default parameter values must be changed**. Things to consider:
  - Delimiter (in the case of .txt files)
  - Header (does it exist? if not, what should be used as column names?)
  - Index Column (if it exists, what is it? if not, what should be used as the index?)
  - Encoding (to be changed if file is not in English, or has names such as: François, José, Weiß, etc.)
  - Date Format (if the file has dates, what is the format?)
  - etc.

Importing Data Using Pandas: CSVs

Source: The content is from the `pandas.read_csv()` documentation available at pandas.pydata.org.

Importing Data w/ Pandas: CSVs [Example 1: Local]

```
import pandas as pd

# rRead the CSV file (the path will depend on where the file is located)
hotel = pd.read_csv('../../data/01_hotel_occupancy.csv')

# Display the first 5 rows of the dataset
hotel.head()
```

```
##          date  hotel_occupancy  month month_name
## 0  2022-01-01             3512      1    January
## 1  2022-02-01             3794      2   February
## 2  2022-03-01             4032      3      March
## 3  2022-04-01             4294      4      April
## 4  2022-05-01             4776      5        May
```

Importing Data w/ Pandas: CSVs [Example 2: Remote]

```
import pandas as pd

# Read the CSV file from the web
unrate = pd.read_csv('https://fred.stlouisfed.org/graph/fredgraph.csv?bgcolor=%23e1e9f0&char

# Display the first 5 rows of the dataset
unrate.head()
```

```
##   observation_date  UNRATE
## 0       1948-01-01     3.4
## 1       1948-02-01     3.8
## 2       1948-03-01     4.0
## 3       1948-04-01     3.9
## 4       1948-05-01     3.5
```

Live Demo: Importing Data Using Pandas on Colab

- **For "Local" Files**, Colab allows you to **upload these files** directly to the environment or **mount Google Drive** to access files. For this demo, I will walk you through both methods.
  - In the future, I will use the **"Upload" method** since it is quicker.
- **For "Remote" Files**, you can use the **URL directly** to read the file.
- Your environment will be **reset after ~ 12 hours** of inactivity. So, **save your work**. Note that you **must save your notebook** to Google Drive.
- Also, **remember to install any required packages** using `!pip install package_name` if they are not already installed.
- Once your environment is reset, you will need to **reinstall the packages** and re-upload (or remount) the data. **Why does Google do this?** Insert your explanation here.

Class Activity: Duke Energy Hourly Load Data

05:00

- **Download** the *Historical Hourly Loads by Class 2024 data* from here.
- **Inspect** the file to understand its structure; noting that I want you to read the **Total Usage** sheet. **Things to note when inspecting:**
  - What is the file extension? Edit me. Insert your answer here.
  - How many sheets are in the file? Edit me. Insert your answer here.
  - What is the name of the sheet you want to read? Edit me. Insert your answer here.
  - Which row should be used as the header? Edit me. Insert your answer here.
- **Upload** the file to your existing Google Colab notebook for today's class.
- **Read** the file using Pandas by consulting their API on Input/Output.

Using Pandas for Data Manipulation

Dateime Manipulation with Pandas

- Pandas has a powerful datetime manipulation capability.
- It allows you to **convert strings to datetime objects**, **extract parts of the date**, **perform arithmetic operations on dates**, and **filter data based on dates**.
- You can create a `Timestamp` object using the `pd.to_datetime()` function, or by using the `datetime` library in Python.

```
import pandas as pd

duke = (
  pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_2(
    sheet_name='Total Usage', header=1
    )
  .assign(
    datetime = lambda x:
      pd.to_datetime(x['REPORT DAY']) +
      pd.to_timedelta(x['HOUR ENDING']-1, unit='h')
      )
  [['datetime', 'TOTAL']]
  )

duke.head(n=1)
```

```
##     datetime       TOTAL
## 0 2023-01-01  1862129.01
```

Datetime Manipulation with Pandas: Without Chaining

- Pandas has a powerful datetime manipulation capability.
- It allows you to **convert strings to datetime objects**, **extract parts of the date**, **perform arithmetic operations on dates**, and **filter data based on dates**.
- You can create a `Timestamp` object using the `pd.to_datetime()` function, or by using the `datetime` library in Python.

```
import pandas as pd

df = pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_2(
    sheet_name='Total Usage', header=1
    )

df['datetime'] = pd.to_datetime(df['REPORT DAY']) +

df = df[['datetime', 'TOTAL']]

df.head(n=1)
```

```
##     datetime       TOTAL
## 0 2023-01-01  1862129.01
```

💡 Why Method Chaining? Detailed Interpretation

- The **parentheses** in this code enable **method chaining**, allowing multiple transformations to be applied to the *DataFrame* sequentially without intermediate variables. This improves readability and keeps the workflow structured.
- The first step reads the data using `pd.read_excel()`, then `.assign()` creates a `datetime` column by combining `REPORT DAY` with `HOUR ENDING`, and finally, column selection (`[['datetime', 'TOTAL']]`) ensures only relevant data is retained.
- This approach is similar to **piping** (`%>%` or `|>`) in R, where each function operates on the output of the previous step, promoting a clear, left-to-right flow of transformations.
- Without parentheses, each step would require separate assignments, making the code longer and less efficient.

💡 Why Method Chaining? Fadel's Perspective

- Improves **readability**
- Avoids **intermediate variables**
- Mimics **R’s dplyr piping**

Indexing and Slicing with Pandas

- **Indexing** is the process of selecting specific rows and columns from a DataFrame.
- **Slicing** is the process of selecting a subset of the data based on specific criteria. We can slice data based on row numbers, column names, or conditions.
- **Pandas** provides several methods for indexing and slicing data, including `.loc[]`, `.iloc[]`, and `.query()`.
- **`.loc[]`** is used for label-based indexing, while **`.iloc[]`** is used for positional indexing.
  - `df.loc[0:5, 'A':'C']` selects rows 0 to 5 and columns A to C, and
  - `df.iloc[0:5, 0:3]` selects the first 5 rows and first 3 columns.
- **`.query()`** is used to filter data based on specific conditions. We will discuss this in more detail in the next slides.

Filtering/Querying Data with Pandas: Syntax

```
import pandas as pd

duke = (
  pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_20240507.xlsx',
    sheet_name='Total Usage', header=1
    )
  .assign(
    datetime = lambda x:
      pd.to_datetime(x['REPORT DAY']) +
      pd.to_timedelta(x['HOUR ENDING']-1, unit='h')
      )
  [['datetime', 'TOTAL']]
  .query("TOTAL >= 3055456.7") # use query to filter data
  )

duke.head(n=1)
```

```
##                 datetime       TOTAL
## 3591 2023-05-30 15:00:00  3087982.04
```

Filtering/Querying Data with Pandas: Syntax w/ Objects

```
import pandas as pd

total_cutoff= 3055456.7

duke = (
  pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_20240507.xlsx',
    sheet_name='Total Usage', header=1
    )
  .assign(
    datetime = lambda x:
      pd.to_datetime(x['REPORT DAY']) +
      pd.to_timedelta(x['HOUR ENDING']-1, unit='h')
      )
  [['datetime', 'TOTAL']]
  .query("TOTAL >= @total_cutoff") # use query to filter with @ for object
  )

duke.head(n=1)
```

```
##                 datetime       TOTAL
## 3591 2023-05-30 15:00:00  3087982.04
```

Querying Data w/ Pandas: No Spaces in Col Names

```
import pandas as pd

duke = (
  pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_20240507.xlsx',
    sheet_name='Total Usage', header=1
    )
  .assign(
    datetime = lambda x:
      pd.to_datetime(x['REPORT DAY']) +
      pd.to_timedelta(x['HOUR ENDING']-1, unit='h')
      )
  [['datetime', 'HOUR ENDING', 'TOTAL']]
  .query("`HOUR ENDING` == 8") # use backticks ` ` for spaces in col names
  )

duke.head(n=1)
```

```
##              datetime  HOUR ENDING       TOTAL
## 7 2023-01-01 07:00:00            8  1807402.51
```

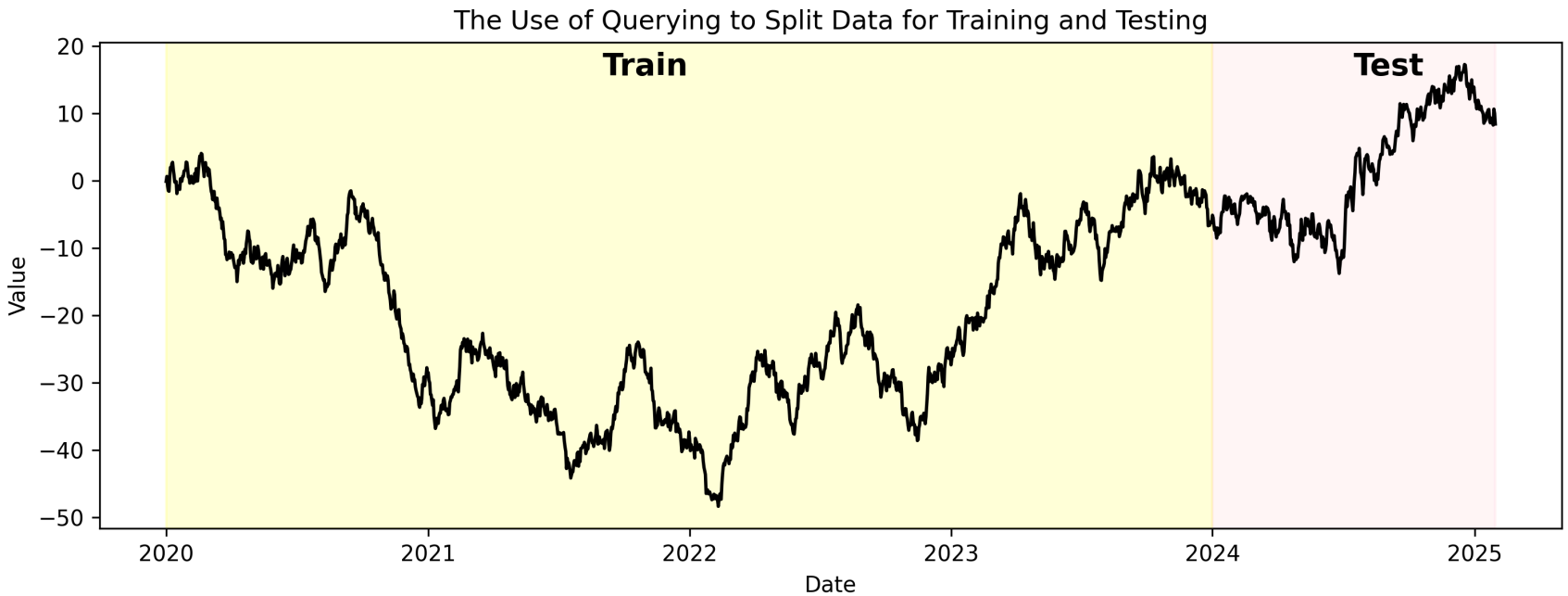
Filtering/Querying Data with Pandas: Multiple Conditions

```
import pandas as pd

duke = (
  pd.read_excel(
    '../../data/Actual_Hourly_Loads_by_Class_2023_20240507.xlsx',
    sheet_name='Total Usage', header=1
    )
  .assign(
    datetime = lambda x:
      pd.to_datetime(x['REPORT DAY']) +
      pd.to_timedelta(x['HOUR ENDING']-1, unit='h')
      )
  [['datetime', 'TOTAL']]
  .query("TOTAL >= 3055456 and datetime >= '2023-05-31'") # use and for multiple conditions
  )

duke.head(n=1)
```

```
##                 datetime       TOTAL
## 3613 2023-05-31 13:00:00  3091924.35
```

How Filtering/Querying is Used in Forecasting?

- **Filtering** and **querying** are essential in forecasting to **select relevant data** for analysis.
- For example, we can use data prior to 2024 for training a forecasting model, and data from 2024 for testing the model to evaluate its performance.



Kahoot Competition #02

To assess your understanding and retention of the topics covered so far, you will **compete in a Kahoot competition (consisting of 5 questions)**:

- Download and inspect the 01_hotel_occupancy.csv
- Go to https://kahoot.it/
- Enter the game pin, which will be shown during class
- Provide your first (preferred) and last name
- Answer each question within the allocated time window (**fast and correct answers provide more points**)

Winning the competition involves having as many correct answers as possible AND taking the shortest duration to answer these questions. The winner 🏆 of the competition will receive a **0.15 bonus on Assignment 02**. Good luck!!!

P.S: The Kahoot competition will have **no negative impact on your grade**. It is a **fun** way of assessing your knowledge, motivating you to ask questions about topics covered that you do not have a full understanding of it, and providing me with some data that I can use to pace today's class.

Summary of Main Points

By now, you should be able to do the following:

- Setting up Python (Colab, Anaconda, and/or VS Code)
- Practice basic data reading (from CSVs and the Web)
- Use panda's datatime, indexing, and slicing capabilities
- (Optional) Discuss generative AI usage in Google Colab

📝 Review and Clarification 📝

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the session ends.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.

📖 Required Readings 📖

- **Read** the following sections from the Pandas Documentation:
  - 10 Minutes to Pandas
  - Pandas I/O
  - Indexing and Selecting Data
  - Time Series / Date functionality
  - Comparison with SQL
- Additionally, you should keep a copy of Pandas Cheat Sheet for a quick reference to the most important functions and methods in Pandas.

🎯 Assignment 🎯

- Go over your notes and complete Assignment 02 on Canvas.