# 🐍 Days 81-84: Portfolio Project - Text-to-Morse Code Converter

This project focuses on building a desktop application that converts plain text into its Morse code equivalent.

## Project Goal

Create a GUI application using Tkinter where a user can type in English text, press a button, and see the corresponding Morse code translation.

## Core Concepts

- **GUI with Tkinter:** Solidifying skills in creating windows, labels, buttons, and text entry widgets.
- **Dictionaries for Mapping:** Using a Python dictionary is the most efficient way to map characters to their Morse code representation.
- **String Manipulation:** Iterating through the user's input string and building the output string.
- **Handling User Events:** Connecting a button click to a function that performs the conversion.

## Implementation Steps & Code Snippets

1. **Setup the UI:**
    - Create the main window, a title label, a `Text` widget for user input, a `Button` to trigger the conversion, and a `Label` or another `Text` widget to display the result.

```python
import tkinter as tk

# --- Morse Code Dictionary ---
MORSE_CODE_DICT = { 'A':'.-', 'B':'-...', 'C':'-.-.', 'D':'-..', 'E':'.',
                    'F':'..-.', 'G':'--.', 'H':'....', 'I':'..', 'J':'.---',
                    'K':'-.-', 'L':'.-..', 'M':'--', 'N':'-.', 'O':'---',
                    'P':'.--.', 'Q':'--.-', 'R':'.-.', 'S':'...', 'T':'-',
                    'U':'..-', 'V':'...-', 'W':'.--', 'X':'-..-', 'Y':'-.--',
                    'Z':'--..', '1':'.----', '2':'..---', '3':'...--',
                    '4':'....-', '5':'.....', '6':'-....', '7':'--...',
                    '8':'---..', '9':'----.', '0':'-----', ', ':'--..--',
                    '.':'.-.-.-', '?':'..--..', '/':'-..-.', '-':'-....-',
                    '(':'-.--.', ')':'-.--.-', ' ':'/'}

# --- UI Setup ---
window = tk.Tk()
window.title("Text to Morse Code Converter")
window.config(padx=50, pady=50)

# Widgets (Labels, Text, Button) go here
# ...
```

2. **Implement Conversion Logic:**
   - Create a function that will be called when the button is pressed.
   - This function should get the text from the input widget ( `text_widget.get("1.0", tk.END)` ).
   - Iterate through each character of the input string (converted to uppercase).
   - Use a `try-except` block to handle characters that are not in your `MORSE_CODE_DICT` .
   - Look up the Morse code for each character in the dictionary and append it to a result string.
   - Display the result in the output widget.

```
def convert_to_morse():
    text_to_convert = input_text.get("1.0", tk.END).upper()
    morse_result = ""
    for char in text_to_convert:
        try:
            morse_result += MORSE_CODE_DICT[char] + " "
        except KeyError:
            # Handle characters not in the dictionary, e.g., ignore or add a special symbol
            pass
    output_label.config(text=morse_result)

# --- Button ---
convert_button = tk.Button(text="Convert", command=convert_to_morse)
convert_button.grid(row=2, column=0)
```

# 🐍 Days 85-86: Portfolio Project - Image Watermarking App

This project involves building a desktop application to add a watermark (text or a logo) to images.

## Project Goal

Create a GUI application that allows a user to open an image file and automatically add a predefined watermark to it.

## Core Concepts

- **Pillow (PIL) Library:** The cornerstone of this project. Used for opening, manipulating, and saving images. Key functions include `Image.open()` , `ImageDraw.Draw()` , and `image.save()` .
- **Tkinter GUI:** Used to create the user interface, including buttons for opening files and triggering the watermarking process.
- **File Dialogs:** Using `tkinter.filedialog` to let the user browse their computer and select an image file. This is a crucial concept for any application that interacts with the user's filesystem.

## Implementation Steps & Code Snippets

1. **Install Pillow:**

- Ensure you have the Pillow library installed: `pip install Pillow` .

2. **UI Setup:**
    - Create a simple Tkinter window with two buttons: "Open Image" and "Add Watermark". You can also add a canvas to display the loaded image.
3. **File Opening Logic:**
    - The "Open Image" button should trigger a function that uses `filedialog.askopenfilename()` .
    - Store the path of the selected file in a global variable or class attribute.

```
from tkinter import filedialog
from PIL import Image, ImageDraw, ImageFont

filepath = ""

def open_image():
    global filepath
    filepath = filedialog.askopenfilename(
        title="Select an Image",
        filetypes=(("JPEG files", "*.jpg"), ("PNG files", "*.png"), ("All files", "*.*"))
    )
    if filepath:
        # Optionally, display the image in the UI
        print(f"Image selected: {filepath}")
```

4. **Watermarking Logic:**
    - The "Add Watermark" button calls the main watermarking function.
    - This function opens the selected image using `Image.open()` .
    - It creates a drawing context with `ImageDraw.Draw(image)` .
    - It defines the text, font, and color for the watermark.
    - It uses `draw.text()` to write the watermark onto the image.
    - Finally, it saves the modified image, usually with a new name like `watermarked_image.jpg` .

```
def add_watermark():
    if not filepath:
        print("Please open an image first.")
        return

    with Image.open(filepath) as im:
        # Create a drawing context
        draw = ImageDraw.Draw(im)

        # Define watermark text and font
        text = "© Your Name 2025"
        font = ImageFont.truetype("arial.ttf", 36) # Make sure you have the font file
        text_width, text_height = draw.textsize(text, font)

        # Position the watermark (e.g., bottom right)
        width, height = im.size
        x = width - text_width - 10
        y = height - text_height - 10

        # Add the text
        draw.text((x, y), text, font=font, fill=(255, 255, 255, 128)) # White with some tra

        # Save the new image
        im.save("watermarked_output.jpg")
        print("Watermark added successfully!")
```

# 🐍 Days 87-88: Portfolio Project - Cafe & WiFi Website

This project marks a shift from desktop applications to web development using the Flask framework.

## Project Goal

Build a simple website that displays a list of cafes from a CSV file. The website should show information like cafe name, location (as a Google Maps link), and WiFi strength.

## Core Concepts

- **Flask Framework:** A lightweight web framework for Python. You'll learn the basics:
    - Routing ( `@app.route('/')` )

    - Rendering templates ( `render_template()` )
    - Running a development server.
- **HTML & CSS (with Bootstrap):** Using HTML to structure the web pages and Bootstrap to quickly style them and make them responsive.
- **Templating with Jinja2:** Flask uses Jinja to embed Python-like code directly into HTML files (e.g., for loops to display data).
- **Working with CSV Data:** Using Python's built-in `csv` module to read data from `cafe-data.csv` and pass it to the web templates.

# Implementation Steps & Code Snippets

1. **Project Setup:**
    - Install Flask: `pip install Flask` .
    - Create a project structure:

    ```
    /cafe-project
        |-- main.py
        |-- cafe-data.csv
        |-- /templates
            |-- index.html
            |-- cafes.html
        |-- /static
            |-- /css
                |-- styles.css
    ```

2. **Basic Flask App ( `main.py` ):**
    - Set up the main application file.

    ```python
    from flask import Flask, render_template
    import csv

    app = Flask(__name__)

    @app.route('/')
    def home():
        return render_template('index.html')

    if __name__ == '__main__':
        app.run(debug=True)
    ```

3. **Reading CSV and Rendering Data:**

- Create a route `/cafes` that reads the `cafe-data.csv` file.
- Convert the CSV data into a list of lists or a list of dictionaries.
- Pass this data to the `cafes.html` template.

```
@app.route('/cafes')
def cafes():
    with open('cafe-data.csv', newline='', encoding='utf-8') as csv_file:
        csv_data = csv.reader(csv_file, delimiter=',')
        list_of_rows = []
        for row in csv_data:
            list_of_rows.append(row)
    return render_template('cafes.html', cafes=list_of_rows)
```

4. **Displaying Data with Jinja ( `cafes.html` ):**

- Use a Jinja `for` loop to iterate through the `cafes` data passed from `main.py` and display it in an HTML table.

```
<div class="container">
  <h1>Cafes with WiFi</h1>
  <table class="table table-dark table-striped">
    <thead>
      <tr>
        {% for header in cafes[0] %}
          <th scope="col">{{ header }}</th>
        {% endfor %}
      </tr>
    </thead>
    <tbody>
      {% for row in cafes[1:] %}
        <tr>
          <td>{{ row[0] }}</td>
          <td><a href="{{ row[1] }}" target="_blank">Map Link</a></td>
          <td>{{ row[2] }}</td>
          <td>{{ row[3] }}</td>
          <td>{{ row[4] }}</td>
          <td>{{ row[5] }}</td>
          <td>{{ row[6] }}</td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
</div>
```

# 🐍 Days 89-90: Portfolio Project - Disappearing Text Writing App

Back to desktop applications, this project is a creative writing tool that forces you to keep typing or else your work disappears.

## Project Goal

Create a simple text editor where, if the user stops typing for a set amount of time (e.g., 5 seconds), all the text they've written is deleted.

## Core Concepts

- **Tkinter Event Handling:** Specifically, binding a key press event ( `<Key>` ) to a function.
- **Scheduling with `window.after()` :** This is the core mechanism of the app. `after()` schedules a function to be called after a certain number of milliseconds. This is how we'll check for inactivity.
- **Canceling Scheduled Events with `after_cancel()` :** To prevent the text from being deleted while the user is typing, we need to cancel the previously scheduled "delete" task every time a new key is pressed.

## Implementation Steps & Code Snippets

1. **UI Setup:**
    - A simple Tkinter window with a large `Text` widget is all that's needed.
2. **Core Logic:**
    - Create a global variable or class attribute to hold the ID of the `after()` job.
    - Create a function `delete_text()` that clears the text widget.
    - Create a function `key_pressed()` that is triggered on every keystroke.
    - Inside `key_pressed()` :
        - Cancel the previous `after()` job using `window.after_cancel(timer_id)` .
        - Schedule a new `after()` job to call `delete_text()` in 5000 milliseconds (5 seconds).
        - Store the new job ID.

```
import tkinter as tk

# --- Constants ---
INACTIVITY_TIME = 5000 # 5 seconds in milliseconds
timer = None

# --- Functions ---
def delete_text():
    text_widget.delete("1.0", tk.END)
    print("Text deleted due to inactivity!")

def on_key_press(event):
    global timer
    # If a timer is already running, cancel it
    if timer:
        window.after_cancel(timer)

    # Start a new timer
    timer = window.after(INACTIVITY_TIME, func=delete_text)

# --- UI Setup ---
window = tk.Tk()
window.title("Disappearing Text App")
window.config(padx=20, pady=20)

text_widget = tk.Text(window, height=20, width=80, font=("Arial", 14))
text_widget.pack()

# Bind the key press event to the text widget
text_widget.bind("<KeyPress>", on_key_press)

# Start the first timer when the app loads
timer = window.after(INACTIVITY_TIME, func=delete_text)

window.mainloop()
```

# 🐍 Days 91-92: Portfolio Project - Image Colour Palette Generator

This project combines web scraping and the Pillow library to extract a color palette from an image found online.

## Project Goal

Build a script that scrapes a website for an image and then uses a library to identify the 10 most dominant colors in that image, creating a color palette.

## Core Concepts

- **Web Scraping:** Using libraries like **BeautifulSoup** and **Requests** to fetch a web page and parse its HTML to find an image URL.
- **`colorgram.py` library:** A specialized library for extracting colors from images. It's much simpler than trying to do this manually with Pillow.
- **Data Structures:** Storing the extracted colors (which are often RGB tuples) in a list.

## Implementation Steps & Code Snippets

1. **Install Libraries:**
    - `pip install colorgram.py`
    - `pip install beautifulsoup4`
    - `pip install requests`
2. **Scrape for an Image URL (Example):**
    - This part is highly dependent on the target website. The goal is to isolate the `src` attribute of an `<img>` tag.
    - *Note: This is a conceptual example. Scraping is fragile and site-specific.*
3. **Extract Colors:**
    - Use `colorgram.extract()` to get the colors. This function can take a file path or a URL.
    - The result is a list of `Color` objects. Each object has an `rgb` attribute.

```python
import colorgram

# This can be a local file path or a URL to an image
image_source = "image.jpg"
number_of_colors = 10

# Extract colors
colors = colorgram.extract(image_source, number_of_colors)

# Store them in a list of RGB tuples
rgb_palette = []
for color in colors:
    r = color.rgb.r
    g = color.rgb.g
    b = color.rgb.b
    new_color = (r, g, b)
    rgb_palette.append(new_color)

print(rgb_palette)
# Output might be: [(236, 224, 212), (198, 13, 32), (247, 237, 227), ...]
```

4. **Application (Optional):**
    - You could use these colors in another project, for example, using the Turtle graphics library to draw a grid of dots representing the color palette (similar to a Damien Hirst spot painting).

# 🐍 Days 93-94: Portfolio Project - Google Dino Game Automation

This is a fun automation project where you write a Python script that plays the Google Chrome "No Internet" dinosaur game.

## Project Goal

Use screen capture and browser automation to detect obstacles (cacti) in the game and trigger the dinosaur to jump automatically.

## Core Concepts

- **Selenium:** A powerful browser automation tool. It's used to open the game ( `chrome://dino` ) and send commands to the browser (like pressing the space bar).
- **Pillow (PIL):** Used for taking screenshots of a specific region of the screen where the obstacles appear.
- **Pixel Analysis:** The core logic involves checking the color of specific pixels in the screenshot. If a pixel color matches the color of a cactus, it means an obstacle is approaching.
- **Timing and Control:** Using the `time` module to create loops and pauses to continuously check the screen.

## Implementation Steps & Code Snippets

1. **Install Libraries:**
    - `pip install selenium`
    - `pip install pillow`
    - You'll also need to download the correct `WebDriver` for your browser (e.g., `chromedriver` ).
2. **Setup Selenium:**
    - Open the Chrome browser and navigate to the game.

    ```
    from selenium import webdriver
    from selenium.webdriver.common.by import By
    from selenium.webdriver.common.keys import Keys
    import time

    chrome_driver_path = "path/to/your/chromedriver"
    driver = webdriver.Chrome(executable_path=chrome_driver_path)
    driver.get("chrome://dino")

    # Wait for page to load and get the body element to send keys
    body = driver.find_element(By.TAG_NAME, "body")
    time.sleep(1)
    body.send_keys(Keys.SPACE) # Start the game
    ```

3. **Game Loop and Screen Capture:**
    - Create a loop that runs as long as the game is active.
    - Inside the loop, use Pillow's `ImageGrab.grab()` to take a screenshot of the area in front of the dinosaur. The `bbox` (bounding box) parameter is crucial here.
4. **Obstacle Detection:**
    - Iterate over the pixels in the screenshot.

- The background color of the game is typically white or light gray. Obstacles (cacti, birds) are dark gray or black.
- If you detect a dark pixel, it signifies an obstacle.

```python
from PIL import ImageGrab

# Coordinates for the box in front of the dino to check for obstacles
# These values need to be found by trial and error for your screen resolution.
OBSTACLE_CHECK_BOX = (250, 400, 300, 450) # (left, top, right, bottom)

def jump():
    body.send_keys(Keys.SPACE)

game_on = True
while game_on:
    # Take a screenshot of the detection area
    image = ImageGrab.grab(bbox=OBSTACLE_CHECK_BOX)

    # Check for non-white pixels
    for x in range(image.width):
        for y in range(image.height):
            pixel_color = image.getpixel((x, y))
            # The game's background is typically (247, 247, 247)
            if pixel_color != (247, 247, 247):
                jump()
                time.sleep(0.1) # Brief pause to avoid multiple jumps
                break # Exit inner loop
        else:
            continue # Only executed if the inner loop did NOT break
        break # Exit outer loop
```

## 🐍 Days 95-96: Portfolio Project - Custom Web Scraper API

This project brings together web scraping and API development. Instead of just running a scraper, you'll build an API that can be called to trigger the scraper and return the results.

# Project Goal

Create a Flask API with a specific endpoint (e.g., `/api/v1/price`). When this endpoint is called, the server runs a web scraper (e.g., to get the price of a product on Amazon) and returns the scraped data in JSON format.

# Core Concepts

- **API Development with Flask:** Going beyond rendering HTML to returning structured data (JSON).
- **JSON (JavaScript Object Notation):** The standard format for data exchange on the web. Python dictionaries are easily converted to JSON.
- **Flask's `jsonify`:** The function used to properly format a Python dictionary into a JSON response with the correct headers.
- **Scraping Logic:** Using BeautifulSoup and Requests to get the target data from a website.

# Implementation Steps & Code Snippets

1. **Project Setup:**
    - Install Flask, BeautifulSoup, and Requests.
    - Create a `main.py` file.
2. **Scraper Function:**
    - Write a standalone function that takes a URL, scrapes it, and returns the desired data (e.g., product price).
    - **Crucially**, add `headers` to your request to mimic a real browser and avoid being blocked.

```
import requests
from bs4 import BeautifulSoup

def get_product_price(url):
    HEADERS = {
        "User-Agent": "Your User Agent String",
        "Accept-Language": "en-US,en;q=0.9"
    }
    response = requests.get(url, headers=HEADERS)
    response.raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')

    # NOTE: These selectors are examples and WILL change.
    price_tag = soup.find(name="span", class_="a-price-whole")
    if price_tag:
        return price_tag.getText().strip(".")
    return "Price not found"
```

3. **Flask API Endpoint:**
   - Create a Flask app.
   - Define a route, for example, `/get-price` .
   - This route will call your scraper function and return the result using `jsonify` .

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/")
def home():
    return "<h1>Price Scraper API</h1><p>Usage: /get-price</p>"

@app.route("/get-price")
def get_price():
    # Example URL
    product_url = "https://www.amazon.com/dp/B0756CYWWD/"
    price = get_product_price(product_url)
    return jsonify(product_name="Sample Product", price=price)

if __name__ == '__main__':
    app.run(debug=True)
```

# 🐍 Day 97: Portfolio Project - Deploying a Website

This day is about taking one of your web projects (like the Cafe & WiFi website) and putting it on the internet for everyone to see.

## Project Goal

Deploy a Flask web application to a hosting service so that it's live on the web with a public URL.

## Core Concepts

- **Web Hosting Platforms:** Understanding the role of services like **PythonAnywhere**, **Heroku**, or **Replit**. PythonAnywhere is often recommended for beginners.
- **WSGI (Web Server Gateway Interface):** The standard that allows a web server (like Apache or Nginx) to communicate with your Python Flask application. You don't need to code this, but you'll configure it on the hosting platform.
- **Environment Variables:** Best practice for managing sensitive information (like API keys) instead of hardcoding them.
- **File Management:** Uploading your project files ( `.py` , `templates` , `static` ) to the server.
- **Dependency Management:** Using a `requirements.txt` file to tell the server which Python packages to install.

## Deployment Steps (Example with PythonAnywhere)

1. **Sign up for PythonAnywhere.**
2. **Upload Files:** Use the "Files" tab to upload your `main.py` , your CSV file, and your `templates` and `static` directories.
3. **Create a `requirements.txt` :** On your local machine, run `pip freeze > requirements.txt` and upload this file.
4. **Open a "Bash console"** on PythonAnywhere and run `pip install -r requirements.txt` to install your project's dependencies (like Flask).
5. **Configure the Web App:**
    - Go to the "Web" tab and create a new web app.
    - Choose the "Flask" framework and the Python version you used.
    - PythonAnywhere will create a `flask_app.py` file for you. You need to edit this file to import your app from `main.py` .
    - In the "Code" section on the Web tab, your `flask_app.py` should look like this:

```
# This file contains the WSGI configuration required to serve up your
# web application at http://<your-username>.pythonanywhere.com/
# ...
import sys

# add your project directory to the sys.path
project_home = '/home/<your-username>/mysite' # Change mysite to your project folder na
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

# import the Flask app object
from main import app as application  # noqa
```

6. **Reload the App:** Click the big "Reload" button on the Web tab. Your site should now be live at `<your-username>.pythonanywhere.com` .

# 🐍 Day 98: Data Science - Analyzing Space Race Data

The final days of the course pivot to data analysis and machine learning, starting with a fun dataset about the Space Race.

## Project Goal

Use **Pandas** and **Matplotlib** to explore a dataset about space missions. Answer questions like: Which country has launched the most missions? How has the number of launches changed over time?

## Core Concepts

- **Jupyter Notebooks / Google Colab:** The ideal environment for interactive data analysis.
- **Pandas DataFrame:** The primary data structure for working with tabular data in Python.
    - Loading data: `pd.read_csv()`
    - Inspecting data: `.head()` , `.info()` , `.describe()`
    - Cleaning data: `.isnull().sum()` , `.dropna()` , changing data types with `.to_datetime()` .
    - Querying & Grouping: `.groupby()` , `.value_counts()` .
- **Matplotlib:** The fundamental library for creating static visualizations in Python.
    - Creating plots: `plt.figure()` , `plt.bar()` , `plt.plot()` , `plt.title()` , `plt.xlabel()` .

# Analysis Workflow & Code Snippets

1. **Load and Inspect Data:**

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

   df = pd.read_csv('mission_launches.csv')
   print(df.shape)
   print(df.isnull().sum()) # Check for missing values
   ```

2. **Data Cleaning:**
   - Drop unnecessary columns with `df.drop()` .
   - Handle missing values, for example, by removing rows with `df.dropna()` .
   - Convert date columns to datetime objects for easier analysis:
     `df['Date'] = pd.to_datetime(df['Date'])` .
3. **Analysis and Visualization:**
   - **How many missions were successful vs. failed?**

     ```
     status_counts = df['Status Mission'].value_counts()
     plt.figure(figsize=(8, 6))
     plt.bar(status_counts.index, status_counts.values)
     plt.title('Mission Status Counts')
     plt.show()
     ```

   - **How many launches per country?** (Requires cleaning the 'Location' column to extract country names).
   - **How many launches per year?**

     ```
     df['Year'] = df['Date'].dt.year
     launches_per_year = df['Year'].value_counts().sort_index()

     plt.figure(figsize=(14, 7))
     plt.plot(launches_per_year.index, launches_per_year.values)
     plt.title('Number of Space Launches Per Year')
     plt.xlabel('Year')
     plt.ylabel('Number of Launches')
     plt.grid(True)
     plt.show()
     ```

# 🐍 Day 99: Data Science - Analyzing Deaths by Police in the US

This project involves working with a more complex and sensitive dataset to practice more advanced Pandas skills.

## Project Goal

Use Pandas to analyze a dataset on police killings in the United States. Answer questions like: What is the age distribution of the deceased? Which states have the highest numbers of incidents? What are the racial demographics of the incidents?

## Core Concepts

- **Advanced Pandas:**
  - `.groupby()` with multiple columns.
  - `.agg()` to apply multiple aggregation functions at once.
  - Merging and joining DataFrames (if using multiple datasets).
  - Working with string methods ( `.str` ) on columns.
- **Seaborn:** A higher-level plotting library built on top of Matplotlib that makes creating beautiful statistical plots easier. `sns.countplot()` , `sns.boxplot()` .

## Analysis Workflow & Code Snippets

1. **Load and Clean:**
   - Load the CSV file into a Pandas DataFrame.
   - Check for missing values ( `.isnull().sum()` ) and decide on a strategy (e.g., drop rows where age is missing).
   - Check data types with `.info()` and convert columns if necessary (e.g., `age` to numeric).
2. **Exploratory Data Analysis (EDA):**
   - **Age Distribution:**

     ```python
     import seaborn as sns

     plt.figure(figsize=(10, 6))
     sns.histplot(df['age'], bins=30, kde=True)
     plt.title('Age Distribution of Deceased')
     plt.xlabel('Age')
     plt.show()
     ```

- **Deaths by Race:**

  ```
  plt.figure(figsize=(12, 7))
  sns.countplot(y=df['race'], order=df['race'].value_counts().index)
  plt.title('Number of Incidents by Race')
  plt.show()
  ```

- **Incidents by State:**

  ```
  state_counts = df['state'].value_counts().head(10)
  plt.figure(figsize=(12, 7))
  sns.barplot(x=state_counts.index, y=state_counts.values)
  plt.title('Top 10 States by Number of Incidents')
  plt.show()
  ```

# 🐍 Day 100: Capstone Project - Predicting App Store Ratings

The final day introduces the fundamentals of machine learning by building a model to predict mobile app ratings.

## Project Goal

Using a dataset of app store data, clean the data, select relevant features, and build a simple linear regression model with Scikit-learn to predict app ratings.

## Core Concepts

- **Machine Learning Fundamentals:**
  - **Features vs. Target:** Identifying which columns are input variables (features, e.g., app size, price) and which is the output variable we want to predict (target, e.g., user rating).
  - **Train-Test Split:** The crucial process of splitting data into a training set (to build the model) and a testing set (to evaluate its performance on unseen data).
- **Scikit-learn:** The go-to library for machine learning in Python.
  - `train_test_split()` for splitting data.
  - `LinearRegression()` for the model itself.
  - Model training with `.fit()` .
  - Making predictions with `.predict()` .

  - Evaluating performance with `.score()` (R-squared).
- **Feature Engineering:** The process of creating new features or transforming existing ones to improve model performance. For this project, it's mostly about selecting numeric features and dropping non-numeric ones.

# ML Workflow & Code Snippets

1. **Load and Preprocess Data:**
   - Load the data into a Pandas DataFrame.
   - This is the most critical step. You'll need to clean the data extensively:
     - Drop rows with missing values.
     - Convert columns like 'Size', 'Installs', and 'Price' from strings (e.g., '1.9M', '1,000,000+', '$2.99') into numerical formats (e.g., 1.9, 1000000, 2.99). This requires significant string manipulation.
     - Select only the numeric columns to use as features.
2. **Define Features (X) and Target (y):**

   ```python
   # After cleaning...
   features = ['Reviews', 'Size', 'Installs', 'Price']
   X = df[features]
   y = df['Rating']
   ```

3. **Train-Test Split:**

   ```python
   from sklearn.model_selection import train_test_split

   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
   ```

4. **Train the Model:**

   ```python
   from sklearn.linear_model import LinearRegression

   # Create a regression model object
   model = LinearRegression()

   # Train the model using the training sets
   model.fit(X_train, y_train)
   ```

5. **Evaluate the Model:**
   - Use the trained model to make predictions on the *test* data.

- Check the R-squared score, which measures how well the model explains the variance in the data (a value closer to 1 is better).

```
# Make predictions using the testing set
y_pred = model.predict(X_test)

# The score (R-squared)
score = model.score(X_test, y_test)
print(f"Model R-squared score: {score:.2f}")

# You can also look at the model's learned coefficients
print("Coefficients: \n", model.coef_)
```

This final project serves as a launchpad into the world of machine learning, demonstrating how all the data cleaning and manipulation skills learned with Pandas are essential prerequisites for building predictive models.

# Congratulations on completing 100 Days of Code! 🎉