🐍 Python Notes: Days 71-80 (Data Science with Pandas, Matplotlib & Scikit-Learn)

This block marks a significant shift into the world of **Data Science**. You'll move from building applications to analyzing and interpreting data. These skills are fundamental for any data-driven project, including the AI/ML aspects of your NEETPrepGPT plan.

🗓️ Day 71: Introduction to Pandas and Data Exploration

Objective

Learn to use the Pandas library to load, inspect, and perform initial analysis on tabular data. The core data structure you'll master is the **DataFrame**.

Key Concepts

- What is Pandas?
- The DataFrame and Series data structures.
- Loading data from a CSV file.
- Inspecting a DataFrame: `.head()`, `.tail()`, `.shape`, `.columns`.
- Accessing columns and rows.
- Performing basic calculations: `.mean()`, `.max()`, `.value_counts()`.

Detailed Notes & Code Snippets

1. Introduction to Pandas

Pandas is the most popular Python library for data manipulation and analysis. It provides two primary data structures:

- **Series:** A one-dimensional labeled array, like a single column of a spreadsheet.

- **DataFrame:** A two-dimensional labeled data structure with columns of potentially different types, like a full spreadsheet or an SQL table.

2. Loading Data and Initial Inspection

You almost always start by loading data from a source, most commonly a `.csv` file.

```
import pandas as pd

# Load the dataset into a DataFrame
df = pd.read_csv('salaries_by_college_major.csv')

# --- Inspection Methods ---

# See the first 5 rows
print(df.head())

# See the last 5 rows
print(df.tail())

# Get the dimensions (rows, columns)
print(f"DataFrame shape: {df.shape}")

# Get the column names
print(f"Column names: {df.columns}")
```

3. Accessing Data

You can access columns like a dictionary and rows using indexing.

```python
# Access a single column (returns a Pandas Series)
starting_salary_col = df['Starting Median Salary']
print(starting_salary_col.head())

# Get the maximum value in a column
print(f"Max starting salary: {starting_salary_col.max()}")

# Find the index of the row with the max value
max_salary_index = starting_salary_col.idxmax()
print(f"Index for max salary: {max_salary_index}") # Outputs an integer index

# Access a specific row by its index label
row_data = df.loc[max_salary_index]
print(row_data)

# Access a specific cell (row index, column name)
major_with_max_salary = df.loc[max_salary_index, 'Undergraduate Major']
print(f"Major with highest starting salary: {major_with_max_salary}")
```

Project Context

Today's project involves analyzing a dataset of salaries by college major. You'll use these inspection techniques to answer questions like "Which major has the highest starting salary?" and "Which major has the highest mid-career salary?".

🗓️ Day 72: Data Cleaning and Manipulation with Pandas

Objective

Learn how to handle common data quality issues, such as missing values (NaNs) and incorrect data types. Clean data is essential for accurate analysis.

Key Concepts

- Identifying missing values: `.isna()`.
- Removing rows/columns with missing values: `.dropna()`.

- Filling missing values: `.fillna()`.
- Changing column data types: `.astype()`.
- Working with dates and times.

Detailed Notes & Code Snippets

1. Finding and Handling Missing Values (NaN)

Real-world data is messy. `NaN` (Not a Number) is Pandas' way of representing missing data.

```
import pandas as pd

df = pd.read_csv('salaries_by_college_major.csv')

# Check for missing values in the entire DataFrame
print(df.isna().sum())

# Let's assume the last row has missing data
# You can see this by inspecting df.tail()

# Drop rows with any NaN values
clean_df = df.dropna()
print(f"Original shape: {df.shape}")
print(f"Shape after dropna: {clean_df.shape}")

# If you wanted to fill missing values instead (e.g., with 0)
# df_filled = df.fillna(0)
```

2. Manipulating Columns

You can create new columns from existing ones or modify them.

```
# Create a new column by performing an operation on existing columns
# Calculate the spread between mid-career 90th percentile and 10th percentile salaries
spread_col = clean_df['Mid-Career 90th Percentile Salary'] - clean_df['Mid-Career 10th Percenti

# Insert the new column into the DataFrame at a specific position
clean_df.insert(1, 'Spread', spread_col)
print(clean_df.head())

# To sort the DataFrame by this new column
low_risk_majors = clean_df.sort_values('Spread')
print(low_risk_majors[['Undergraduate Major', 'Spread']].head())
```

3. Grouping Data with `.groupby()`

This is one of the most powerful features in Pandas. It allows you to split data into groups, apply a function, and combine the results.

```
# Group by 'Group' column and find the average salaries for each group
grouped_data = clean_df.groupby('Group').mean(numeric_only=True)
print(grouped_data)
```

Project Context

The project continues with the salary dataset, focusing on cleaning it and then using `.groupby()` to find the majors with the highest potential and lowest risk (smallest salary spread).

🗓️ Day 73: Aggregation, Merging, and Pivoting DataFrames

Objective

Learn advanced data manipulation techniques, including how to combine multiple datasets and reshape data for better analysis.

Key Concepts

- Advanced `.groupby()` with `.agg()`.

- Combining DataFrames with `.merge()` (like SQL joins).
- Reshaping DataFrames with `.pivot_table()`.

Detailed Notes & Code Snippets

1. Combining DataFrames (`.merge()`)

Often, your data is split across multiple files. `pd.merge()` allows you to combine them based on a common column.

```
import pandas as pd

# Imagine two dataframes
df1 = pd.DataFrame({'student_id': [1, 2, 3], 'subject': ['Math', 'Science', 'History']})
df2 = pd.DataFrame({'student_id': [1, 2, 3], 'grade': [85, 92, 78]})

# Merge them on the common 'student_id' column
merged_df = pd.merge(df1, df2, on='student_id')
print(merged_df)
```

2. Reshaping with `.pivot_table()`

Pivoting is useful for summarizing data in a matrix-like format.

```
# Using the Nobel Prize dataset from the course
df = pd.read_csv('nobel_prize_data.csv')

# Create a pivot table to see the number of prizes per year and category
prize_counts = df.pivot_table(index='year', columns='category', values='prize', aggfunc='count')
print(prize_counts.tail())
```

Project Context

Today's project involves analyzing the **Nobel Prize dataset**. You'll need to clean it, manipulate date formats, and use `.pivot_table()` and other aggregations to uncover trends, like which countries or categories have won the most prizes over time.

🗓️ Day 74: Introduction to Data Visualization with Matplotlib

Objective

Learn to create basic plots and charts to visually represent data using the **Matplotlib** library. A picture is worth a thousand rows of data.

Key Concepts

- Creating basic plots: `plt.plot()`, `plt.scatter()`.
- Customizing plots: labels, titles, colors, line styles, figure size.
- Working with subplots.

Detailed Notes & Code Snippets

1. Basic Plotting

Matplotlib is the foundational plotting library in Python.

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

# Using the grouped data from Day 72
clean_df = pd.read_csv('salaries_by_college_major.csv').dropna()
spread_col = clean_df['Mid-Career 90th Percentile Salary'] - clean_df['Mid-Career 10th Percentil
clean_df.insert(1, 'Spread', spread_col)
highest_spread = clean_df.sort_values('Spread', ascending=False)


# Plotting Starting Median Salary vs. Spread
plt.figure(figsize=(10, 6)) # Set the figure size (width, height in inches)

plt.scatter(highest_spread['Starting Median Salary'], highest_spread['Spread'])

# Adding labels and title for clarity
plt.xlabel('Starting Median Salary ($)')
plt.ylabel('Salary Spread ($)')
plt.title('Starting Salary vs. Salary Spread for College Majors')

plt.grid(True) # Add a grid
plt.show() # Display the plot
```

Project Context

You will use the **LEGO dataset** for this project. The goal is to create various plots to answer questions like: "How has the average number of parts in a LEGO set changed over the years?" and "Which LEGO themes have the most sets?".

🗓️ Day 75: Advanced Visualization with Seaborn and Plotly

Objective

Build upon Matplotlib by using **Seaborn** for more aesthetically pleasing statistical plots and get an introduction to **Plotly** for interactive charts.

Key Concepts

- Seaborn for statistical plots: `regplot`, `boxplot`, `histplot`.
- Creating interactive plots with Plotly Express.

Detailed Notes & Code Snippets

1. Seaborn for Regression Plots

Seaborn is built on top of Matplotlib and makes creating complex statistical plots much easier.

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

# Load Google trends data about programming languages
df = pd.read_csv('programming_languages_search_trends.csv')
df.date = pd.to_datetime(df.date) # Convert date column to datetime objects

# Create a regression plot to see the trend of Python searches over time
plt.figure(figsize=(12, 7))
sns.regplot(x=df.index, y=df['python'], scatter_kws={'alpha':0.3}, line_kws={'color':'red'})

plt.title('Search Trend for "Python" Over Time')
plt.xlabel('Date')
plt.ylabel('Search Interest')
plt.show()
```

2. Plotly for Interactive Visualizations

Plotly creates interactive, web-based visualizations that you can zoom, pan, and hover over for more information.

```
import plotly.express as px

# Using the same programming languages data
fig = px.line(df, x='date', y=['python', 'java', 'javascript'],
              title='Interactive Search Trends for Programming Languages')

fig.show() # This will open an interactive plot in your browser or notebook
```

Project Context

The main project uses Google Trends data to analyze the popularity of different programming languages over time. You will use Seaborn and Plotly to visualize these trends and compare them.

🗓️ Day 76: Advanced Pandas & NumPy

Objective

Dive deeper into advanced Pandas features and get an introduction to **NumPy**, the fundamental package for numerical computing in Python.

Key Concepts

- Pandas: Multi-level indexing.
- NumPy: The `ndarray` object.
- Vectorization and broadcasting in NumPy for high-performance computation.

Detailed Notes & Code Snippets

1. NumPy Arrays

NumPy's core object is the `ndarray` (n-dimensional array). It's faster and more memory-efficient than Python lists for numerical operations.

```
import numpy as np

# Create a NumPy array from a list
my_list = [1, 2, 3, 4, 5]
my_array = np.array(my_list)

# Perform vectorized operations (no loops needed!)
print(my_array * 2) # Output: [ 2  4  6  8 10]
print(my_array + 5) # Output: [ 6  7  8  9 10]

# Generate arrays
zeros = np.zeros(5) # [0. 0. 0. 0. 0.]
ones = np.ones((2, 3)) # 2x3 array of 1s
```

2. The Power of Vectorization

Vectorization allows you to perform operations on entire arrays at once, which is executed in optimized, pre-compiled C code. This is *much* faster than iterating with a Python `for` loop.

```
# Using NumPy is significantly faster for large datasets
large_array = np.arange(1_000_000)

# NumPy vectorized operation (very fast)
%timeit large_array ** 2

# Python list comprehension (slower)
large_list = list(range(1_000_000))
%timeit [x**2 for x in large_list]
```

Project Context

You will revisit the **LEGO dataset** and apply NumPy for faster calculations. You will also tackle more complex data manipulation tasks that require a deeper understanding of Pandas indexing and grouping.

🗓️ Day 77: Linear Regression with Scikit-Learn

Objective

Build your first machine learning model! Understand the theory and practical application of **Linear Regression** using the **Scikit-Learn** library.

Key Concepts

- What is Linear Regression? (Finding the best-fit line).
- The Scikit-Learn library.
- Splitting data into training and testing sets.
- Creating, training (`.fit()`), and using a model (`.predict()`).
- Evaluating model performance (`.score()`).

Detailed Notes & Code Snippets

1. Theory of Linear Regression

Linear regression models the relationship between a dependent variable (what you want to predict) and one or more independent variables (features) by fitting a straight line to the data. The equation is $y = mx + b$, where the model learns the best values for the slope (m) and the intercept (b).

2. Implementation with Scikit-Learn

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# 1. Load and prepare the data
df = pd.read_csv('boston_housing.csv')
X = df[['RM', 'LSTAT']] # Features (e.g., avg rooms, % lower status)
y = df['PRICE']        # Target (what we want to predict)

# 2. Split data into training and testing sets
# 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Create and train the model
model = LinearRegression()
model.fit(X_train, y_train) # The "learning" happens here

# 4. Evaluate the model
train_score = model.score(X_train, y_train) # R-squared value for training data
test_score = model.score(X_test, y_test)   # R-squared value for testing data

print(f"Training R^2 Score: {train_score:.4f}")
print(f"Testing R^2 Score: {test_score:.4f}")

# 5. Make predictions on new, unseen data
new_house_features = [[6.5, 5.0]] # e.g., 6.5 rooms, 5% lower status population
predicted_price = model.predict(new_house_features)
print(f"Predicted price for the new house: ${predicted_price[0]*1000:.2f}")
```

Project Context

You'll use a dataset like the **Boston Housing dataset** to predict house prices based on features like the number of rooms, crime rate, etc. This is a classic "hello world" project for machine learning.

🗓️ Day 78: Logistic Regression and Classification

Objective

Learn about **Logistic Regression**, a powerful algorithm for solving classification problems (i.e., predicting a category, not a continuous number).

Key Concepts

- Classification vs. Regression.
- The Sigmoid function.
- Building a Logistic Regression model in Scikit-Learn.
- Evaluating classification models: Confusion Matrix, Accuracy, Precision, Recall.

Detailed Notes & Code Snippets

1. Theory of Logistic Regression

While Linear Regression outputs a continuous value, Logistic Regression outputs a probability (between 0 and 1). It uses the sigmoid function to map any real-valued number into this range. We can then set a threshold (e.g., 0.5) to classify the outcome into one of two categories (e.g., "Yes" or "No", "Survived" or "Died").

2. Implementation with Scikit-Learn

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load and prepare the data (e.g., Titanic dataset)
df = pd.read_csv('titanic.csv')
# (Data cleaning and feature engineering steps would go here)
# ... for simplicity, assume we have features X and target y
X = df[['Pclass', 'Age', 'Fare']].fillna(df['Age'].mean()) # Simplified features
y = df['Survived']

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Create and train the model
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train)

# 4. Make predictions
predictions = log_model.predict(X_test)

# 5. Evaluate
accuracy = accuracy_score(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)

print(f"Model Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

Project Context

The quintessential project for this day is predicting survival on the **Titanic**. You will use passenger data (age, class, sex, etc.) to build a model that classifies whether a passenger likely survived or not.

🗓️ Day 79: Advanced Machine Learning Models

Objective

Get a high-level overview of more advanced and powerful machine learning models beyond linear/logistic regression.

Key Concepts

- **Decision Trees:** A flow-chart-like structure.
- **Random Forests:** An ensemble of many decision trees.
- **Support Vector Machines (SVM):** A model that finds the optimal hyperplane to separate data points.
- The concept of **Overfitting** and **Underfitting**.

Detailed Notes

This day is more conceptual, introducing you to the existence and general idea of more complex models.

- **Decision Tree:** Easy to interpret. It splits the data based on feature values, creating a tree of decisions. Prone to overfitting (learning the training data too well).
- **Random Forest:** The "wisdom of the crowd" for models. It builds many different decision trees on random subsets of the data and averages their predictions. This reduces overfitting and generally improves performance. It's often a great "go-to" model.
- **Overfitting:** Your model learns the training data, including its noise and quirks, so perfectly that it fails to generalize to new, unseen data. It will have a high training score but a low testing score.
- **Underfitting:** Your model is too simple to capture the underlying patterns in the data. It will perform poorly on both the training and testing sets.

The Scikit-Learn API is consistent, so using these models is syntactically similar to what you've already done:

```python
from sklearn.ensemble import RandomForestClassifier

# Same workflow: create, fit, predict, score
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_accuracy = rf_model.score(X_test, y_test)
print(f"Random Forest Accuracy: {rf_accuracy:.4f}")
```

Project Context

You might apply a Random Forest model to the same Titanic or Boston Housing dataset to see if you can achieve better performance than with the simpler models from the previous days.

🗓️ Day 80: Capstone Project - LEGO Dataset Analysis

Objective

Combine everything you've learned in the data science section (Days 71-79) to perform a complete, end-to-end data analysis project.

Project Steps

1. **Ask Interesting Questions:** Start by formulating questions you want to answer with the data.
    - What is the relationship between the number of parts and the price of a LEGO set?
    - How many unique themes are there? Which are the most popular?
    - How has the color palette of LEGO bricks changed over time?
2. **Data Wrangling & Cleaning (Pandas):**
    - Load multiple CSV files (`sets.csv` , `colors.csv` , `themes.csv`).
    - Handle missing values in the datasets.
    - Merge the different DataFrames together to create one master DataFrame for analysis.
3. **Data Exploration & Visualization (Pandas, Matplotlib, Seaborn):**
    - Use `.groupby()` and `.agg()` to summarize the data.
    - Create plots (bar charts, line charts, scatter plots) to visualize your findings and answer the questions you formulated. For example, plot the number of sets released per year.
4. **(Optional) Machine Learning (Scikit-Learn):**

- Build a simple linear regression model to predict the number of parts in a set based on the year it was released.

Example Code Snippet (Merging and Plotting)

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

# Load datasets
sets = pd.read_csv('sets.csv')
themes = pd.read_csv('themes.csv')

# Merge to get theme names
sets_with_themes = pd.merge(sets, themes, left_on='theme_id', right_on='id', suffixes=('_set',

# Explore sets per year
sets_by_year = sets_with_themes.groupby('year').count()['set_num']

# Plotting
plt.figure(figsize=(14, 8))
plt.plot(sets_by_year.index[:-2], sets_by_year.values[:-2]) # Exclude last 2 years for complete
plt.title('Number of LEGO Sets Released Per Year')
plt.xlabel('Year')
plt.ylabel('Number of Sets')
plt.grid(True)
plt.show()
```

This capstone solidifies your new data science skills, preparing you for the next phase of the course, which often involves building full-stack web applications that can serve up these kinds of analyses.