Python for Data Analysts: A Step-by-Step Guide for Beginners

Overview:

Python is an essential tool for data analysts, providing a powerful and flexible environment for data manipulation, analysis, and visualization. This step-by-step guide covers fundamental operations that every data analyst should master, from loading data to performing complex analyses and visualizations.

First Phase

Step 1: Import Necessary Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

Step 2: Load Data

From a CSV file on the desktop:

```
data = pd.read_csv(r'C:\path_to_your_file\filename.csv')
```

From the web:

```
url = 'https://example.com/filename.csv'

data = pd.read_csv(url)
```

Step 3: Convert CSV to DataFrame

The data object above is already a pandas DataFrame.

Step 4: Remove Unneeded Columns

```
columns_to_drop = ['column1', 'column2']  # Replace with actual column names

data = data.drop(columns=columns_to_drop)
```

Step 5: Rename Column Headers

```
new_column_names = {
  'old_column1': 'new_column1',
  'old_column2': 'new_column2'
}
data = data.rename(columns=new_column_names)
```

Step 6: Describe the Dataset

```
print(data.describe())  # Summary of numerical columns
print(data.info())      # General info, including null values
```

Step 7: Handle/Remove Null or N/A Values

Remove rows with null values:

```
data = data.dropna()
```

Fill null values with a specific value:

```
data['column_name'] = data['column_name'].fillna(0)  # Replace with a default value
```

Step 8: Merge Tables

```
table1 = pd.read_csv('table1.csv')
table2 = pd.read_csv('table2.csv')

# Merge on a common column (e.g., 'id')
merged_data = pd.merge(table1, table2, on='id', how='inner')   # Options: 'inner', 'outer', 'left', 'right'
```

Step 9: Change Data Types

```
data['column_name'] = data['column_name'].astype('int')     # Convert to integer
data['date_column'] = pd.to_datetime(data['date_column'])   # Convert to datetime
```

Step 10: Perform Mathematical Calculations

```
# Example: Add a new column that is the sum of two existing columns
data['new_column'] = data['column1'] + data['column2']

# Example: Calculate the mean of a column
mean_value = data['column_name'].mean()
```

Step 11: Create Visualizations

Line Chart

```
plt.figure(figsize=(10, 6))
```

```
plt.plot(data['x_column'], data['y_column'], label='Line 1')
plt.title('Line Chart')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.legend()
plt.show()
```

Bar Chart

```
plt.figure(figsize=(10, 6))
plt.bar(data['x_column'], data['y_column'], color='skyblue', label='Bar 1')
plt.title('Bar Chart')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.legend()
plt.show()
```

Histogram

```
plt.figure(figsize=(10, 6))
plt.hist(data['column_name'], bins=20, color='green', alpha=0.7, label='Histogram')
plt.title('Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

Seaborn Example (Bar Plot)

```
plt.figure(figsize=(10, 6))
sns.barplot(x='x_column', y='y_column', data=data, palette='Blues')
plt.title('Seaborn Bar Chart')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.show()
```

Additional Notes:

*For **filtering rows** based on conditions:*

```
filtered_data = data[data['column_name'] > 50]  # Example: Filter rows where column > 50
```

For grouping data:

```
grouped_data = data.groupby('group_column')['value_column'].sum().reset_index()
```

For saving the cleaned/processed data back to a file:

```
data.to_csv(r'C:\path_to_save\processed_file.csv', index=False)
```

More Step Guides

1. Create Pivot Tables

```
pivot_table = data.pivot_table(
    values='value_column',
    index='index_column',
    columns='column_name',
    aggfunc='sum'
)
print(pivot_table)
```

---

2. Sort Data by a Column

```
sorted_data = data.sort_values(by='column_name', ascending=False)
print(sorted_data)
```

---

3. Filter Rows Based on Multiple Conditions

```
filtered_data = data[(data['column1'] > 50) & (data['column2'] == 'category')]
print(filtered_data)
```

---

4. Apply Custom Functions to Columns

```
def custom_function(x):
    return x * 2 if x > 50 else x

data['new_column'] = data['existing_column'].apply(custom_function)
```

---

5. Cumulative Sum and Running Totals

```
data['cumulative_sum'] = data['value_column'].cumsum()
```

---

6. Detect and Remove Duplicates

Find Duplicates:

```
duplicates = data[data.duplicated()]
print(duplicates)
```

Remove Duplicates:

```
data = data.drop_duplicates()
```

---

7. Group and Aggregate

```
aggregated_data = data.groupby('group_column')['value_column'].agg(['sum', 'mean', 'max', 'min'])

print(aggregated_data)
```

---

8. Create a New Column Based on a Condition

```
data['new_column'] = np.where(data['column'] > 50, 'High', 'Low')
```

---

9. Generate Random Data

```
random_data = np.random.randint(0, 100, size=(10, 3))

random_df = pd.DataFrame(random_data, columns=['A', 'B', 'C'])

print(random_df)
```

---

10. Normalize or Standardize Data

Normalize (scale to range [0, 1]):

```
data['normalized_column'] = (data['column'] - data['column'].min()) / (data['column'].max() - data['column'].min())
```

Standardize (convert to Z-scores):

```
data['standardized_column'] = (data['column'] - data['column'].mean()) / data['column'].std()
```

---

11. Correlation Matrix

```
correlation_matrix = data.corr()

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()
```

---

12. Split a Column into Multiple Columns

```
data[['new_col1', 'new_col2']] = data['column_to_split'].str.split('_', expand=True)
```

---

13. Create a Time Series Plot

```
data['date_column'] = pd.to_datetime(data['date_column'])

data.set_index('date_column', inplace=True)

data['value_column'].plot(figsize=(10, 6), title='Time Series Plot')

plt.xlabel('Date')

plt.ylabel('Value')

plt.show()
```

---

14. Find Outliers Using IQR

```
Q1 = data['column_name'].quantile(0.25)

Q3 = data['column_name'].quantile(0.75)

IQR = Q3 - Q1


outliers = data[(data['column_name'] < Q1 - 1.5 * IQR) | (data['column_name'] > Q3 + 1.5 * IQR)]

print(outliers)
```

---

15. Extract Text Patterns

```
data['extracted_text'] = data['text_column'].str.extract(r'(\d{4})')  # Example: Extract 4-digit numbers
```

---

16. Create Dummy Variables

```
data_with_dummies = pd.get_dummies(data, columns=['categorical_column'], drop_first=True)
```

---

17. Reshape Data (Melt and Pivot)

Melt (Wide to Long):

```
melted_data = pd.melt(data, id_vars=['id_column'], var_name='variable', value_name='value')

print(melted_data)
```

Pivot (Long to Wide):

```
pivoted_data = melted_data.pivot(index='id_column', columns='variable', values='value')

print(pivoted_data)
```

---

18. Create a Box Plot

```
sns.boxplot(x='categorical_column', y='numerical_column', data=data)

plt.title('Box Plot')

plt.show()
```

---

19. Rolling Window Calculations

```
data['rolling_mean'] = data['value_column'].rolling(window=3).mean()
```

---

20. Export Data to Excel

```
data.to_excel(r'C:\path_to_save\filename.xlsx', index=False)
```

---

21. Split Data into Train and Test Sets

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2, random_state=42)
```

---

22. Aggregate Multiple Columns with Different Functions

```
agg_data = data.groupby('group_column').agg({'col1': 'sum', 'col2': 'mean', 'col3': 'max'})

print(agg_data)
```

---

23. Count Unique Values in a Column

```
unique_counts = data['column_name'].nunique()

print(f"Number of unique values: {unique_counts}")
```

---

24. Generate Pairplots with Seaborn

```
sns.pairplot(data, hue='categorical_column', diag_kind='kde')

plt.title('Pairplot')

plt.show()
```

---

25. Perform String Manipulations

Convert to lowercase:

```
data['column'] = data['column'].str.lower()
```

Check if a string contains a substring:

```
data['contains_keyword'] = data['column'].str.contains('keyword')
```

Other Common Visualizations

1. Heatmap

***Purpose**: Visualize the correlation or relationship between variables.*

```
plt.figure(figsize=(10, 6))

sns.heatmap(data.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()
```

---

2. Scatter Plot

***Purpose**: Explore the relationship between two numerical variables.*

```
plt.figure(figsize=(10, 6))

sns.scatterplot(x='x_column', y='y_column', data=data, hue='category_column',
style='category_column', palette='viridis')

plt.title('Scatter Plot')

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.show()
```

---

3. Stacked Bar Chart

***Purpose**: Show proportions or parts of a whole over categories.*

```
data.groupby(['category', 'sub_category'])['value'].sum().unstack().plot(kind='bar', stacked=True, figsize=(10, 6))

plt.title('Stacked Bar Chart')

plt.xlabel('Category')

plt.ylabel('Value')

plt.legend(title='Sub-Category')

plt.show()
```

---

4. Bubble Chart

***Purpose**: Visualize three variables with X, Y, and bubble size.*

```
plt.figure(figsize=(10, 6))

plt.scatter(data['x_column'], data['y_column'], s=data['size_column'] * 100, alpha=0.5, c=data['color_column'], cmap='viridis')

plt.title('Bubble Chart')

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.colorbar(label='Color Metric')

plt.show()
```

---

5. Pairplot

***Purpose**: Explore pairwise relationships in a dataset.*

```
sns.pairplot(data, hue='category_column', diag_kind='kde', palette='husl')

plt.suptitle('Pairplot', y=1.02)

plt.show()
```

---

6. Pie Chart

***Purpose**: Show proportions of categories.*

```
data['category'].value_counts().plot.pie(

  autopct='%1.1f%%', figsize=(8, 8), startangle=90, colormap='viridis')

plt.title('Pie Chart')

plt.ylabel('')  # Remove default y-label

plt.show()
```

---

7. Violin Plot

***Purpose**: Visualize the distribution of data and its probability density.*

```
plt.figure(figsize=(10, 6))
```

```
sns.violinplot(x='category_column', y='value_column', data=data, palette='Set2')

plt.title('Violin Plot')

plt.xlabel('Category')

plt.ylabel('Value')

plt.show()
```

---

8. Box Plot

***Purpose**: Display data spread and detect outliers.*

```
plt.figure(figsize=(10, 6))

sns.boxplot(x='category_column', y='value_column', data=data, palette='Set3')

plt.title('Box Plot')

plt.xlabel('Category')

plt.ylabel('Value')

plt.show()
```

---

9. Histogram with KDE (Density Curve)

***Purpose**: Show distribution of a single variable.*

```
plt.figure(figsize=(10, 6))

sns.histplot(data['value_column'], kde=True, bins=30, color='blue')

plt.title('Histogram with KDE')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()
```

---

10. Treemap

***Purpose**: Represent hierarchical data in a nested structure.*

```
import squarify

sizes = data['value_column']

labels = data['category_column']


plt.figure(figsize=(10, 6))

squarify.plot(sizes=sizes, label=labels, alpha=0.8, color=sns.color_palette('Set2'))

plt.title('Treemap')

plt.axis('off')

plt.show()
```

---

11. Word Cloud

***Purpose**: Visualize the frequency of words in a dataset.*

```
from wordcloud import WordCloud
```

```
text = ' '.join(data['text_column'].dropna())

wordcloud = WordCloud(background_color='white', colormap='viridis').generate(text)


plt.figure(figsize=(10, 6))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud')

plt.show()
```

---

12. Area Chart

***Purpose**: Show trends over time with emphasis on magnitude.*

```
plt.figure(figsize=(10, 6))

plt.fill_between(data['x_column'], data['y_column'], color='skyblue', alpha=0.5)

plt.plot(data['x_column'], data['y_column'], color='Slateblue', alpha=0.8)

plt.title('Area Chart')

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.show()
```

---

13. Donut Chart

***Purpose**: Pie chart variation with a center cut out.*

```
sizes = data['value_column']

labels = data['category_column']


plt.figure(figsize=(8, 8))

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, pctdistance=0.85,
colors=sns.color_palette('Set2'))

centre_circle = plt.Circle((0, 0), 0.70, fc='white')

plt.gca().add_artist(centre_circle)

plt.title('Donut Chart')

plt.show()
```

---

14. Facet Grid

***Purpose**: Visualize data subsets across multiple categories.*

```
g = sns.FacetGrid(data, col='category_column', col_wrap=4, height=4)

g.map(plt.hist, 'value_column', bins=20, color='skyblue')
```

```
g.fig.suptitle('Facet Grid', y=1.02)

plt.show()
```

---

15. Geospatial Heatmap

***Purpose**: Visualize geospatial data (e.g., latitude and longitude).*

```
import geopandas as gpd

from shapely.geometry import Point


geometry = [Point(xy) for xy in zip(data['longitude'], data['latitude'])]

geo_df = gpd.GeoDataFrame(data, geometry=geometry)


geo_df.plot(marker='o', color='red', markersize=10, figsize=(10, 8))

plt.title('Geospatial Heatmap')

plt.show()
```

---

16. Regplot (Regression Line)

***Purpose**: Add a regression line to visualize the trend.*

```
plt.figure(figsize=(10, 6))

sns.regplot(x='x_column', y='y_column', data=data, color='teal', line_kws={"color": "red"})

plt.title('Regression Plot')

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.show()
```

---

17. Time Series with Multiple Lines

***Purpose**: Show trends for multiple categories over time.*

```
plt.figure(figsize=(12, 6))

for label, group in data.groupby('category_column'):

    plt.plot(group['time_column'], group['value_column'], label=label)

plt.title('Time Series with Multiple Lines')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend(title='Category')

plt.show()
```

---

18. Population Pyramid

***Purpose**: Show age distribution split by gender (or similar categories).*

```
male = data[data['gender'] == 'Male']['age'].value_counts().sort_index()

female = data[data['gender'] == 'Female']['age'].value_counts().sort_index()
```

```
plt.figure(figsize=(10, 6))
plt.barh(male.index, male.values, color='blue', label='Male')
plt.barh(female.index, -female.values, color='pink', label='Female')
plt.title('Population Pyramid')
plt.xlabel('Count')
plt.ylabel('Age')
plt.legend()
plt.show()
```
