Working with Data Files 1

Course:
INFO-6145 Data Science and Machine Learning

FANSHAWE

Revised by:
Mohammad Noorchenarboo

September 19, 2024

Contents

Current Section

Stages of Data Analysis

- Data Gathering: What data and metrics do you need?
- Data Cleaning: Remove errors, handle missing data.
- Data Analysis: Apply statistical/ML methods.
- Data Interpretation: Transform results into conclusions.
- Data Visualization: Create charts, graphs for reporting.

Working with CSV Files in Python

Saving and Writing CSV Files:

Python Code for Writing CSV

```
import csv
fields = ['Name', 'Email']
rows = [['Nikhil', 'nikhil.gfg@gmail.com'],
        ['Sanchit', 'sanchit.gfg@gmail.com']]
filename = "email_records.csv"
with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

Key Points:

- CSV is widely used for tabular data.
- Ensure correct delimiters and encoding for large files.

Sources of Data

Internal Sources:

- Company databases, emails, documents.

External Sources:

- Public datasets (Kaggle, Google BigQuery).
- Government and open-access databases, publications.

Additional Notes:

- Validate external data for consistency and format.
- Ensure internal data security and privacy.

Data Collection and Verification

Data Collection Methods:

- Interviews, surveys, observations, experiments.

Dataset Verification:

Key Steps

- Check column names and data types.
- Handle missing data, detect and address outliers.
- Split data into training and test sets.

Data Analysis Tools: Pandas and NumPy

Pandas:

- Data manipulation, DataFrames, filtering, aggregation.

NumPy:

- Mathematical functions, linear algebra, random generation.
- Supports ‘np.inf‘ for handling infinity.

NumPy Example

```
import numpy as np
df = pd.DataFrame([10, 3000, -4000, np.inf, -np.inf])
```

Creating and Manipulating DataFrames

Creating DataFrames:

- Use the DataFrame constructor with dictionaries or lists.

DataFrame Example

```
df = pd.DataFrame({'X': [788, 596], 'Y': [849, 489]})
print(df)
```

Data Cleaning:

- Remove rows with '.drop()', reset index with '.reset$_i$*ndex*()'.

Drop Rows Example

```
df = df.drop(df.index[[2, 4]]).reset_index(drop=True)
```

Splitting and Combining DataFrames

Splitting DataFrames:

- Useful for splitting training and test datasets.
- Use ‘sample()‘ to split data randomly.

Splitting Example

```
part_70 = df.sample(frac=0.7, random_state=10)
part_30 = df.drop(part_70.index)
```

Combining Series:

- Use ‘pd.concat()‘ to combine multiple Series into a DataFrame.

Combining Example

```
s1 = pd.Series([100, 200])
s2 = pd.Series([10, 20])
df = pd.concat([s1, s2], axis=1)
```

Current Section

pandas.cut() for Binning Data

Overview:

- `pandas.cut()` is used to segment and sort data into bins.
- Often used when converting continuous data into categorical data based on ranges.

Example: Binning Age into Categories

Using `pandas.cut()`

```
import pandas as pd
df["AgeCategory"] = pd.cut(df["Age"], [0,20,40,60,80])
```

Key Points:

- The second argument is a list of bin edges.
- Bins are created as intervals between each value.

pandas.append() for Adding Rows

Overview:

- The `append()` function adds rows to the end of a DataFrame.
- Useful when appending new data dynamically.

Example: Appending a New Row

Using `append()`

```
import pandas as pd
df = pd.DataFrame({"col1": range(3), "col2": range(3)}
new_row = pd.DataFrame({"col1": [3], "col2": [3]})
df = df.append(new_row, ignore_index=True)
print(df)
```

Key Points:

- `ignore_index=True` reindexes the DataFrame after appending.
- Append can accept either a dictionary or another DataFrame.

pandas.iloc for Index-based Selection

Overview:

- `iloc` allows for selection of rows and columns based on integer positions.
- It is useful when selecting by row/column index rather than by label.

Example: Selecting the First Row

Using `iloc`

```
import pandas as pd
d = {'col1': [123, 456], 'col2': [789, 1011]}
df = pd.DataFrame(data=d)
print("Row 1:", df.iloc[0])
```

Key Points:

- `iloc` uses numerical indices starting from 0.
- You can specify both rows and columns using this method (e.g., `df.iloc[0, 1]`).

pandas.loc for Label-based Selection

Overview:

- `loc` selects data based on labels or a boolean condition.
- Useful when you need to filter data by row/column names.

Example: Selecting Rows by Label

Using `loc`

```
import pandas as pd
df = pd.DataFrame({'Name': ['John', 'Jane'], 'Age': [2
row = df.loc[df['Name'] == 'John']
print(row)
```

Key Points:

- `loc` allows for label-based indexing and boolean conditions.
- It can also be used to modify specific rows based on conditions.

Creating DataFrames

Overview:

- DataFrames can be created directly from dictionaries or lists.

DataFrame Example

```
import pandas as pd
df = pd.DataFrame({'col1': range(3), 'col2': range(3)}
print(df)
```

Key Points:

- DataFrames are 2-dimensional, tabular data structures.
- Each column can be a different data type (e.g., int, float, string).

Source

https://www.w3resource.com/python-exercises/pandas/index-dataframe.php