Programowanie Aplikacji Geoinformacyjnych

1060-GI000-ISP-5002

Laboratorium 11. Formaty danych

Formaty danych

- Długi i szeroki format danych
- Pliki danych Parquet
- Star schema

Długi i szeroki format danych

(*Wide and long data formats*)

Szeroki format danych - struktura

- W formacie szerokim każdy obiekt lub obserwacja obejmuje jeden wiersz, a każda zmienna lub pomiar jest reprezentowana przez osobną kolumnę.

Subject	Measurement_1	Measurement_2	Measurement_3
A	5	3	4
B	6	8	7
C	9	6	5

Szeroki format danych - korzyści

- **Wygoda wykonywania analiz**: Niektóre metody i modele statystyczne preferują format szeroki, ponieważ każda zmienna znajduje się w osobnej kolumnie.
- **Czytelność** (human interpretation): Przy mniejszej liczbie wierszy i większej liczbie kolumn może być łatwiej ludziom czytać i rozumieć zbiór danych.

Długi format danych - struktura

- W formacie długim każdy obiekt lub obserwacja może mieć wiele wierszy, a pomiary są ułożone w jednej kolumnie, często z dodatkową kolumną wskazującą typ zmiennej.
- Zwykle wygląda to jak tabela z większą liczbą wierszy i mniejszą liczbą kolumn.

Subject	Measurement_Type	Value
A	Measurement_1	5
A	Measurement_2	3
A	Measurement_3	4
B	Measurement_1	6
B	Measurement_2	8
B	Measurement_3	7
C	Measurement_1	9
C	Measurement_2	6
C	Measurement_3	5

Długi format danych - korzyści

- **Elastyczność manipulacji i wizualizacji danych**: Format długi jest często preferowany do użycia z bibliotekami manipulacji danymi, takimi jak Pandas w Pythonie, lub do tworzenia wykresów za pomocą narzędzi takich jak ggplot2 w R.
- **Łatwiejsze stosowanie transformacji**: Wiele procesów analizy i transformacji danych, takich jak transpozycja lub agregowanie, jest prostszych do wykonania w formacie długim.
- **Kompatybilność z narzędziami statystycznymi**: Wiele nowoczesnych narzędzi i bibliotek do nauki o danych jest zaprojektowanych tak, aby efektywniej pracować z danymi w formacie długim.

Podsumowanie

- **Format szeroki** jest przydatny do określonych typów analiz i może być łatwiejszy do odczytania na pierwszy rzut oka, ale może być nieporęczny przy skomplikowanych manipulacjach danymi.
- **Format długi** oferuje większą elastyczność transformacji danych i jest lepiej przystosowany do zaawansowanej analizy danych oraz narzędzi do wizualizacji danych.

Przykład – odczyt danych w formacie szerokim

```
import pandas as pd

# CSV file in wide format

file_path = 'wide_format.csv'

wide_df = pd.read_csv(file_path)

print(wide_df.head())
```

Przykład – zapis danych w formacie szerokim

```
# Save the wide format DataFrame to a CSV file

wide_df.to_csv('wide_format_output.csv', index=False)
```

Przykład – odczyt danych w formacie długim

```
# CSV file in long format

file_path = 'long_format.csv'

long_df = pd.read_csv(file_path)

print(long_df.head())
```

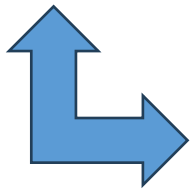
Przykład – zapis danych w formacie długim

```
# Save the long format DataFrame to a CSV file

long_df.to_csv('long_format_output.csv', index=False)
```

Przykład – konwersja formatów

Subject	Measurement_1	Measurement_2	Measurement_3
A	5	3	4
B	6	8	7
C	9	6	5



Subject	Measurement_Type	Value
A	Measurement_1	5
A	Measurement_2	3
A	Measurement_3	4
B	Measurement_1	6
B	Measurement_2	8
B	Measurement_3	7
C	Measurement_1	9
C	Measurement_2	6
C	Measurement_3	5

Przykład – konwersja formatów

```
# Convert wide format to long format using pd.melt

long_df = pd.melt(wide_df, id_vars=['Subject'],
var_name='Measurement_Type', value_name='Value')

print(long_df.head())
```

Przykład – konw

```
# Convert wide format to long fo

long_df = pd.melt(wide_df, id_va
var_name='Measurement_Type', val

print(long_df.head())
```

pandas.melt

pandas.melt(*frame*, *id_vars*=None, *value_vars*=None, *var_name*=None, *value_name*='value', *col_level*=None, *ignore_index*=True) [source]

Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.

This function is useful to massage a DataFrame into a format where one or more columns are identifier variables (*id_vars*), while all other columns, considered measured variables (*value_vars*), are "unpivoted" to the row axis, leaving just two non-identifier columns, 'variable' and 'value'.

Parameters:

id_vars : *scalar, tuple, list, or ndarray, optional*

Column(s) to use as identifier variables.

value_vars : *scalar, tuple, list, or ndarray, optional*

Column(s) to unpivot. If not specified, uses all columns that are not set as *id_vars*.

var_name : *scalar, default None*

Name to use for the 'variable' column. If None it uses `frame.columns.name` or 'variable'.

value_name : *scalar, default 'value'*

Name to use for the 'value' column, can't be an existing column label.

col_level : *scalar, optional*

If columns are a MultiIndex then use this level to melt.

ignore_index : *bool, default True*

If True, original index is ignored. If False, the original index is retained. Index labels will be repeated as necessary.

Przykład – konwersja formatów

```
# Convert long format to wide format using pd.pivot

wide_df = long_df.pivot(index='Subject', columns='Measurement_Type',
values='Value').reset_index()

print(wide_df.head())
```

Przykład – konw

```
# Convert long format to wide

wide_df = long_df.pivot(index=
values='Value').reset_index()

print(wide_df.head())
```

pandas.pivot

`pandas.pivot(data, *, columns, index=<no_default>, values=<no_default>)` [source]

Return reshaped DataFrame organized by given index / column values.

Reshape data (produce a "pivot" table) based on column values. Uses unique values from specified *index* / *columns* to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the User Guide for more on reshaping.

Parameters:

data : ***DataFrame***

columns : ***str or object or a list of str***

Column to use to make new frame's columns.

index : ***str or object or a list of str, optional***

Column to use to make new frame's index. If not given, uses existing index.

values : ***str, object or a list of the previous, optional***

Column(s) to use for populating new frame's values. If not specified, all remaining columns will be used and the result will have hierarchically indexed columns.

Returns:

DataFrame

Returns reshaped DataFrame.

Przykład – wizualizacja danych

```
data = {

    'Subject': ['A', 'B', 'C'],

    'Measurement_1': [5, 6, 9],

    'Measurement_2': [3, 8, 6],

    'Measurement_3': [4, 7, 5]

}

wide_df = pd.DataFrame(data)
```

Przykład – wizualizacja danych szerokich

```
# Plotting wide format data

wide_df.plot(x='Subject', y=['Measurement_1', 'Measurement_2',
'Measurement_3'], kind='bar')

plt.title('Wide Format Data Visualization')

plt.xlabel('Subject')

plt.ylabel('Value')

plt.show()
```

Przykład – wizuali

```
# Plotting wide format data

wide_df.plot(x='Subject', y=['Meas
'Measurement_3'], kind='bar')

plt.title('Wide Format Data Visual

plt.xlabel('Subject')

plt.ylabel('Value')

plt.show()
```

pandas.DataFrame.plot

`DataFrame.plot(*args, **kwargs)` [source]

Make plots of Series or DataFrame.

Uses the backend specified by the option `plotting.backend`. By default, matplotlib is used.

Parameters:

data : *Series or DataFrame*

The object for which the method is called.

x : *label or position, default None*

Only used if data is a DataFrame.

y : *label, position or list of label, positions, default None*

Allows plotting of one column versus another. Only used if data is a DataFrame.

kind : *str*

The kind of plot to produce:

- 'line' : line plot (default)
- 'bar' : vertical bar plot
- 'barh' : horizontal bar plot
- 'hist' : histogram
- 'box' : boxplot
- 'kde' : Kernel Density Estimation plot
- 'density' : same as 'kde'
- 'area' : area plot
- 'pie' : pie plot
- 'scatter' : scatter plot (DataFrame only)

Przykład – wizualizacja danych długich

```
# Plotting long format data

import seaborn as sns

sns.barplot(x='Subject', y='Value', hue='Measurement_Type', data=long_df)

plt.title('Long Format Data Visualization')

plt.xlabel('Subject')

plt.ylabel('Value')

plt.show()
```

Przykład – wizuali

```
# Plotting long format data

import seaborn as sns

sns.barplot(x='Subject', y='Value

plt.title('Long Format Data Visua

plt.xlabel('Subject')

plt.ylabel('Value')

plt.show()
```

seaborn.barplot

```
seaborn.barplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None,
estimator='mean', errorbar=('ci', 95), n_boot=1000, seed=None, units=None, weights=None,
orient=None, color=None, palette=None, saturation=0.75, fill=True, hue_norm=None,
width=0.8, dodge='auto', gap=0, log_scale=None, native_scale=False, formatter=None,
legend='auto', capsize=0, err_kws=None, ci=<deprecated>, errcolor=<deprecated>,
errwidth=<deprecated>, ax=None, **kwargs)
```

Show point estimates and errors as rectangular bars.

A bar plot represents an aggregate or statistical estimate for a numeric variable with the height of each rectangle and indicates the uncertainty around that estimate using an error bar. Bar plots include 0 in the axis range, and they are a good choice when 0 is a meaningful value for the variable to take.

See the tutorial for more information.

Note

By default, this function treats one of the variables as categorical and draws data at ordinal positions (0, 1, ... n) on the relevant axis. As of version 0.13.0, this can be disabled by setting `native_scale=True`.

Parameters: **data** : *DataFrame, Series, dict, array, or list of arrays*

Dataset for plotting. If `x` and `y` are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

x, y, hue : *names of variables in `data` or vector data*

Inputs for plotting long-form data. See examples for interpretation.

order, hue_order : *lists of strings*

Order to plot the categorical levels in; otherwise the levels are inferred from the data objects.

estimator : *string or callable that maps vector -> scalar*

Podsumowanie przykładów

- **Format szeroki**: Łatwy do odczytania i odpowiedni do niektórych analiz statystycznych.
- **Format długi**: Elastyczny do manipulacji danymi i wizualizacji.
- Oba formaty mają swoje mocne strony, a wybór zależy od konkretnego zastosowania i potrzeb analizy danych.

Pliki danych Parquet

Format danych pliku Parquet

- Format pliku Parquet jest **zorientowany kolumnowo**, co oznacza, że przechowuje dane według kolumn, a nie wierszy.
- Sprawia to, że jest szczególnie **wydajny** do zapytań analitycznych i przetwarzania danych.
- Chociaż sam format Parquet nie narzuca, czy dane są w formacie szerokim, czy długim, może przechowywać dane w obu formatach, w zależności od tego, jak dane są skonstruowane i zapisane w pliku.
- Zasadniczo można mieć dane w formacie szerokim, gdzie kolumny reprezentują różne pomiary i dane w formacie długim, gdzie wiersze reprezentują różne obserwacje lub pomiary.

Przykład zapisu danych

```
import pandas as pd

# Sample DataFrame
data = {
    'Subject': ['A', 'B', 'C'],
    'Measurement_1': [5, 6, 9],
    'Measurement_2': [3, 8, 6],
    'Measurement_3': [4, 7, 5]
}
df = pd.DataFrame(data)

# Writing to Parquet file
df.to_parquet('sample_data.parquet')
```

Przykład odczytu danych

```
# Reading from Parquet file

df_read = pd.read_parquet('sample_data.parquet')
```

Parquet - korzyści

- **Efektywne przechowywanie** - przechowując dane kolumnowo, pliki Parquet osiągają lepsze współczynniki kompresji, zmniejszając koszty przechowywania.
- **Optymalizacja wydajności odczytu** - przechowywanie kolumnowe umożliwia selektywny odczyt tylko potrzebnych kolumn, poprawiając wydajność odczytu danych i zmniejszając operacje I/O.
- **Ewolucja schematu** - schemat danych jest przechowywany w stopce pliku i pozwala na ewolucję schematu, ułatwiając radzenie sobie ze zmianami w strukturze danych z biegiem czasu.
- **Kompatybilność** - Parquet jest szeroko stosowanym formatem i jest kompatybilny z różnymi narzędziami i platformami big data, takimi jak Apache Spark, Hadoop i Amazon Redshift.

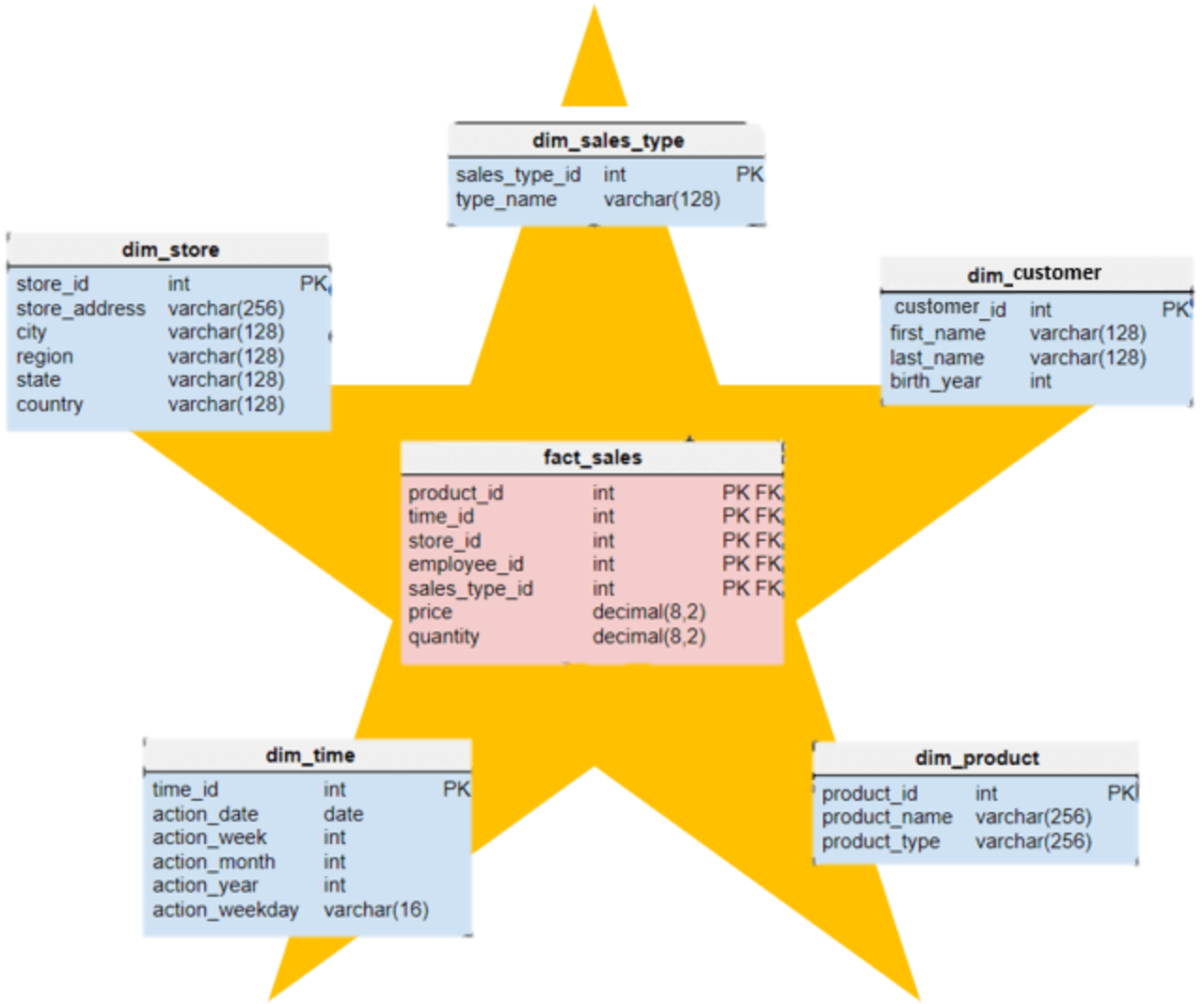
Star schema

Star schema

- Schemat gwiazdy (star schema) to rodzaj schematu danych stosowanego w hurtowniach danych (data warehouse), który służy do reprezentowania danych wielowymiarowych.
- Schemat ten nazywany jest schematem gwiazdy, ponieważ diagram przypomina gwiazdę z punktami promieniującymi od centralnej tabeli.

Star s

- Schemat g ych stosowane(który służy do repreze(
- Schemat te /aż diagram przypomina tralnej tabeli.



Źródło: https://iterationinsights.com/

Star schema - komponenty

Tabela faktów (fact table):

- Centralna tabela w schemacie gwiazdy (star schema), która zawiera dane ilościowe do analizy (np. kwota sprzedaży, ilość).
- Zawiera klucze obce odnoszące się do kluczy głównych w tabelach wymiarów.

Tabele wymiarów (dimension tables):

- Otaczają tabelę faktów i zawierają atrybuty opisowe związane z danymi faktów (np. szczegóły produktu, czas, lokalizacja).
- Każda tabela wymiarów ma klucz główny, który łączy się z kluczem obcym w tabeli faktów.

Prosty przykład star schema

Dimension Table: Time

TimeID	Date	Month	Year
202101	2021-01-01	January	2021
202102	2021-02-01	February	2021

Dimension Table: Store

StoreID	StoreName	Location
1	Store A	New York
2	Store B	Los Angeles

Fact Table: Sales

SaleID	ProductID	TimeID	StoreID	Amount	Quantity
1	101	202101	1	100	2
2	102	202102	2	200	3

Dimension Table: Product

ProductID	ProductName	Category
101	Widget	Gadgets
102	Thingamajig	Tools

Przykład – wielkość sprzedaży wg. produktów

```
SELECT p.ProductName, SUM(f.Amount) AS TotalSales

FROM Sales f

JOIN Product p ON f.ProductID = p.ProductID

GROUP BY p.ProductName;
```

Przykład – liczba produktów wg. sklepów

```
SELECT s.StoreName, SUM(f.Quantity) AS TotalQuantity

FROM Sales f

JOIN Store s ON f.StoreID = s.StoreID

GROUP BY s.StoreName;
```

Przykład – wielkość sprzedaży w danym roku

```
SELECT t.Month, SUM(f.Amount) AS TotalSales

FROM Sales f

JOIN Time t ON f.TimeID = t.TimeID

WHERE t.Year = 2021

GROUP BY t.Month;
```

Star schema - korzyści

- **Prostota**: Łatwość zrozumienia i projektowania. Prosta struktura pozwala użytkownikom łatwo pytać o dane bez skomplikowanych połączeń.
- **Wydajność**: Optymalizowany do operacji z dużą liczbą odczytów. Użycie denormalizowanych tabel zmniejsza potrzebę skomplikowanych połączeń i poprawia wydajność zapytań.
- **Elastyczność**: Upraszcza proces dodawania nowych wymiarów lub faktów. Pozwala to na łatwe rozszerzenie schematu danych, aby spełnić nowe wymagania dotyczące danych.
- **Efektywność**: Umożliwia szybsze pobieranie i przetwarzanie danych, co jest idealne dla aplikacji OLAP (Online Analytical Processing).

Przykład – star schema i dane przestrzenne

Microsoft Copilot

Fact Table: `TrafficIncidents`

This table records traffic incidents in a city.

IncidentID	LocationID	TimeID	Severity	VehicleCount	WeatherID
1	101	20231201	3	2	1
2	102	20231201	2	1	2
3	103	20231202	4	3	1

Dimension Table: `Location`

This table contains geospatial information about locations where incidents occur.

LocationID	Latitude	Longitude	City	District
101	52.2297	21.0122	Warszawa	Śródmieście
102	52.4064	16.9252	Poznań	Stare Miasto
103	51.1079	17.0385	Wrocław	Krzyki

Dimension Table: `Time`

This table includes details about the time of the incidents.

TimeID	Date	Month	Year
20231201	2023-12-01	12	2023
20231202	2023-12-02	12	2023

Dimension Table: `Weather`

This table records the weather conditions during the incidents.

WeatherID	WeatherType	Temperature	Precipitation
1	Clear	5°C	0 mm
2	Rainy	3°C	10 mm

Przyk strzenne

Microsoft Copilot

Fact Table: `TrafficIncidents`

This table records traffic incidents in a ci

IncidentID	LocationID	TimeID
1	101	2023120
2	102	2023120
3	103	2023120

Dimension Table: `Location`

This table contains geospatial informatic

LocationID	Latitude	Lon
101	52.2297	21.0
102	52.4064	16.9
103	51.1079	17.0

SQL Queries

Here are some example SQL queries to analyze the data:

1. **Total Number of Incidents by City**

Sql — Kopiuj

```sql
SELECT l.City, COUNT(f.IncidentID) AS TotalIncidents
FROM TrafficIncidents f
JOIN Location l ON f.LocationID = l.LocationID
GROUP BY l.City;
```

2. **Average Severity of Incidents by Weather Type**

Sql — Kopiuj

```sql
SELECT w.WeatherType, AVG(f.Severity) AS AverageSeverity
FROM TrafficIncidents f
JOIN Weather w ON f.WeatherID = w.WeatherID
GROUP BY w.WeatherType;
```

3. **Number of Incidents per Day**

Sql — Kopiuj

```sql
SELECT t.Date, COUNT(f.IncidentID) AS DailyIncidents
FROM TrafficIncidents f
JOIN Time t ON f.TimeID = t.TimeID
GROUP BY t.Date;
```

ents.

Month	Year
12	2023
12	2023

incidents.

ature	Precipitation
	0 mm
	10 mm