Data Mining

Beer and Diapers


One Midwest grocery chain used the data mining capacity of Oracle software to analyze local buying patterns. They discovered that **when men bought diapers on Thursdays and Saturdays, they also tended to buy beer**. Further analysis showed that these shoppers **typically did their weekly grocery shopping on Saturdays**. On Thursdays, however, they only bought a few items.

The retailer concluded that they purchased the beer to have it available for the upcoming weekend. The grocery chain could use this newly discovered information in various ways to increase revenue. For example, they could move the beer display closer to the diaper display. And, they could make sure beer and diapers were sold at full price on Thursdays.

What is Data Mining?

Data Mining is the process of understanding your data and identifying its characteristics.

- finding anomalies
- discovering patterns
- testing correlations
- predicting outcomes

It's an interdisciplinary field of computer science and statistics.

Data Mining Applications

Data Mining is widely used in a variety of industries and disciplines.

- Communications
- Insurance
- Retail
- Bioinformatics
- ...

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) often uses visual methods to summarize data characteristics.

EDA Toolkit

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.

seaborn is a Python data visualization library based on **matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics.

Python: class

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def introduce(self):
        print(f"Hello, my name is {self.name}. I am {self.age}.")

p = Person("John", 36)
p.introduce()
```

```
Hello, my name is John. I am 36.
```

- The **`__init__()`** function is called automatically every time the class is being used to create a new object.
- The **self** parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

pandas

Installation

```
$ pip install pandas
```

Import

```
import pandas as pd
```

Creating a Series

```
s = pd.Series([1, 10e1, 10e2, 10e3, 10e4])
```

Creating a DataFrame

```
d = pd.DataFrame({
    "col1": range(100),
    "col2": [i*2 for i in range(100)],
    "col3": [i**2 for i in range(100)],
})
```

pandas: viewing data

View first and last few data rows

```
d.head()
```

```
d.tail()
```

View a specific number of first or last few data rows

```
d.head(10)
```

```
d.tail(2)
```

View column names

```
d.columns
```

View row index

```
d.index
```

pandas: selecting data

Select single column

```
d["col1"]
```

Select multiple columns

```
d[["col1", "col3"]]
```

Select a slice of data rows

```
d[13:27]
```

Select by position

```
d.iloc[55]
```

pandas: selecting data cont'd

Select by label (index)

```
d.loc[55]
```

```
d.index = [f"a{i}" for i in range(100)]
d.loc["a33"]
```

Select on a multi-axis by label

```
d.loc[[f"a{i}" for i in [10, 20, 30, 40]], ["col1", "col3"]]
```

```
d.loc[:, ["col1", "col3"]]
```

pandas: selecting data cont'd

Select by boolean indexing

```
d[d["col3"] < 400]
```

```
d[(d["col3"] < 400) & (d["col1"] > 3)]
```

```
d[(d["col3"] > 400) | (d["col1"] < 3)]
```

```
d[~(d["col3"] < 400)]
```

pandas: missing data

```
import numpy as np

d.loc["a0", "col1"] = np.nan
d.loc["a1", "col2"] = np.nan
d.loc["a2", "col3"] = np.nan

d.dropna()
```

```
d.fillna(-100)
```

```
d.isna()
```

pandas: operations

```
d.max()
```

```
d.min()
```

```
d.count()
```

```
d.sum()
```

```
d.mean()
```

```
d.describe()
```

Perform operation by row

```
d.mean(axis=1)
```

pandas: operations cont'd

Apply functions to the data

```
d.apply(lambda col: col.sum() / col.count())
```

Apply the functions by row

```
d.apply(lambda row: row["col1"] + row["col3"], axis=1)
```

Note: A **lambda** function is a small anonymous function.

pandas: casting type

```
a = d["col3"].astype("str")
```

String methods can be applied if a series is of str type.

```
a = a.str.rstrip(".0")
a.str.zfill(6)
```

pandas: histogramming

```
s = pd.Series(np.random.randint(0, 7, size=100))
s.value_counts()
```

Select only the top 3

```
s.value_counts()[:3]
```

Sort in ascending order

```
s.value_counts(ascending=True)[:3]
```

pandas: sorting

```
df = pd.DataFrame({
    'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
    'col2': [2, 1, 9, 8, 7, 4],
    'col3': [0, 1, 9, 4, 2, 3],
})

df.sort_values(by=['col1'])
```

```
df.sort_values(by=['col1', 'col2'])
```

```
df.sort_values(by=['col1', 'col2'], ascending=False)
```

pandas: grouping

```
df = pd.DataFrame(
    {
        "A": [
            "foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"
        ],
        "B": ["x", "x", "y", "z", "y", "y", "x", "z",],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)

df.groupby('A').sum()
```

```
df.groupby(['A', 'B']).sum()
```

pandas: grouping cont'd

```
for key, group_df in df.groupby('A'):
    print(f"{key}: {type(group_df)}")
    group_df.head()
    print()
```

```
bar: <class 'pandas.core.frame.DataFrame'>
     A  B         C         D
1  bar  x  0.112899  0.723876
3  bar  z -1.049243 -0.400599
5  bar  y  0.387344 -0.707578

foo: <class 'pandas.core.frame.DataFrame'>
     A  B         C         D
0  foo  x  0.563033  0.896916
2  foo  y  0.082203 -0.653892
4  foo  y -0.031266 -0.400012
6  foo  x -0.029963 -0.302663
7  foo  z  0.745815  0.849511
```

pandas: getting data in/out

```
df.to_csv('foo.csv')
```

```
pd.read_csv('foo.csv')
```

Package **openpyxl** and **xlrd** are needed to deal with MS Excel file.

```
df.to_excel('foo.xlsx')
```

```
pd.read_excel('foo.xlsx')
```

Also work with Python object serialization

```
df.to_pickle('foo.pkl')
```

```
pd.read_pickle('foo.pkl')
```

seaborn

Installation

```
$ pip install seaborn
```

Import

```
import seaborn as sns
```

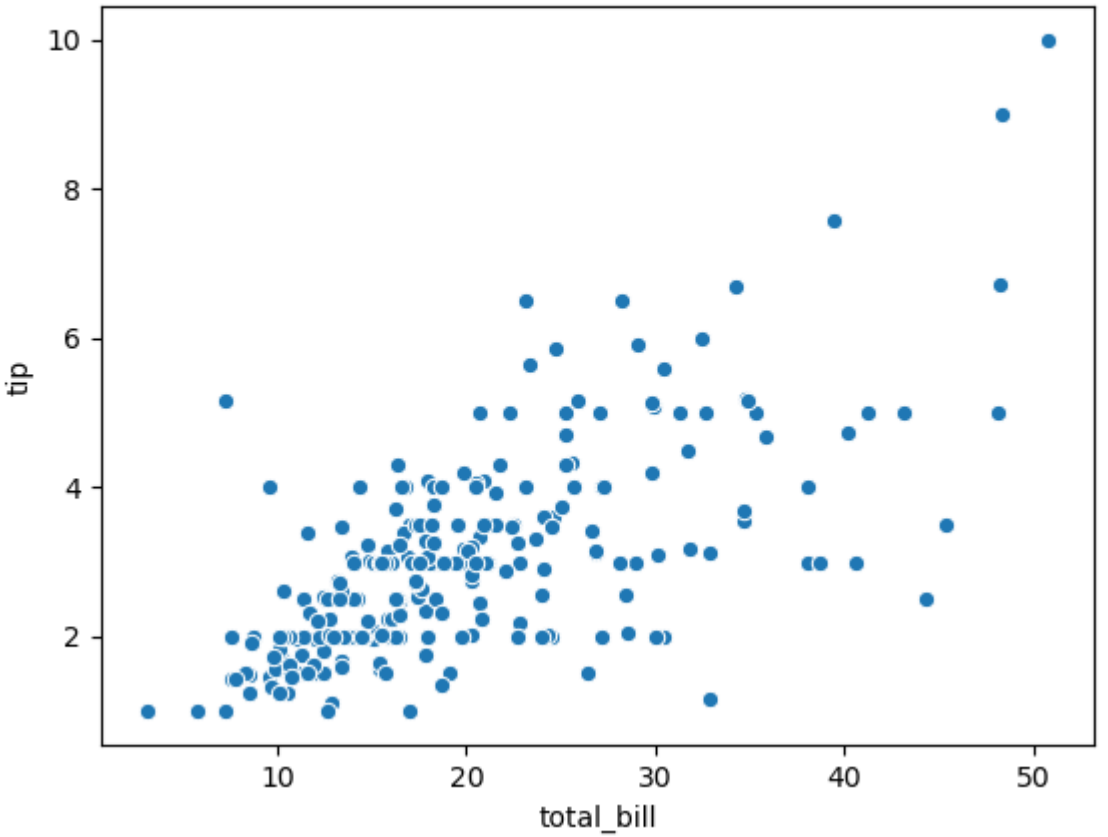
seaborn: scatterplot

```
import seaborn as sns
import matplotlib.pyplot as plt
tips = sns.load_dataset("tips")

ax = sns.scatterplot(x="total_bill", y="tip", data=tips)

plt.show()
```

Result



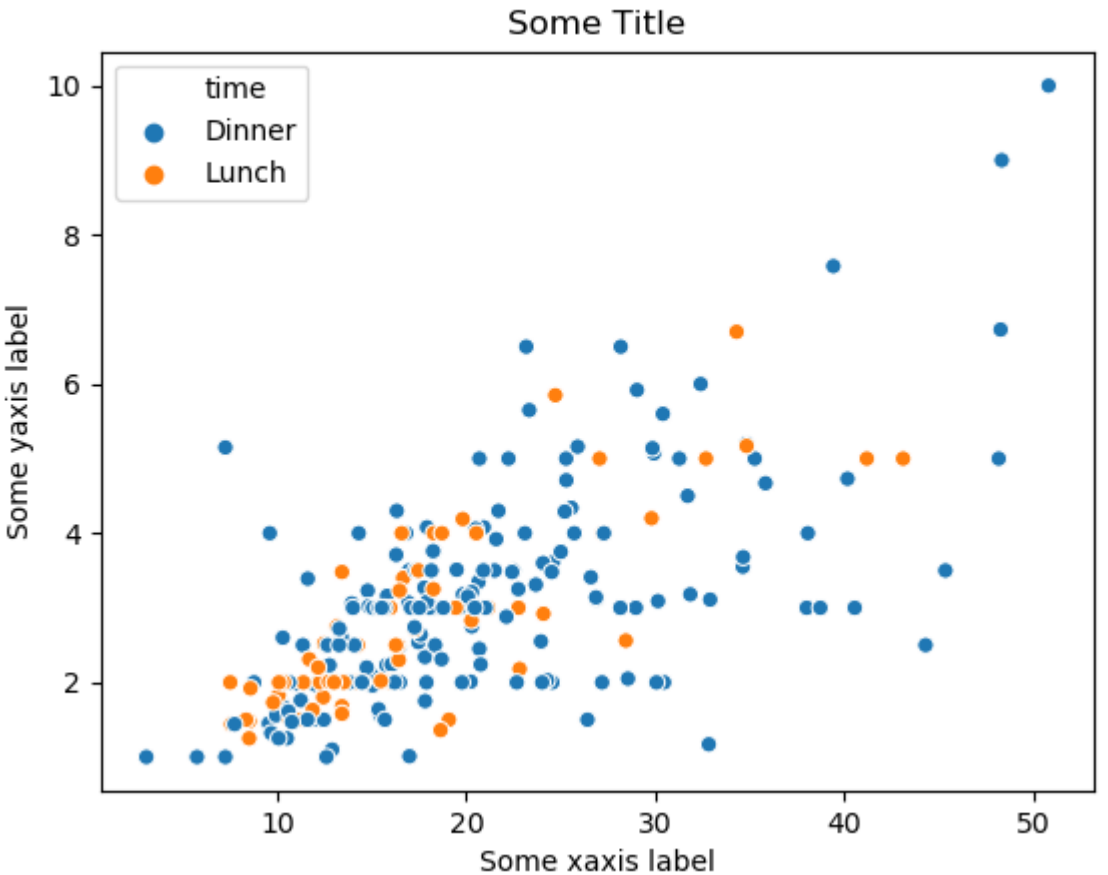
seaborn: scatterplot cont'd

Add hue to the plot (grouping the data points by time of day).

```
ax = sns.scatterplot(x="total_bill", y="tip", hue="time", data=tips)
plt.title("Some Title")
plt.xlabel("Some xaxis label")
plt.ylabel("Some yaxis label")

plt.show()
```

Result with hue

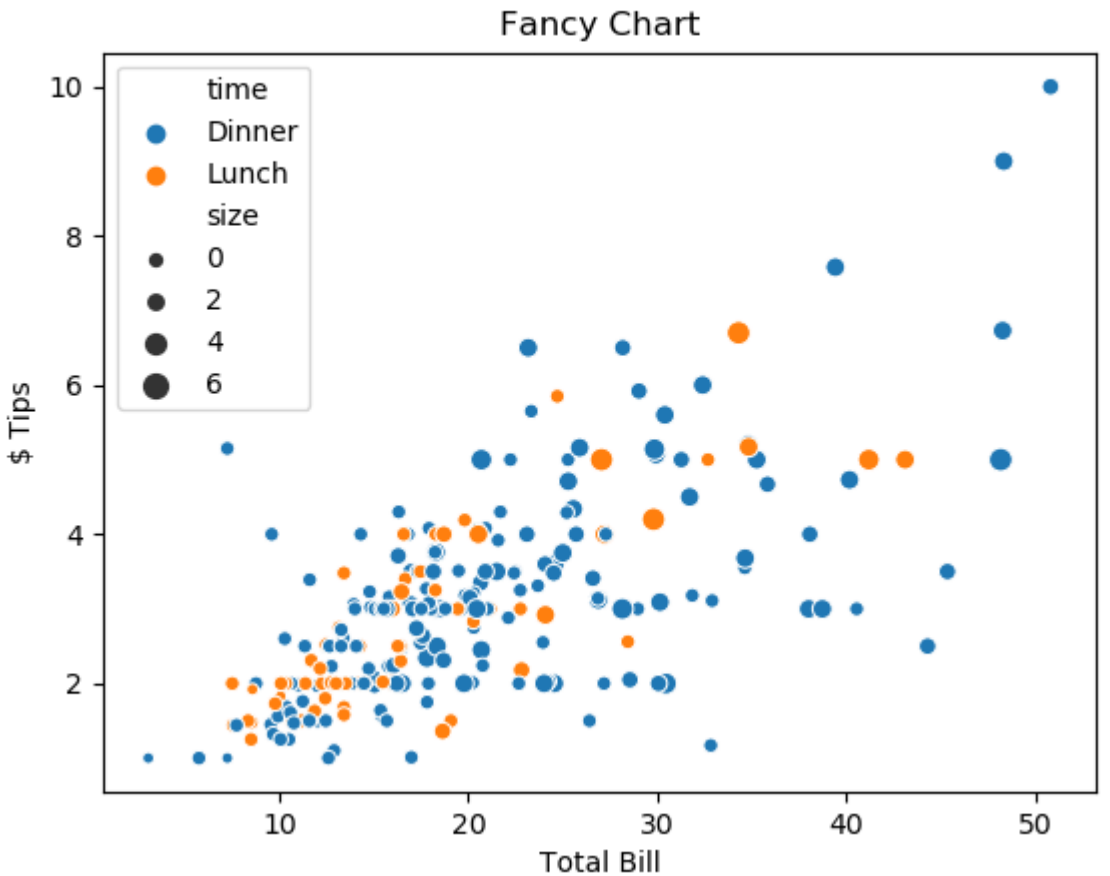


seaborn: scatterplot even fancier

```
ax = sns.scatterplot(
    x="total_bill", y="tip", hue="time", size="size", data=tips
)
plt.title("Fancy Chart")
plt.xlabel("Total Bill")
plt.ylabel("$ Tips")

plt.show()
```

Result that's fancier

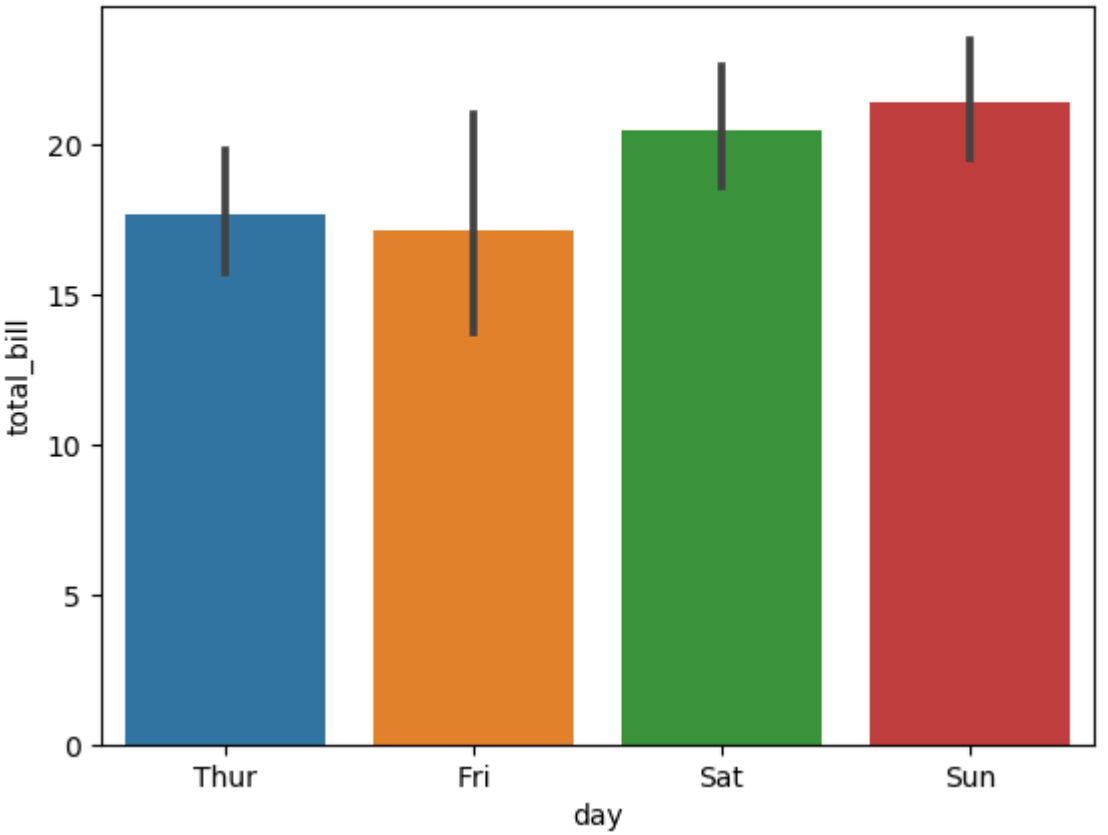


seaborn: barplot

```
ax = sns.barplot(x="day", y="total_bill", data=tips)

plt.show()
```

Barplot result

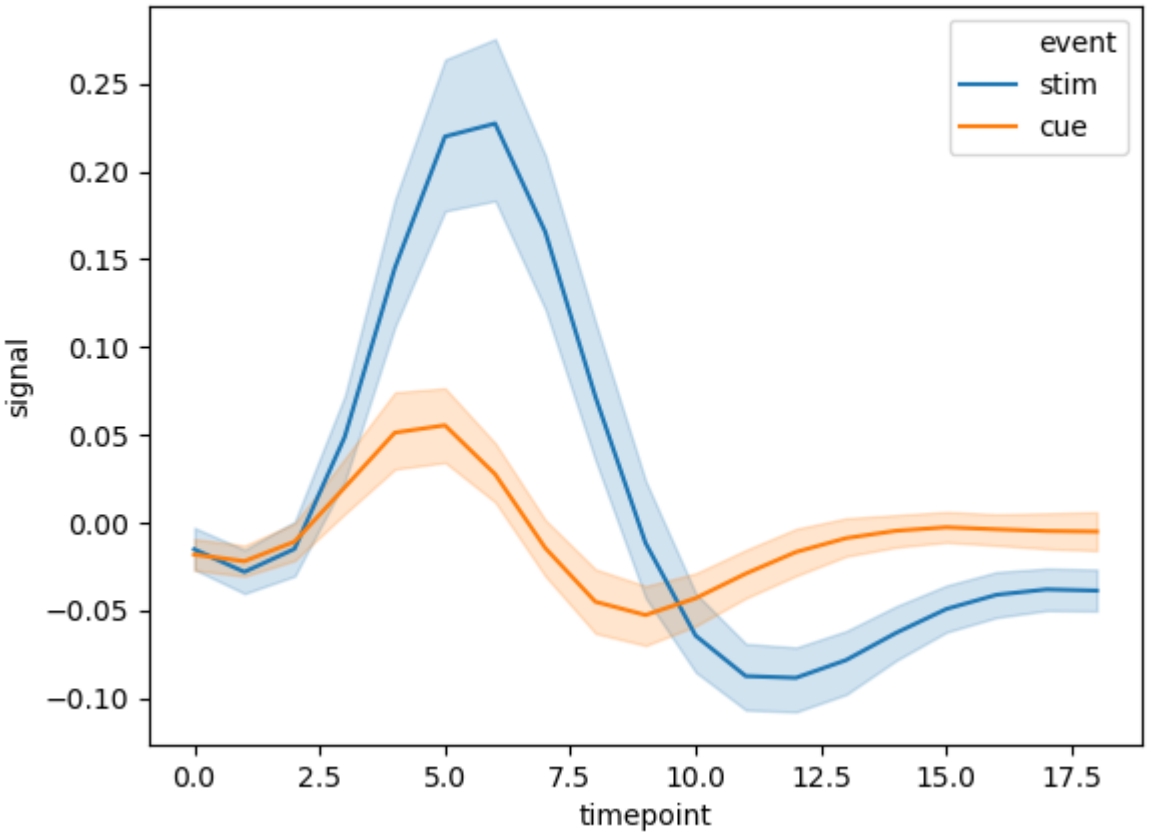


seaborn: lineplot

```
fmri = sns.load_dataset("fmri")
ax = sns.lineplot(x="timepoint", y="signal", hue="event", data=fmri)

plt.show()
```

Lineplot result

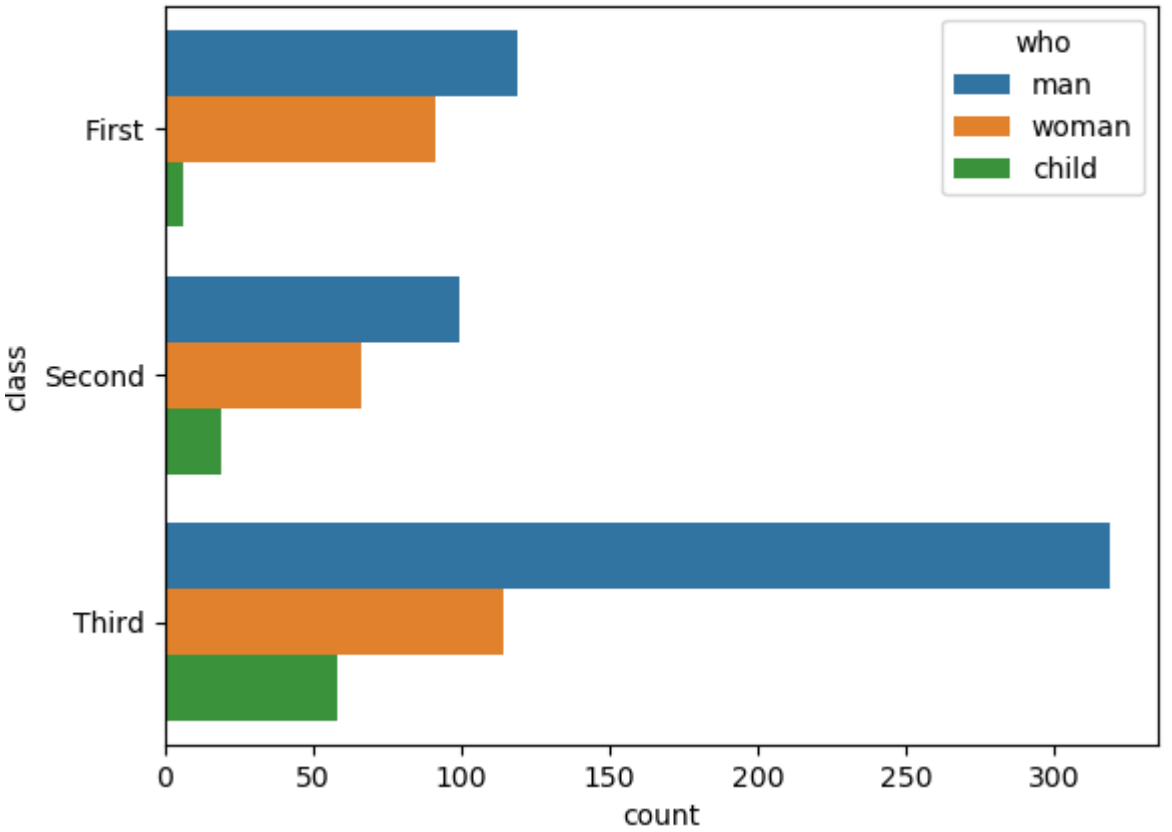


seaborn: countplot

```
titanic = sns.load_dataset("titanic")
ax = sns.countplot(y="class", hue="who", data=titanic)

plt.show()
```

Countplot result

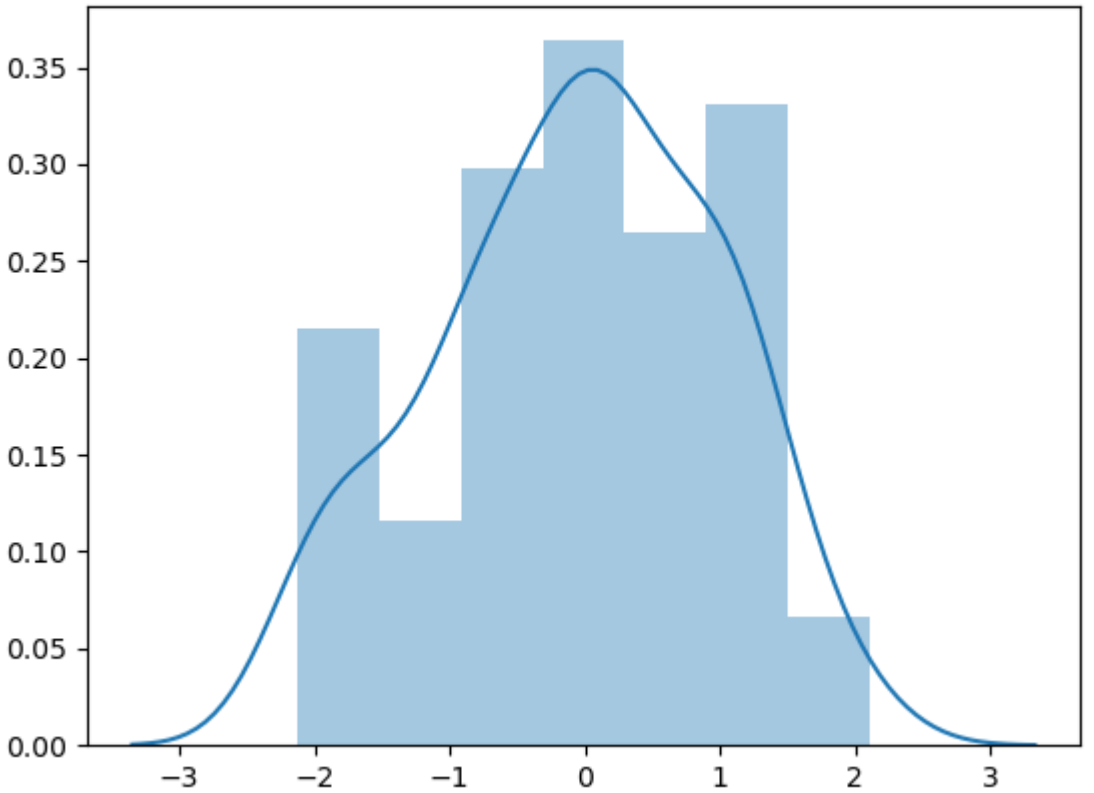


seaborn: distplot

```
import numpy as np
x = np.random.randn(100)
ax = sns.distplot(x)

plt.show()
```

Distplot result

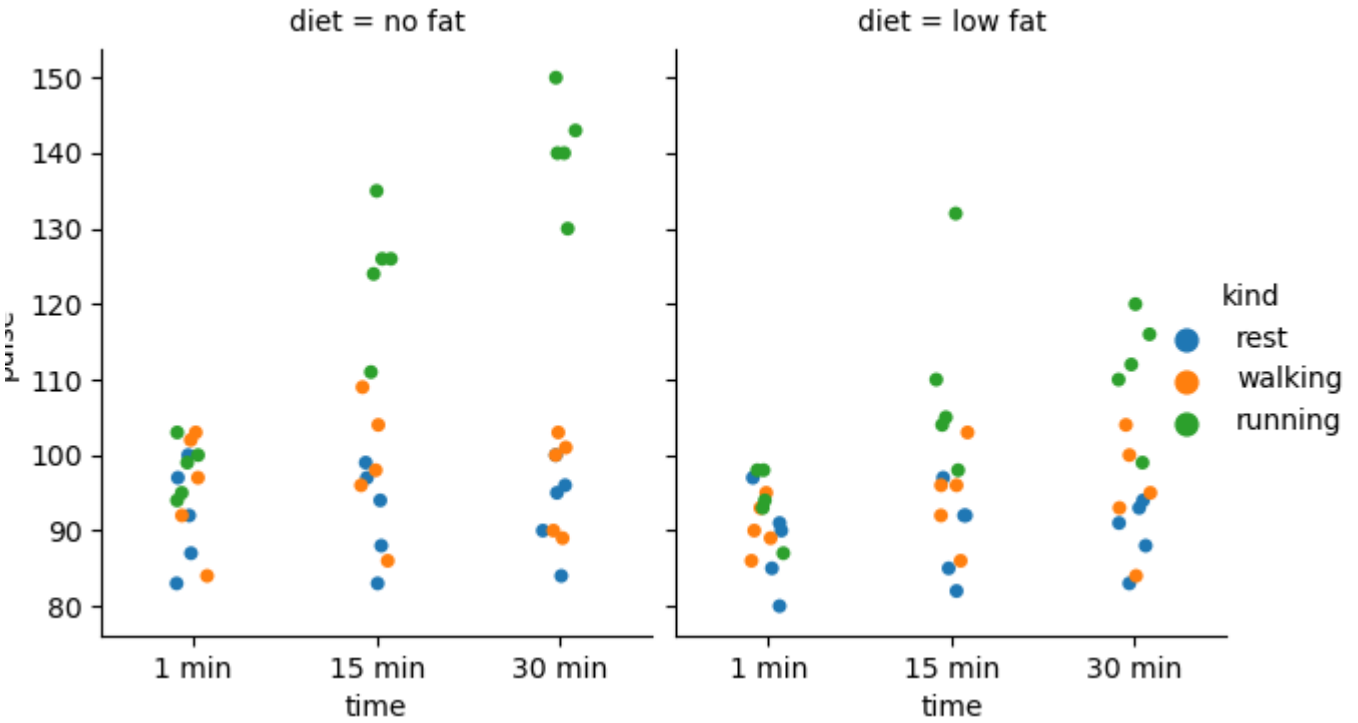


seaborn: catplot

```
exercise = sns.load_dataset("exercise")
g = sns.catplot(
    x="time", y="pulse", hue="kind", col="diet", data=exercise
)

plt.show()
```

Catplot result



seaborn: jointplot

A jointplot is a scatterplot with marginal histograms.

See example at https://scrapy-darksky-api.herokuapp.com/

What is association rule mining?

Association rule mining it a process to discover interesting relations between variables in large datasets.

In case of the Beer and Diapers story, it uncovers the pattern that **when diapers are bought, beer are also likely to be bought**.

Toolkit

mlxtend (machine learning extensions) is a Python library of useful tools for the day-to-day data science tasks

Installation

```
$ pip install mlxtend
```

mlxtend: mining association rules

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori


dataset = [
    ['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
    ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
    ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
    ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
    ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs'],
]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
```

```
frequent_itemsets
```

```
    support                      itemsets
0       0.8                       (Eggs)
1       1.0                (Kidney Beans)
2       0.6                       (Milk)
3       0.6                      (Onion)
4       0.6                     (Yogurt)
5       0.8          (Eggs, Kidney Beans)
6       0.6                 (Eggs, Onion)
7       0.6          (Milk, Kidney Beans)
8       0.6         (Kidney Beans, Onion)
9       0.6        (Yogurt, Kidney Beans)
10      0.6  (Eggs, Kidney Beans, Onion)
```

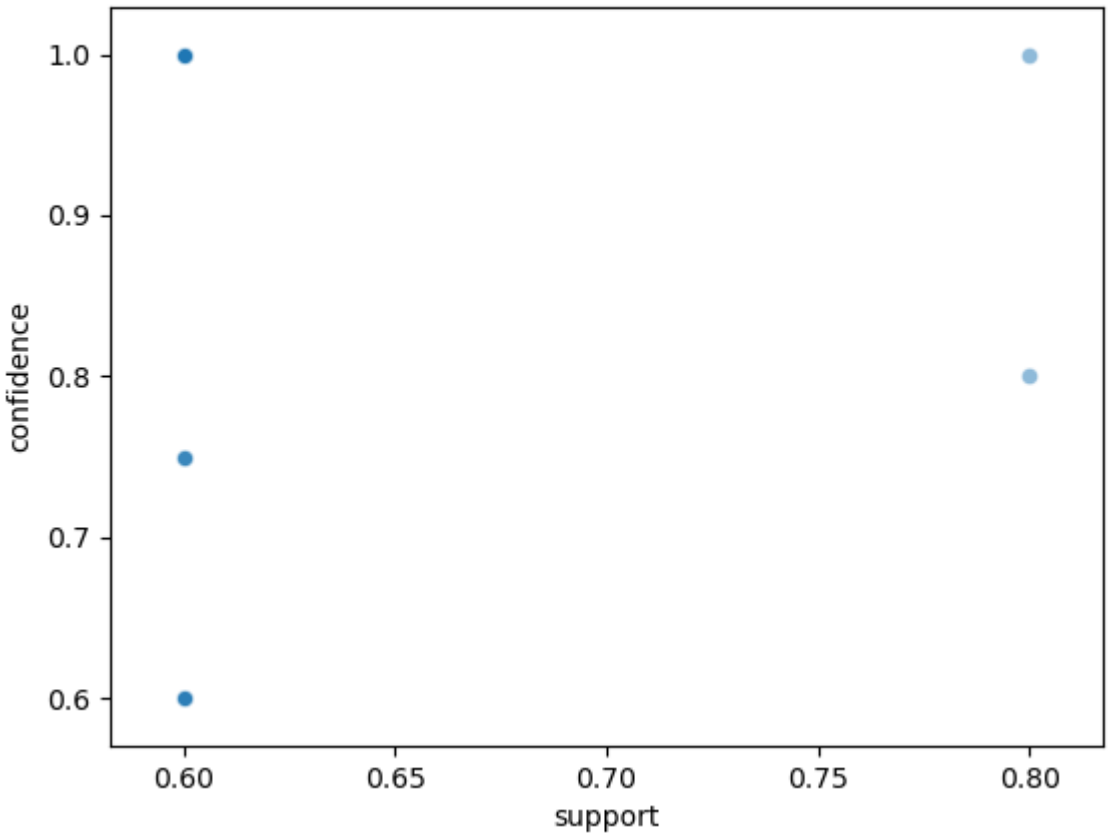
```
from mlxtend.frequent_patterns import association_rules

rules = association_rules(frequent_itemsets, min_threshold=0.1)
rules
```

```
                antecedents                consequents  ...
0                    (Eggs)             (Kidney Beans)  ...
1             (Kidney Beans)                    (Eggs)  ...
2                    (Eggs)                   (Onion)  ...
3                   (Onion)                    (Eggs)  ...
4                    (Milk)             (Kidney Beans)  ...
5             (Kidney Beans)                    (Milk)  ...
6             (Kidney Beans)                   (Onion)  ...
7                   (Onion)             (Kidney Beans)  ...
8                  (Yogurt)             (Kidney Beans)  ...
9             (Kidney Beans)                  (Yogurt)  ...
10   (Eggs, Kidney Beans)                    (Onion)  ...
11           (Eggs, Onion)             (Kidney Beans)  ...
12  (Onion, Kidney Beans)                    (Eggs)  ...
13                   (Eggs)  (Onion, Kidney Beans)  ...
14            (Kidney Beans)           (Eggs, Onion)  ...
15                  (Onion)   (Eggs, Kidney Beans)  ...
```

```
ax = sns.scatterplot(
    x="support", y="confidence", alpha=0.5, data=rules
)

plt.show()
```



```
length = frequent_itemsets["itemsets"].apply(len)
frequent_itemsets["length"] = length
frequent_itemsets
```

	support	itemsets	length
0	0.8	(Eggs)	1
1	1.0	(Kidney Beans)	1
2	0.6	(Milk)	1
3	0.6	(Onion)	1
4	0.6	(Yogurt)	1
5	0.8	(Eggs, Kidney Beans)	2
6	0.6	(Eggs, Onion)	2
7	0.6	(Milk, Kidney Beans)	2
8	0.6	(Kidney Beans, Onion)	2
9	0.6	(Yogurt, Kidney Beans)	2
10	0.6	(Eggs, Kidney Beans, Onion)	3

```
rules.sort_values("confidence", ascending=False).head(3)
```

```
  antecedents     consequents  ...
0      (Eggs)  (Kidney Beans)  ...
3     (Onion)          (Eggs)  ...
4      (Milk)  (Kidney Beans)  ...
```

Process

Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually split up into two separate steps:

1. A **minimum support** threshold is applied to find all frequent itemsets in a database.
2. A **minimum confidence** constraint is applied to these frequent itemsets in order to form rules.

Step 1 is very computational intensive.

Explanation

Given:

- transaction: T
- itemsets: X, Y
- association rule: $X \Rightarrow Y$

Support

$$\mathbf{supp}(X) = \frac{|\{t \in T; X \subseteq t\}|}{|T|}$$

Support is an indication of how frequently the itemset appears in the dataset.

Explanation cont'd

Confidence

$\mathbf{conf}(X \Rightarrow Y) = \mathbf{supp}(X \cup Y)/\mathbf{supp}(X)$

Confidence is an indication of how often the rule has been found to be true.

Explanation cont'd

Lift

$$\mathbf{lift}(X \Rightarrow Y) = \frac{\mathbf{supp}(X \cup Y)}{\mathbf{supp}(X) \times \mathbf{supp}(Y)}$$

Lift is the ratio of the observed support to that expected if X and Y were independent.

Questions?