Exercise1

Mean and median

Summary statistics are exactly what they sound like - they summarize many numbers in one statistic. For example, mean, median, minimum, maximum, and standard deviation are summary statistics. Calculating summary statistics allows you to get a better sense of your data, even if there's a lot of it.

sales is available and pandas is loaded as pd.

Instructions

- Explore your new DataFrame first by printing the first few rows of the sales DataFrame.
- Print information about the columns in sales.
- Print the mean of the weekly_sales column.
- Print the median of the weekly_sales column.

In [ ]:

```
# Print the head of the sales DataFrame
print(____)

# Print the info about the sales DataFrame
print(____)

# Print the mean of weekly_sales
print(____)

# Print the median of weekly_sales
print(____)

#_____________________________________________________#
#Solutions

# Print the head of the sales DataFrame
print(sales.head())

# Print the info about the sales DataFrame
print(sales.info())

# Print the mean of weekly_sales
print(sales["weekly_sales"].mean())

# Print the median of weekly_sales
print(sales["weekly_sales"].median())
```

Exercise2

Summarizing dates

Summary statistics can also be calculated on date columns that have values with the data type datetime64. Some summary statistics — like mean — don't make a ton of sense on dates, but others are super helpful, for example, minimum and maximum, which allow you to see what time range your data covers.

sales is available and pandas is loaded as pd.

Instructions

- Print the maximum of the date column.
- Print the minimum of the date column.

In [ ]:

```
# Print the maximum of the date column
____

# Print the minimum of the date column
____
#_____________________________________________________#
#Solution

# Print the maximum of the date column
print(sales["date"].max())

# Print the minimum of the date column
print(sales["date"].min())
```

Exercise3

Efficient summaries

While pandas and NumPy have tons of functions, sometimes, you may need a different function to summarize your data.

The `.agg()` method allows you to apply your own custom functions to a DataFrame, as well as apply functions to more than one column of a DataFrame at once, making your aggregations super-efficient. For example,

```
df['column'].agg(function)
```

In the custom function for this exercise, "IQR" is short for inter-quartile range, which is the 75th percentile minus the 25th percentile. It's an alternative to standard deviation that is helpful if your data contains outliers.

sales is available and pandas is loaded as pd.

Instructions

- Use the custom iqr function defined for you along with .agg() to print the IQR of the temperature_c column of sales.

- Update the column selection to use the custom iqr function with .agg() to print the IQR of temperature_c, fuel_price_usd_per_l, and unemployment, in that order.
- Update the aggregation functions called by .agg(): include iqr and np.median in that order.

In [ ]:

```
# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Print IQR of the temperature_c column
print(____)

# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Update to print IQR of temperature_c, fuel_price_usd_per_l, & unemployment
print(sales[["temperature_c", ____, ____]].agg(iqr))



#_____________________________________________________#
#Solution

# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Print IQR of the temperature_c column
print(sales['temperature_c'].agg(iqr))

# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Update to print IQR of temperature_c, fuel_price_usd_per_l, & unemployment
print(sales[["temperature_c", 'fuel_price_usd_per_l','unemployment']].agg(iqr))

# Import NumPy and create custom IQR function
import numpy as np
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Update to print IQR and median of temperature_c, fuel_price_usd_per_l, & unemployment
print(sales[["temperature_c", "fuel_price_usd_per_l", "unemployment"]].agg([iqr, np.median]
```

Exercise4

Cumulative statistics

Cumulative statistics can also be helpful in tracking summary statistics over time. In this exercise, you'll calculate the cumulative sum and cumulative max of a department's weekly sales, which will allow you to identify what the total sales were so far as well as what the highest weekly sales were so far.

A DataFrame called `sales_1_1` has been created for you, which contains the sales data for department 1 of store 1. pandas is loaded as pd.

Instructions

- Sort the rows of `sales_1_1` by the date column in ascending order.
- Get the cumulative sum of weekly_sales and add it as a new column of sales_1_1 called cum_weekly_sales.
- Get the cumulative maximum of weekly_sales, and add it as a column called cum_max_sales.
- Print the date, weekly_sales, cum_weekly_sales, and cum_max_sales columns.

In [ ]:

```
# Sort sales_1_1 by date
sales_1_1 = ____

# Get the cumulative sum of weekly_sales, add as cum_weekly_sales col
sales_1_1[____] = ____

# Get the cumulative max of weekly_sales, add as cum_max_sales col
____

# See the columns you calculated
print(sales_1_1[["date", "weekly_sales", "cum_weekly_sales", "cum_max_sales"]])

#_____________________________________________________#
#Solution

# Sort sales_1_1 by date
sales_1_1 = sales_1_1.sort_values("date")

# Get the cumulative sum of weekly_sales, add as cum_weekly_sales col
sales_1_1["cum_weekly_sales"] = sales_1_1["weekly_sales"].cumsum()

# Get the cumulative max of weekly_sales, add as cum_max_sales col
sales_1_1["cum_max_sales"] = sales_1_1["weekly_sales"].cummax()

# See the columns you calculated
print(sales_1_1[["date", "weekly_sales", "cum_weekly_sales", "cum_max_sales"]])
```

Exercise5

Dropping duplicates

Removing duplicates is an essential skill to get accurate counts because often, you don't want to count the same thing multiple times. In this exercise, you'll create some new DataFrames using unique values from sales.

sales is available and pandas is imported as pd.

Instructions

- Remove rows of sales with duplicate pairs of store and type and save as store_types and print the head.
- Remove rows of sales with duplicate pairs of store and department and save as store_depts and print the head.

- Subset the rows that are holiday weeks using the is_holiday column, and drop the duplicate dates, saving as holiday_dates.
- Select the date column of holiday_dates, and print.

In [ ]:

```
# Drop duplicate store/type combinations
store_types = ____
print(store_types.head())

# Drop duplicate store/department combinations
store_depts = ____
print(store_depts.head())

# Subset the rows where is_holiday is True and drop duplicate dates
holiday_dates = sales[sales[____]].drop_duplicates(____)

# Print date col of holiday_dates
print(____)

#_____#
#Solution

# Drop duplicate store/type combinations
store_types = sales.drop_duplicates(subset=["store",'type'])
print(store_types.head())

# Drop duplicate store/department combinations
store_depts = sales.drop_duplicates(subset=["store",'department'])

print(store_depts.head())

# Subset the rows where is_holiday is True and drop duplicate dates
holiday_dates = sales[sales['is_holiday']].drop_duplicates('date')

# Print date col of holiday_dates
print(holiday_dates['date'])
```

Exercise6

Counting categorical variables

Counting is a great way to get an overview of your data and to spot curiosities that you might not notice otherwise. In this exercise, you'll count the number of each type of store and the number of each department number using the DataFrames you created in the previous exercise:

```
#Drop duplicate store/type combinations
store_types = sales.drop_duplicates(subset=["store", "type"])

#Drop duplicate store/department combinations
store_depts = sales.drop_duplicates(subset=["store", "department"])
```

The `store_types` and `store_depts` DataFrames you created in the last exercise are available, and pandas is imported as pd.

Instructions

- Count the number of stores of each store type in store_types.
- Count the proportion of stores of each store type in store_types.
- Count the number of different departments in store_depts, sorting the counts in descending order.
- Count the proportion of different departments in store_depts, sorting the proportions in descending order.

In [ ]:

```
# Count the number of stores of each type
store_counts = ____
print(store_counts)

# Get the proportion of stores of each type
store_props = ____
print(store_props)

# Count the number of each department number and sort
dept_counts_sorted = ____
print(dept_counts_sorted)

# Get the proportion of departments of each number and sort
dept_props_sorted = ____.____(sort=____, normalize=____)
print(dept_props_sorted)

#_____#
#Solution

# Count the number of stores of each type
store_counts = store_types["type"].value_counts()
print(store_counts)

# Get the proportion of stores of each type
store_props = store_types["type"].value_counts(normalize=True)
print(store_props)

# Count the number of each department number and sort
dept_counts_sorted = store_depts["department"].value_counts(sort=True)
print(dept_counts_sorted)

# Get the proportion of departments of each number and sort
dept_props_sorted = store_depts["department"].value_counts(sort=True, normalize=True)
print(dept_props_sorted)
```

Exercise7

What percent of sales occurred at each store type?

While `.groupby()` is useful, you can calculate grouped summary statistics without it.

Walmart distinguishes three types of stores: "supercenters," "discount stores," and "neighborhood markets," encoded in this dataset as type "A," "B," and "C." In this exercise, you'll calculate the total sales made at each store type, without using `.groupby()`. You can then use these numbers to see what proportion of Walmart's total sales were made at each type.

sales is available and pandas is imported as pd.

Instructions

- Calculate the total weekly_sales over the whole dataset.
- Subset for type "A" stores, and calculate their total weekly sales.
- Do the same for type "B" and type "C" stores.
- Combine the A/B/C results into a list, and divide by sales_all to get the proportion of sales by type.

In [ ]:

```
# Calc total weekly sales
sales_all = ____["____"].____()

# Subset for type A stores, calc total weekly sales
sales_A = ____[____["____"] == "____"]["____"].____()

# Subset for type B stores, calc total weekly sales
sales_B = ____

# Subset for type C stores, calc total weekly sales
sales_C = ____

# Get proportion for each type
sales_propn_by_type = [sales_A, ____, ____] / ____
print(sales_propn_by_type)

#_____________________________________________________#
#Solution

# Calc total weekly sales
sales_all = sales["weekly_sales"].sum()

# Subset for type A stores, calc total weekly sales
sales_A = sales[sales["type"] == "A"]["weekly_sales"].sum()

# Subset for type B stores, calc total weekly sales
sales_B = sales[sales["type"] == "B"]["weekly_sales"].sum()

# Subset for type C stores, calc total weekly sales
sales_C = sales[sales["type"] == "C"]["weekly_sales"].sum()

# Get proportion for each type
sales_propn_by_type = [sales_A, sales_B, sales_C] / sales_all
print(sales_propn_by_type)
```

Exercise8

Calculations with .groupby()

The `.groupby()` method makes life much easier. In this exercise, you'll perform the same calculations as last time, except you'll use the `.groupby()` method. You'll also perform calculations on data grouped by two variables to see if sales differ by store type depending on if it's a holiday week or not.

sales is available and pandas is loaded as pd.

Instructions

- Group sales by "type", take the sum of "weekly_sales", and store as sales_by_type.
- Calculate the proportion of sales at each store type by dividing by the sum of sales_by_type. Assign to sales_propn_by_type.
- Group sales by "type" and "is_holiday", take the sum of weekly_sales, and store as sales_by_type_is_holiday.

In [ ]:

```
# Group by type; calc total weekly sales
sales_by_type = ____.____("____")["____"].____()

# Get proportion for each type
sales_propn_by_type = ____ / sum(____)
print(sales_propn_by_type)

# From previous step
sales_by_type = sales.groupby("type")["weekly_sales"].sum()

# Group by type and is_holiday; calc total weekly sales
sales_by_type_is_holiday = ____
print(sales_by_type_is_holiday)

#_____#
#Solution

# Group by type; calc total weekly sales
sales_by_type = sales.groupby("type")["weekly_sales"].sum()

# Get proportion for each type
sales_propn_by_type = sales_by_type / sum(sales_by_type)
print(sales_propn_by_type)

# From previous step
sales_by_type = sales.groupby("type")["weekly_sales"].sum()

# Group by type and is_holiday; calc total weekly sales
sales_by_type_is_holiday = sales.groupby(['type','is_holiday'])["weekly_sales"].sum()
print(sales_by_type_is_holiday)
```

Exercise9

Multiple grouped summaries

Earlier in this chapter, you saw that the `.agg()` method is useful to compute multiple statistics on multiple variables. It also works with grouped data. NumPy, which is imported as np, has many different summary statistics functions, including: np.min, np.max, np.mean, and np.median.

sales is available and pandas is imported as pd.

Instructions

- Import numpy with the alias np.

- Get the min, max, mean, and median of weekly_sales for each store type using .groupby() and .agg(). Store this as sales_stats. Make sure to use numpy functions!
- Get the min, max, mean, and median of unemployment and fuel_price_usd_per_l for each store type. Store this as unemp_fuel_stats.

In [ ]:

```
# Import numpy with the alias np
____

# For each store type, aggregate weekly_sales: get min, max, mean, and median
sales_stats = ____

# Print sales_stats
print(sales_stats)

# For each store type, aggregate unemployment and fuel_price_usd_per_l: get min, max, mean,
unemp_fuel_stats = ____

# Print unemp_fuel_stats
print(unemp_fuel_stats)

#_____________________________________________________#
#Solution

# Import numpy with the alias np
import numpy as np

# For each store type, aggregate weekly_sales: get min, max, mean, and median
sales_stats = sales.groupby("type")["weekly_sales"].agg([np.min, np.max, np.mean, np.median

# Print sales_stats
print(sales_stats)

# For each store type, aggregate unemployment and fuel_price_usd_per_l: get min, max, mean,
unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([np.

# Print unemp_fuel_stats
print(unemp_fuel_stats)
```

Exercise10

Pivoting on one variable

Pivot tables are the standard way of aggregating data in spreadsheets. In pandas, pivot tables are essentially just another way of performing grouped calculations. That is, the `.pivot_table()` method is just an alternative to `.groupby()`.

In this exercise, you'll perform calculations using `.pivot_table()` to replicate the calculations you performed in the last lesson using `.groupby()`.

sales is available and pandas is imported as pd.

Instructions

- Get the mean `weekly_sales` by type using `.pivot_table()` and store as `mean_sales_by_type`.
- Get the `mean` and `median` (using NumPy functions) of `weekly_sales` by type using `.pivot_table()` and store as `mean_med_sales_by_type`.
- Get the `mean` of weekly_sales by type and `is_holiday` using `.pivot_table()` and store as `mean_sales_by_type_holiday`.

In [ ]:

```
# Pivot for mean weekly_sales for each store type
mean_sales_by_type = sales.____

# Print mean_sales_by_type
print(mean_sales_by_type)

# Import NumPy as np
import numpy as np

# Pivot for mean and median weekly_sales for each store type
mean_med_sales_by_type = sales.pivot_table(____)

# Print mean_med_sales_by_type
print(mean_med_sales_by_type)

# Pivot for mean weekly_sales by store type and holiday
mean_sales_by_type_holiday = sales.pivot_table(____)

# Print mean_sales_by_type_holiday
print(mean_sales_by_type_holiday)

#_____________________________________________________#
#Solution

# Pivot for mean weekly_sales for each store type
mean_sales_by_type = sales.pivot_table(values='weekly_sales', index='type')

# Print mean_sales_by_type
print(mean_sales_by_type)

# Import NumPy as np
import numpy as np

# Pivot for mean and median weekly_sales for each store type
mean_med_sales_by_type = sales.pivot_table(values='weekly_sales', index='type', aggfunc=[np

# Print mean_med_sales_by_type
print(mean_med_sales_by_type)

# Pivot for mean weekly_sales by store type and holiday
mean_sales_by_type_holiday = sales.pivot_table(values='weekly_sales', index='type', columns

# Print mean_sales_by_type_holiday
print(mean_sales_by_type_holiday)
```

Exercise11

Fill in missing values and sum values with pivot tables

The `.pivot_table()` method has several useful arguments, including `fill_value` and `margins`.

- `fill_value` replaces missing values with a real value (known as imputation). What to replace missing values with is a topic big enough to have its own course (Dealing with Missing Data in Python), but the simplest thing to do is to substitute a dummy value.
- `margins` is a shortcut for when you pivoted by two variables, but also wanted to pivot by each of those variables separately: it gives the row and column totals of the pivot table contents.

In this exercise, you'll practice using these arguments to up your pivot table skills, which will help you crunch numbers more efficiently!

sales is available and pandas is imported as pd.

Instructions

- Print the mean weekly_sales by department and type, filling in any missing values with 0.
- Print the mean weekly_sales by department and type, filling in any missing values with 0 and summing all rows and columns.

In [ ]:

```
# Print mean weekly_sales by department and type; fill missing values with 0
print(sales.pivot_table(____))

# Print the mean weekly_sales by department and type; fill missing values with 0s; sum all
print(sales.pivot_table(values="weekly_sales", index="department", columns="type", ____))
#_____________________________________________________#
#Solution

# Print mean weekly_sales by department and type; fill missing values with 0
print(sales.pivot_table(values='weekly_sales', index='department',columns='type',fill_value

# Print the mean weekly_sales by department and type; fill missing values with 0s; sum all
print(sales.pivot_table(values="weekly_sales", index="department", columns="type",fill_valu
```