

# Data visualization with Python and JavaScript

Diving into exploratory data analysis with Python, a JavaScript library for data visualization, and Jupyter



Veronika Rovnik  
Sep 3 · 6 min read

STEP-BY-STEP GUIDE

## DATA VISUALIZATION WITH PYTHON AND JAVASCRIPT



Any data science or data analytics project can be generally described with the following steps:

1. Acquiring a business understanding & defining the goal of a project
2. Getting data
3. Preprocessing and exploring data
4. Improving data, e.g., by feature engineering
5. Visualizing data
6. Building a model
7. Deploying the model
8. Scoring its performance

This time, I would like to bring your attention to the data cleaning and exploration phase since it's a step which value is hard to measure, but the impact it brings is difficult to overestimate. Insights gained during this stage can affect all further work.

There are multiple ways you can start exploratory data analysis with:

1. Load data and preprocess it: clean it from unnecessary artifacts, deal with missing values. Make your dataset comfortable to work with.
2. Visualize as much data as possible using different kinds of plots & a pivot table.

## Purpose

In this tutorial, I would like to show **how to prepare your data** with Python and explore it using a JavaScript library for **data visualization**. To get the most value out of exploration, I recommend using **interactive visualizations** since they make exploring your data faster and more comfortable.

Hence, we will present data in an interactive **pivot table** and **pivot charts**.

Hopefully, this approach will help you facilitate the data analysis and visualization process in Jupyter Notebook.

## Set up your environment

Run your Jupyter Notebook and let's start. If Jupyter is not installed on your machine, choose the way to get it.

## Get your data

Choosing the data set to work with is the number one step.

If your data is already cleaned and ready to be visualized, jump to the **Visualization** section.

For demonstration purposes, I've chosen the data for the prediction of Bike Sharing Demand. It's provided as data for the Kaggle's competition.

## Imports for this tutorial

Classically, we will use the "pandas" library to read data into a dataframe.

Additionally, we will need `json` and `IPython.display` modules. The former will help us serialize/deserialize data and the latter — render HTML in the cells.

Here's the full code sample with imports we need:

```
from IPython.display import HTML

import json

import pandas as pd
```

## Read data

```
df = pd.read_csv('train.csv')
```

## Clean & preprocess data

Before starting data visualization, it's a good practice to see what's going on in the data.

```
df.head()
```

```
In [38]: df.head()
```

Out[38]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
df.info()
```

```
In [3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
datetime      10886 non-null object
season        10886 non-null int64
holiday        10886 non-null int64
workingday     10886 non-null int64
weather        10886 non-null int64
temp          10886 non-null float64
atemp         10886 non-null float64
humidity       10886 non-null int64
windspeed     10886 non-null float64
casual         10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: float64(3), int64(8), object(1)
```

```
memory usage: 1020.7+ KB
```

First, we should check the percentage of missing values.

```
missing_percentage = df.isnull().sum() * 100 / len(df)
```

There are a lot of strategies to follow when dealing with missing data. Let me mention the main ones:

1. Dropping missing values. The only reason to follow this approach is when you need to quickly remove all NaNs from the data.
2. Replacing NaNs with values. This is called **imputation**. A common decision is to replace missing values with zeros or with a mean value.

Luckily, we don't have any missing values in the dataset. But if your data has, I suggest you look into a quick guide with the pros and cons of different imputation techniques.

## Manage features data types

Let's convert the type of "datetime" column from object to datetime:

```
df['datetime'] = pd.to_datetime(df['datetime'])
```

Now we are able to engineer new features based on this column, for example:

- a day of the week
- a month
- an hour

```
df['weekday'] = df['datetime'].dt.dayofweek
```

```
df['hour'] = df['datetime'].dt.hour
```

```
df['month'] = df['datetime'].dt.month
```

These features can be used further to figure out trends in rent.

Next, let's convert string types to categorical:

```
categories = ['season', 'workingday', 'weekday', 'hour', 'month',  
'weather', 'holiday']  
  
for category in categories:  
    df[category] = df[category].astype('category')
```

Read more about when to use the categorical data type [here](#).

Now, let's make values of categorical more meaningful by replacing numbers with their categorical equivalents:

```
df['season'] = df['season'].replace([1, 2, 3, 4], ['spring',  
'summer', 'fall', 'winter'])  
  
df['holiday'] = df['holiday'].replace([0, 1], ['No', 'Yes'])
```

By doing so, it will be easier for us to interpret data visualization later on. We won't need to look up the meaning of a category each time we need it.

## Visualize data with a pivot table and charts

Now that you cleaned the data, let's visualize it.

The data visualization type depends on the question you are asking.

In this tutorial, we'll be using:

- a pivot table for tabular data visualization
- a bar chart

## Prepare data for the pivot table

Before loading data to the pivot table, convert the dataframe to an array of JSON objects. For this, use the `to_json()` function from the `json` module.

The `records` orientation is needed to make sure the data is aligned according to the format the pivot table requires.

```
json_data = df.to_json(orient="records")
```

## Create a pivot table

Next, define a pivot table object and feed it with the data. Note that the data has to be deserialized using the `loads()` function that decodes JSON:

```

pivot_table = {
    "container": "#pivot-container",
    "componentFolder": "https://cdn.flexmonster.com/",
    "toolbar": True,
    "report": {
        "dataSource": {
            "type": "json",
            "data": json.loads(json_data)
        },
        "slice": {
            "rows": [{
                "uniqueName": "weekday"
            }],
            "columns": [{
                "uniqueName": "[Measures]"
            }],
            "measures": [{
                "uniqueName": "count",
                "aggregation": "median"
            }],
            "sorting": {
                "column": {
                    "type": "desc",
                    "tuple": [],
                    "measure": {
                        "uniqueName": "count",
                        "aggregation": "median"
                    }
                }
            }
        }
    }
}

```

In the above pivot table initialization, we specified a simple report that consists of a slice (a set of fields visible on the grid), data source, options, formats, etc. We also specified a container where the pivot table should be rendered. The container will be defined a bit later.

Plus, here we can add a mapping object to prettify the field captions or set their data types. Using this object eliminates the need in modifying the data source.

Next, convert the pivot table object to a JSON-formatted string to be able to pass it for rendering in the HTML layout:

```
pivot_json_object = json.dumps(pivot_table)
```

## Define a dashboard layout

Define a function that renders the pivot table in the cell:

```
1  def render_pivot_table(pivot_table):
2      code = '''
3      <head>
4          <script src="https://cdn.flexmonster.com/flexmonster.js"></script>
5      </head>
6      <body>
7          <h1 style="margin-bottom: 15px;">Exploratory data analysis with a pivot table</h1>
8          <div id="pivot-container"></div>
9          <script>
10             new Flexmonster({0});
11          </script>
12      </body>
13      '''.format(pivot_table)
14      #convert the code string to HTML
15      return HTML(code)
```

html\_layout.py hosted with ❤ by GitHub

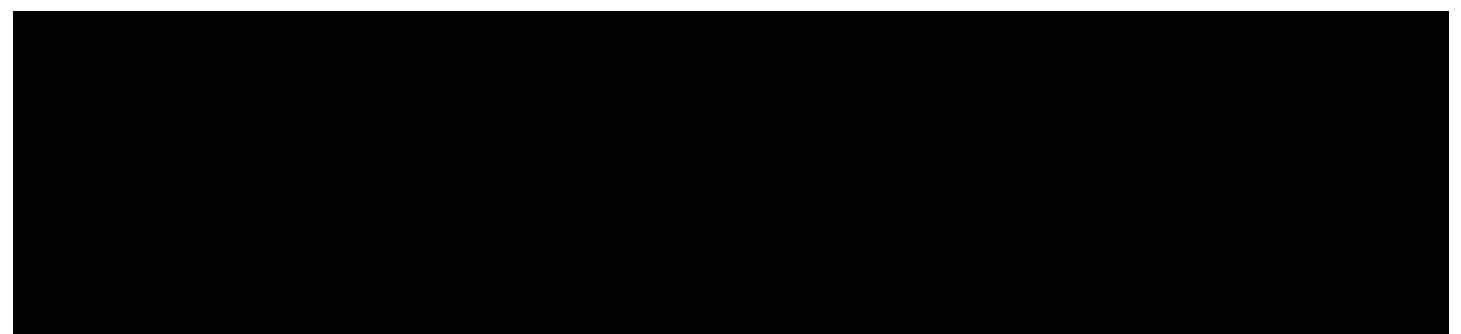
[view raw](#)

In this function, we call `HTML()` from the `IPython.display` module — it will render the layout enclosed into a multi-line string.

Next, let's call this function and pass to it the pivot table previously encoded into JSON:

```
render_pivot_table(pivot_json_object)
```

Likewise, you can create and render **as many data visualization components as you need**. For example, pivot charts that visualize aggregated data:



## What's next

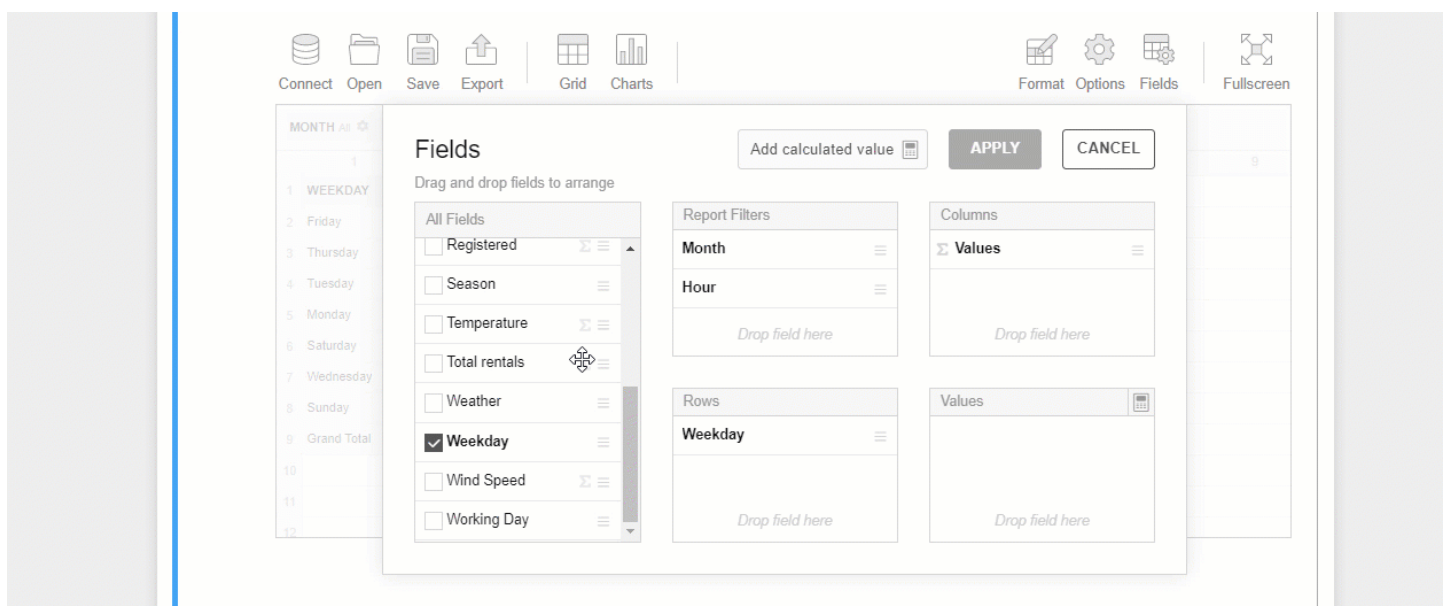
Now that you embedded the pivot table into Jupyter, it's time to start exploring your data:

- *drag and drop fields to rows, columns, and measures of the pivot table*
- *set Excel-like filtering*
- *highlight important values with conditional formatting*

At any moment, you can save your results to a **JSON** or **PDF/Excel/HTML** report.

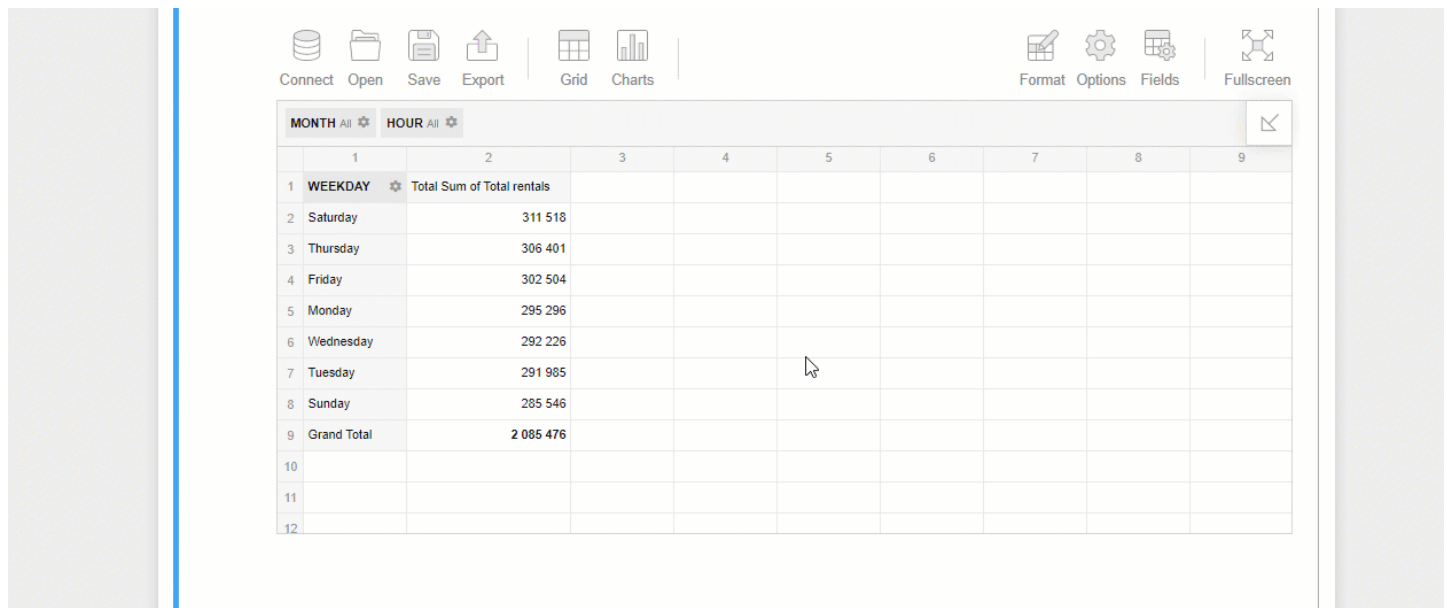
## Examples

Here is how you can try identifying trends on bikes usage depending on the day of the week:





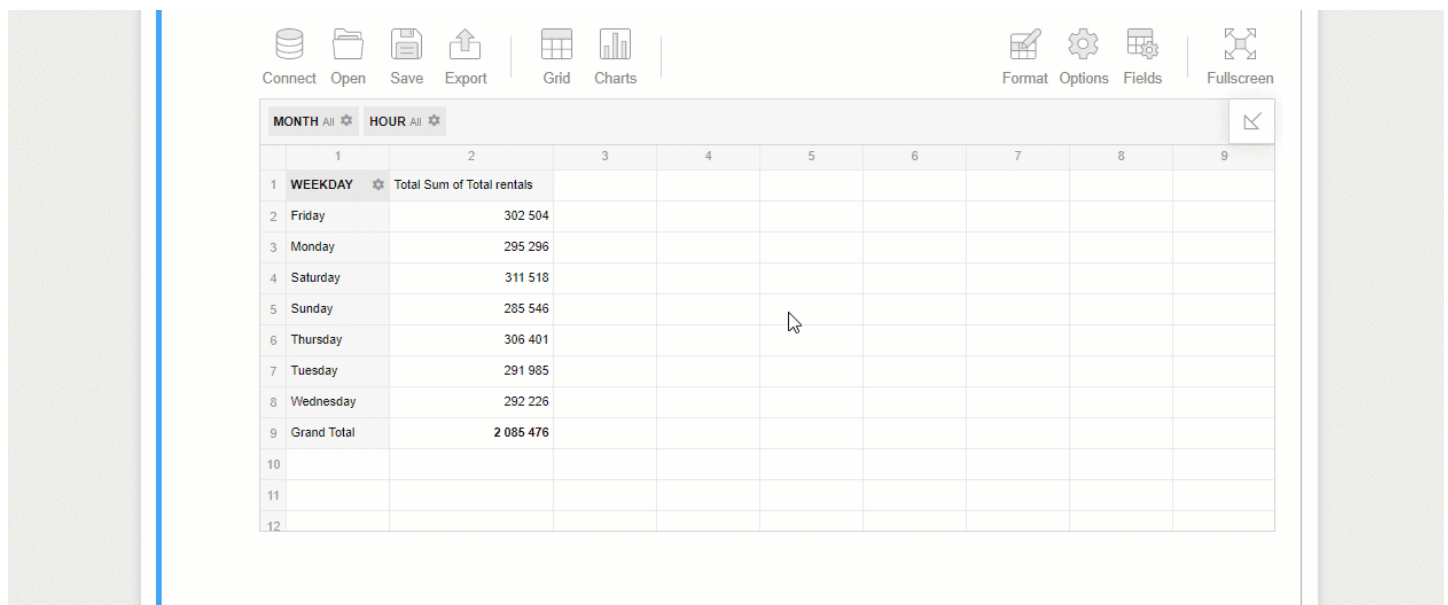
You can also figure out if any weather conditions affect the number of rents by registered and unregistered users:



The screenshot shows a data visualization tool interface. At the top, there are icons for 'Connect', 'Open', 'Save', 'Export', 'Grid', and 'Charts'. On the right, there are icons for 'Format', 'Options', 'Fields', and 'Fullscreen'. Below these icons, there are tabs for 'MONTH All' and 'HOUR All'. The main area displays a pivot table with the following data:

	1	2	3	4	5	6	7	8	9
1 WEEKDAY	Total Sum of Total rentals								
2 Saturday		311 518							
3 Thursday		306 401							
4 Friday		302 504							
5 Monday		295 296							
6 Wednesday		292 226							
7 Tuesday		291 985							
8 Sunday		285 546							
9 Grand Total		2 085 476							
10									
11									
12									

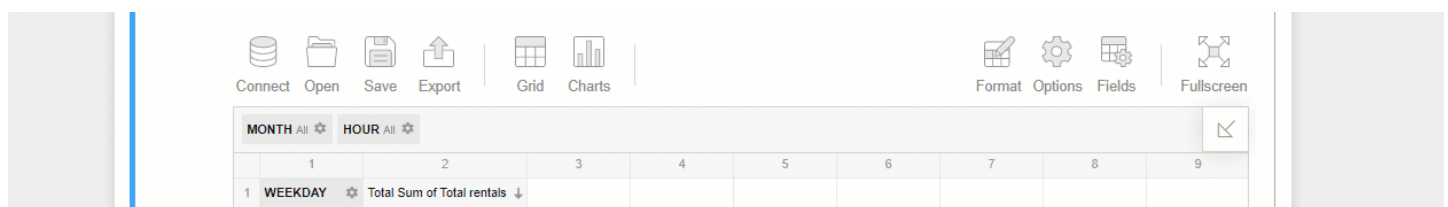
To dig deeper into the data, **drill through** aggregated values by double-clicking and see the raw records they are composed of:



The screenshot shows the same data visualization tool interface as the previous one. The 'Friday' row (row 4) is selected, and the pivot table displays the following data:

	1	2	3	4	5	6	7	8	9
1 WEEKDAY	Total Sum of Total rentals								
2 Friday		302 504							
3 Monday		295 296							
4 Saturday		311 518							
5 Sunday		285 546							
6 Thursday		306 401							
7 Tuesday		291 985							
8 Wednesday		292 226							
9 Grand Total		2 085 476							
10									
11									
12									

Or simply switch to the pivot charts mode and give your data an even more comprehensible look:



The screenshot shows the same data visualization tool interface as the previous ones. The 'Friday' row (row 4) is selected, and the pivot table displays the following data:

	1	2	3	4	5	6	7	8	9
1 WEEKDAY	Total Sum of Total rentals								
2 Friday		302 504							
3 Monday		295 296							
4 Saturday		311 518							
5 Sunday		285 546							
6 Thursday		306 401							
7 Tuesday		291 985							
8 Wednesday		292 226							
9 Grand Total		2 085 476							
10									
11									
12									

2	Saturday	311 518							
3	Thursday	306 401							
4	Friday	302 504							
5	Monday	295 296							
6	Wednesday	292 226							
7	Tuesday	291 985							
8	Sunday	285 546							
9	Grand Total	2 085 476							
10									
11									
12									

## Bringing it all together

By completing this tutorial, you learned a new way to interactively explore your multi-dimensional data in Jupyter Notebook using Python and the JavaScript data visualization library. I hope this will make your exploration process more insightful than before.

## Useful links

- [Jupyter Notebook sample](#)
- [Pivot Table live demo](#)
- [Pythonic Data Cleaning With Pandas and NumPy](#)
- [Exploratory Data Analysis With Python and Pandas on Coursera](#)

[Data Visualization](#)[JavaScript](#)[Python](#)[Data Science](#)[Data](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

