Member-only story

How to Create PDF Reports with Python — The Essential Guide

Create PDF reports with beautiful visualizations in 10 minutes or less

Dario Radečić · Follow

Published in Towards Data Science

6 min read · Jan 18, 2021

Listen Share

Photo by Jesse G-C on Unsplash

Reports are everywhere, so any tech professional must know how to create them. It's a tedious and time-consuming task, which makes it a perfect candidate for automation with Python.

You can benefit from an automated report generation whether you're a data scientist or a software developer. For example, data scientists might use reports to show performance or explanations of machine learning models.

This article will teach you how to make data-visualization-based reports and save them as PDFs. To be more precise, you'll learn how to combine multiple data visualizations (dummy sales data) into a single PDF file.

And the best thing is — it's easier than you think!

The article is structured as follows:

- Data generation
- Data visualization
- Create a PDF page structure
- Create PDF reports
- Conclusion

You can download the Notebook with the source code here.

Data generation

You can't have reports without data. That's why you'll have to generate some first — more on that in a bit.

Let's start with the imports. You'll need a bunch of things — but the `FPDF` library is likely the only unknown. Put simply, it's used to create PDFs, and you'll work with it a bit later. Refer to the following snippet for the imports:

```python
import os
import shutil
import numpy as np
import pandas as pd
import calendar
from datetime import datetime
from fpdf import FPDF

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

pdf_reports.py hosted with ❤ by GitHub view raw

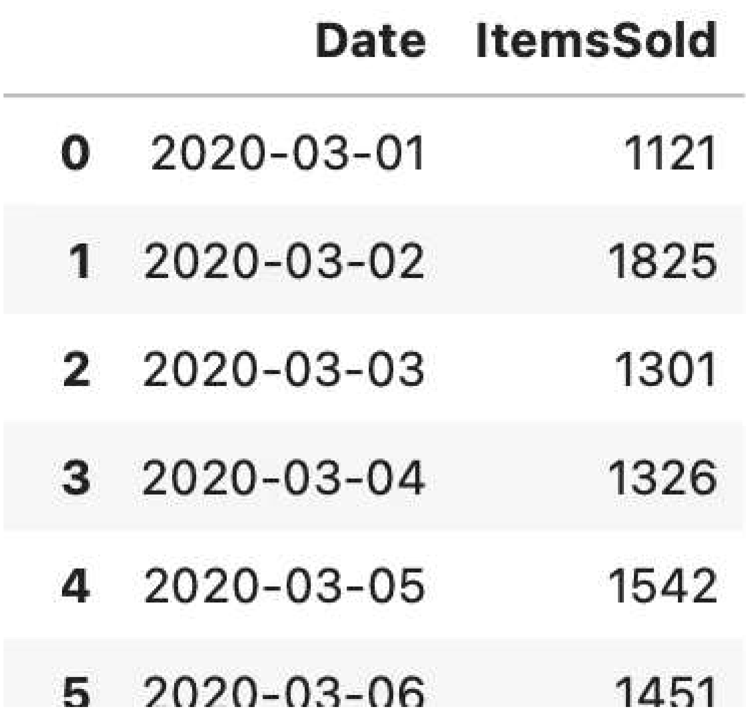
Let’s generate some fake data next. The idea is to declare a function that returns a data frame of dummy sales data for a given month. It does that by constructing a date range for the entire month and then assigning the sales amount as a random integer within a given range.

You can use the `calendar` library to get the last day for any year/month combination. Here’s the entire code snippet:

```python
def generate_sales_data(month: int) -> pd.DataFrame:
    # Date range from first day of month until last
    # Use ```calendar.monthrange(year, month)``` to get the last date
    dates = pd.date_range(
        start=datetime(year=2020, month=month, day=1),
        end=datetime(year=2020, month=month, day=calendar.monthrange(2020, month)[1])
    )

    # Sales numbers as a random integer between 1000 and 2000
    sales = np.random.randint(low=1000, high=2000, size=len(dates))

    # Combine into a single dataframe
    return pd.DataFrame({
        'Date': dates,
        'ItemsSold': sales
    })

# Test
generate_sales_data(month=3)
```

pdf_reports.py hosted with ❤ by GitHub view raw

A call to `generate_sales_data(month=3)` generated 31 data points for March of 2020. Here's how the first couple of rows look like:



	Date	ItemsSold
0	2020-03-01	1121
1	2020-03-02	1825
2	2020-03-03	1301
3	2020-03-04	1326
4	2020-03-05	1542
5	2020-03-06	1451

Image 1 — Sample of generated data (image by author)

And that's it — you now have a function that generates dummy sales data. Let's see how to visualize it next.

Data visualization

Your next task is to create a function that visualizes the earlier created dataset as a line plot. It's the most appropriate visualization type, as you're dealing with time series data.

Here's the function for data visualization and an example call:

```python
def plot(data: pd.DataFrame, filename: str) -> None:
    plt.figure(figsize=(12, 4))
    plt.grid(color='#F2F2F2', alpha=1, zorder=0)
    plt.plot(data['Date'], data['ItemsSold'], color='#087E8B', lw=3, zorder=5)
    plt.title(f'Sales 2020/{data["Date"].dt.month[0]}', fontsize=17)
    plt.xlabel('Period', fontsize=13)
    plt.xticks(fontsize=9)
    plt.ylabel('Number of items sold', fontsize=13)
    plt.yticks(fontsize=9)
    plt.savefig(filename, dpi=300, bbox_inches='tight', pad_inches=0)
    plt.close()
    return

# Test
december = generate_sales_data(month=12)
plot(data=december, filename='december.png')
```

pdf_reports.py hosted with ❤ by GitHub view raw

In a nutshell — you're creating data visualization, setting the title, playing around with fonts — nothing special. The visualization isn't shown to the user but is instead saved to the machine. You'll see later how powerful this can be.

An example call will save a data visualization for December of 2020. Here's how it looks like:

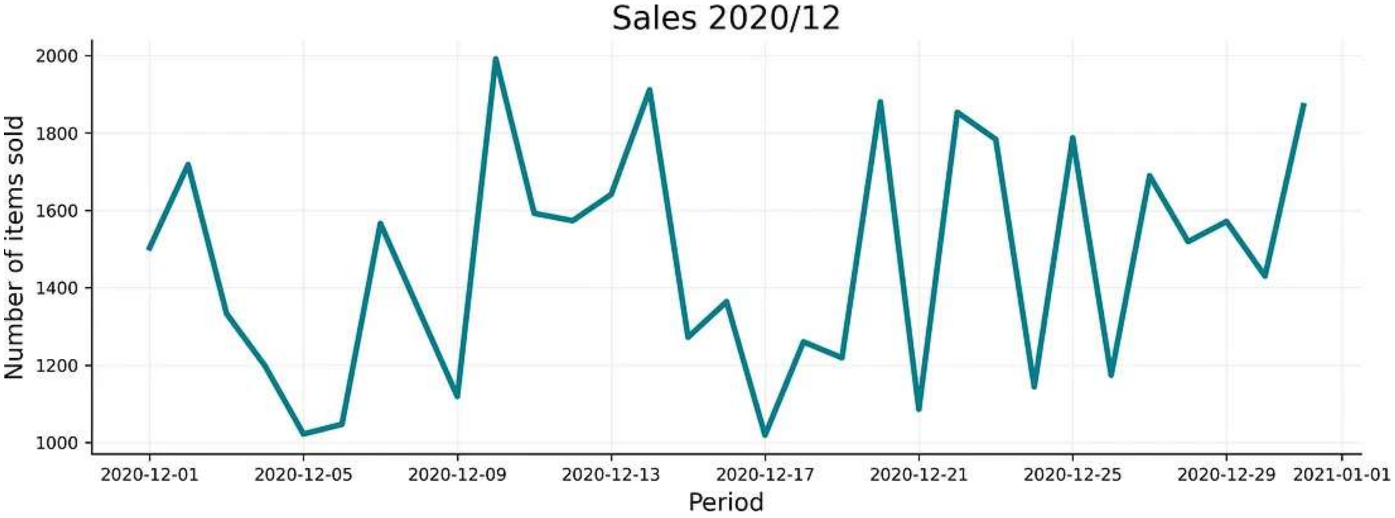


Image 2 — Sales for December/2020 plot (image by author)

And that's your visualization function. There's only one step remaining before you can create PDF documents, and that is to save all the visualization and define the report page structure.

Create a PDF page structure

The task now is to create a function that does the following:

- Creates a folder for charts — deletes if it exists and re-creates it
- Saves a data visualization for every month in 2020 except for January — so you can see how to work with different number of elements per page (feel free to include January too)
- Creates a PDF matrix from the visualizations — a 2-dimensional matrix where a row represents a single page in the PDF report

Here's the code snippet for the function:

```python
PLOT_DIR = 'plots'

def construct():
    # Delete folder if exists and create it again
    try:
        shutil.rmtree(PLOT_DIR)
        os.mkdir(PLOT_DIR)
    except FileNotFoundError:
        os.mkdir(PLOT_DIR)

    # Iterate over all months in 2020 except January
    for i in range(2, 13):
        # Save visualization
        plot(data=generate_sales_data(month=i), filename=f'{PLOT_DIR}/{i}.png')

    # Construct data shown in document
    counter = 0
    pages_data = []
    temp = []
    # Get all plots
    files = os.listdir(PLOT_DIR)
    # Sort them by month - a bit tricky because the file names are strings
    files = sorted(os.listdir(PLOT_DIR), key=lambda x: int(x.split('.')[0]))
    # Iterate over all created visualization
    for fname in files:
        # We want 3 per page
        if counter == 3:
            pages_data.append(temp)
            temp = []
            counter = 0

        temp.append(f'{PLOT_DIR}/{fname}')
        counter += 1

    return [*pages_data, temp]
```

pdf_reports.py hosted with ❤ by GitHub view raw

It’s possibly a lot to digest, so go over it line by line. The comments should help. The idea behind sorting is to obtain the month integer representation from the string — e.g., 3 from “3.png” and use this value to sort the charts. Delete this line if the order doesn’t matter, but that’s not the case with months.

Here's an example call of the `construct()` function:

```python
plots_per_page = construct()
plots_per_page
```

pdf_reports.py hosted with ❤ by GitHub view raw

You should see the following in your Notebook after running the above snippet:

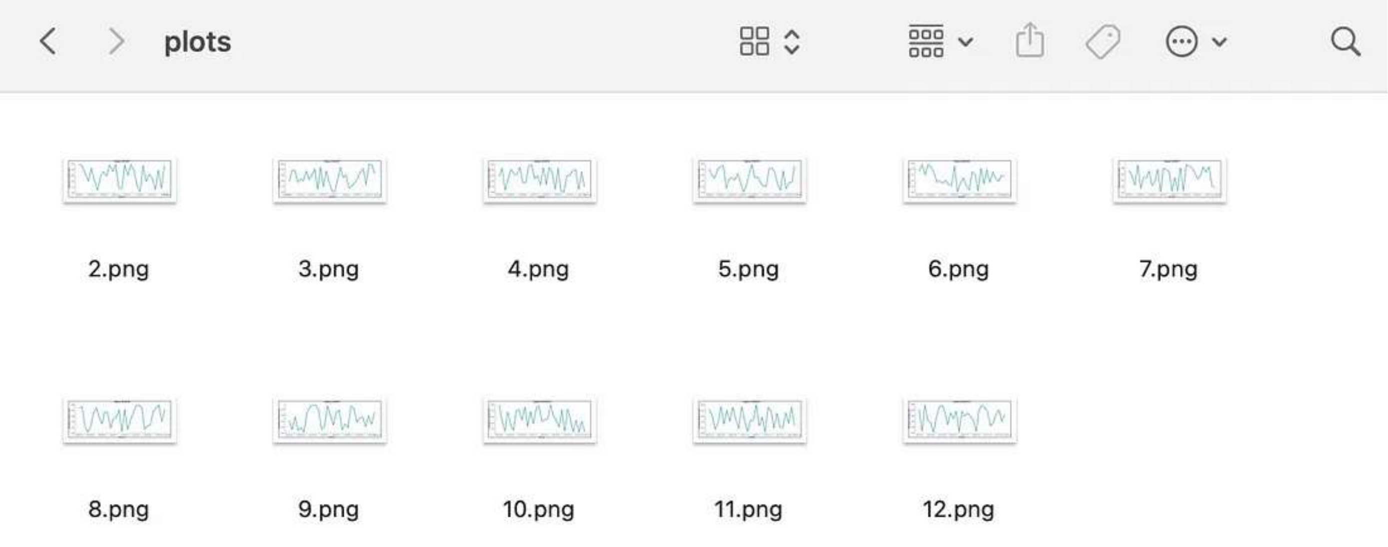
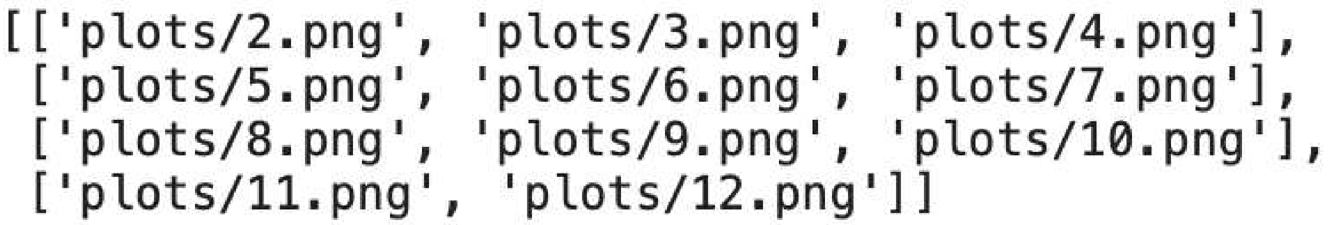


Image 3 — Generated visualizations (image by author)

In case you're wondering — here's how the `plots/` folder looks on my machine (after calling the `construct()` function):



```
[['plots/2.png', 'plots/3.png', 'plots/4.png'],
 ['plots/5.png', 'plots/6.png', 'plots/7.png'],
 ['plots/8.png', 'plots/9.png', 'plots/10.png'],
 ['plots/11.png', 'plots/12.png']]
```

Image 4 — PDF report content matrix (image by author)

And that's all you need to construct PDF reports — you'll learn how to do that next.

Create PDF reports

This is where everything comes together. You'll now create a custom `PDF` class that inherits from the `FPDF` . This way, all properties and methods are available in our class, if you don't forget to call `super().__init__()` in the constructor. The constructor will also hold values for page width and height (A4 paper).

Your `PDF` class will have a couple of methods:

- `header()` – used to define the document header. A custom logo is placed on the left (make sure to have one or delete this code line), and a hardcoded text is placed on the right
- `footer()` – used to define the document footer. It will simply show the page number
- `page_body()` – used to define how the page looks like. This will depend on the number of visualizations shown per page, so positions are margins are set accordingly (feel free to play around with the values)
- `print_page()` – used to add a blank page and fill it with content

Here's the entire code snippet for the class:

```python
class PDF(FPDF):
    def __init__(self):
        super().__init__()
        self.WIDTH = 210
        self.HEIGHT = 297

    def header(self):
        # Custom logo and positioning
        # Create an `assets` folder and put any wide and short image inside
        # Name the image `logo.png`
        self.image('assets/logo.png', 10, 8, 33)
        self.set_font('Arial', 'B', 11)
        self.cell(self.WIDTH - 80)
        self.cell(60, 1, 'Sales report', 0, 0, 'R')
        self.ln(20)

    def footer(self):
        # Page numbers in the footer
        self.set_y(-15)
        self.set_font('Arial', 'I', 8)
        self.set_text_color(128)
        self.cell(0, 10, 'Page ' + str(self.page_no()), 0, 0, 'C')

    def page_body(self, images):
        # Determine how many plots there are per page and set positions
        # and margins accordingly
        if len(images) == 3:
            self.image(images[0], 15, 25, self.WIDTH - 30)
            self.image(images[1], 15, self.WIDTH / 2 + 5, self.WIDTH - 30)
            self.image(images[2], 15, self.WIDTH / 2 + 90, self.WIDTH - 30)
        elif len(images) == 2:
            self.image(images[0], 15, 25, self.WIDTH - 30)
            self.image(images[1], 15, self.WIDTH / 2 + 5, self.WIDTH - 30)
        else:
            self.image(images[0], 15, 25, self.WIDTH - 30)

    def print_page(self, images):
        # Generates the report
        self.add_page()
        self.page_body(images)
```

pdf_reports.py hosted with ❤ by GitHub view raw

Now it’s time to instantiate it and to append pages from the 2-dimensional content matrix:

```
pdf = PDF()

for elem in plots_per_page:
    pdf.print_page(elem)

pdf.output('SalesRepot.pdf', 'F')
```

pdf_reports.py hosted with ❤ by GitHub view raw

The above cell will take some time to execute, and will return an empty string when done. That’s expected, as your report is saved to the folder where the Notebook is stored.

Here’s how to first page of the report should look like:

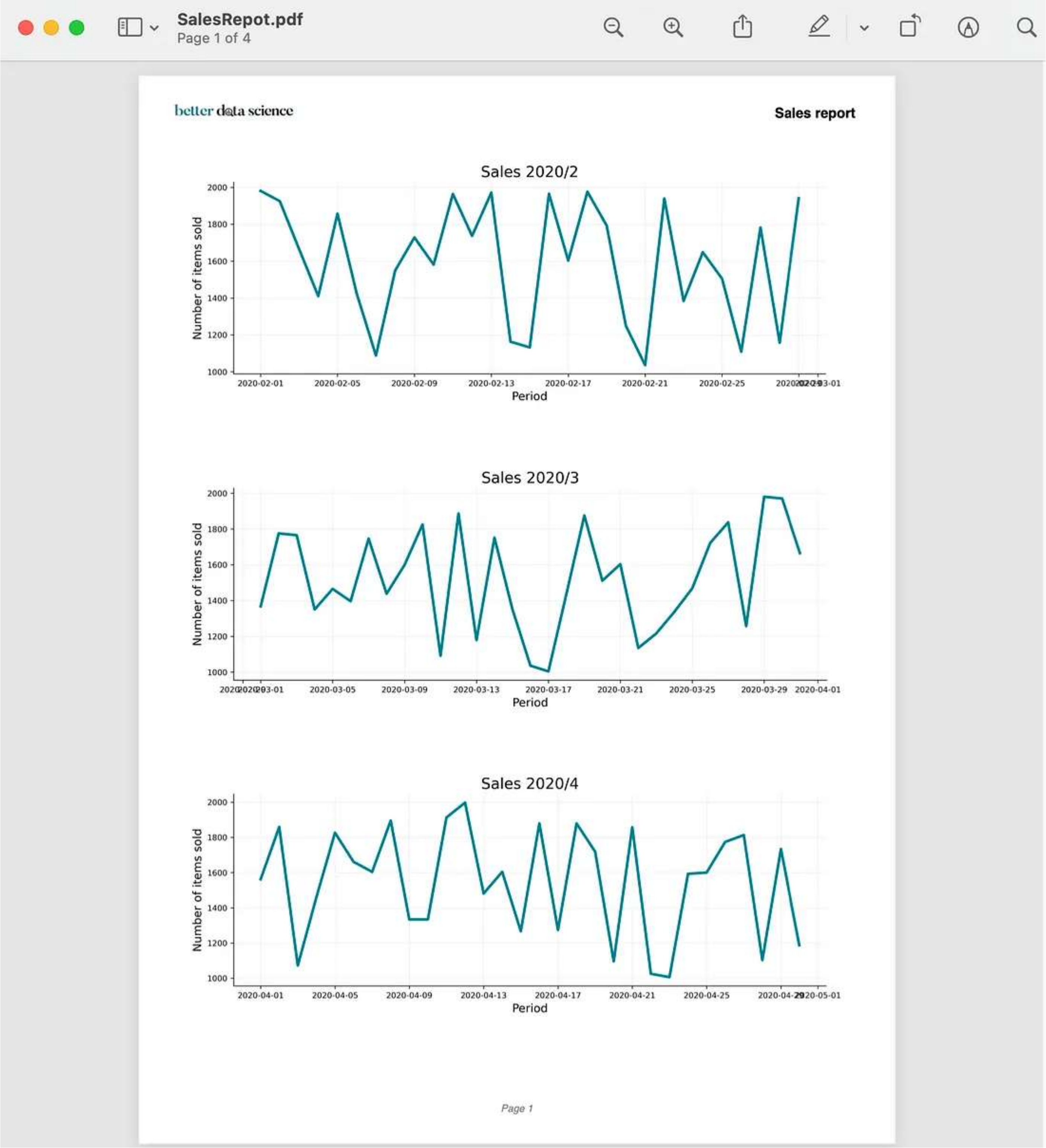


Image 5 — First page of the PDF report (image by author)

Of course, yours will look different due to the different logo and due to sales data being completely random.

And that’s how you create data-visualization-powered PDF reports with Python. Let’s wrap things up next.

Conclusion

You’ve learned many things today — how to create dummy data for any occasion, how to visualize it, and how to embed visualizations into a single PDF report. Embedding your visualizations will require minimal code changes — mostly for positioning and margins.

Let me know if you’d like to see a guide for automated report creation based on machine learning model interpretations (SHAP or LIME) or something else related to data science.

Thanks for reading.

Loved the article? Become a Medium member to continue learning without limits. I’ll receive a portion of your membership fee if you use the following link, with no extra cost to you.

Join Medium with my referral link - Dario Radečić

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story…

medium.com

Join my private email list for more helpful insights.

Learn more

- Top 5 Books to Learn Data Science in 2021
- SHAP: How to Interpret Machine Learning Models With Python
- Top 3 Classification Machine Learning Metrics — Ditch Accuracy Once and For All
- ROC and AUC — How to Evaluate Machine Learning Models
- Precision-Recall Curves: How to Easily Evaluate Machine Learning Models

Originally published at https://betterdatascience.com on January 18, 2021.

Towards Data Science

Data Science

Python

Programming

Machine Learning

Follow

Written by Dario Radečić

40K Followers · Writer for Towards Data Science

Data Scientist & Tech Writer | betterdatascience.com

More from Dario Radečić and Towards Data Science

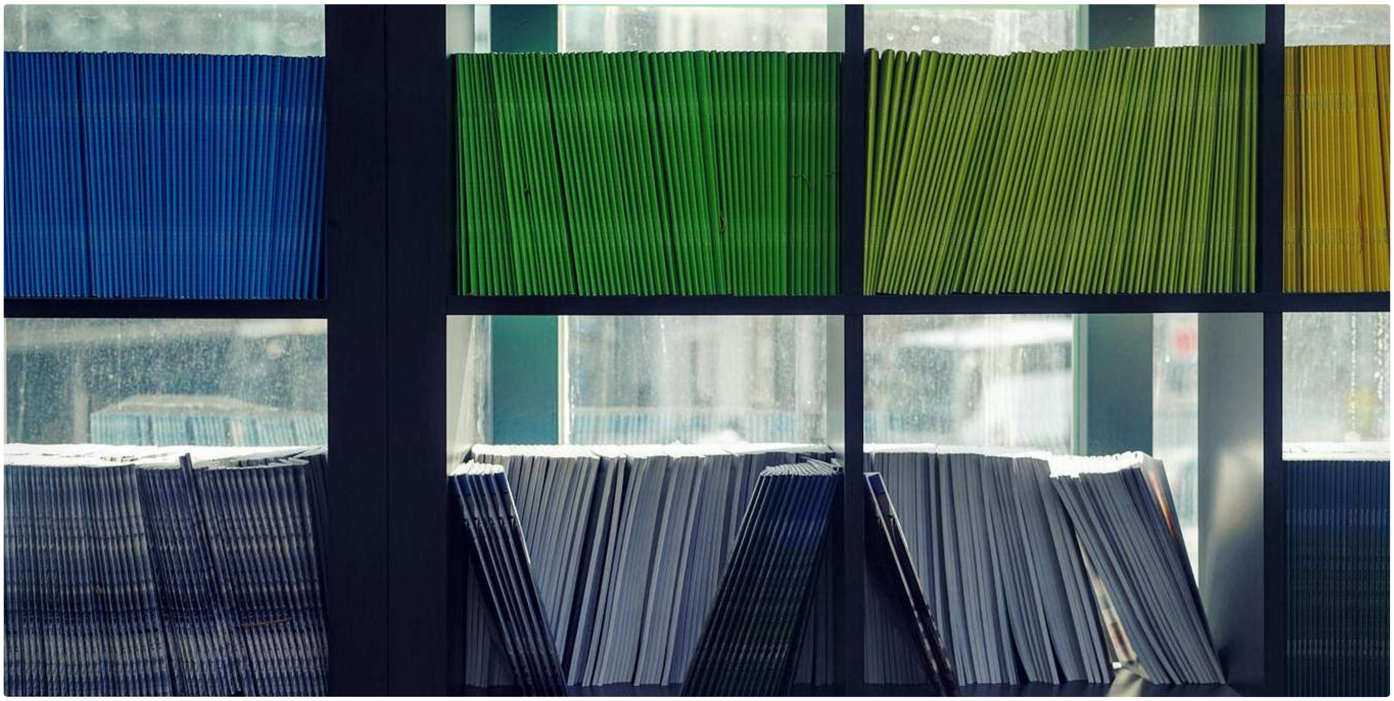
Dario Radečić in Towards Data Science

Pip Install Specific Version—How to Install a Specific Python Package Version with Pip

Want to install a specific Python package version with Pip? This article will show you how with hands-on examples and guides.

· 6 min read · Apr 5

73 2



Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

3K 39

Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

✦ · 12 min read · Apr 25

2.2K 18

Dario Radečić in Towards Data Science

How to Schedule Python Scripts With Cron — The Only Guide You'll Ever Need

Automate your Python script execution — works on Linux and macOS.

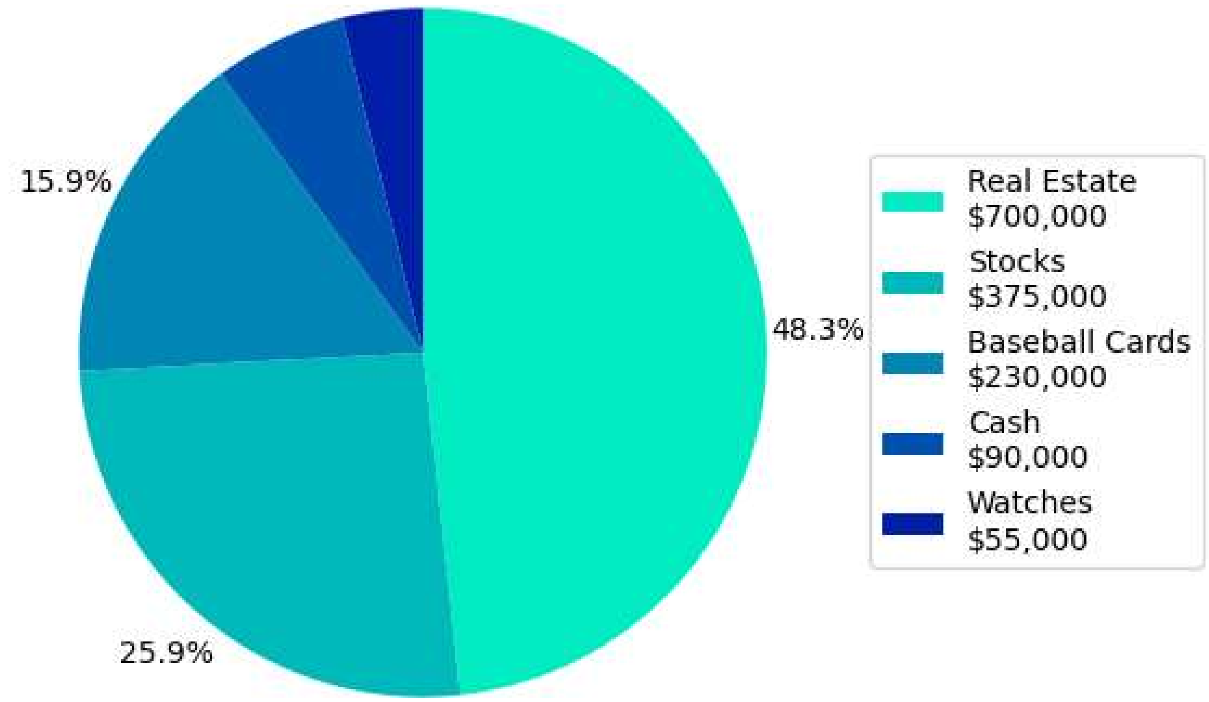
✦ · 6 min read · May 7, 2021

413 3

See all from Dario Radečić

See all from Towards Data Science

Recommended from Medium



Better Everything

Beautiful Pie Charts with Python

Making pie charts in Python is fairly simple as you will soon see. It can be done with the matplotlib package which is commonly used to…

· 7 min read · Feb 17

58

Utkarsha Bakshi in Geek Culture

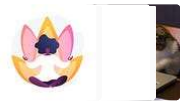
Generating PDF from HTML Template Using AWS Lambda

In this article, we will learn how to generate a PDF file from an jinja2 HTML template using a Python AWS Lambda function. We will use...

✦ · 5 min read · Feb 3

25 1

Lists



Stories to Help You Grow as a Software Developer

19 stories · 64 saves

Leadership

30 stories · 27 saves

What is ChatGPT?

9 stories · 59 saves

Stories to Help You Level-Up at Work

19 stories · 53 saves

Gabe Araujo, M.Sc. in Level Up Coding

🐼Introducing PandasAI: The Generative AI Python Library 🐼

Pandas AI is an additional Python library that enhances Pandas, the widely-used data analysis and manipulation tool, by incorporating…

✦ · 9 min read · May 17

686 7

Himanshu Sharma in MLearning.ai

Comparing Python Libraries for Visualization

Plotly, Seaborn, or Matplotlib, Which is better?

✦ · 7 min read · Jan 26

53

Gabe Araujo, M.Sc. in Level Up Coding

🧬 How I Used Python to Make Everyday Tasks Easier

Hey there! As a busy person with a lot on my plate, I'm always looking for ways to make my life easier.

✦ · 8 min read · May 1

548 2

Bobby in Level Up Coding

6 Pythonic Ways to Replace if-else Statements

Avoid if-else the Pythonic Ways

✦ · 6 min read · Feb 27

371 11

See more recommendations