Interactive Weather Data Visualizations with Plotly

Exploring the NOAA Global Surface Summary of the D

Will Norris
Mar 26, 2021 10 min read

Making Sense of Big Data

Photo by Michael D on Unsplash

The Global Surface Summary of the Day (GSOD) set from over 9,000 weather stations dating bac created and is maintained by the National Ocea

Administration (NOAA) and generates a daily summary of hourly surface measurements for 18 climate variables like mean dew point, mean wind speed, and min/max temperate.

This is a great data set to use for practicing visualizing spatial data since it has many stations across the globe with a relatively large time series to play around with. It also is updated daily, which makes it a great candidate for practicing with da

While Python might not be the first language or mind for visualizing spatial data, there are sever capable of creating fantastic geographic visualiz become a very well rounded interactive plotting and its geographic plotting capabilities have cor the last few years. Because of its excellent docu popularity I will focus only on Plotly in this post, that there are other libraries available.

This data set is somewhat large at 4.2 GB so loa will take a little effort; it is not a massive data s but it is larger than the average "practice" data s We will start by walking through how to access explore visualizing the processed data on intera Plotly. Make a cup of coffee, get comfortable, ar larger weather data set and visualize our finding

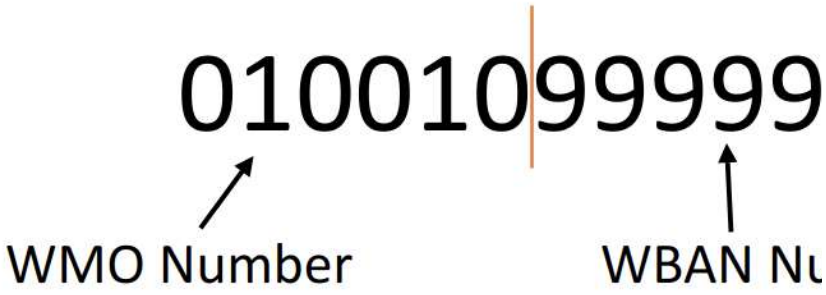
File Structure

There are multiple ways to obtain this data set. download version of it is available from Kaggle, daily like the NOAA repositories and the Kaggle

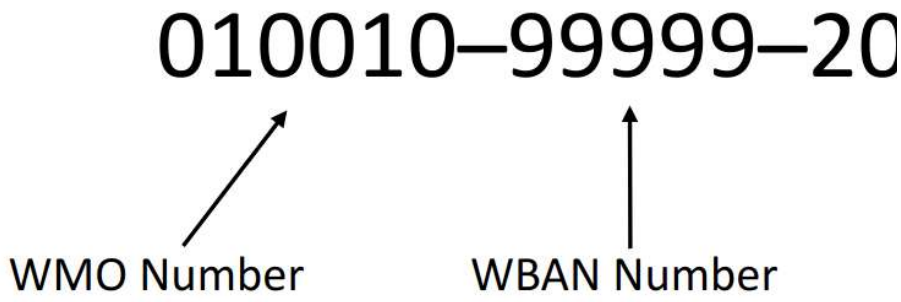
difficult to format in Python. This is how the NOAA repository is structured:

```
1929.tar.gz/
1930.tar.gz/
    03005099999.csv
    03026099999.csv
    ...
...
```

The data is stored in yearly compressed tarballs files for each station active that year. Every .csv its station ID, which is made up of the station's numbers.



NOAA CSV File Structure (Created By Author)



Kaggle Gzip File Structure (Created By Author)

WMO Number

The World Meteorological Organization (WMO) is an agency of the United Nations responsible for weather, climate, and water resources. The first two digits are the country code, the next three digits are set by the National Meteorological Service (NMS). If the last number is a zero that means the WMO number is, or was in the past, official.

WBAN Number

WBAN stands for Weather-Buereau-Army-Navy. T
is an identifier for digital data storage that start
US Weather Bureau, Canada's Transportation De
three major military branches to share data with
expanded to include some German and Korean s

Note: The important piece to remember is that t
numbers create a unique station ID that is our p
referencing each weather station once loaded in

Downloading Our Data

The best way to get this data is directly from NC
download a tarball for each year that contain a
station active during that year. I have included a
module for downloading and processing all of th
find in the documentation section at the end.

```
1  import requests
2  import re
3  import os
4
5  def get_data(directory="noaa_gsod"):
6      """
7      Download all data from GSOD that is currently in the Bulk Dow
8
```

```
    Parameters
    ----------
    directory: str
        The directory you want to download the csv files to. Will create it if it
        doesn't exist
    """
    if not os.path.exists(directory):
        os.makedirs(directory)
    base_url = "https://www.ncei.noaa.gov/data/global-summary-of-the-day/archive/"
    rt = requests.get(base_url).text
    years = re.findall("(?<!\>)\d{4}", rt)
    for year in years:
        url = base_url+str(year)+'.tar.gz'
        r = requests.get(url, stream=True)
        if r.status_code == 200:
            with open('noaa_data/'+str(year)+'.tar.gz', 'wb') as
                f.write(r.raw.read())
```

get_data.py hosted with ❤ by GitHub

To download the NOAA data I decided to make a function using requests. It starts on the home d set and uses regex to find all of the years availa the base URL page. With each year in a list it is the final URLs and download each tarball. Reme code is available on Github and can be importe in Python.

Data Processing

The most important question to ask yourself wh datasets is what data you want to have access t into a dataframe. There were over 11,000 station means ~10 years of daily data would be quite a l work with. Instead you may want to do some ag article I decided to aggregate the data by month daily data for one day of each year. Aggregating t

will greatly reduce the size of our final dataframe and still let us find trends over a decently long time series.

It may seem weird to keep one specific day un-aggregated, and it is! The real reason we want to save a single day's data is so that we can load in the most recent day to a live dashboard with Dash in a future post. For now it will still be useful for creating similar plots to what we will explore with Dash later.

Step 0: Imports

Since processing this data is a little more involv
practice datasets, I have included a GitHub repc
module that can be imported locally into your P

```
import gsodpy.gsodProcess as gsod
import gsodpy.gsodDownloader as gsodDown
import datetime
```

In order to import modules locally like this, the
must be in your project directory.

Step 1: Define Time Series

The first thing we need to do is decide how mar
and which day we will store un-aggregated. We
to create our desired file list.

```
num_years = 10
target_day = datetime.datetime(2020,3,20)
years, files = get_years_files(num_years)
```

```
1  def get_years_files(num_years):
2      """
3      Create list of files and list of most recent num_years
```

```
    Parameters
    ----------
    num_years: int
      number of years in time series
    """
    files = os.listdir("noaa_data")
    files.sort()
    files = files[-num_years:]
    years = [int(re.findall('\d+',file)[0]) for file in files]

    return years, files
```

get_years.py hosted with ❤ by GitHub

Step 2: Process Files During Time Frame

Once we have defined our time series, we are re
cracking open tarballs and processing .csv files!
functions I wrote to unpack our data.

```
def process_all_years(yearfiles, target_day):
    """
    Open each zip file in tar, check if station was operating in

    Parameters
    ----------
    yearfiles: str
        List of zipped tarballs to unpack
    target_day: datetime.datetime
        Target day to store unaggregated data
    """
    df = pd.DataFrame([])
    df_day = pd.DataFrame([])
    for yearfile in yearfiles[:-1]: # skip most recent year for r
        print("Processing file: {}".format(yearfile))
        tar = tarfile.open("noaa_data/"+yearfile, "r")
        print("Number of stations in file: {}".format(len(tar.get
        for member in tar:
            de = pd.read_csv(io.BytesIO(tar.extractfile(member).r
            de = process_df(de)
            df_day = df_day.append(de[(de['MONTH']==target_day.mc
            de = de.groupby(['STATION', 'WMO', 'WBAN', 'YEAR', 'M
            df = df.append(de)
        tar.close()
```

```
    df = add_meta(df)
    df_day = add_meta(df_day)
    df_day.reset_index(inplace = True, drop=True)
    return df, df_day


def process_df(df):
    """
    Clean and process the raw weather station dataframes

    Parameters
    ----------
    df: pd.DataFrame
        Raw dataframe from station csv file to clean and process
    """
    df['WMO'] = df['STATION'].apply(lambda x: str(x)[0:6])
    df['WBAN'] = df['STATION'].apply(lambda x: str(x)[6:11])
    df['MAX'] = df['MAX'].apply(lambda x: str(x).strip("*"))
    df['MIN'] = df['MIN'].apply(lambda x: str(x).strip("*"))
    df[['STATION','TEMP','DEWP','WDSP','MAX','MIN']] = df[['STATI
    df['DATE'] = pd.to_datetime(df['DATE'], format='%Y%m%d', errc
    df['YEAR'] = pd.DatetimeIndex(df['DATE']).year
    df['MONTH'] = pd.DatetimeIndex(df['DATE']).month
    df['DAY'] = pd.DatetimeIndex(df['DATE']).day
    df.drop(['TEMP_ATTRIBUTES', 'DEWP_ATTRIBUTES', 'MAX_ATTRIBUTE
             'PRCP_ATTRIBUTES','SLP_ATTRIBUTES', 'MIN_ATTRIBUTES'
             'SLP', 'STP', 'STP_ATTRIBUTES'], axis=1, inplace=Tru
    return df


def add_meta(df):
    """
    Add metadata column to dataframe for plotly text boxes

    Parameters
    ----------
    df: pd.DataFrame
        The final weather station dataframe to add metadata to
    """
    df['META'] = df['NAME']
    df['ELEV_LABEL'] = df['ELEVATION'].apply(lambda x: 'Elevatior
    df['META'] = df[['META','ELEV_LABEL']].apply(lambda x: x.str.
    df['addmeta'] = df['TEMP'].apply(lambda x: "Temp: {} C".forma
    df['META'] = df[["META", "addmeta"]].apply(lambda x: x.str.ca
    df = df.drop(['NAME', 'ELEV_LABEL', 'addmeta'], axis=1)
    return df
```

process_gsod.py hosted with ❤ by GitHub view raw

To save you some time, here is the basic psuedo-code for this workflow:

```
- Loop through each tarball (years)
    - Loop through each csv file stored in tarball (stations)
        - Open .csv file for current station/year with
        - Call process_df() to clean the dataframe
        - Append all data for this station on target_
        - Aggregate data by month
        - Append monthly aggregate to df
- Call add_meta() to create metadata column on df and
- Return completed df and df_day
```

Overall the process is not complex, and I'm sure more concise given more time. Remember that number of stations in each year it takes quite a of this data.

We can go ahead and call our processing functi dataframes:

```
df, df_day = gsod.process_all_years(files, target_day)
```

> **Note**: If you notice that your machine is strug of the data try starting with a short time serie want a longer time series you can also select of stations in the tarball to reduce the numbe you need to process.

Step 3: Save Processed Data

I highly recommend pickling your finished dataframes so that you can load them in much faster later.

```
import pandas as pd

# Save DataFrames to pickle in current directory
df.to_pickle("df_monthly.pkl")
df_day.to_pickle("df_daily.pkl")

# Read them back in later:
df = pd.read_pickle("df_monthly.pkl")
df_day = pd.read_pickle("df_daily.pkl")
```


Visualizations

Now that we have some meaningful data loaded get to the fun part and start making visualizatio

If you use the free version of Plotly you are limit 512kb. This isn't a problem if you are making you However, for me to share interactive plots here Plotly's Chart Studio to upload them to their clo am limited to roughly 300 weather stations that randomly with this function:

```
def sample_stations(df_month, num_samples=300):
    stationID = df_month["STATION"].unique().tolist()
    station_subset = random.sample(stationID, num_samples)
    return df_month[df_month['STATION'].isin(station_subset)]
```

sample_stations.py hosted with ❤ by GitHub

Basic Maps

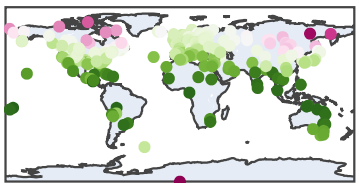
To start off, lets take a look at one month in our time series to see where some of our stations are on the map. This is about as basic of a map we can create with Plotly and this data; considering how little configuration it took, I think it looks pretty good.

Plotly's main data structure is the `figure`, which is an instance of the `plotly.graph_objects.Figure` class. The `figure` is rendered via Plotly.js using JavaScript, which is why we are a looking maps in Python.


```
month = 2
year = 2016
df_month = sample_stations(df, 300)
tempsdf = df_month[(df_month['MONTH']==month) & (df_month['YEAR']

fig = go.Figure(data=go.Scattergeo(
        text = tempsdf['META'],
        lat = tempsdf['LATITUDE'],
        lon = tempsdf['LONGITUDE'],
        mode = 'markers',
        marker_color = tempsdf['TEMP'],
        ))

fig.update_layout(
        title = 'Temperature on {},{}'.format(month,year),
        geo_scope='world',
        geo = dict(
                showcountries = False,
                showcoastlines=False,
                resolution = 110)
    )
fig.show()
```

basicmap.py hosted with ❤ by GitHub

Temperature on 2,2016



I love plotting spatial data with Plotly because o
responsive interface that can be embedded alm
Quality base maps have always been difficult fo
Python packages, but Plotly's are great.

While this plot may be fine, there are absolutely
changes I would make to it. The coastlines are c
color scale could use adjusting. Overall it could
our data points stand out more. Luckily all of th
accomplished with Plotly!

Styled Plots: Extreme Temperatures on Given Da

One reason I chose to keep a single day of the c
taking monthly aggregates was so that we could
warmest and coldest places on a given day.

First we need to find our extreme temperatures

```
1  def create_extremes_df(df, day, num_extremes):
2      """
```

```
    Create extremes dataframe to plot coldest and warmest places on a given day
    """
    extremes = pd.DataFrame([])
    extremes = extremes.append(df[df['DATE']==day].sort_values(by="TEMP", ascending=False).head(
    extremes = extremes.append(df[df['DATE']==day].sort_values(by="TEMP", ascending=False).tail(
    return extremes
```

extremes.py hosted with ❤ by GitHub view raw

This plot uses the same structure as our first or more time customizing the marker and figure pr notice that I am able to pass a dictionary to cer `marker` to customize them further. In most cases are objects that are being initialized by the infor dictionary. For example, the `marker` parameter in `plotly.graph_objects.[scattergeo.Marker]` (https://plotly.github.io/plotly.py-docs/generated/plotly.graph_objects.scattergeo.html#pl ttergeo.Marker) object. While this increases comp from the scattergeo.Marker documentation how gives us over the markers on our plot.

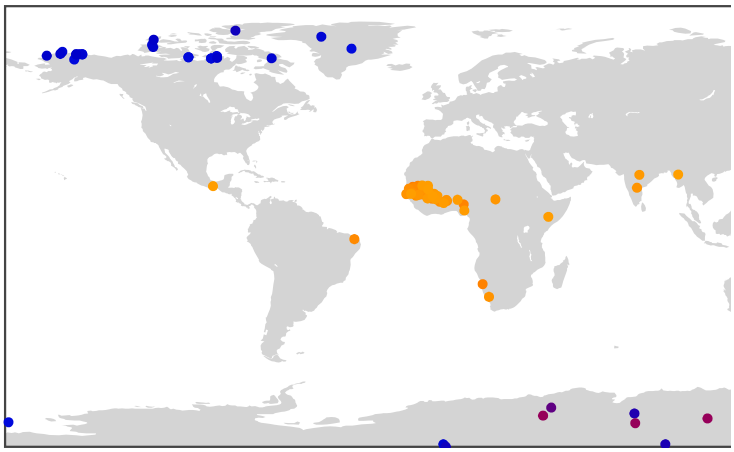
LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
Contributor Portal

```
scl = [0,"rgb(150,0,90)"],[0.125,"rgb(0, 0, 200)"],[0.25,"rgb(0,
[0.375,"rgb(0, 152, 255)"],[0.5,"rgb(44, 200, 150)"],\
[0.75,"rgb(255, 234, 0)"],[0.875,"rgb(255, 111, 0)"],[1,"rgb(255,

fig = go.Figure(data=go.Scattergeo(
    lat = extremes['LAT'],
    lon = extremes['LON'],
    text = extremes['META'],
    marker = dict(
        color = (extremes['TEMP']-32)*5/9,
        colorscale = scl,
        cmin = -50,
        cmax = 50,
        size = 5,
        colorbar = dict(
            thickness = 10,
            titleside = "right",
            outlinecolor = "rgba(68, 68, 68, 0)",
            tickvals = [-50,-30,-15,0,15,30,50],
```

```
            ticks = "outside",
            ticklen = 3,
            ticksuffix = " C",
            showticksuffix = "all"
        )
    )
))

fig.update_layout(
    geo = dict(
        scope = 'world',
        showland = True,
        landcolor = "rgb(212, 212, 212)",
        showlakes = True,
        lakecolor = "rgb(255, 255, 255)",
        showsubunits = False,
        showcountries = False,
        showcoastlines=False,
        resolution = 110
        ),
    title='{} Coldest and Hotest Temperatures on {}'.format(num_e

fig.show()
```

scattergeo.py hosted with ❤ by GitHub

40 Coldest and Hostest Temperatures on (3,20,202



Mapbox Integration

I have always been a sucker for Mapbox. It has great base maps and is super responsive. Plotly has limited Mapbox integration that can enhance your geographical plots. Especially when zooming in you will notice a pleasing amount of detail in the base map compared to the base Plotly version. Mapbox has many effects on the way Plotly maps are rendered, so I will be covering M in an entire post.


```
mapbox_access_token = open(".mapbox_token").read()

fig = go.Figure(go.Scattermapbox(
        lat=extremes['LATITUDE'],
        lon=extremes['LONGITUDE'],
        mode='markers',
            marker = dict(
            color = (extremes['TEMP']-32)*5/9,
            colorscale = scl,
            cmin = -50,
            cmax = 50,
            size = 5,
            colorbar = dict(
                thickness = 10,
                titleside = "right",
                outlinecolor = "rgba(68, 68, 68, 0)",
                tickvals = [-50,-30,-15,0,15,30,50],
                ticks = "outside",
                ticklen = 3,
                ticksuffix = " C",
                showticksuffix = "all"
            )
        ),
        text=extremes['META'],
    ))

fig.update_layout(
    autosize=True,
    hovermode='closest',
    mapbox=dict(
        accesstoken=mapbox_access_token,
        bearing=0,
        pitch=0,
```

```
            zoom=0
        ),
    )

    fig.show()
```

scattermap.py hosted with ❤ by GitHub view raw

40 Coldest and Hostest Temperatures on (3,20,202

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

Contributor Portal

Adding Sliders: Average Temperatures by Month

One feature that I really like about Plotly is how adding a slider to view how data changes over ti lets take a look at the average monthly tempera

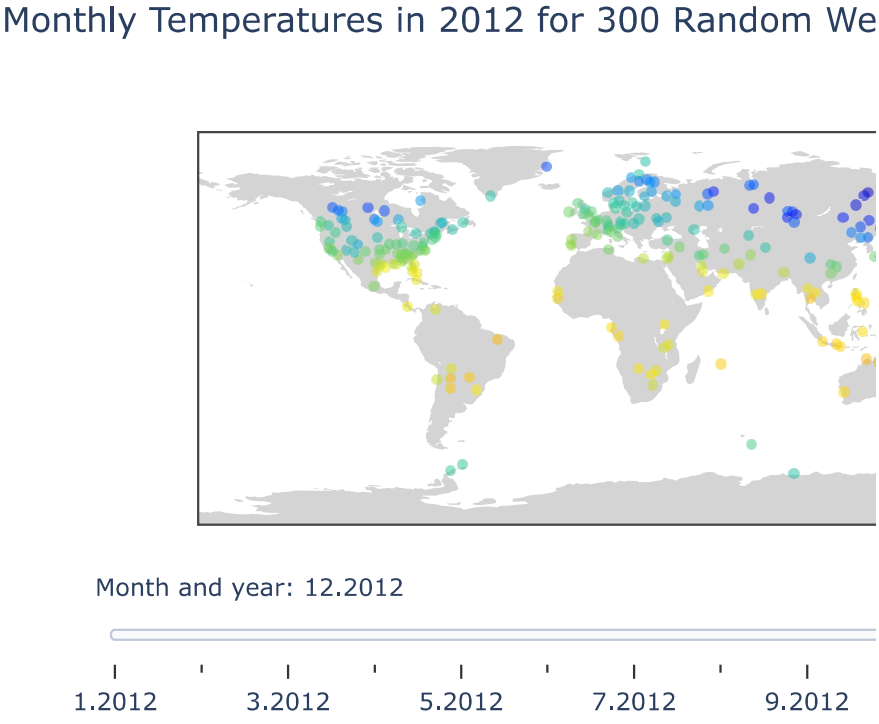
You will notice that I am using a slightly differen this plot. This is technically the "old" method of But for more complex plots like this it is nice to the data separately.

```
data = [ dict(
    visible = False,
    name = '',
    type = 'scattergeo',
    text = df_month[(df_month['MONTH']==month) & (df_month['YEAR'
```

```
    lat = df_month[(df_month['MONTH']==month) & (df_month['YEAR']==year)]['LATITUDE'],
    lon = df_month[(df_month['MONTH']==month) & (df_month['YEAR']==year)]['LONGITUDE'],
    marker = dict(
        color = (df_month[(df_month['MONTH']==month) & (df_month['YEAR']==year)]['TEMP']-32)*5/
        colorscale = scl,
        cmin = -50,
        cmax = 50,
        opacity = 0.5,
        size = 5,
        colorbar = dict(
            thickness = 10,
            titleside = "right",
            outlinecolor = "rgba(68, 68, 68, 0)",
            tickvals = [-50,-30,-15,0,15,30,50],
            ticks = "outside",
            ticklen = 3,
            ticksuffix = " C",
            showticksuffix = "all"
        )
    )
) for year in [years[0]] for month in range(1,13)]
data[-1]['visible'] = True

steps = []
for i in range(len(data)):
    step = dict(
        method = 'restyle',
        args = ['visible', [False] * len(data)],
        label = [str(month)+"."+str(year) for year in [years[-1]]]
    )
    step['args'][1][i] = True
    steps.append(step)

sliders = [dict(
    active = len(steps)-1,
    currentvalue = {"prefix": "Month and year: "},
    steps = steps
)]

layout = dict(
    sliders=sliders,
    geo = dict(
        scope = 'world',
        showland = True,
        landcolor = "rgb(212, 212, 212)",
        showlakes = True,
        lakecolor = "rgb(255, 255, 255)",
```

```
            showsubunits = False,
            showcountries = False,
            showcoastlines=False,
            resolution = 110
            ),
        )
    fig = dict( data=data, layout=layout )
```

avg_month_temp.py hosted with ❤ by GitHub view raw



LATEST
EDITOR'S PICKS
DEEP DIVES
CONTRIBUTE
NEWSLETTER
Sign in
Contributor Portal

Creating this plot looks more complex than the
the bulk of the code is simply restructuring data
groupings for Plotly to easily send to its JavaScr
have ever worked with json data, you will probal
of dictionaries can easily be converted to json.

Wrapping Up

Spatial data is never the easiest to work with in
multidimensional, often quite large, and normal
are not well suited to plotting geographical coor

removes a lot of the headache by providing easy to use functionality with quality base maps that I generally had to use javascript to access a couple years ago. When it comes to spatial plotting in Python, Ploty is generally my first choice.

There is so much more to learn with plotting spatial data with Plotly so don't be afraid to sift through the documentation and examples online. The same developers also make Dash, w for building live dashboards.

Documentation

- Github Repo for Processing GSOD Data
- GSOD Data Repository (NOAA)
- Kaggle GSOD Data
- Plotly
- Mapbox

WRITTEN BY

Will Norris

See all from Will Norris

Topics:

Editors Pick | Geospatial | Making Sense Of Big D

Python

Share this article:

Related Articles

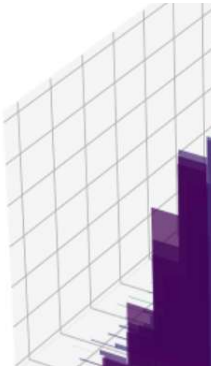
ARTIFICIAL INTELLIGENCE

What Do Large Language Models "Understand"?

A deep dive on the meaning of understanding and how it applies to LLMs

Tarik Dzekman

August 21, 2024 31 min read



DATA SCIENCE

Must-Know Bivariate No Explained

Derivation and powerful conce

Luigi Battistoni

August 14, 2024

DATA SCIENCE

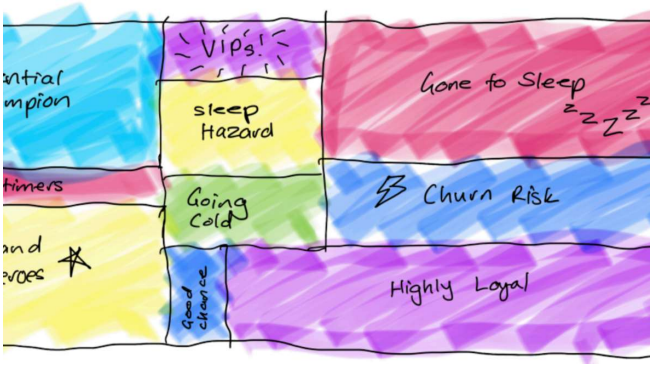
DEEP LEARNING

Deep Dive in by Hand ✍️

Optimizing Marketing Campaigns with Budgeted Multi-Armed Bandits

With demos, our new solution, and a video

Vadim Arzamasov



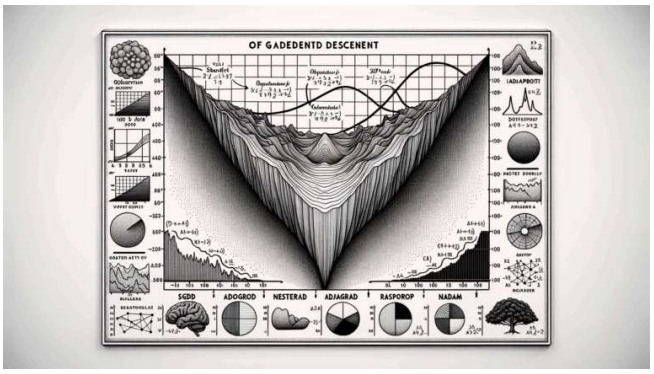
ANALYTICS

Methods for Modelling Customer Lifetime Value: The Good Stuff and the Gotchas

Part three of a comprehensive, practical guide to CLV techniques and real-world use-cases

Katherine Munro

November 17, 2023 12 min read



DATA SCIENCE

The Math Behind Keras 3 Optimizers: Deep Understanding and Application

This is a bit different from what the books say.

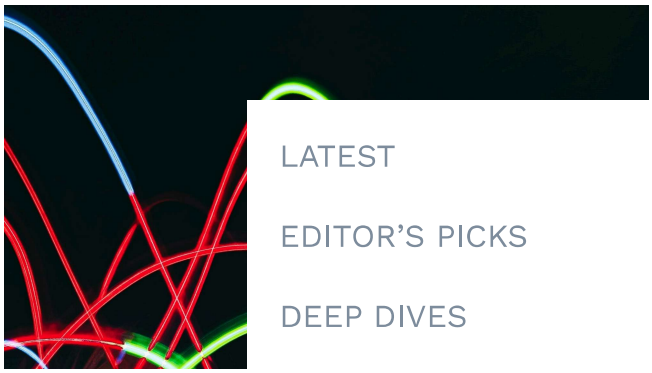
Peng Qian

August 17, 2024 9 min read

Explore the wisdom of LSTM leading into xLSTMs - a probable competition to the present-day LLMs

Srijanie Dey, PhD

July 9, 2024 13 min read



DATA SCIENCE

Squashing th into Penalize Regression f

How to build pe regression mod

Álvaro Méndez Ci

August 16, 2024

Your home for data science and Al. The world's leading publ
data analytics, data engineering, machine learning, and artifi
professionals.

Subscribe to Our Newsletter

ABOUT · ADVERTISE · PRIVACY POLICY · TERI

COOKIES SETTINGS

LATEST

EDITOR'S PICKS

DEEP DIVES

CONTRIBUTE

NEWSLETTER

Sign in

Contributor Portal