Understanding Databases with SQLAlchemy: Python Data Playbook

UP AND RUNNING WITH SQLALCHEMY

Xavier Morera

HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA

@xmorera www.xaviermorera.com

What is SQLAlchemy?

Python SQL toolkit

Object Relational Mapper

Supported Databases

SQLite

PostgreSQL

MySQL

Oracle

Microsoft
SQL Server

To name a few...

Working with SQLAlchemy

Raw SQL

Using SQL statements
What many know

Object-relational Mapper

Using objects
The Pythonic way

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sqlalchemy_mysql   |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE sqlalchemy_mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```


Getting
SQLAlchemy

Getting SQLAlchemy

```
$ pip install sqlalchemy
$ pip install mysql-connector-python
```

Install SQLAlchemy Using pip

Using a package manager: pip

- Pip Installs Packages

Install connector, based on your desired database

- Like mysql-connector-python for MySQL

SQLAlchemy 1.2.17

```
pip install SQLAlchemy
```

Latest version

Last released: about 11 hours ago

Database Abstraction Library

Navigation

Project description

Release history

Download files

Project links

Homepage

Statistics

View statistics for this project via

Project description

The Python SQL Toolkit and Object Relational Mapper

Introduction

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. SQLAlchemy provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

Major SQLAlchemy features include:

- An industrial strength ORM, built from the core on the identity map, unit of work, and data mapper patterns. These patterns allow transparent persistence of objects using a declarative configuration system. Domain models can be constructed and manipulated naturally, and changes are synchronized with the current transaction automatically

```
from sqlalchemy import create_engine, select, insert
import sqlalchemy
import sqlalchemy as db
help(db)
```

Import SQLAlchemy

Import classes that you need

Or import the library

- Recommended to use an alias

Loading Data with SQLAlchemy

Raw SQL

Object-relational Mapper

Loading Data with SQLAlchemy

Raw SQL

Strings with the full query

Access to available SQL commands

Performance advantages

Flexibility

Object-relational Mapper

May be more robust

Mitigates possibility of syntax errors

Scalability

More secure

Pythonic way

```
engine = db.create_engine(
'mysql+mysqlconnector://root:mysql@localhost:3306/sqlalchemy_mysql')


connection = engine.connect()
results = engine.execute('SELECT * FROM posts LIMIT 10')
```

Load Data Using SQL Queries

Create an engine

- Pass connection string

Open a connection and execute the query

```
first_result = results.fetchone()
first_result
first_results.items()
type(first_result.items())
result.fetchmany(3)
other_results = results.fetchall()
type(other_results)
```

Load Data using SQL Queries

Get results one at a time

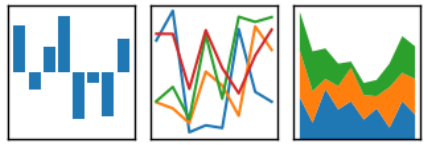
A few at a time

Or fetch all results

Loading Data with SQLAlchemy & Pandas

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Data manipulation and analysis library

Widely used in the Python world

View and work with your data

- Data we just loaded using SQLAlchemy

```
import pandas as pd

query = 'SELECT * FROM posts'

posts_df = pd.read_sql_query(query, engine)
```

Loading Data with SQLAlchemy & Pandas

Start in a very similar way, with the import

- Define the query

Use pandas's read_sql_query method to load into a DataFrame

```
posts_df.columns
posts_df.count()
posts_df.head()
posts_df[['ViewCount', 'AnswerCount']].max()
posts_df[['ViewCount', 'AnswerCount']].min()
posts_df[['ViewCount', 'AnswerCount']].sum()
posts_df.groupby('OwnerUserId')
```

Work & View your Data using Pandas

Inspect and work with your data

Numerical functional available as pandas leverages numpy

Looking at data gives you precise information

- But some things are harder to infer

“An image is worth a thousand words… or in our case thousands of rows with many columns”

Heard it a few times…

```
posts_df[[‘AnswerCount, ‘ViewCount’]].dropna()[:50]
```

Why Visualizations?

Numbers can give precise data

But hard to look at the bigger picture

Visualizing & Graphing Data with Matplotlib

2D plotting library

- Plots, histograms, bar charts, scatterplots...
- With just a few lines of code

Publication quality figures

- Shell, notebook, and more

Easy things easy and hard things possible

```
import matplotlib.pyplot as plt
x = posts_df['AnswerCount']
y = posts_df['ViewCount']
import numpy as np
colors = np.random.rand(4500)
plt.scatter(x, y, c=colors)
```

Creating Visualizations with Matplotlib

Import matplotlib.pyplot

Specify data positions

Specify which type of visualization

```
plt.title("Posts: Views vs. Answers")
plt.xlabel("Answers")
plt.ylabel("Views")

plt.show()
```

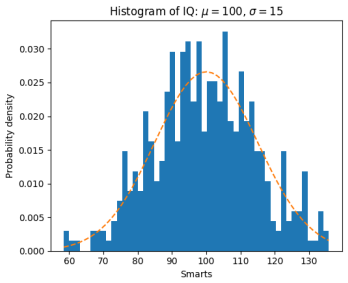
Creating Visualizations with Matplotlib

Add title and labels

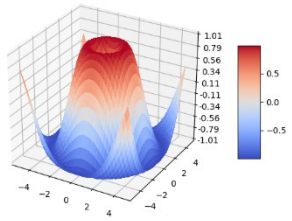
And show

Works a bit differently in a shell vs. a notebook

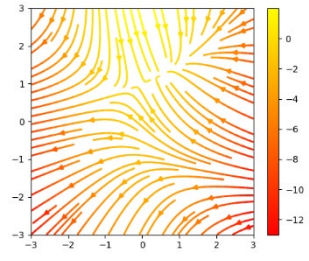
Visualizing & Graphing Data with Matplotlib



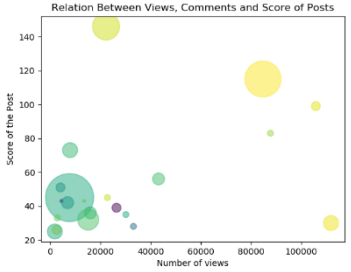
Histogram



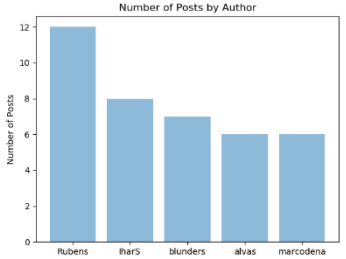
3D Surface



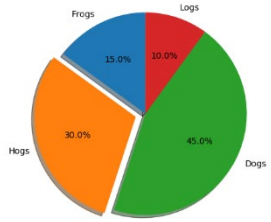
Streamplot



Scatter plot



Bar chart



Pie Chart

Takeaway

SQLAlchemy

- Python SQL toolkit
- Object-relational mapper

Multiple types of databases

Raw SQL or ORM

Takeaway

- Install and import
- Import
- Create engine, connect, and execute
- Retrieving results from the database

Takeaway

Load into a Pandas DataFrame

- Use the available functionality

Visualize with Matplotlib