ARCHITECTURE

ANALYZING SWIGGY

Version:	1.0
Last Revised Date:	16-09-2021
Written By:	Balaji Mummidi

Contents

1. INTRODUCTION:

1.1 WHAT IS ARCHITECTURE DESIGN DOCUMENT?

Any software needs the architectural design to represents the design of software. IEEE defines architectural design as “The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

The software that is built for computer-based systems can exhibit one of these many architectures.

Each style will describe a system category that consists of:

• A set of components (eg: a database, computational modules) that will perform a function required by the system.

• The set of connectors will help in coordination, communication, and cooperation between the components.

• Conditions that how components can be integrated to form the system.

• Semantic models that help the designer to understand the overall properties of the system.

1.2 Scope

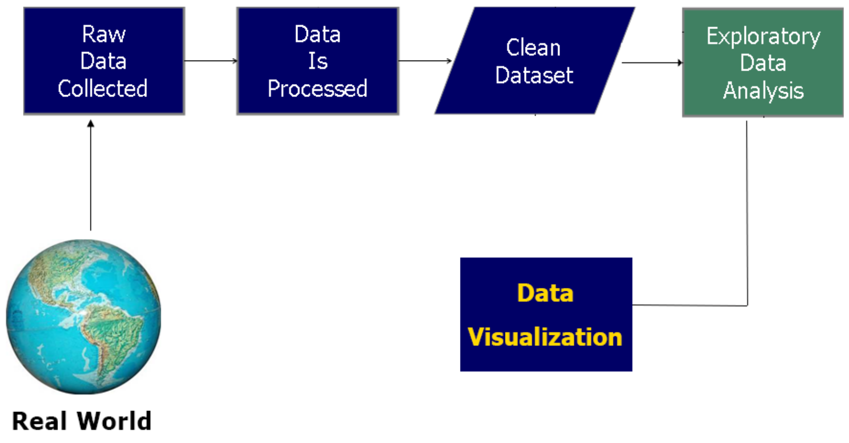
Architecture Design Document (ADD) is an architecture design process that follows a step-by-step

refinement process. The process can be used for designing data structures, required software

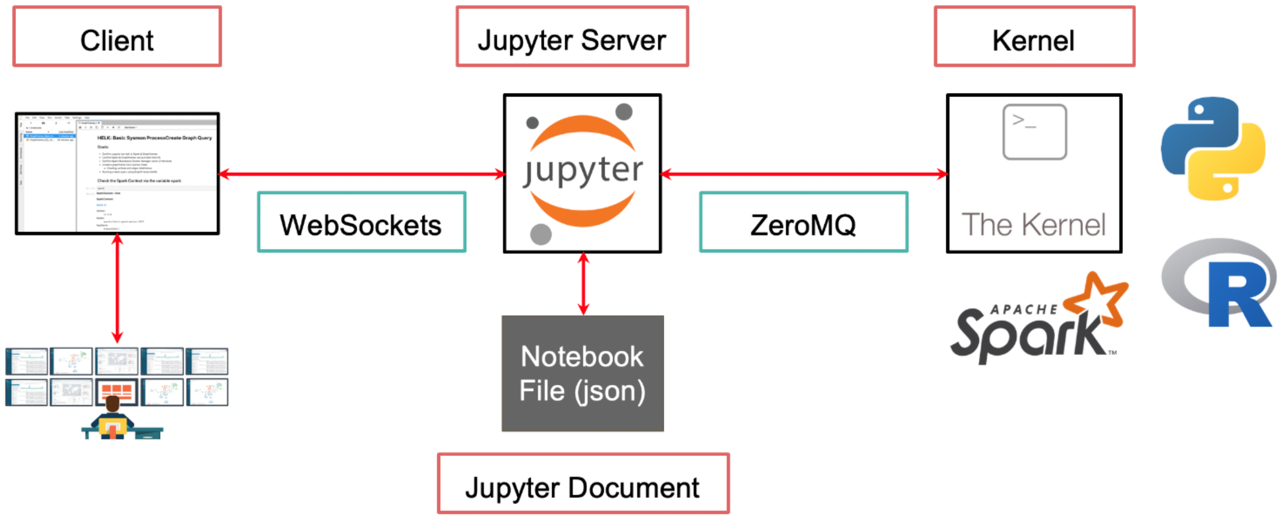
architecture, source code and ultimately, performance algorithms. Overall, the design principles

may be defined during requirement analysis and then refined during architectural design work.

2. Architecture



2.1 Jupyter Notebook Architecture

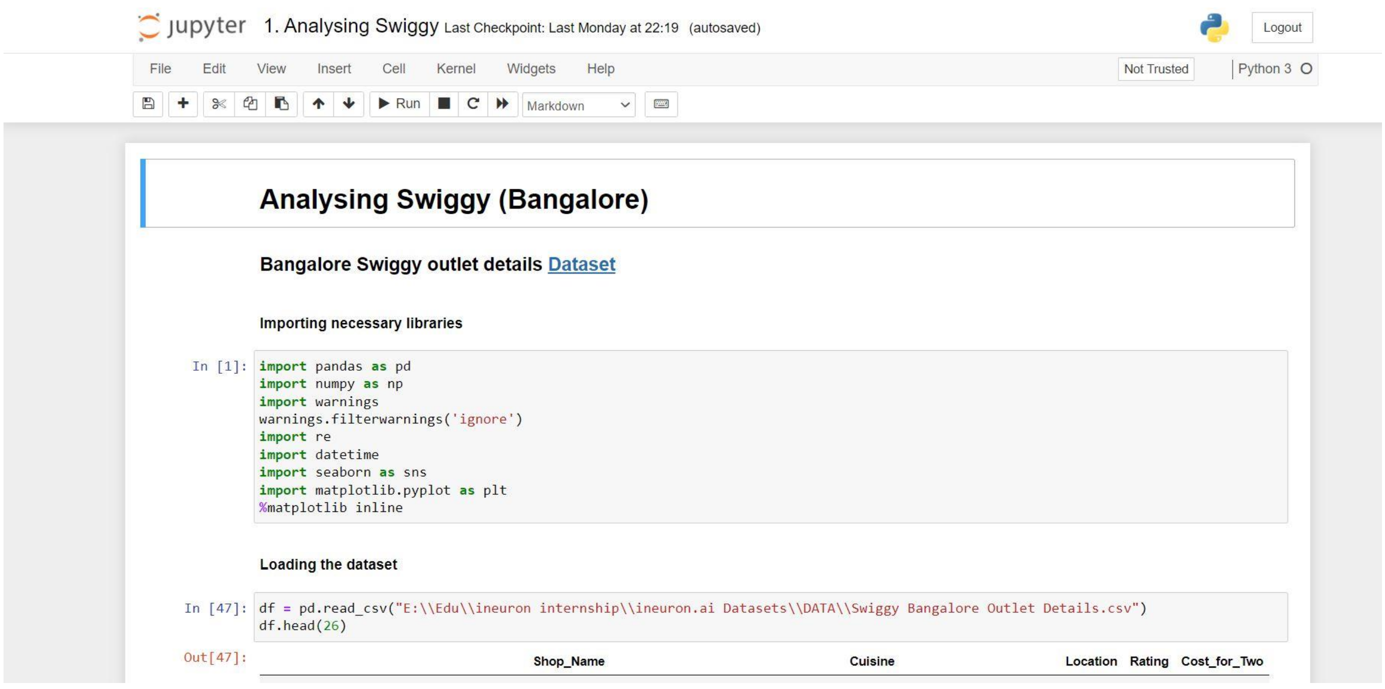


A Jupyter notebook's architecture comprises four components: a web browser, a notebook server, a notebook file, and a kernel. The user relies on a web browser to interact with the notebook server, which reads the notebook file and returns it to the browser to render as a web page. This user interface rendered in the web page includes individual cells—sections of either code or formatted text—in which the user can interactively type.

A code cell can be executed by the user: the browser sends the code in a cell to the server, which routes it to the kernel, which runs the code and returns the result to the browser via the notebook server. The browser renders this result inline in the notebook beneath the code cell. Thus, computational output such as individual calculations, tables, or figures appears beside the code that generated it.

Computational notebooks save, store, and present the resulting output of their code within the notebook itself. By integrating these elements, they blend what the code is (code cells) with what the code does (narrative) and with what the code produces (output). The notebook becomes a "one-stop shop" that allows the user to focus what the code is doing and why and how it is doing it (à la literate programming), as well as to explore and tinker with it while immediately seeing how those changes translate into different results (à la interactive computing). Another important feature is notebooks' distributed architecture. The notebook server could be running on your own computer, in a server room down the hall, or even on the other side of the world—yet you have easy access to it through any web browser. The server itself is language-agnostic, but notebook kernels are language-specific. Hundreds of Jupyter kernels exist for dozens of different programming languages. The most popular languages for data science and spatial analysis are all supported, including Python, R, and Julia.

2.2 Notebook Interface


jupyter
1. Analysing Swiggy Last Checkpoint: Last Monday at 22:19 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

Analysing Swiggy (Bangalore)

Bangalore Swiggy outlet details Dataset

Importing necessary libraries

```
In [1]: import pandas as pd
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')
        import re
        import datetime
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
```

Loading the dataset

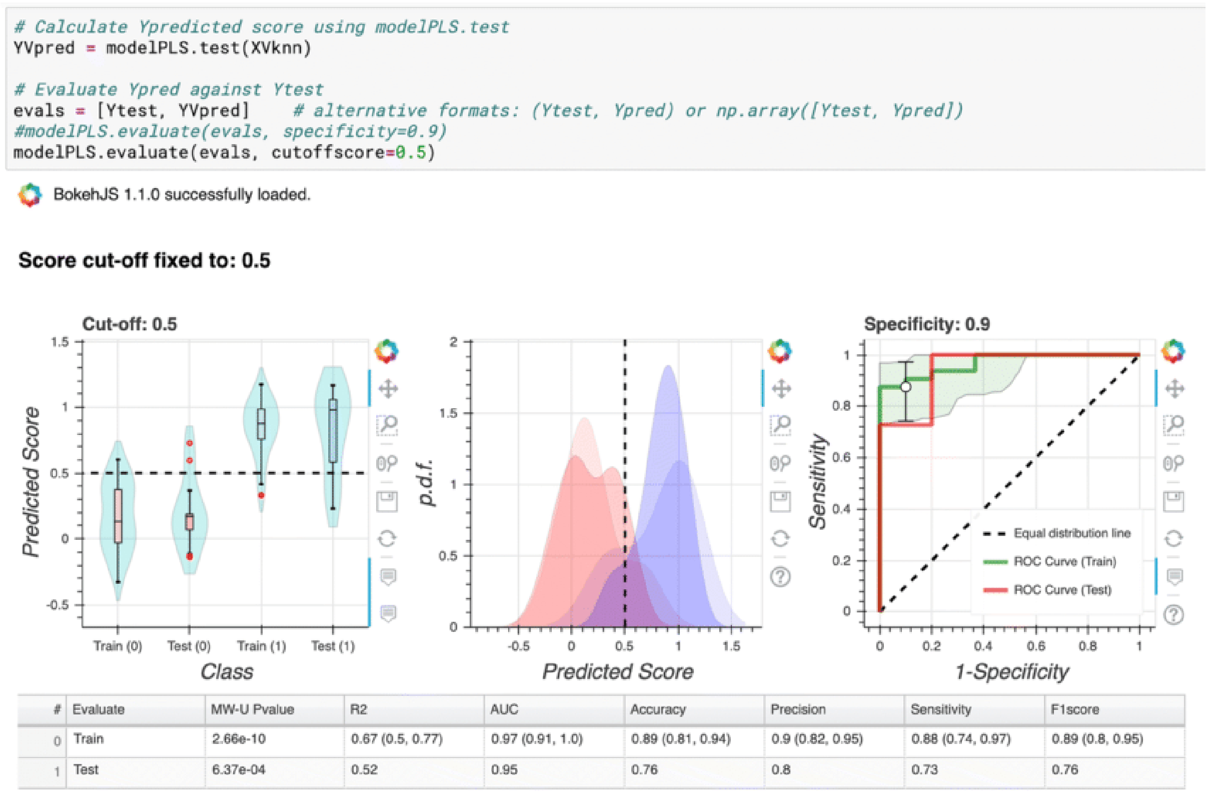
```
In [47]: df = pd.read_csv("E:\\Edu\\ineuron internship\\ineuron.ai Datasets\\DATA\\Swiggy Bangalore Outlet Details.csv")
         df.head(26)

Out[47]:          Shop_Name                    Cuisine                Location   Rating   Cost_for_Two
```

Fig – Jupyter Notebook Interface

In the Jupyter ecosystem, the user interface in the web browser is called JupyterLab. JupyterLab shares many common elements with other computational notebooks. It primarily consists of a main work area and a sidebar to browse files and running kernels. The main work area displays the currently open notebooks. Here, notebooks can be created, edited, and executed. Users can add or remove notebook cells, move cells around to reorder their execution, type code or markdown text into cells, or execute one or more cells.

2.3 Notebooks in Action

Today, the geographic data science ecosystem is most robust in R and Python, where many packages exist to support spatial analysis and modeling such as geopandas (geospatial data handling), PySAL (advanced spatial analytics and modeling), matplotlib (data visualization), OSMnx (street network modelling and analytics), cenpy (working with US Census Bureau data), contextily (basemap tiles), folium (web mapping), and many more: see the Additional Resources section for links. These tools allow geospatial data scientists to fully replace traditional desktop GIS software with reproducible, universal analytics workflows in computational notebooks.



```
# Calculate Ypredicted score using modelPLS.test
YVpred = modelPLS.test(XVknn)

# Evaluate Ypred against Ytest
evals = [Ytest, YVpred]     # alternative formats: (Ytest, Ypred) or np.array([Ytest, Ypred])
#modelPLS.evaluate(evals, specificity=0.9)
modelPLS.evaluate(evals, cutoffscore=0.5)
```

BokehJS 1.1.0 successfully loaded.

Score cut-off fixed to: 0.5

#	Evaluate	MW-U Pvalue	R2	AUC	Accuracy	Precision	Sensitivity	F1score
0	Train	2.66e-10	0.67 (0.5, 0.77)	0.97 (0.91, 1.0)	0.89 (0.81, 0.94)	0.9 (0.82, 0.95)	0.88 (0.74, 0.97)	0.89 (0.8, 0.95)
1	Test	6.37e-04	0.52	0.95	0.76	0.8	0.73	0.76

Fig – Sample figure of notebook in action

3 Conclusion

The open science movement promotes transparency and reproducibility in the workflows underlying data analyses. Traditional desktop GIS's reliance on GUIs makes the software easy to learn and use, but difficult to fully document, understand, and replicate complex spatial analysis projects. Computational notebooks offer an alternative paradigm for scientists, educators, and analysts across disciplines—including quantitative geography. On one hand, notebooks' reliance on code creates a steeper learning curve. On the other hand, they lend themselves much more readily to transparency, reproducibility, documentation, dissemination, and modern data science workflows.