**Project Report – ML Dashboard and PDF Chatbot Web Application**

**Submitted by : Niteesh Kumar Singh**

**Table of Contents**

[**1.Introduction** 3](#_Toc195436532)

[**2. Background and Motivation** 4](#_Toc195436533)

[**3. Application Architecture** 4](#_Toc195436534)

[**3.1 System Overview and Component Roles** 4](#_Toc195436535)

[**3.2 Data Flow and Integration Points** 5](#_Toc195436536)

[**4. Machine Learning Model Development** 5](#_Toc195436537)

[**4.1 Data Preparation and Feature Engineering** 5](#_Toc195436538)

[**4.2 Model Evaluation and Export** 6](#_Toc195436539)

[**5. PDF Chatbot Design and Implementation** 6](#_Toc195436540)

[**5.1 Vector Search and Embedding Strategy** 6](#_Toc195436541)

[**5.2 Question Answering with Transformer Models** 7](#_Toc195436542)

[**6. Visualization and Frontend Design** 7](#_Toc195436543)

[**6.1 Dashboard Charts and Insights** 7](#_Toc195436544)

[**6.2 User Experience and App Flow** 8](#_Toc195436545)

[**7. Conclusion and Recommendations** 9](#_Toc195436546)

[**8. Appendix** 10](#_Toc195436547)

[**8.1 Visualization Screenshots of my project:** 10](#_Toc195436548)

# **1.Introduction**

The field of data science has become dynamic in recent years and requires an application of machine learning for prediction and NLP for interacting with data. The given report focuses on the process of designing, developing, and deploying a web application having mutual functionality developed during the Imarticus’ Data Science Internship – Assessment. It is the general objective of the project to demonstrate how machine learning-predictive modeling can be incorporated with smart blueprint chatbot that can search for information in structural pdfs.

This application will allow the users to upload a csv file having dataset, make real-time prediction using regression model and visualize the findings through a beautiful dashboard. At the same time, it enables users to search through subject-specific PDFs, including ones on Python, SQL, and Machine Learning, based on plain language terms. The app is developed using Python language and Streamlit, the main Machine Learning and NLP libraries used in the application include Scikit-learn, FAISS, HuggingFace Transformers, and Sentence Transformers. Hence, one gets an overall end-to-end, click-able data piece to showcase foundational software engineering and enhanced insight into the quantitative analysis.

## **2. Background and Motivation**

Each of the above fields is usually studied as a subject of machine learning or NLP, but their combination creates powerful tools for data analysis. In most business and technical scenarios across organizations, computer users are often required to work with both the structured data such as tabular data and the unstructured data like documents or system policies in PDF format. These qualities if incorporated in a single system made the need for this integrated project possible. At the internship, it is crucial to demonstrate how the concepts of data engineering, models making, models deployment, vector search, and the design of the interface are implemented. Instead of just notebooks, it became an attempt to create something that is both usable and deployable, and most important interpretable.

This project offers instant estimation of traffic volume or “click through rate” considering such tentative variables including product type and price, time of the day, among others. Besides, using the chatbot it is possible to search topics in the academic or cheat-sheet PDFs, for instance, in the format “What is JOIN in SQL?” to write scripts for “Explaining for loops in Python” and the answers should be garnered from the documents uploaded. It was not just to show segmentation algorithm, but usefulness, interactivity in a way that os easily understandable by the end users without much information processing by the brain.

## **3. Application Architecture**

### **3.1 System Overview and Component Roles**

Used in its current setting, the architecture of this application is optimized multi-purpose and has a minimalist look and feel. Indeed, the application consists of two major parts—an ML dashboard and a chatbot that are both available in the same Streamlit application but in different tabs. The ML dashboard deals with data pre-processing, loading of the model and prediction process. On the other hand, the chatbot uses vector indexing to facilitate indexing of the text chunks in the PDFs by employing them as documents and employing a transformer-based QA model to contextualize the answers.

In the ML dashboard, there is a pre-trained regression model which is stored in serialized form as “regression\_model.pkl”. can parse standart CSV files with flat fields like product\_type, price and either hour\_of\_day or more complex timestamp. When a timestamp is detected, then the extract of the hour is dynamic. As for the backend of the chatbot, it employs FAISS for the identification of the most semantically similar text chunks in relation to a certain query. These chunks are then having to be handed over to the Hugging Face distil XLM vL bottom sub/super network distilbert-base-uncased-distilled-squad model which retrieves an adequate answer to the input question.

### **3.2 Data Flow and Integration Points**

This application opens in two tabs only, but before that let me describe its first tab. In this particular tab, a user inputs data in a CSV format; this data is then analyzed and categorized before going through the modeling stage. It provides the new column of the form of prediction called ‘Predicted Clicks’ and this forms the final and new form of the data set. This augmented dataset is represented and shown in the interface with three kinds of visualization that are a bar chart indicating average predicted of click through per product type, time series line chart of predicted click through (if a timestamp is provided) and lastly a comparison line chart of actual and predicted click through (if actual labels are provided). Such features allow other people who are not allowed backend access to readily understand results from models.

In the chatbot tab, the user can decide to search in any of the given documents within python.pdf, sql.pdf or ml.pdf. Each of them submits a question in natural language that is then transformed into a sentence embedding using a MiniLM model. These embeddings are compared to the indexed PDF chunks stored in FAISS, SELECT and a relevant text is fetched, and a sentence-based answer is returned from the QA model. As for the full context of the match, it is also shown to provide transparency as to the source file of the data.

## **4. Machine Learning Model Development**

### **4.1 Data Preparation and Feature Engineering**

The construction of ML model utilized in the next dashboard, a dataset which considers fields such as product type, price, and time-based metrics was compiled. The feature engineering was aimed at transforming product\_type into ordinal, converting categorical variable into dummy/indicator, and deriving hour of the day from the timestamp if any. In particular, the price and hour features were maintained as numeric, use N\_Binarize() for binarization of the several features found in the textual data used. The numerical inputs were normalized using the Scikit-learn’s StandardScaler while categorical inputs were one-hot coded.

As for the target variable, clicks were continuous, which is why regression as the type of analysis was chosen. The data was then split between train and test classes and the models used to consider comprise of the Linear Regression, Ridge as well as the Random Forest Regressor. The last chosen model was the one that gave satisfactory performance and reasonable speed and was saved with joblib.

### **4.2 Model Evaluation and Export**

The performance on the other hand was assessed by the mean square error (MSE) and R² score. Because the data has been fabricated and due to lesser features the initial values are much higher when the number of iterations increases, but after data manipulation accuracy has been improved and MSE was also lower. The model was then serialized and saved as regression\_model.pkl in the file format in order to make it easily accessible for use later on. This model was installed in the Streamlit app during the time of execution and was tested on real and fake datasets.

Another advantage comes from the fact that the model is lightweight; hence, it can generate the real-time predictions immediately. The preprocessing and pipeline components are also saved within the model to be reused when it is used in the actual production setting. It becomes integrated with the data uploaded by the business user, and the results are presented in the format of the charts and output to the business user.

## **5. PDF Chatbot Design and Implementation**

### **5.1 Vector Search and Embedding Strategy**

It addresses only questions and answers, it involves the use of the Retrieval-Augmented Generation (RAG) model. First, the input of each type of the uploaded PDFs (Python, SQL, ML) is split into more complex passages of the means of text processing. These blocks are converted into the final query vectors of the fixed length using the all-MiniLM-L6-v2 model from Sentence Transformers. The chunks that are vectorized are then indexed using FAISS, which facilitate fast similarity search over documents among huge data sets.

When a question is typed in, the latter is transformed into an element of the same vector space. Thus, there is carried out the cosine similarity search to select the 5 most similar to the input text blocks in the FAISS. This means that even if the query might be some distortion from the way it appears in the PDF, similar in terms of meaning, documents that are likely to be related will be captured. In order to remove noise and bring the subsequent step, only the fragments of the text are extracted for the context that is relevant to it only.

### **5.2 Question Answering with Transformer Models**

The context section is appended to the original question as an input into a HuggingFace QA pipeline with distilbert-base-uncased-distilled-squad model. It involves understanding the text and identifying the most varied answer to the posed question or the most relevant query in the case of question exposure. This makes it very accurate without having to produce what we could describe as hallucinations or non-sensical text.

The capability also extends from brief, straightforward-referential questions such as “What is supervised learning?” or “How does a year come out of a date in SQL?” The output includes the direct answer to the question and the name of the PDF used with the piece of text being used for the answer highlighted. It also promotes user trust in the reported answers and creates interest and willingness to read through technical texts in the form of a chatbot.

## **6. Visualization and Frontend Design**

### **6.1 Dashboard Charts and Insights**

The model projections are supported by multiple types of visual layers in the dashboard. A bar chart represents the average of the predicted click which will help for categorizing high or low engagement of the product types. A line chart shows the actual or predicted numbers of clicks on the Y-axis, if the date stamp is present; this can then help a business user find a time-related pattern. Furthermore, if actual click values are included in the input dataset, the app also provides an output of the chart of actual versus predicted click.

The ability of these visualizations to be integrated in the Streamlit interface and automatically update according to the data uploaded by the user. This has been facilitated by the clean look and feel design in order to make the app very friendly to any user, including the non-technical individuals and as such very useful. To enhance the tolerance of the program, missing columns and the wrong format are also handled.

### **6.2 User Experience and App Flow**

One of the concepts constantly present is the principles of simplicity and speed of its usage. Loading the interface is not slow and although it opens large files the appearance of errors is not fatal. They range from section headings, contextually helpful messages and previews of information or data presented on that particular page. Each chart and output table has an individual and clear look and all are updated from the input if the arrows are pulLED. Just like with the web-chat, the input prompts and the generated answers in the chatbot tab include the option to expand context as well as the traceability of the source.

In summary, the frontend is effective in getting the user from the Data Upload, Prediction, Analysis, and Interaction with Data and Documents functions. It is operational and adaptable as a tool for e-commerce, technical education application, or even in automated customer services.

# **7. Conclusion and Recommendations**

Reading and summarising this proposal, it can be concluded that this is an effective approach that combines supervised machine learning and NLP-based question answering system into one application. There is utilised good pipeline data, approach to deploy the model, semantic vector search and good use of streamlit for web development. Through the integration of real-time prediction and humanized interface in the form of well-organized document, the application provides a stable and friendly environment.

This dual-function platform can be extended further, and is suitable for the development of the society as a whole, as well as for different categories of individuals and families. Additional future developments will include PDF upload through the frontend, language translation, more elaborate ensemble models, and real-time logging in case of the product’s deployment for enterprise use. Moreover, taking into account a feedback system in which the user provides a rating to an answer, or marks it as correct or incorrect could help enhance the performance of the chatbot overtime.

# **8. Appendix**

## **8.1 Visualization Screenshots of my project:**

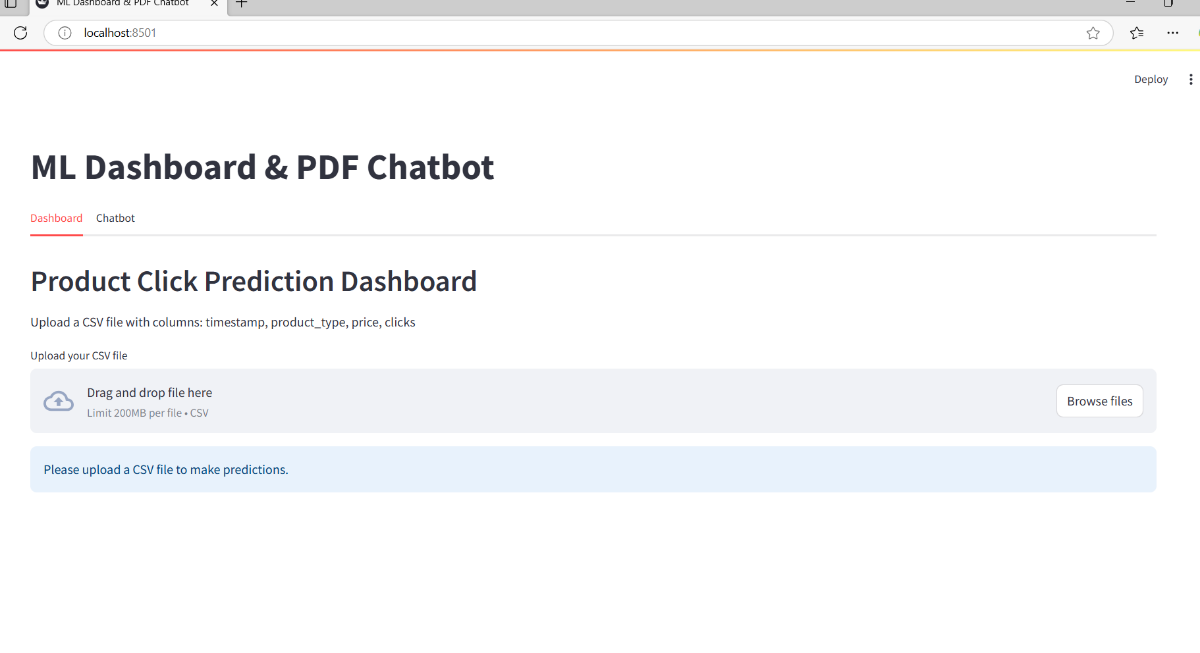


Figure 1 Code Output : Home UI

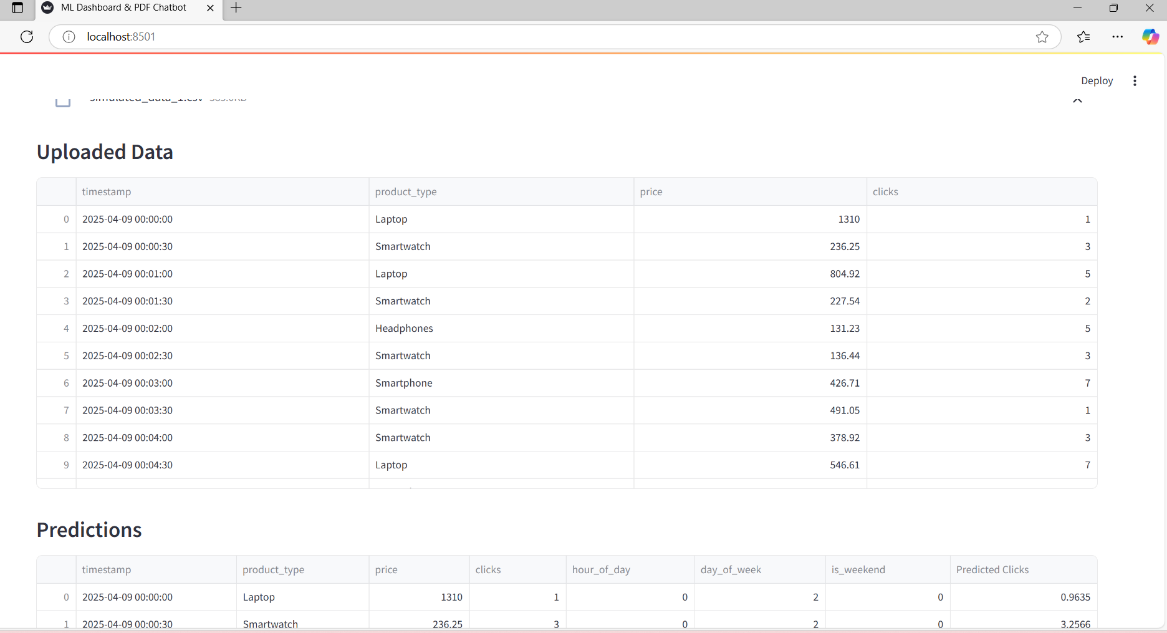


Figure 2 Code output : data Format

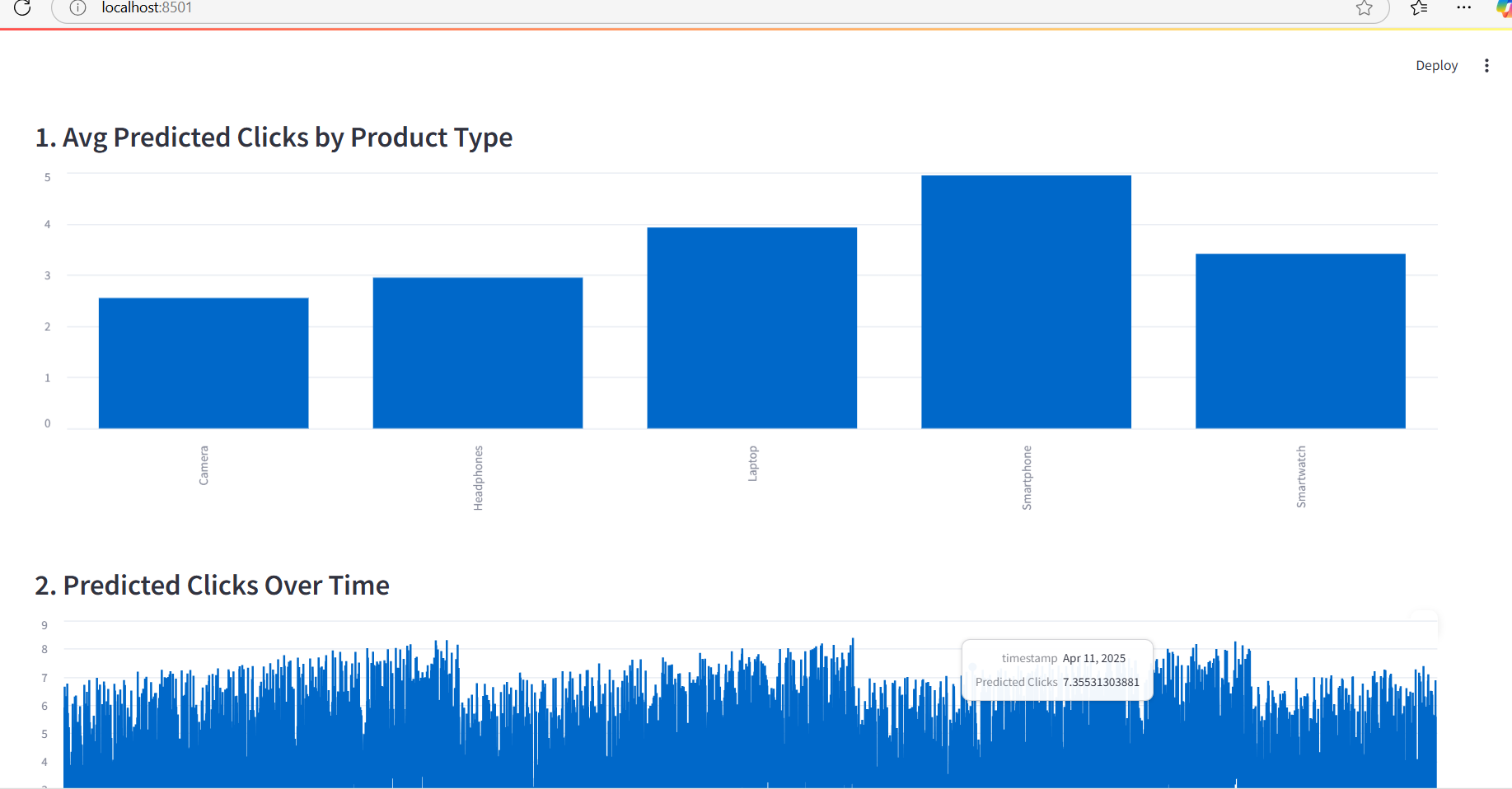
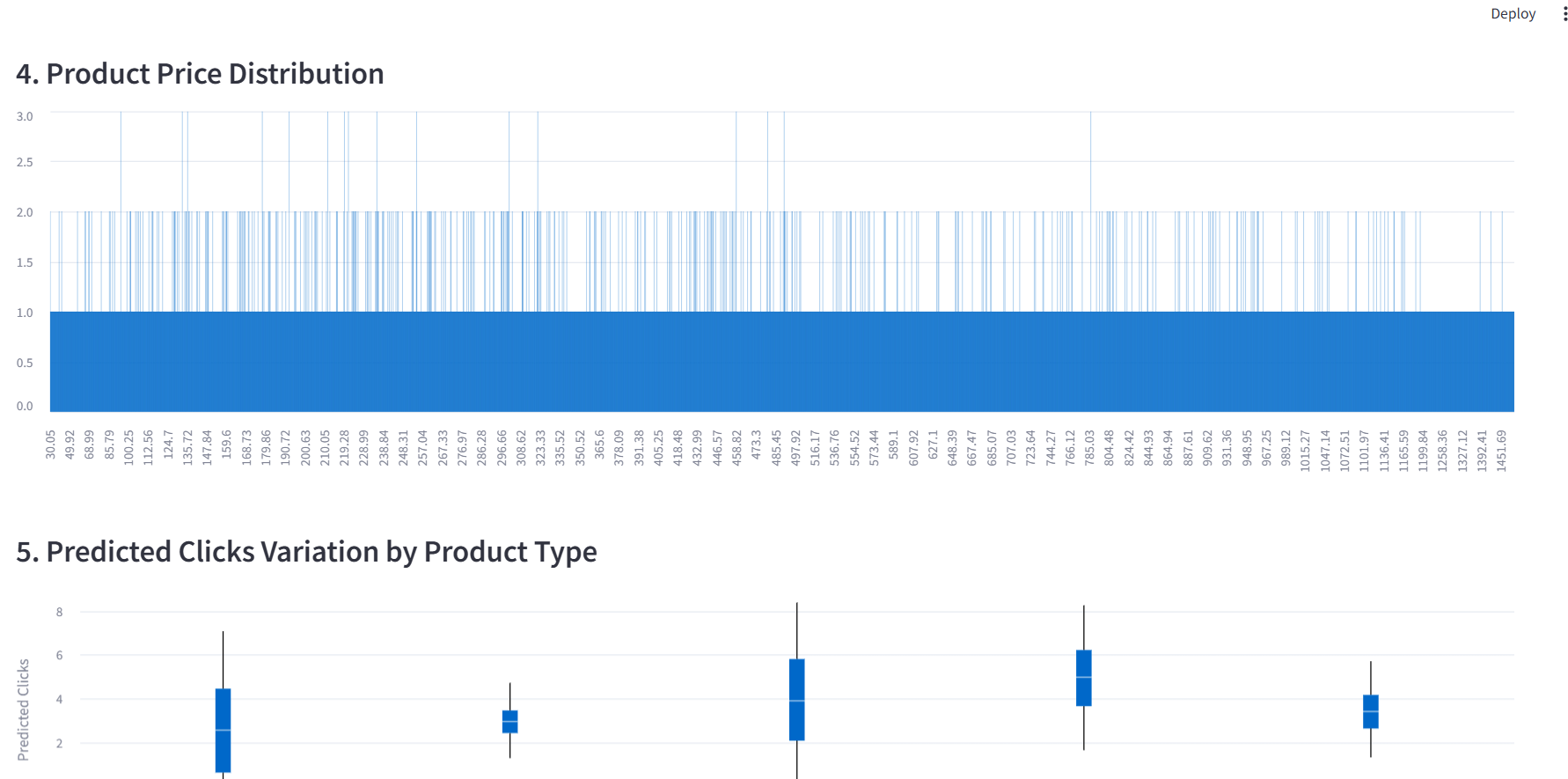


Figure 3 Code Output Dashboard



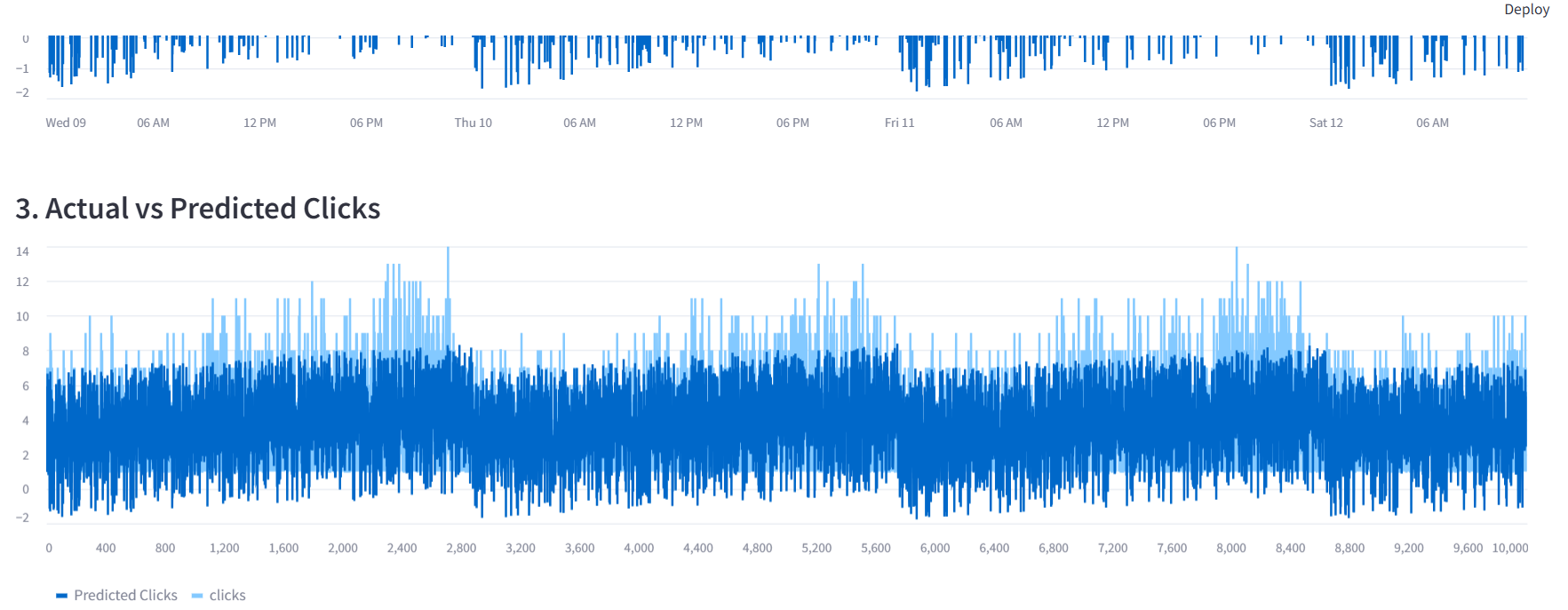


Figure 4 code Output Dashboard

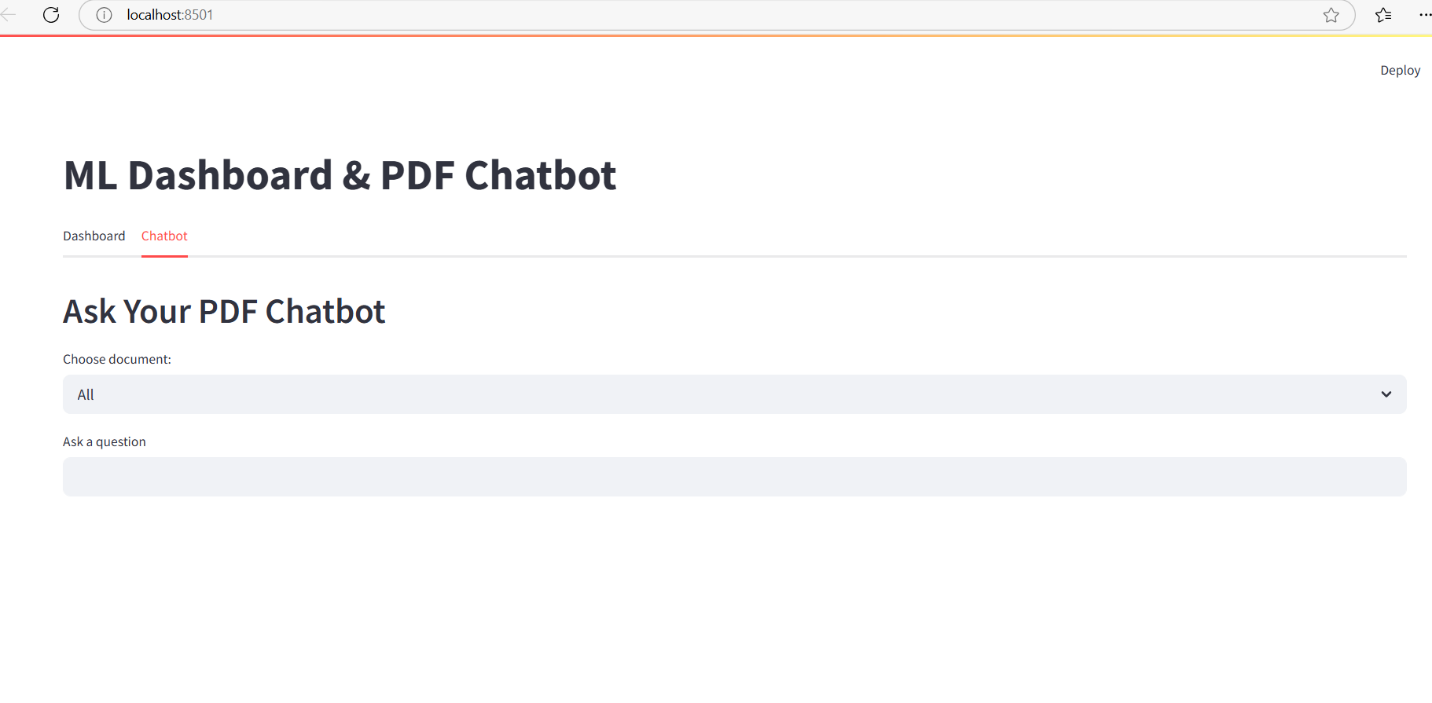


Figure 5 code output Chatbot

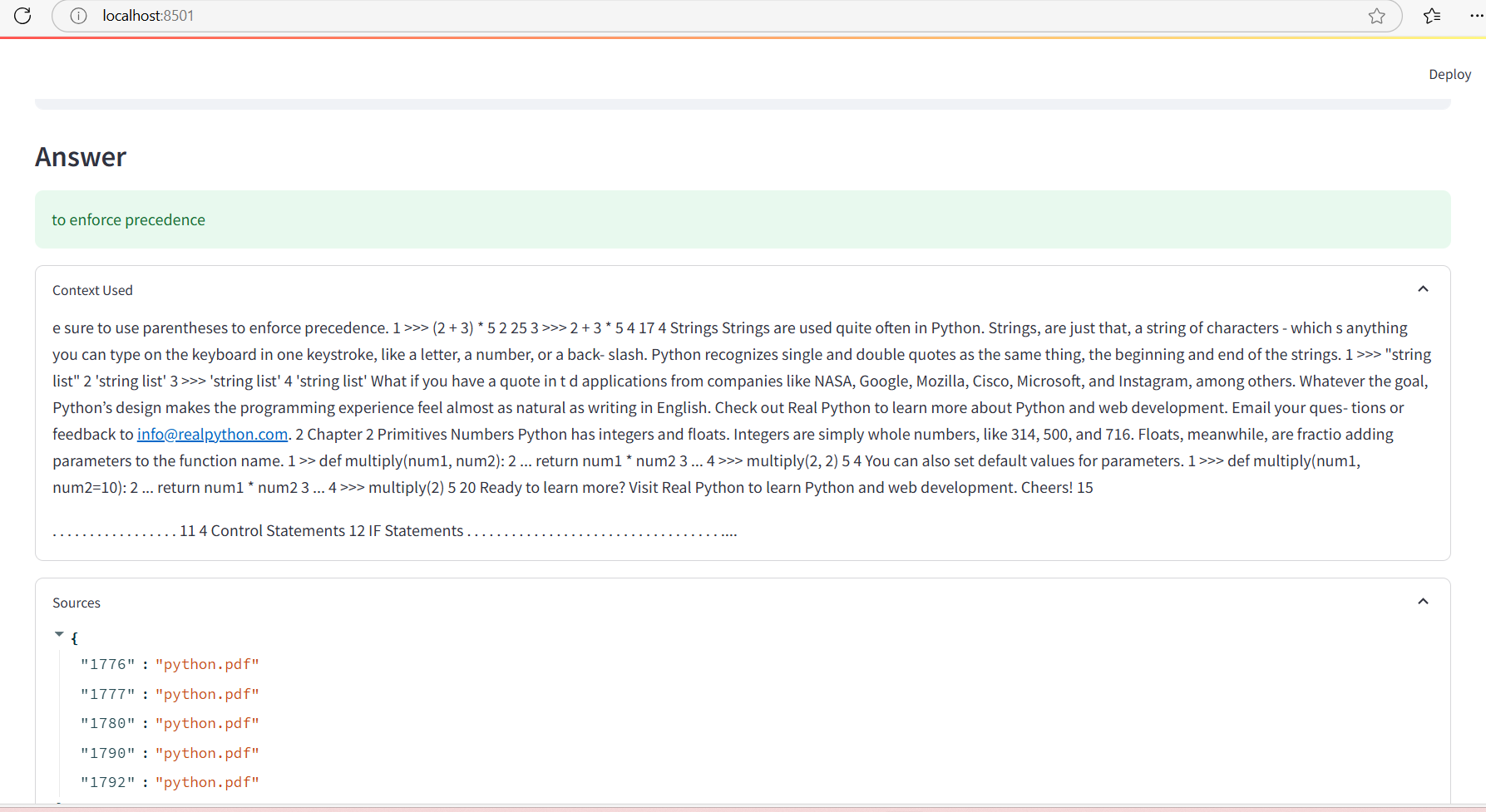
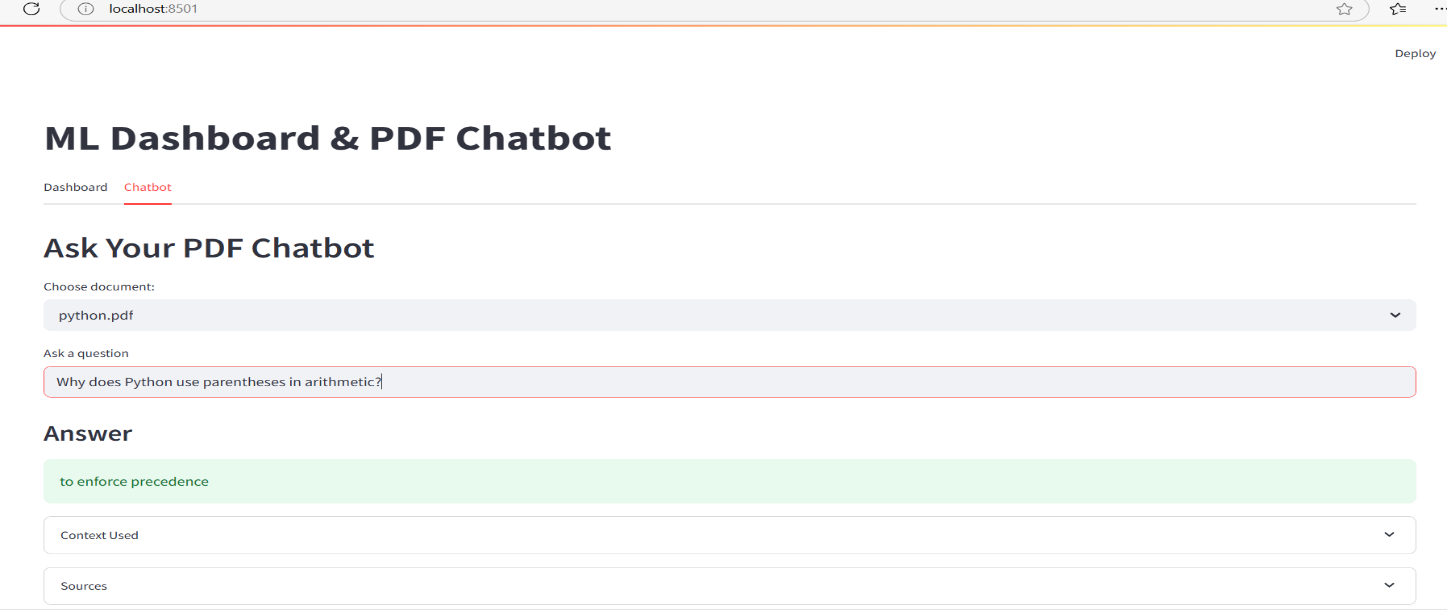


Figure 6 code output Chatbot workflow