**Project Proposal: Project 3, Team 1**

**Team Members: Geoff McDaniel, Chris Alvarez and Doug Francis**

**Title:** **Automated Graphing and Data Processing through Voice Commands and Human Prompts**

**Overview:** This proposal outlines the development of an innovative application designed to revolutionize Exploratory Data Analysis (EDA) by automating custom graphing processes of any dataset provided. Utilizing human prompts and voice inputs, this application aims to streamline the creation and manipulation of data visualizations.

**Detailed Description:** The primary objective is to create a user-friendly, web-based application that facilitates automatic generation of custom graphs for any given tabular dataset through simple voice commands or text prompts. This technology will enable users to engage with data analysis in a more intuitive and efficient manner, significantly reducing the manual effort involved in data processing.

**Technologies:**

1. **Streamlit:** This web framework is specifically tailored for data science applications, allowing Python developers to easily build and deploy web applications. Streamlit will serve as the foundation for our web-based interface, ensuring a seamless user experience.
2. **Langchain and Langraph:** These tools will be employed to establish a robust communication link between Python and a Large Language Model (LLM). The LLM will be responsible for generating custom Python code based on user inputs, which will then be executed to create graphs.
3. **Python and Matplotlib:** The Python code generated by the LLM will leverage Matplotlib, among other potential libraries, to produce a wide array of graph types. These graphs will then be rendered and displayed within the Streamlit application, offering users a dynamic and interactive data visualization tool.

By integrating these technologies, our application will not only enhance data visualization capabilities but also transform the approach to EDA, making it more accessible and efficient for users across various domains.

**Not in Scope / Limitations:** The application will only work with csv files that are in a “long” tabular format, such as a database table. Prior to use with this application, data may need to be cleaned or reformatted to this format.

1. **Application / Model Optimization**
   1. Per the accepted project proposal, we did not do a standard model that needed hyper tuning, etc. Instead we did an application that used text (or voice if that comes together) to generate human graphing requests on any data set that is in “tall” form, like a normalized database table or (tall) csv file.
   2. Optimization of the application.
      1. In the LangGraph workflow, both a system prompt and a user generated prompt were passed to the LLM. The system message and the user prompt were used on a variety of data. The application takes the code generated by the LLM and parses it out from the returned LLM message. In turn this is pushed into a graphing file, that Streamlit can then call to generate the graphs. The system message was continuously revised since many of the initial graphs would fail. This tuning was essential to improve performance.
      2. The initial one page implementation was too busy. Hence the “Select data…” page was made the initial request generation page and a “create\_graphs…” page was added to process subsequent graphing requests.
      3. Graphing page was improved by writing out the actual request before the generated graphs. To prevent Streamlit write functions from failing, the user request was stripped of leading and tailing spaces and linefeeds.
      4. Complex graphing requests can sometimes fail. So there is a ‘save graph file’ as well as a restore graph file button to undo changes (if the graph file was saved prior processing the new request.
         1. While originally envisioned for use by non-Python programmers, the occasional failures might be frustrating. For a python programmer, they can go into the code and correct the usually simple mistakes made by the LLM.
      5. 3D plotting was added using it’s own page. The user specifies 3 or more columns, as well as an optional ‘colorby’ variable and the application generates a rotatable and resizable 3D plot for all combinations of the columns specified. By all combinations we mean that if three columns are specified, there is only one combination, so one 3D graph is produced. If the user specifies four columns, then three graphs are produced (if the four columns are A, B, C and D, then the 3D plots would be {x,y,z} includes [{A, B, C}, {A, B, D}, {B, C, D}]. If five columns are selected, then six graphs would be produced. Limitation: The 3D graphing routine seems to stop generating images if too many columns are selected. However, the request can be broken down into smaller bites to avoid an issue.
      6. Considerable debugging was needed since the Streamlit session state was not visible in the LangGraph workflow. Global variable were used to overcome these issues.
      7. Management of the Streamlit session state between pages with the different options was probably the most challenging aspect of this project, requiring considerable optimization and debugging.
      8. Application was updated to allow the user to specify alternate graphing files. The default file is over-written each time the application is used. However, if alternative graphing files are specified, these are persistent saving work.
2. **Application / Model Performance**
   1. The application’s performance in its final state was tested with four datasets. PDF’s of some of the testing are saved in the data directory as PDF’s. See the data directory. The file is called “Application Performance Score Card.xlsx”.
   2. Overall, of 23 requests, 20 worked first time. After adjustments 2 or the 3 failures passed on subsequent testing after minor change to system prompt and other minor Python code changes.