The ETL processor we created works to process data through the three main mechanisms; Extract, Transform, and Load. A user can input a local file, allowing the processor to extract/access the data. Once that file is uploaded, it's converted to a pandas dataframe first so the user can make further transformations. Specifically our ETL processor allows the user to keep, add, or drop columns. The user can also choose the output format of the data, converting it to either a CSV, JSON, or Sqlite file. The last mechanism of any ETL is to load the data, ours is written so that the data can either be stored as a CSV, JSON, or Sqlite file.

In our example, we uploaded a local file from Charlottesville Open Census Data, /content/US_Census_Tract_Area_2010.csv. It has 12 records and 353 columns of multiple demographic attributes. We formatted the ETL processor to allow the user to keep, add, or drop columns. The Census Tract 2010 data contains so much information so we aimed to clean up and narrow the attributes of the dataframe. For example, the columns County and State hold no particular relevance, since all the values are the same for each row (all of the data is from one place). To narrow the attributes of the data frame and filter out low-quality data, we specified the processor to only keep the following columns: ID, AREA_, Population, White, Black, AmIndian, Asian, Hawiian, and Other.

Throughout the process of building our ETL pipeline we found some features to be easier than expected, encountered challenges, and addressed some limitations. Throughout development we faced simple problems associated with data transformation and error handling. JSON structures in particular are more complex as flattening a JSON structure or transforming requires more planning and additional steps. For example, we had to make sure to include the function pd.json_normalize() as it helps transform a complex JSON structure into a flat dataframe, allowing for further analysis. In contrast, we found certain features such as using libraries like pandas to be easier than expected. Once the data was ingested from a local source, using pandas to convert the data to a dataframe from various file formats turned out to be very straightforward and efficient. It was additionally interesting to see the limitations of an ETL pipeline. Our ETL pipeline would definitely be challenged when dealing with high-volume data operations or if it were to draw from multiple sources.

Overall, an ETL pipeline that ingests, transforms, and loads data will be extremely useful for future data projects. Our pipeline functions to transform the columns of a dataset, however the options for data transformations are limitless. For example in future marketing analytic projects, an ETL pipeline could clean and aggregate data concerning sales and demographic insights allowing for the opportunity of various data operations. In conclusion, developing this ETL pipeline taught us the many benefits and limitations of data processing systems.