# Differences in the AI generated code and your own code

Name: Siji Mariam Mathew

Student ID: A00325707

## Table of Contents

## 1. Introduction

This report provides a comparative analysis of two implementations for the Q3(a) Polars (import for data analysis) and Altair (import for plots)

Here the original code and an alternative solution generated using AI tools. Both implementations utilize Polars for data processing and Altair for visualization. The goal is to analyze the similarities and differences in logic, efficiency, and visualization while highlighting areas for improvement.

**Polars** is a high-performance Data Frame library optimized for large datasets. Its lazy evaluation mechanism enables efficient data transformations, making it ideal for complex data manipulation tasks. **Altair**, on the other hand, provides declarative visualizations that are interactive and intuitive, making it suitable for creating advanced charts.

## 2. Original Code Implementation

The original code processes the avocado dataset to group data by year and calculate key statistics such as mean, median, and standard deviation. These statistics are visualized using an interactive grouped bar chart.

### Key Steps

- **Loading Data:** The dataset is loaded with scan_csv, which uses lazy evaluation to optimize memory consumption.
- **Calculating Statistics**: Grouping is performed by year, and Polars functions are used to calculate statistical measures.
- **Data Transformation:** The grouped data is transformed into a format compatible with Altair visualizations.
- **Visualization:** A bar chart is created with interactive tooltips to display statistics by year.

Example of grouping and statistical calculation:

**Data Loading**

```
avocado_data = pl.scan_csv('avocado.csv')
```

**Grouping and Statistical Calculations**:

```
wd_2 = avocado_data.group_by('year').agg([
```

```
    pl.col('Total Volume').mean().alias('mean'),
    pl.col('Total Volume').median().alias('median'),
    pl.col('Total Volume').std().alias('std')
```

**Visualization with Altair**:

```
chart = alt.Chart(stats).mark_bar().encode(
    x='year:O',
    y='Value:Q',
    color='Statistic:N',
    column='Statistic:N',
    tooltip=['year', 'Statistic', 'Value']
).properties(
    title='Total Volume Statistics by Year'
).interactive()
])
```

The analysis begins by loading the avocado sales data from a CSV file using the scan_csv function from the Polars library. This function is used in a lazy loading manner, meaning the data is not loaded into memory all at once but is processed in chunks as needed. This method is particularly useful when working with large datasets as it avoids loading everything into memory upfront. The dataset, avocado.csv, contains multiple columns, including year and Total Volume, which are the focus of this analysis.

Once the data is loaded, it is grouped by the year column. This is done using the group_by method, which organizes the dataset by the year of sale. The agg function is then used to calculate three key statistical metrics for the Total Volume column within each year:

1. **Mean (Average)**: This calculates the average total volume of avocados sold each year.
2. **Median**: The median gives the middle value of the total volume for each year, providing a sense of the central tendency of the data.
3. **Standard Deviation**: This measures the spread or variability of the total volume of sales within each year.

The results of these calculations are stored in a new DataFrame, which contains columns for the year and the computed values for the mean, median, and std (standard deviation).

For visualization, the statistical results are plotted using the Altair library. A bar chart is created to display the total volume statistics by year. The x-axis represents the year, and it is treated as a categorical variable. The y-axis represents the Value column, which contains the calculated statistics (mean, median, and standard deviation), and is treated as a quantitative variable. The chart uses different colours to distinguish between the three statistics, with each statistic displayed in a separate subplot (column) for clear comparison.

In addition, a tooltip is added to the chart, which shows detailed information when a user hovers over any bar. This includes the year, the type of statistic (mean, median, or standard deviation), and the corresponding value. Finally, the chart is interactive, allowing users to zoom, pan, and explore the data in more detail. The chart is titled "Total Volume Statistics by Year" and provides a clear, visual representation of the trends in avocado sales over time.

### 3. AI-Generated Code Implementation

The AI-generated solution implements similar tasks with some variations in logic and structure. The main steps include:

**Simulating Data**: A mock dataset is created using Python dictionaries.

**Grouping and Calculations**: Statistics are calculated for grouped data using Polars.

**Data Transformation**: The data is melted to a tidy format for visualization.

**Visualization**: A grouped bar chart is created to display the calculated statistics

.Example of dataset simulation:

```
data = {

    "Year": [2015, 2015, 2016, 2016, 2017, 2017, 2018, 2018],
    "Region": ["TotalUS", "West", "TotalUS", "West", "TotalUS",
"West", "TotalUS", "West"],
    "Type": ["Conventional", "Conventional", "Organic", "Organic",
"Conventional", "Conventional", "Organic", "Organic"],
    "Total Volume": [1000000, 2000000, 1500000, 2500000,
1200000, 2200000, 1600000, 2600000]
    }

df = pl.DataFrame(data)
```

Example of grouping and statistical calculation:

```
stats_df = df.groupby("Year").agg([
    pl.col("Total Volume").mean().alias("mean"),
    pl.col("Total Volume").median().alias("median"),
    pl.col("Total Volume").std().alias("std")
])
```

Example of Altair visualization:

```
chart = alt.Chart(melted_pd).mark_bar().encode(
    x="Year:O",
    y="Value:Q",
    color="Statistic:N",
    column="Statistic:N",
    tooltip=["Year", "Statistic", "Value"]
).properties(
    title="Total Volume Statistics by Year"
).interactive()
```

## 4. Comparison

The table below summarizes the key differences:

| **Aspect** | **Original Implementation** | **AI-Generated Implementation** |
|---|---|---|
| Data Handling | Uses `scan_csv` for lazy evaluation | Simulates data directly with Polars |
| Grouping and Aggregation | Detailed column aliasing and transformations | Simplified column aliasing |
| Visualization | Interactive, tooltip-rich charts | Simpler grouped bar charts |
| Performance | Optimized for large datasets | Suitable for smaller datasets |

### 4.1 Data Handling

**Original Implementation:** The original approach uses scan_csv for lazy evaluation, which allows for memory-efficient processing of large datasets, ensuring that the system does not overwhelm the available memory when working with large-scale data.
**AI-Generated Solution:** The AI-generated approach simulates the data directly. This is simpler and more convenient for rapid prototyping, but it doesn't have the ability to manage real-world datasets dynamically and efficiently like the original method does.

### 4.2 Statistical Calculations

**Original Implementation:** The original solution takes advantage of detailed column aliasing and advanced transformation techniques. This approach fully utilizes Polars' potential to perform complex data transformations and statistical calculations, making it more adaptable to various data analysis needs.
**AI-Generated Solution:** The AI-generated solution uses simpler column aliasing, which may make the code easier to read and understand, but it doesn't fully tap into the more advanced features of Polars, possibly limiting its flexibility for handling more complex tasks.

### 4.3 Visualization

**Original Implementation:** The original solution creates interactive charts with rich tooltips, which enhance the user experience by allowing for detailed exploration of the data directly within the visualization. This provides a more dynamic and engaging way to interpret results.
**AI-Generated Solution**: The AI-generated solution uses basic grouped bar charts that are static but still provide clear and functional visual representation of the data. While these charts are effective for quick views, they do not offer the same level of interactivity and customization as the original implementation.

### 4.4 Performance

**Original Implementation:** The original method is optimized for handling large datasets, which is essential for professional-grade dashboards and data-intensive applications. This approach ensures that the system can handle large volumes of data efficiently.
**AI-Generated Solution:** The AI-generated solution is more suited to small to medium datasets or quick prototypes. While it offers ease of use, it may not

perform as well with larger datasets, making it less appropriate for applications requiring robust performance at scale.

### 4.5 Strengths and Weaknesses

**Original Implementation**

*Strengths*:

- The original implementation is designed to handle large datasets efficiently, making it well-suited for professional environments where large volumes of data need to be processed without causing performance issues or excessive memory consumption. This is achieved through memory-efficient techniques like lazy evaluation with scan_csv.
- The visualizations in the original approach are highly detailed and interactive, offering rich tooltips that allow users to explore the data more deeply. This makes the visualizations more engaging and user-friendly, as it provides additional context and clarity, enhancing the user experience when interacting with the data.

*Weaknesses*:

- One of the challenges of the original implementation is that it requires a certain level of expertise with the Polars library and Altair for visualization. Users need to be familiar with these tools to fully utilize their capabilities. This may be a barrier for those who are new to data science or programming.
- The approach, while powerful, can be slightly more complex for beginners. The need to manage detailed data transformations and interactive visualizations can make the learning curve steeper for someone who is just starting out with data analysis or visualization tools.

**AI-Generated Solution**

*Strengths*:

- The AI-generated solution is much simpler and more beginner-friendly. It uses straightforward techniques that allow users to quickly generate results without needing deep knowledge of complex libraries like Polars or Altair. This makes it a good starting point for those who are just learning how to work with data analysis and visualization.

- The quick prototyping capabilities of the AI-generated solution are another strength. It allows for fast development and testing of ideas, which is useful in environments where speed is important, such as during early stages of data analysis or when trying to quickly validate hypotheses.

*Weaknesses*:

- One of the limitations of the AI-generated solution is its lack of interactivity. Unlike the original implementation, it does not offer advanced interactive features like tooltips or the ability to explore the data in a more dynamic way. This limits its utility for users who need deeper engagement with the data.
- The AI-generated solution is less efficient for handling large datasets. It is better suited to smaller datasets or scenarios where performance is not as critical. When dealing with large datasets, the AI-generated solution may not perform as well, leading to slower processing times or higher memory usage, making it less ideal for production-level applications.

## 6.Case Study: Sometimes how we create our plot code is better

The following is the custom code i have created to generate a box plot for avocado sales from 2015 to 2018, filtered by regions "TotalUS", "West", and "WestTexNewMexico".

```
filtered_data = avocado_data.filter(
    (pl.col('year').is_between(2015, 2018)) &
    (pl.col('region').is_in(["TotalUS", "West", "WestTexNewMexico"]))
)
# Convert the filtered data to a Pandas DataFrame (since Altair works with Pandas)
df = filtered_data.collect().to_pandas()

# Create the box plot
chart3 = alt.Chart(df).mark_boxplot(ticks=True).encode(
    x=alt.X("region:O", title=None, axis=alt.Axis(labels=False, ticks=False), scale=alt.Scale(padding=1)),
    y=alt.Y("Total Volume:Q"),
    color="region:N",
    column=alt.Column('year:N', header=alt.Header(orient='bottom'))
).properties(
    width=60
```

```
).configure_facet(
    spacing=0
).configure_view(
    stroke=None
)

# Display the plot

chart3.show()
```

### 6.1 Advantages of Writing Your Own Code

1. **Customization and Control**: Writing your own code gives you full control over the logic, filtering, and visual output. In the example above, we manually filter the data to focus on a specific time range (2015-2018) and region, and then convert it to a Pandas Data Frame for compatibility with Altair. This process ensures that the data we use is precisely tailored to our analysis.

   In contrast, if we used AI to generate this code, we might not get the exact results we want. AI tools often work by pattern matching, and might not fully understand the specific context or nuances of the problem, leading to code that requires more post-processing or adjustments.

2. **Efficiency in Debugging and Iteration**: When you write the code yourself, you can quickly troubleshoot errors and adjust the logic. If there's an issue with the data or the visualization, you can directly pinpoint where things went wrong and make specific changes. AI-generated code may not always provide insight into why something failed or offer a clear path for fixing it.
3. **Understanding the Code**: Writing code manually forces you to understand each step of the process. For instance, in the box plot creation code above, you are directly involved in how the data is filtered, how the chart is built, and how the aesthetic elements like colors and axes are customized. This understanding becomes invaluable when you need to explain or modify the analysis later on. With AI-generated code, you might not fully understand what's happening behind the scenes, which can make it harder to adapt the solution to your needs.
4. **Learning and Skill Development**: By writing your own code, you develop a deeper understanding of the libraries you're working with (such as Polars, Pandas, and Altair). This knowledge is essential for becoming proficient in data analysis, and it improves problem-solving skills. Relying solely on AI

for coding can result in missed opportunities for learning and may inhibit growth in your coding abilities.

### 6.2 AI Can Be Helpful, but Input Matters

AI tools like OpenAI's GPT models or Copilot can assist by quickly generating code based on prompts. Here's an example of how AI might generate a similar box plot creation:

```
# Sample dataset with 'year', 'region', and 'Total Volume'
avocado_data = pd.read_csv('avocado_data.csv')

# Filter the dataset based on year and region
filtered_data = avocado_data[(avocado_data['year'] >= 2015) & (avocado_data['year'] <= 2018)]
filtered_data = filtered_data[filtered_data['region'].isin(['TotalUS', 'West', 'WestTexNewMexico'])]

# Create a box plot using Altair
chart = alt.Chart(filtered_data).mark_boxplot().encode(
    x='region:N',
    y='Total Volume:Q',
    color='region:N',
    column='year:N'
)

chart.show()
```

This AI-generated code is concise and works in most cases. However, there are a few important things to consider:

**Input Quality**: The AI-generated code's quality heavily depends on the input you provide. If you ask the AI to generate a box plot without specifying the regions or years, it may produce a generic solution. Clear and detailed input helps the AI understand the context and generate better results. For example, specifying "box plot for avocado data from 2015 to 2018 for regions 'TotalUS', 'West', and 'WestTexNewMexico'" would result in more tailored output.

**Lack of Context**: AI tools often don't have the full context of your analysis goals. While they can generate functional code, they might not anticipate nuances such as handling missing data or filtering based on specific combinations of categories. These might need to be manually added.

**Post-Processing Needs**: The AI may generate code that works well for a general use case, but often lacks fine-tuning for specific requirements (e.g., formatting, axes customization, or advanced visualization options). You may still need to adapt or extend the code, requiring a solid understanding of the underlying libraries.

## 7. Conclusion

In summary, while AI tools can be powerful for quickly generating code, writing your own code offers unmatched flexibility and control. By crafting your own solutions, you learn and gain a deeper understanding of the tools you're using, which is essential for long-term growth in data science. However, the input you give to AI is crucial—it directly influences the quality and relevance of the generated code. Therefore, while AI can save time in some situations, developing your coding skills is invaluable for creating custom, efficient, and accurate data analyses.