This open-source Python program, overall, reads data using the Torque API, handles missing values and outliers, calculates statistics, normalizes scores using two techniques - Min-Max normalization and Z-score normalization, performs data pivoting, calculates overall scores, ranks applications, and saves the results in a new CSV file.

We use the following normalization techniques used in the code:   
• Min-Max Normalization: This method scales the data to a fixed range, usually 0 to 1. It is useful when you know the boundaries of the input variables, which helps scale the output but maintains the original distribution. The drawback of this method is that it does not handle outliers very well.   
• Z-score Normalization: This method scales data based on the mean and standard deviation. This is a useful technique when dealing with features with a normal distribution, and it deals better with outliers compared to Min-Max normalization.

In the context of our script, the scores given by different judges are being normalized. This could be useful if different judges have different scoring scales or tendencies (for example, if some judges are generally harsher or more lenient in their scoring). By normalizing the scores, we can adjust for these differences and make the scores from different judges more directly comparable. This approach seems to be the underlying logic behind Carrot’s normalization technique as well.

**However**, the normalization process in a script that just does this will assume that the scores from each judge follow either a uniform or normal distribution, which may not be the case. If the scores don't follow these distributions, the normalization could distort the data and lead to misleading results. Therefore, a prior analysis of the score distributions is performed **to account for this**, where we compute simple descriptive statistics for the scores of each judge. We calculate the skewness and kurtosis of the scores, which give an idea about the symmetry and tail heaviness of the distribution respectively.

(*Skewness gives an idea of the symmetry of the distribution around the mean. A skewness value close to 0 suggests that the data is fairly symmetric. Negative skewness indicates that the data is skewed to the left, while positive skewness indicates that the data is skewed to the right. Kurtosis measures the "tailedness" of the distribution. A high kurtosis value indicates a distribution with heavy tails, which means there are more outliers. A low kurtosis value indicates a distribution with light tails, meaning fewer outliers.*)

**Another consideration** is the handling of missing data. Neither normalization technique is well-equipped to handle missing data, and there might be missing or inconsistent data. This should also be taken into account when assessing the usefulness of the script's normalization techniques. **To account for this**, we add checks for missing data and outliers as well as conduct a simple analysis of score distributions for each judge. We do so by replacing missing trait scores with the median of that column, and handle outliers by replacing any scores above the upper whisker with the value of the upper whisker.

In summary, to make our normalization techniques useful depending on the nature of the scores' distribution, the presence of outliers, and how missing or inconsistent data are handled is accounted for.

**To summarize**, the script follows the following steps:

1. Import necessary libraries: pandas, numpy, datetime, seaborn, torque, and matplotlib.
2. Read data from Torque API into a Pandas DataFrame, selecting specific columns.
3. Rename the column "Trait Score" to "Raw Trait Score" in the DataFrame.
4. Check for missing data in the DataFrame and print the count of missing values for each column.
5. Fill missing values in the "Raw Trait Score" column with the median of that column.
6. Calculate the upper whisker value and replace any data point in the "Raw Trait Score" column greater than the upper whisker with the upper whisker value.
7. Iterate over unique values in the "Anonymous Judge Name" column and calculate skewness and kurtosis for the "Raw Trait Score" of each judge's scores. Print the skewness and kurtosis values.
8. Normalize the scores using two different methods: min-max normalization and z-score normalization. Add the normalized scores as new columns in the DataFrame.
9. Reorder the columns in the DataFrame, moving the "Raw Trait Score" column before the normalized score columns.
10. Pivot the DataFrame to get scores by trait, creating a new table with mean values of "Raw Trait Score" and normalized scores for each trait.
11. Calculate the overall sum of normalized scores per normalization style and add the sums as new columns in the pivoted table.
12. Rank applications based on the overall score using both normalization styles and add the ranks as new columns in the pivoted table.
13. Reset the index of the pivoted table and store it in the output\_table DataFrame.
14. Generate a timestamp and append it to the output file name.
15. Save the output\_table as a new CSV file using the generated file name.

**Potential Limitations** with this approach:

1. Handling missing data: The code fills missing values in the "Raw Trait Score" column with the median. While this is a common approach, it might not always be the best strategy. Depending on the nature of the data, it could be more appropriate to use other techniques such as interpolation or imputation based on other variables. You could consider exploring different missing data handling techniques and selecting the one that suits your data best.
2. Outlier detection and treatment: The code uses a basic approach to detect and handle outliers by replacing data points above the upper whisker with the upper whisker value. This method assumes that any data point above the upper whisker is an outlier. However, outliers can have different interpretations and impact the analysis differently. It would be beneficial to explore alternative outlier detection methods (e.g., using statistical tests or domain knowledge) and consider different ways to handle outliers (e.g., removing them, transforming the data, or using robust statistical methods).
3. Statistical measures: The code calculates skewness and kurtosis as measures of distribution characteristics. While these measures can provide insights into the data, they may not always capture the full complexity of the distribution given operational nuances of LFC's competitions. It would be useful to consider additional descriptive statistics or visualization techniques to gain a more comprehensive understanding of the data distribution.