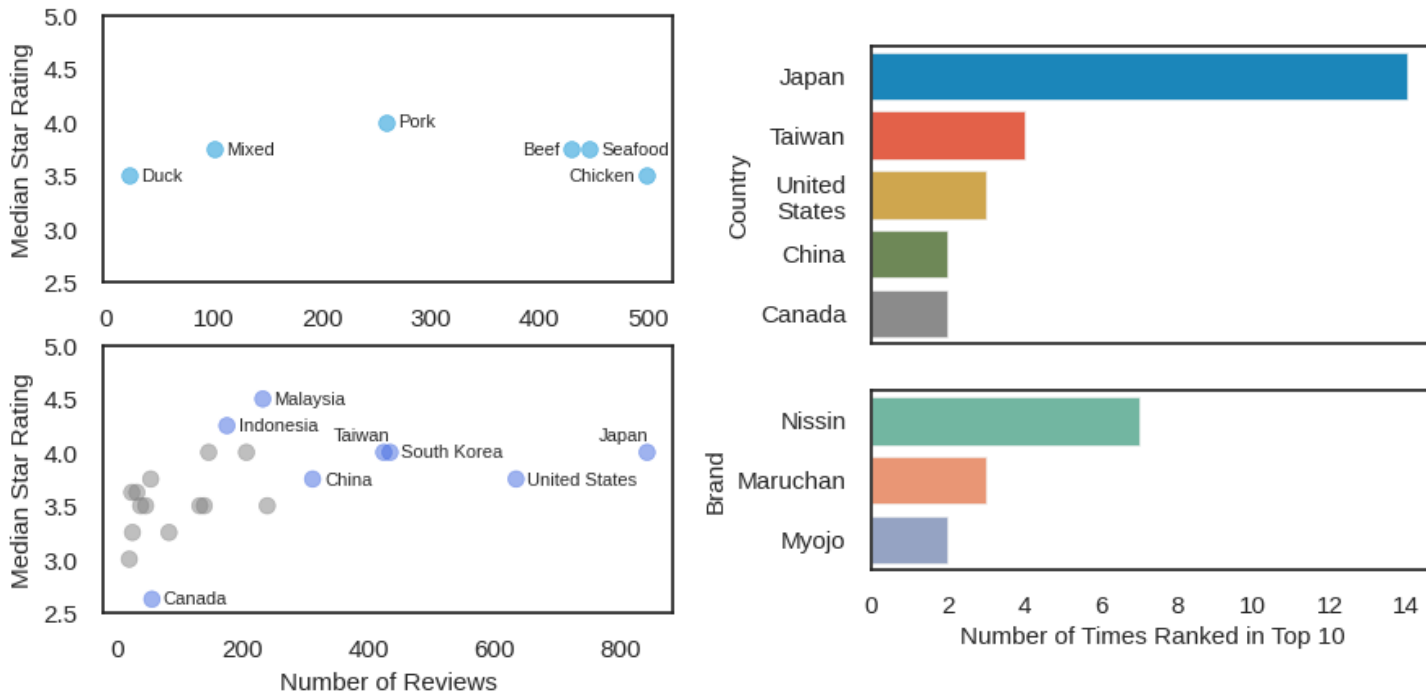


Discovering the Best Ramen Through the Ramen Rater

Since 2002, the Ramen Rater has been the go-to guide for ramen lovers globally, providing over 4,000 reviews of various ramen. This report presents a digestible summary of the best in the ramen world.



Flavour Favourites. Among the meat-based flavours, pork ramen stands out with the highest median star rating of 4 stars whereas chicken leads with the highest number of reviews, indicating its abundance of options in the market. Seafood and beef follow in the number of reviews, while duck—perhaps due to its niche market—sits at the bottom, suggesting a weaker market demand. Despite the market abundance of certain flavours, the Ramen Rater may have a special preference for pork ramen.

Popularity and Rating by Country. Within the global ramen scene, Malaysia leads with the highest median ramen rating of 4.5 stars, followed by Indonesia at 4.25 stars. Among the more frequently reviewed countries, Japan, South Korea, and Taiwan share the spotlight with the highest median star ratings of 4.0, suggesting that the intense market competition could be a contributing factor to the consistently high quality of ramen.

Ramen Rater's Top 10. Of all the countries making it to the Ramen Rater's Top 10 between 2013 and 2016, Japan is the undisputed leader, with brands like Nissin, Maruchan, and Myojo frequently making the cut. Taiwanese and American ramens also receive commendations, though South Korean varieties, despite their abundant reviews and high ratings, are rarely ranked among his Top 10's over the years.

Recommendations

1. For the highest-rated ramen experience, Malaysia's offerings are a must-try.
2. Pork ramen stands as the top choice for those seeking the highest-rated meat-based flavor.
3. Among specific brands, Japan's Nissin emerges as the best option, especially if one is looking for a choice aligned with the Ramen Rater's Top 10.

While the "best" ramen can vary based on personal taste preferences, these findings offer a guide for those seeking the highest-rated experiences according to the Ramen Rater's evaluations.

Data Assumptions and Caveats. We assumed the star ratings and Top 10 lists as metrics of ramen quality, but these metrics can be influenced by the Ramen Rater's subjective taste and personal criteria. Furthermore, the meat flavours were categorized using popular words in ramen names, so niche language-specific terminologies were excluded from the analysis. This might result in an oversimplified view of the flavor landscape and potentially overlook key trends in specific markets or regions.

data_cleaning

November 30, 2023

1 Data cleaning and preprocessing

1. Validate the reviews in **Reviewed** and **The Ramen Rater - Big List Of A**
 - Reviews #428 and #138 are different due to input error, and the version in **Reviewed** seems to be the corrected version.
 - Use **Reviewed** as the reference dataset.
2. Merge top 10 ranks from **The Ramen Rater - Big List Of A** to **Reviewed**
 - Extract the years and ranks from the **Top Ten** column and store them in **T_year** and **T_rank**.
3. Clean up inconsistent country names
 - For the simplicity of the report, assign countries with less than 20 reviews in **Reviewed** to a new category **Other**.
 - Create a custom dictionary to catch spelling and input errors and assign them to the correct country names.
4. Standardize **stars** to the closest 0.25 increment
 - Reassign values like 4.5/5 and 5/4.5 as 4.5.
 - Reassign values like 4-4 and 5-5 as 4 and 5, respectively.
 - Round values that are not divisible by 0.25 to the nearest 0.25 increment.
 - Take the average of values like 2.5/3.5, then round to the nearest 0.25 increment.
 - **Stars** with values like NR are replaced as missing values.
 - **Handling missing values:** There are very few rows with missing values in **Stars** and **Style**, so we can drop these rows without affecting key trends.
5. Clean up the **Variety** column
 - Convert all characters to lowercase and remove special characters
6. Group rare **Style** categories into **Other**
 - Rare categories include **Restaurant**, **Bottle**, **Can**, and **Bar**.
7. Exploratory data analysis with autoviz
8. Create a new **Flavor** column for common meat flavors based on the word cloud generated from the **Variety** column
9. Export **Reviewed** dataset as the working dataset

```
[ ]: from autoviz import AutoViz_Class
import pandas as pd
from openpyxl import load_workbook
import numpy as np
import matplotlib.pyplot as plt
import re
%matplotlib inline
AV = AutoViz_Class()
file_name = "data/4300The-Big-List.xlsx"
```

Imported v0.1.732. After importing autoviz, execute '%matplotlib inline' to display charts inline.

```
AV = AutoViz_Class()
dfte = AV.AutoViz(filename, sep=',', depVar='', dfte=None, header=0,
verbose=1, lowess=False,
                    chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30,
save_plot_dir=None)
```

```
[ ]: # Define a function to get the sheet names from an Excel file
def get_sheetnames_xlsx(filepath):
    wb = load_workbook(filepath, read_only=True, keep_links=False)
    return wb.sheetnames

# Get the sheet names from the specified Excel file
sheet_names = get_sheetnames_xlsx(file_name)
print(sheet_names)
```

```
['Reviewed', 'The Ramen Rater - Big List Of A', 'Re-Reviewed', 'Awaiting
Review']
```

```
[ ]: # Read data from Excel file into a dictionary
data = {}
for sheet in sheet_names:
    data[sheet] = pd.read_excel(
        "data/4300The-Big-List.xlsx", sheet_name=sheet, dtype={"Stars": str}
    )
```

```
[ ]: for df_name, df in data.items():
    print(df_name)
    print(df.info())
    print("\n")
```

```
Reviewed
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4300 entries, 0 to 4299
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
#   :--:  --:  --:  --:
#   0   Column      Non-Null Count  Dtype
#   1   Column      Non-Null Count  Dtype
#   2   Column      Non-Null Count  Dtype
#   3   Column      Non-Null Count  Dtype
#   4   Column      Non-Null Count  Dtype
#   5   Column      Non-Null Count  Dtype
#   6   Column      Non-Null Count  Dtype
```

```

---  -----  -----  -----
0  Review #  4300 non-null  int64
1  Brand     4300 non-null  object
2  Variety   4300 non-null  object
3  Style     4300 non-null  object
4  Country   4300 non-null  object
5  Stars     4299 non-null  object
6  T         0 non-null    float64
dtypes: float64(1), int64(1), object(5)
memory usage: 235.3+ KB
None

```

The Ramen Rater - Big List Of A

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2700 entries, 0 to 2699
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  -----  -
0  Review #    2700 non-null  int64
1  Brand       2700 non-null  object
2  Variety     2700 non-null  object
3  Style       2698 non-null  object
4  Country     2700 non-null  object
5  Stars       2700 non-null  object
6  Top Ten     41 non-null    object
dtypes: int64(1), object(6)
memory usage: 147.8+ KB
None

```

Re-Reviewed

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -----  -
0  Date        80 non-null    object
1  Brand       80 non-null    object
2  Variety     80 non-null    object
3  Style       80 non-null    object
4  Country     80 non-null    object
5  Stars       80 non-null    object
dtypes: object(6)
memory usage: 3.9+ KB
None

```

```

Awaiting Review
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Brand       25 non-null    object
1   Variety     25 non-null    object
2   Style       25 non-null    object
3   Country     25 non-null    object
dtypes: object(4)
memory usage: 928.0+ bytes
None

```

- Found missing values in Reviewed and The Ramen Rater - Big List of A

1.1 Validate reviews in the sheets Reviewed and The Big List

```

[ ]: # Return the reviews that are different
df1 = data["Reviewed"]
df2 = data["The Ramen Rater - Big List Of A"]
diff = []
for idx in df2["Review #"]:
    different = False
    for col in ["Brand", "Variety", "Style", "Country", "Stars"]:
        if (
            df1[df1["Review #"] == idx][col].values[0]
            != df2[df2["Review #"] == idx][col].values[0]
        ):
            different = True
            diff += [idx]
            break
    continue

```

```

[ ]: print("These 2 reviews are different:")
diff

```

These 2 reviews are different:

```

[ ]: [428, 138]

```

```

[ ]: print(df1[df1["Review #"] == 428])
print(df2[df2["Review #"] == 428])

```

	Review #	Brand	Variety	Style	Country	Stars	T
3872	428	Kamfen	E Men Chicken	Pack	China	3.75	NaN

	Review #	Brand	Variety	Style	Country	Stars	Top Ten
2272	428	Kamfen	E Menm Chicken	NaN	China	3.75	NaN

```
[ ]: print(df1[df1["Review #"] == 138])
      print(df2[df2["Review #"] == 138])
```

	Review #	Brand	Variety	Style	Country	Stars	T
4162	138	Unif	100 Furong Shrimp	Pack	Taiwan	3	NaN

	Review #	Brand	Variety	Style	Country	Stars	Top Ten
2562	138	Unif	100 Furong Shrimp	NaN	Taiwan	3	NaN

Reviews #428 and #138 are different due to input error, and the version in Reviewed seems to be the corrected version. Use Reviewed as the reference dataset.

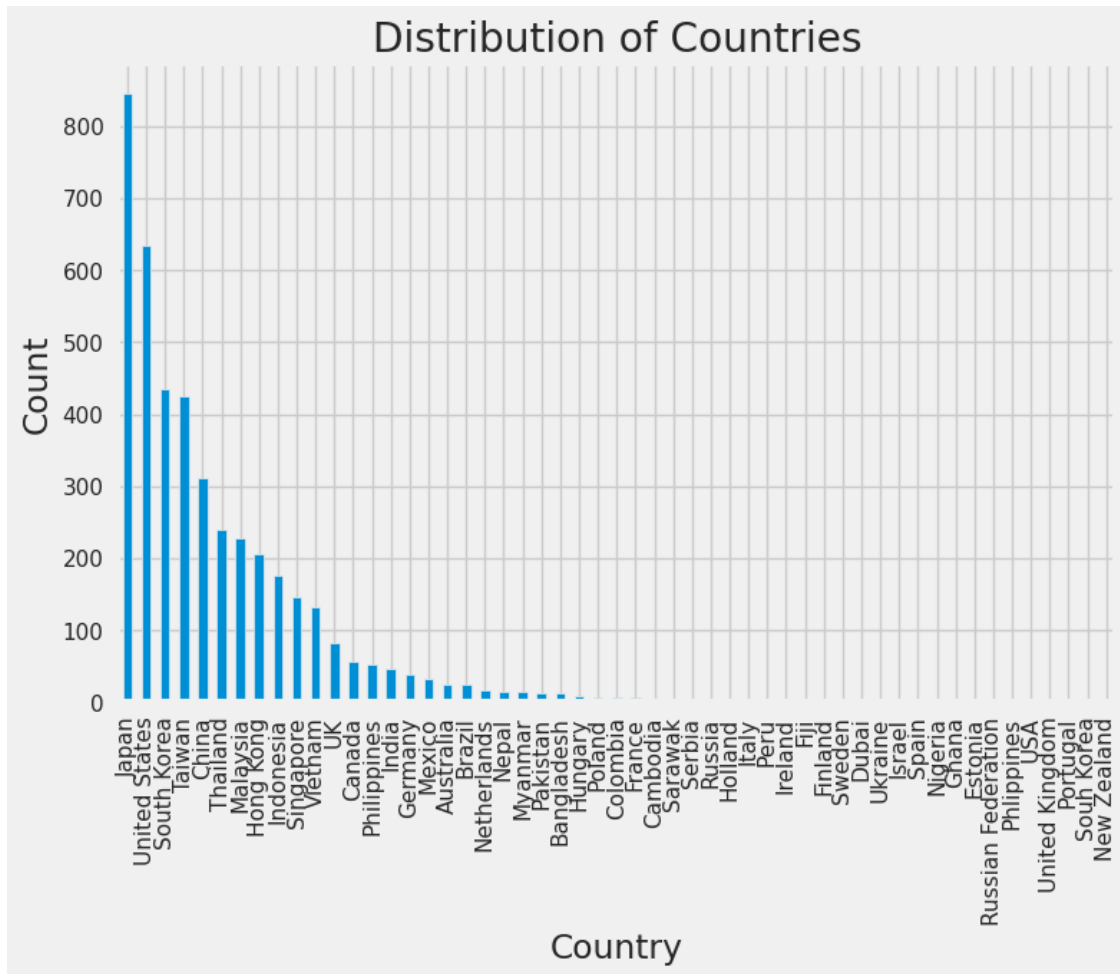
1.2 Merge top 10 ranks from The Ramen Rater - Big List Of A to Reviewed

Extract the years and ranks from the Top Ten column and store them in T_year and T_rank

```
[ ]: data["Reviewed"]["T_year"] = data["The Ramen Rater - Big List Of A"][
      "Top Ten"
    ].str.extract(r"(\d{4})")
data["Reviewed"]["T_rank"] = data["The Ramen Rater - Big List Of A"][
      "Top Ten"
    ].str.extract(r"#(\d{1,2})")
```

1.3 Clean up inconsistent country names

```
[ ]: data["Reviewed"]["Country"].value_counts().plot(kind="bar")
plt.xlabel("Country")
plt.ylabel("Count")
plt.title("Distribution of Countries")
plt.show()
```



```
[ ]: # Create dictionary to catch rare mistakes
exceptions = {
    "Russia": ["Russia", "Russian Federation"],
    "United States": ["United States", "US", "USA"],
    "UAE": ["Dubai"],
    "Netherlands": ["Holland"],
    "Malaysia": ["Sarawak"],
    "South Korea": ["South Korea", "Souh Korea"],
    "United Kingdom": ["UK"],
    "Philippines": ["Philippines", "Phlippines"],
    "United Kingdom": ["UK"],
}
```

```
[ ]: # Define a function to correct exceptions in country names
def correct_exception(input_country: str, exceptions: dict = exceptions):
    for key, val in exceptions.items():
        if input_country in val:
```

```

        return key
    else:
        return input_country

# Apply the correct_exception function to the "Country" column in each sheet
for sheet in sheet_names:
    data[sheet]["Country"] = data[sheet]["Country"].apply(
        lambda x: correct_exception(x)
    )

```

```

[ ]: # Get the countries with counts less than 20
other_countries = pd.DataFrame(
    {
        "Counts": data["Reviewed"]["Country"].value_counts(),
        "Country": data["Reviewed"]["Country"].value_counts().index,
    }
)
other_countries = other_countries[other_countries["Counts"] < 20]["Country"]

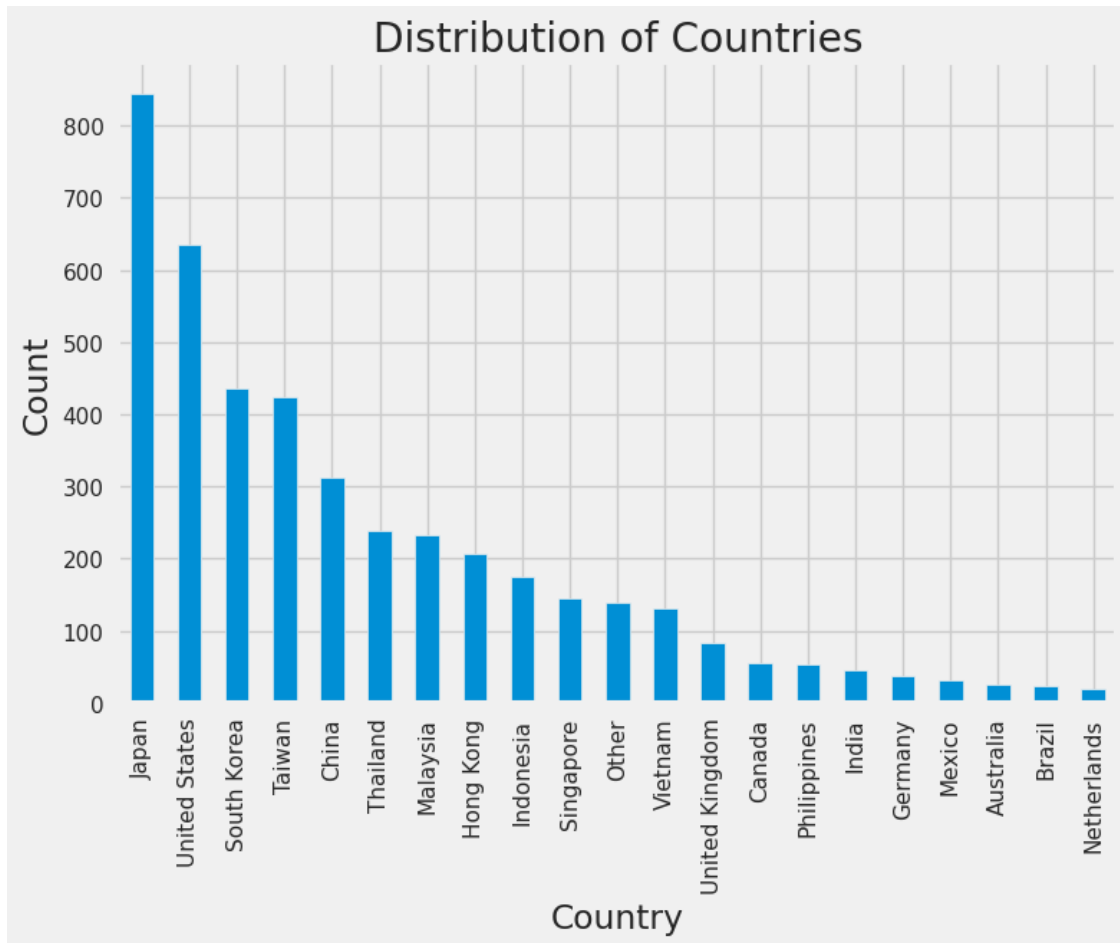
# Update the country values to "Other" for countries with counts less than 20
for sheet in sheet_names:
    data[sheet]["Country"] = data[sheet]["Country"].apply(
        lambda x: "Other" if x in other_countries else x
    )

```

```

[ ]: data["Reviewed"]["Country"].value_counts().plot(kind="bar")
plt.xlabel("Country")
plt.ylabel("Count")
plt.title("Distribution of Countries")
plt.show()

```

1.4 Standardize stars to the closest 0.25 incremental rating

- Reassign values like 4.5/5 and 5/4.5 as 4.5.
- Reassign values like 4-4 and 5-5 as 4 and 5, respectively.
- Round values that are not divisible by 0.25 to the nearest 0.25 increment.
- Take the average of values like 2.5/3.5, then round to the nearest 0.25 increment.

```
[ ]: # Standardize stars to closest 0.25 incremental rating
stars = []
```

```
def standardize_stars(x: str):
    def conditions(x1, x2):
        if x1 == 5 and x2 <= 5:
            return float(x2)
        elif x2 == 5 and x1 <= 5:
            return float(x1)
        elif x1 == x2:
```

```

        return float(x1)
    elif x1 != x2:
        x = (x1 + x2) / 2 # average the 2 values
        x = 0.25 * np.round(float(x) / 0.25, 0) # round to nearest 0.25
        return x
    else:
        return np.nan

try:
    if float(x) % 0.25 != 0: # if not divisible by 0.25
        x = 0.25 * np.round(float(x) / 0.25, 0) # round to nearest 0.25
        return x
    return float(x)
except ValueError:
    if "/" in x:
        x1, x2 = [float(i) for i in x.split("/")]
        return conditions(x1, x2)
    if "-" in x:
        x1, x2 = [float(i) for i in x.split(" ")[0].split("-")][1:3]
        return conditions(x1, x2)
    else:
        return np.nan

for sheet in sheet_names:
    try:
        stars += list(data[sheet]["Stars"].apply(lambda x:
↪standardize_stars(x)))
    except KeyError:
        pass

```

```

[ ]: # Check to see if there are any star ratings that are not divisible by 0.25 or
↪are not np.nan
for i in list(set(stars)):
    if i % 0.25 != 0 and not np.isnan(i):
        print(i)

```

```

[ ]: # Standardize stars in each sheet
for sheet in sheet_names:
    try:
        # Apply the standardize_stars function to the "Stars" column
        data[sheet]["Stars"] = data[sheet]["Stars"].apply(
            lambda x: standardize_stars(x)
        )
        # Convert the "Stars" column to float
        data[sheet].astype({"Stars": "float"}).dtypes
    except KeyError:

```

```
pass
```

Handling missing values: There are very few rows with missing values in Stars and Style, so we can drop these rows without affecting key trends.

```
[ ]: # Drop the row with a missing `Stars` value in `Review`
data["Reviewed"].dropna(subset=["Stars"], inplace=True)

# Drop the 2 rows with missing `Style` values in `The Ramen Rater - Big List Of
↪A`
data["The Ramen Rater - Big List Of A"].dropna(subset=["Style"], inplace=True)
```

1.5 Clean up the Variety column by converting all to lowercase and removing special characters

```
[ ]: # Clean up the `Variety` column by replacing non-alphanumeric characters and
↪converting to lowercase
def replace_non_alphanumeric(string):
    string = re.sub(r"\W+", " ", string)
    return string.lower()

for sheet in sheet_names:
    data[sheet]["Variety"] = data[sheet]["Variety"].apply(
        lambda x: replace_non_alphanumeric(x)
    )
```

2 Group rare Style categories into Other

Rare categories include Restaurant, Bottle, Can, and Bar.

```
[ ]: def group_style(x):
    if x in ["Restaurant", "Bottle", "Can", "Bar", "Box"]:
        return "Other"
    else:
        return x

for sheet in sheet_names:
    data[sheet]["Style"] = data[sheet]["Style"].apply(lambda x: group_style(x))
```

2.1 Exploratory data analysis with autoviz

```
[ ]: dft = AV.AutoViz("", dfte=data["Reviewed"])
```

Shape of your Data Set loaded: (4292, 9)

```
#####
```

```
##### C L A S S I F Y I N G   V A R I A B L E S
```

```
#####
```

```
#####
```

Classifying variables in data set...

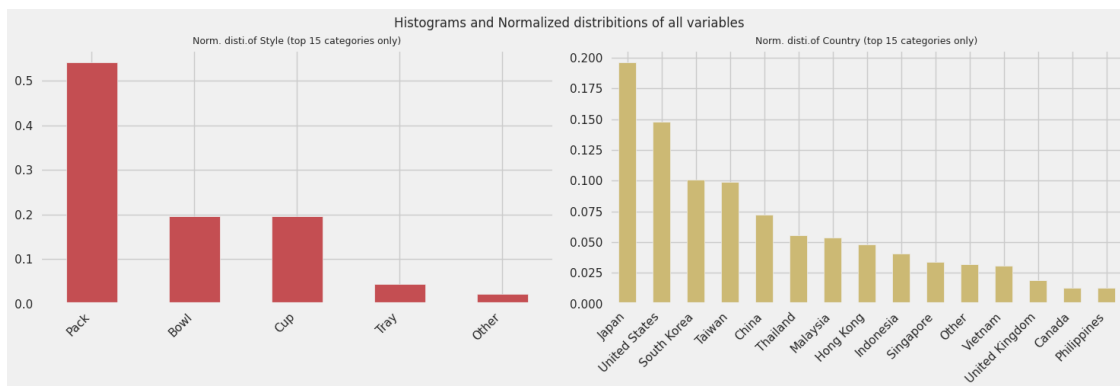
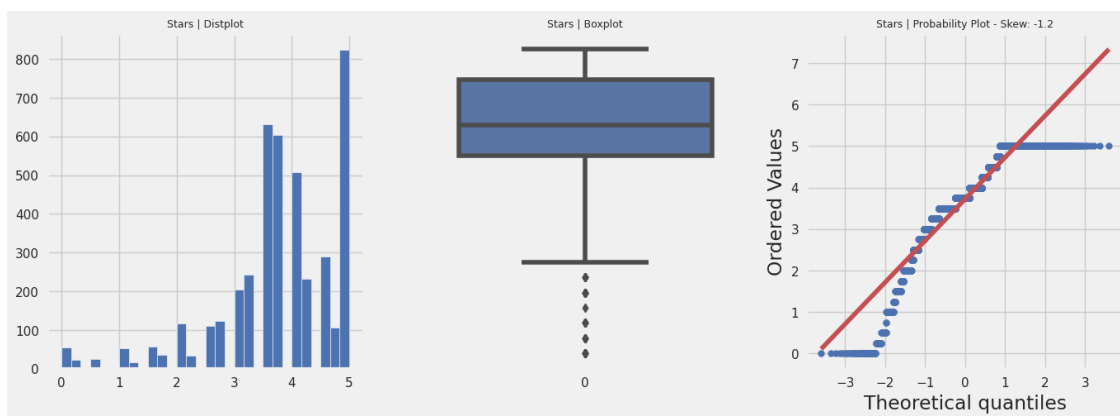
9 Predictors classified...

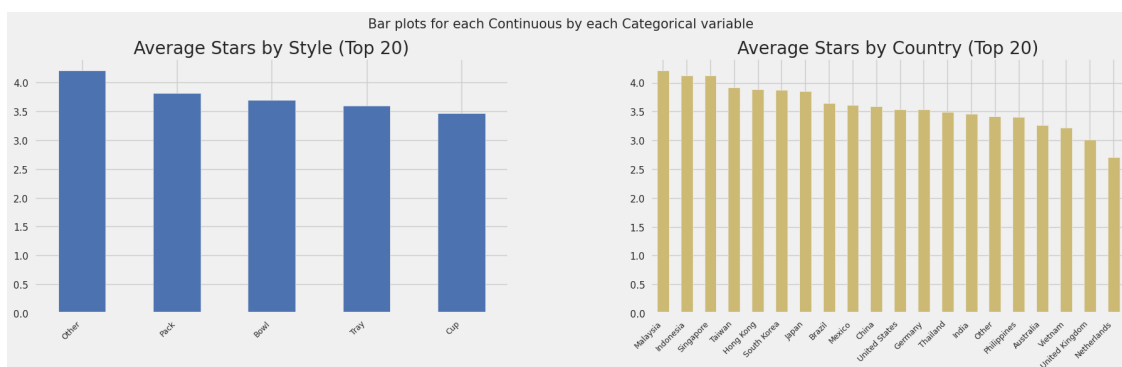
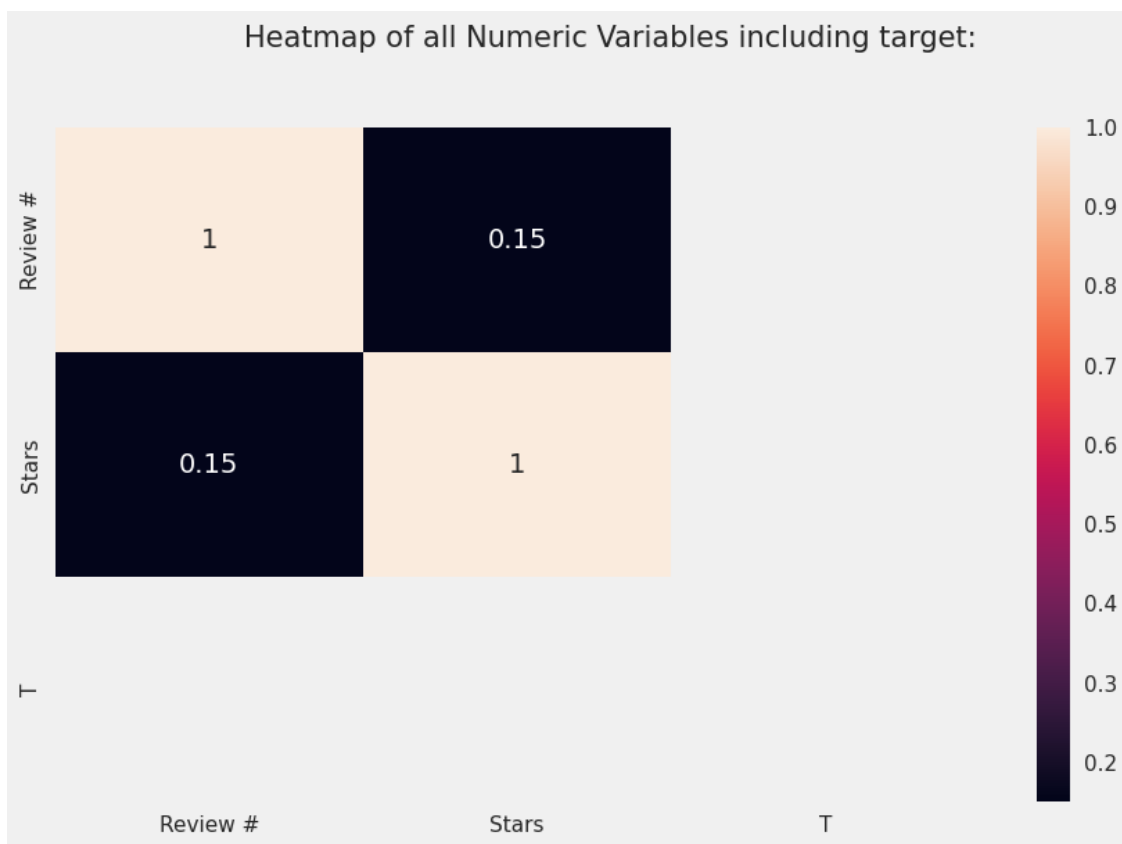
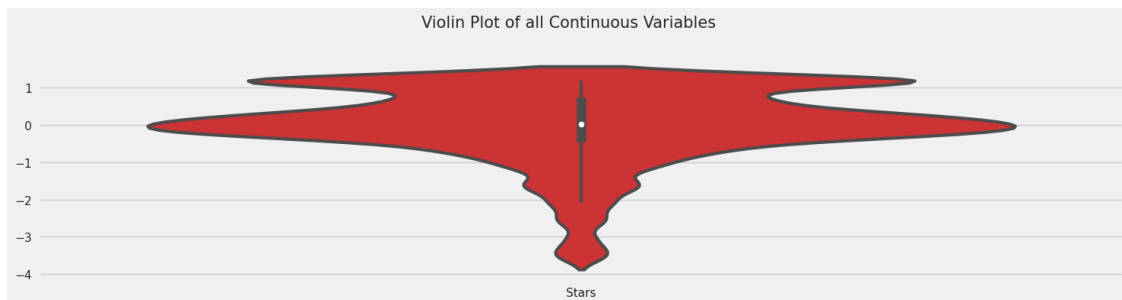
4 variable(s) removed since they were ID or low-information variables

List of variables removed: ['Review #', 'T', 'T_year', 'T_rank']

To fix data quality issues automatically, import FixDQ from autoviz...

<pandas.io.formats.style.Styler at 0x7f245364f1d0>





```

[nltk_data] Downloading collection 'popular'
[nltk_data] |
[nltk_data] | Downloading package cmudict to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package cmudict is already up-to-date!
[nltk_data] | Downloading package gazetteers to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package gazetteers is already up-to-date!
[nltk_data] | Downloading package genesis to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package genesis is already up-to-date!
[nltk_data] | Downloading package gutenburg to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package gutenburg is already up-to-date!
[nltk_data] | Downloading package inaugural to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package inaugural is already up-to-date!
[nltk_data] | Downloading package movie_reviews to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package movie_reviews is already up-to-date!
[nltk_data] | Downloading package names to /home/alice/nltk_data...
[nltk_data] | Package names is already up-to-date!
[nltk_data] | Downloading package shakespeare to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package shakespeare is already up-to-date!
[nltk_data] | Downloading package stopwords to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package stopwords is already up-to-date!
[nltk_data] | Downloading package treebank to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package treebank is already up-to-date!
[nltk_data] | Downloading package twitter_samples to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package twitter_samples is already up-to-date!
[nltk_data] | Downloading package omw to /home/alice/nltk_data...
[nltk_data] | Package omw is already up-to-date!
[nltk_data] | Downloading package omw-1.4 to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package omw-1.4 is already up-to-date!
[nltk_data] | Downloading package wordnet to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package wordnet is already up-to-date!
[nltk_data] | Downloading package wordnet2021 to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package wordnet2021 is already up-to-date!

```

```

[nltk_data] | Downloading package wordnet31 to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package wordnet31 is already up-to-date!
[nltk_data] | Downloading package wordnet_ic to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package wordnet_ic is already up-to-date!
[nltk_data] | Downloading package words to /home/alice/nltk_data...
[nltk_data] | Package words is already up-to-date!
[nltk_data] | Downloading package maxent_ne_chunker to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package maxent_ne_chunker is already up-to-date!
[nltk_data] | Downloading package punkt to /home/alice/nltk_data...
[nltk_data] | Package punkt is already up-to-date!
[nltk_data] | Downloading package snowball_data to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package snowball_data is already up-to-date!
[nltk_data] | Downloading package averaged_perceptron_tagger to
[nltk_data] |   /home/alice/nltk_data...
[nltk_data] | Package averaged_perceptron_tagger is already up-
[nltk_data] |   to-date!
[nltk_data] |
[nltk_data] Done downloading collection popular

```

[illegible]


```

        "seafood",
        "shrimp",
        "fish",
        "crab",
        "lobster",
        "abalone",
        "prawn",
        "pancit",
    ],
    "Duck": ["duck"],
}
val = value.lower()
for k, labels in common_flavours.items():
    for v in labels:
        if v in val:
            output.append(k)
output = list(set(output))
if len(output) > 1:
    return "Mixed"
elif len(output) == 1:
    return output[0]

data["Reviewed"]["Flavor"] = [extract_flavor(x) for x in data["Reviewed"]["Variety"]]

```

2.3 Export Reviewed dataset as the working dataset

```
[ ]: data["Reviewed"].to_csv("data/reviewed.csv", index=False)
```