```python
import pandas as pd
food_trucks = pd.read_csv('food_trucks.csv')
```

```python
food_trucks.head(5)
```

	vendorID	avg_transaction_cost	mnths_operational	days_yr	avg_cost_item	number_trucks	dist_lobland	bev_percent
0	1	12.40	3.0	163	6.83	2	0.12	30.0
1	2	12.10	1.8	140	3.62	5	269.24	40.5
2	3	13.52	6.0	139	3.86	5	8.17	35.7
3	4	15.56	4.0	124	4.05	3	99.09	32.8
4	5	15.08	8.5	186	6.51	5	33.42	35.6

```python
# Drop the 'vendorID' column from the DataFrame
ven= food_trucks.drop('vendorID', axis=1)

print(ven.head)
```

```
<bound method NDFrame.head of      avg_transaction_cost  mnths_operational  days_yr  avg_cost_item  \
0                   12.40                3.0      163           6.83
1                   12.10                1.8      140           3.62
2                   13.52                6.0      139           3.86
3                   15.56                4.0      124           4.05
4                   15.08                8.5      186           6.51
..                    ...                ...      ...            ...
243                 14.70                2.5      153          10.13
244                 14.75                1.1      170           7.88
245                 15.89                0.7      162           3.22
246                 16.17                1.2      153           4.62
247                 14.10                2.8      148           3.26

     number_trucks  dist_lobland  bev_percent
0                2          0.12         30.0
1                5        269.24         40.5
2                5          8.17         35.7
3                3         99.09         32.8
4                5         33.42         35.6
..             ...           ...          ...
243              3        127.16         40.2
244              4         36.29         41.6
245              4         44.35         37.8
246              4         91.98         41.1
247              6         30.15         34.8

[248 rows x 7 columns]>
```

In summary, VendorID cannot be used in clustering models because it is a categorical variable that does not provide meaningful information on the similarity between vendors. Clustering models use numerical values to measure similarity, and categorical variables cannot be used in Euclidean distance calculations that are typically used in clustering algorithms. Therefore, numerical attributes like Sales and Profit should be used to group similar vendors together in a clustering model.

Can be explained with an example ,We can represent vendors as points in a two-dimensional space, with Sales as the x-coordinate and Profit as the y-coordinate. We can use Euclidean distance to calculate the distance between any two vendors in this space. However, including VendorID as a third attribute in our calculations would not provide any meaningful information about the similarity or dissimilarity of vendors, and may even introduce noise or confusion into the model. Therefore, we should focus on numerical attributes like Sales and Profit that can be used to calculate distance and group similar vendors together.

```python
#B (describe function )

food_trucks.describe()
```

	vendorID	avg_transaction_cost	mnths_operational	days_yr	avg_cost_item	number_trucks	dist_lobland	bev_percent
count	248.000000	248.000000	248.000000	248.000000	248.000000	248.000000	248.000000	248.000000
mean	124.500000	14.334274	3.622581	148.822581	5.008468	3.044355	52.435484	40.824194
std	71.735626	1.412888	3.779335	18.942080	1.683380	1.719750	52.002194	5.775277
min	1.000000	8.810000	0.000000	88.000000	-0.270000	0.000000	0.120000	25.400000
25%	62.750000	13.360000	1.175000	136.750000	3.780000	2.000000	16.305000	37.200000
50%	124.500000	14.395000	2.600000	149.000000	5.090000	3.000000	38.315000	40.700000
75%	186.250000	15.242500	4.500000	162.000000	6.105000	4.000000	68.637500	44.200000
max	248.000000	18.320000	27.300000	202.000000	10.130000	9.000000	271.730000	58.600000

The describe() function in Python is a useful tool for any analyst working with a dataset. It provides a summary of basic statistical measures for each column in the dataset, such as the minimum and maximum values, mean, standard deviation, and quartiles. This information can be used to quickly gain insights into the range, spread, and distribution of the data. Additionally, the count provided for each column can help identify missing values or other data quality issues that may need to be addressed before proceeding with further analysis. By using the describe() function, an analyst can quickly gain a sense of the overall structure and characteristics of a dataset, which is valuable when building models or conducting other data analysis tasks.

```python
#c Missing values/impossible values

#missing value -
print(food_trucks.isnull())
```

```
     vendorID  avg_transaction_cost  mnths_operational  days_yr  \
0       False                 False              False    False
1       False                 False              False    False
2       False                 False              False    False
3       False                 False              False    False
4       False                 False              False    False
..        ...                   ...                ...      ...
243     False                 False              False    False
244     False                 False              False    False
245     False                 False              False    False
246     False                 False              False    False
247     False                 False              False    False

     avg_cost_item  number_trucks  dist_lobland  bev_percent
0            False          False         False        False
1            False          False         False        False
2            False          False         False        False
3            False          False         False        False
4            False          False         False        False
..             ...            ...           ...          ...
243          False          False         False        False
244          False          False         False        False
245          False          False         False        False
246          False          False         False        False
247          False          False         False        False

[248 rows x 8 columns]
```

There isnt any missing values in this data . As seen when conducting the function from is.null()

```python
#impossible values
food_trucks2 =food_trucks[food_trucks["avg_cost_item"]>0]

food_trucks2.describe()
```

	vendorID	avg_transaction_cost	mnths_operational	days_yr	avg_cost_item	number_trucks	dist_lobland	bev_percent
count	247.000000	247.000000	247.000000	247.000000	247.000000	247.000000	247.000000	247.000000
mean	124.761134	14.346235	3.612146	148.862348	5.029838	3.048583	52.636154	40.818623
std	71.763073	1.403120	3.783427	18.970165	1.652745	1.721949	52.011478	5.786336
min	1.000000	8.810000	0.000000	88.000000	0.670000	0.000000	0.120000	25.400000
25%	63.500000	13.375000	1.150000	136.500000	3.785000	2.000000	17.065000	37.200000
50%	125.000000	14.400000	2.600000	149.000000	5.090000	3.000000	38.940000	40.700000
75%	186.500000	15.245000	4.500000	162.000000	6.120000	4.000000	68.655000	44.200000
max	248.000000	18.320000	27.300000	202.000000	10.130000	9.000000	271.730000	58.600000

food_trucks2 =food_trucks[food_trucks["avg_cost_item"]>0] and food_trucks2.describe(), we are filtering out any rows in the food_trucks DataFrame where the avg_cost_item column has a value of 0 or less. This is because it is unlikely that a food truck would sell an item for free or at a negative price. By removing these impossible values, we can obtain more accurate and meaningful summary statistics of the avg_cost_item column, such as its mean and standard deviation, which can help us gain insights into the pricing strategy of food trucks.

```python
import pandas as pd
from scipy import stats

# Standardize the numerical variables
numerical_cols = ["avg_transaction_cost", "mnths_operational", "days_yr", "avg_cost_item", "number_trucks", "dist_lobland", "bev_percent"]
ven.loc[:, numerical_cols] = stats.zscore(ven.loc[:, numerical_cols])


foodtrucks_standardized = ven.copy()

# View the standardized dataset
print(foodtrucks_standardized.head())
```

```
   avg_transaction_cost  mnths_operational   days_yr  avg_cost_item  \
0             -1.371790          -0.165066  0.749975       1.084257
1             -1.584551          -0.483224 -0.466708      -0.826478
2             -0.577485           0.630330 -0.519607      -0.683619
3              0.869287           0.100066 -1.313096      -0.570522
4              0.528870           1.293159  1.966658       0.893779

   number_trucks  dist_lobland  bev_percent
0      -0.608499     -1.008059    -1.878019
1       1.139468      4.177573    -0.056248
2       1.139468     -0.852945    -0.889058
3      -0.025844      0.898979    -1.392214
4       1.139468     -0.366406    -0.906408
```

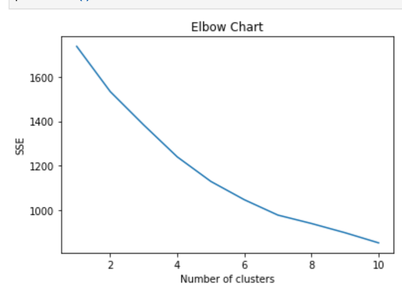
All the data nneds to be standardized because each variables are in different values .In this dataset, the variables have different units and scales. For example, the "avg_transaction_cost" variable ranges from 12.1 to 15.56, while the "days_yr" variable ranges from 124 to 186. Therefore, if the analysis involves comparing or combining these variables, standardization may be necessary to eliminate the effects of the different scales.

```python
#E elbow chart

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=48)
    kmeans.fit(foodtrucks_standardized)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 11), sse)
plt.title('Elbow Chart')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



F-The number of clusters can be decided based on the point on the elbow plot where the rate of decrease in SSE starts to slow down significantly, indicating diminishing returns on adding more clusters. This point is called the "elbow point". Seen from the elbow chart , the elbow point is on 5 where we can see the diminishing returns. Not evidently seen but the slop is regressing more after 5

```python
# Build k-means model with 5 clusters
kmeans = KMeans(n_clusters=5, n_init=10, random_state=48)
kmeans.fit(foodtrucks_standardized)

# Get the cluster labels
cluster_labels = kmeans.labels_
```

```python
# Add the cluster labels to the original dataframe
foodtrucks_standardized["cluster"] = kmeans.labels_

# Generate summary statistics for each cluster
cluster_summary = foodtrucks_standardized.groupby("cluster").mean()

# Display the summary statistics
print(cluster_summary)
```

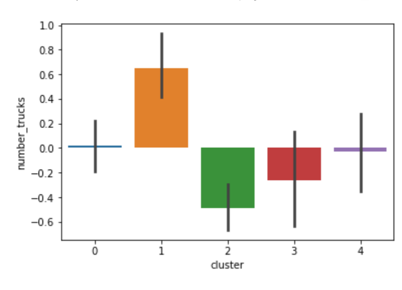
```
         avg_transaction_cost  mnths_operational   days_yr  avg_cost_item  \
cluster
0                   -0.591564          -0.373272 -0.121307       0.724134
1                    0.155783          -0.202880 -0.509201      -0.887368
2                    0.562101          -0.240818  0.356253       0.122342
3                   -0.157766          -0.265093 -0.219043      -0.090810
4                   -0.190838           2.238796  0.710791      -0.062140

         number_trucks  dist_lobland  bev_percent
cluster
0             0.016999     -0.228569    -0.321094
1             0.652330     -0.234560    -0.542054
2            -0.491968     -0.257879     0.824398
3            -0.264203      2.572382     0.150377
4            -0.025844     -0.321853    -0.226537
```

These clusters are groups of food trucks that have similar characteristics based on the variables used in the k-means clustering algorithm. Cluster 0 represents food trucks with high average cost item and low average transaction cost, while Cluster 1 represents food trucks with low average cost item and moderate average transaction cost. Cluster 2 represents food trucks with high bev_percent and moderate days_yr, while Cluster 3 represents food trucks with high dist_lobland and moderate number_trucks. Finally, Cluster 4 represents food trucks with high mnths_operational and moderate days_yr.

```python
sns.barplot(data=foodtrucks_standardized, x="cluster", y="number_trucks")
```

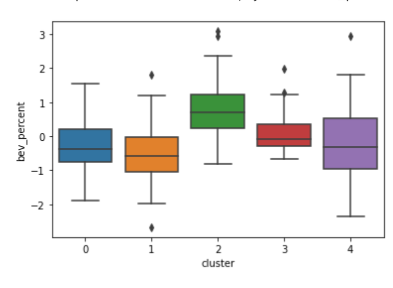
```
<AxesSubplot:xlabel='cluster', ylabel='number_trucks'>
```



This code generates a bar plot that shows the average number of trucks for each cluster. The x-axis represents the cluster labels, and the y-axis represents the average number of trucks in each cluster. The height of each bar indicates the average number of trucks in the corresponding cluster. The plot allows us to compare the number of trucks across different clusters and identify any significant differences. In this case, it appears that Cluster 1 has the highest average number of trucks, while Cluster 2 has the lowest average number of trucks.

```python
sns.boxplot(data=foodtrucks_standardized, x="cluster", y="bev_percent")
```

```
<AxesSubplot:xlabel='cluster', ylabel='bev_percent'>
```



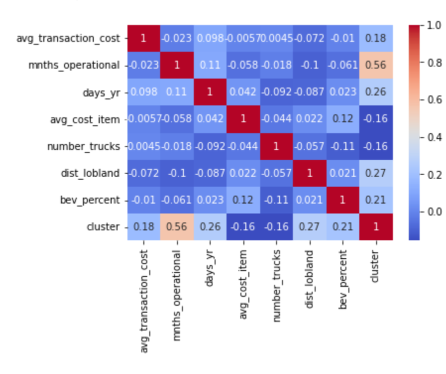
This code creates a boxplot that displays the distribution of beverage sales percentages for each of the five clusters. The x-axis represents the cluster labels, and the y-axis represents the percentage of beverage sales. Each boxplot shows the median (the horizontal line inside the box), the interquartile range (the box), and the range of data (the whiskers). Outliers are also shown as individual points beyond the whiskers. This visualization can help identify any differences or similarities in the beverage sales percentage across the five clusters.

```python
import seaborn as sns

# create correlation matrix of standardized data
corr_matrix = foodtrucks_standardized.corr()

# plot heatmap of correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
```

```
<AxesSubplot:>
```



This code generates a heatmap using seaborn's heatmap function, with the correlation matrix of the standardized food truck data as input. The annot parameter is set to True, which displays the correlation coefficients in each cell of the heatmap, and the cmap parameter is set to "coolwarm", which uses a blue-to-red color map to indicate negative-to-positive correlations.

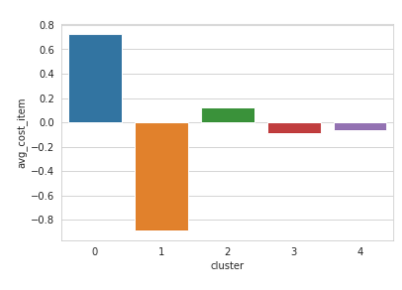
This visualization allows us to see which pairs of variables are highly correlated (either positively or negatively) and which are not. for example,there is a moderate negative correlation between the number of trucks and the average transaction cost. This suggests that as the number of trucks increases, the average transaction cost tends to decrease, which could be due to economies of scale or increased competition in the market.

```python
import seaborn as sns

sns.set_style("whitegrid")

sns.barplot(data=foodtrucks_standardized, x="cluster", y="avg_cost_item", ci=None)
```

```
<AxesSubplot:xlabel='cluster', ylabel='avg_cost_item'>
```



This bar plot shows the differences in average cost per item between the 5 clusters of food trucks. The height of each bar represents the average cost per item for each cluster, and the error bars (not visible in this case because the ci parameter is set to None) would show the confidence interval for each average.

Cluster 1 has the lowest average cost per item, while cluster 0 has the highest. Cluster 3 and 4 have relatively similar average costs, while cluster 2 minimal average cost.

Suppose the mean and standard deviation of the original "average_cost_per_item" variable are $8and2$, respectively. Then, for cluster 0, the standardized average cost per item is 0.72. To obtain the actual average cost per item for cluster 0, you would perform the following calculation:

Actual average cost per item for cluster 0 = (0.72 * $2)+$8 = $9.44

Similarly, for cluster 2, the standardized average cost per item is 1.98. To obtain the actual average cost per item for cluster 2, you would perform the following calculation:

Actual average cost per item for cluster 2 = (0,12 * $2)+$8 = $8.24

This information can be useful for food truck owners and operators to understand where they stand relative to their peers and competitors, and to make decisions about pricing and menu offerings based on this information. For example, if a food truck operator in cluster 1 is finding it difficult to attract customers, they may consider lowering their prices or adjusting their menu to make their offerings more appealing to potential customers. Conversely, a food truck operator in cluster 2 may consider raising their prices or adding more high-end menu items to appeal to customers who are willing to pay more.

J -Question

clusters -0,1,2,3,4

0)High-cost, low-transaction vendors: This cluster has the highest average cost-per-item but the lowest average transaction cost. These vendors may be specialty food trucks that offer high-end, unique products that attract a smaller customer base willing to pay a premium. They may also be located in high-end areas with high rent prices.

1)Low-cost, high-transaction vendors: This cluster has the lowest average cost-per-item but the highest average transaction cost. These vendors may offer more affordable and accessible food options that appeal to a broader customer base, allowing them to generate more sales volume despite their lower prices. They may also be located in high-traffic areas such as busy city centers or popular event venues.

2)Remote vendors: This cluster has the highest distance from the central business district, suggesting that these vendors operate in more remote or less accessible areas. They may cater to customers who work or live in these areas and have limited food options, or they may target specific events or festivals that take place in these areas.

3)New vendors: This cluster has the lowest number of months operational, suggesting that these vendors are relatively new to the food truck business. They may be still in the process of establishing their brand or building a customer base, and may offer more unique or experimental food options to attract customers.

4)Niche vendors- has negative values for all variables except for the number of months operational and days per year. This suggests that these vendors have a lower than average cost per item but also have fewer trucks, serve a smaller geographic area, and may have a lower sales volume. Given the high value for months operational, it is possible that these vendors have a loyal customer base that supports their business. They may also have a unique selling point, such as a specialty food item, that attracts customers despite their smaller size. Overall, this cluster may represent small, niche vendors with a dedicated customer base.

k- question

Lobster Land can benefit from this model in several ways. Firstly, it can help the company to identify different types of food truck vendors operating in the market and gain a better understanding of their strengths and weaknesses. This can be useful in developing strategies to compete with different vendors based on their strengths and weaknesses. For example, if Lobster Land identifies a high-cost, low-transaction vendor operating in the same area, it could target customers who are willing to pay a premium for high-end seafood offerings.

Secondly, this model can help Lobster Land to identify potential areas for expansion. For instance, if it identifies a cluster of remote vendors in an area where there is a lack of seafood options, it could explore the feasibility of setting up a food truck in that area to cater to customers who are not currently being served.

Finally, this model can also help Lobster Land to identify potential collaborations with other food truck vendors. For example, if it identifies a cluster of niche vendors with a loyal customer base, Lobster Land could potentially collaborate with them to offer unique seafood offerings that appeal to a broader customer base. Overall, by utilizing this model, Lobster Land can gain valuable insights into the food truck industry and develop strategies that can help it to grow and compete more effectively in the market.

```python
woodie_roller = pd.read_csv('woodie.csv')
```

```python
woodie_roller.head(5)
```

	bundleID	start_high	maxspeed	steepest_angle	seats_car	drop	track_color	avg_rating
0	1	Yes	40	50	2	100	red	7.613468
1	2	Yes	40	50	2	100	blue	5.266737
2	3	Yes	40	50	2	100	green	4.871951
3	4	Yes	40	50	2	100	white	4.453202
4	5	Yes	40	50	2	200	red	5.476815

B

which elements are numerical and categorical -

bundleID: this variable represents a series of sequential integers from 1 to 288 and can be treated as a numeric identifier.

maxspeed: this variable represents the maximum speed reached by the roller coaster during the ride, and has numeric values of 40, 60, or 80 mph.

steepest_angle: this variable represents the number of degrees associated with the steepest drop on the ride and has numeric values of 50 or 75 degrees.

seats_car: this variable represents the number of seats in each car of the roller coaster, and can be treated as a numeric variable with values of 2 or 4.

drop: this variable represents the size of the largest vertical drop during the ride, and has numeric values of 100, 200, or 300 feet.

avg_rating: this variable represents the average rating that the bundle received on a score from 0 to 10 and can be treated as a numeric variable.

The following variable is categorical:

start_high: this variable has two categorical options, "Yes" or "No", representing whether the roller coaster starts at a high altitude or not.

track_color: this variable has four categorical options, "green", "blue", "white", and "red", representing the color of the roller coaster track.

```python
#C
# drop the bundleID variable
woodie_roller.drop('bundleID', axis=1, inplace=True)
```

```python
woodie_roller['start_high'] = woodie_roller['start_high'].replace({'Yes': 1, 'No': 0})
```

```python
# create dummy variables for all remaining categorical variables
woodie_roller = pd.get_dummies(woodie_roller, drop_first=True)
```

```python
woodie_roller.head()
```

	start_high	maxspeed	steepest_angle	seats_car	drop	avg_rating	track_color_green	track_color_red	track_color_white
0	1	40	50	2	100	7.613468	0	1	0
1	1	40	50	2	100	5.266737	0	0	0
2	1	40	50	2	100	4.871951	1	0	0
3	1	40	50	2	100	4.453202	0	0	1
4	1	40	50	2	200	5.476815	0	1	0

C -A

Dummifying numeric variables is a way to convert them into categorical variables, which can be more useful for certain types of analysis, such as linear regression. In the case of linear regression, numerical variables can create issues related to multicollinearity, where two or more variables are highly correlated with each other, leading to unstable and unreliable estimates of the model coefficients. By converting the numeric variables into categories, we can avoid this problem and create a more robust and interpretable model. Additionally, dummifying numeric variables can also capture any nonlinear relationships that may exist between the variable and the outcome, which may not be captured by a linear model that treats the variable as continuous.

```python
from sklearn.linear_model import LinearRegression
```

```python
#D
# define input and output variables
X = woodie_roller.drop('avg_rating', axis=1)
y = woodie_roller['avg_rating']

# create linear regression model and fit it to the data
model = LinearRegression()
model.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```python
#E
print("Model Coefficients:")
for i, col in enumerate(X.columns):
    print(f"{col}: {model.coef_[i]}")
```

```
Model Coefficients:
start_high: 1.0566803271965342
maxspeed: 0.037355766464625
steepest_angle: -0.02074300623303966
seats_car: -0.22107219765519887
drop: 0.005996595805234815
track_color_green: -0.06556164272106117
track_color_red: 1.791007549871522
track_color_white: -0.23571177239927163
```

Based on the linear model we have developed, we can see that the most significant predictor of a roller coaster's average rating is the maximum speed it reaches during the ride, as indicated by the positive coefficient of 0.037. Specifically, for every 1 mph increase in maximum speed, we can expect to see an increase in the average rating by approximately 0.037 points, holding all other variables constant. This indicates that Lobster Land may want to consider investing in roller coasters with higher maximum speeds to improve the overall experience for their guests.

Additionally, we can see that the steepest angle of the roller coaster's largest drop is also a significant predictor of the average rating, as indicated by the negative coefficient of -0.0207. For every 1 degree increase in the steepest angle, we can expect to see a decrease in the average rating by approximately 0.0207 points, holding all other variables constant. This suggests that Lobster Land may want to prioritize designing roller coasters with slightly less steep angles to improve the overall guest satisfaction.

Interestingly, the color of the roller coaster's track did not appear to have a significant impact on the average rating, as indicated by the relatively small and insignificant coefficients for the different track colors. This may indicate that guests are more focused on the actual ride experience rather than the visual aesthetics of the roller coaster. However, it is worth noting that this data is based on a limited set of colors, and additional research could be conducted to investigate the impact of a wider range of colors on the guest experience.

Finally, we can see that the other variables in our model, such as starting height, seat configuration, and drop size, did not have a significant impact on the average rating, as indicated by their relatively small and insignificant coefficients. While these variables may still be important considerations in the design and construction of roller coasters, they may not be as critical in determining the overall guest experience and satisfaction.

Overall, our model provides valuable insights into the factors that drive guest satisfaction with roller coasters at Lobster Land. By focusing on factors such as maximum speed and steepest angle, Lobster Land can make strategic investments in the design and construction of new roller coasters to enhance the overall guest experience and drive higher ratings and increased revenue.

Part III: Wildcard: Marketing & Segments (1 point)

The ad **"we will always need human connection. download bumble"** is targeting consumers who are seeking romantic relationships or connections with others. This is evident from the tagline, which highlights the need for human connection, and the call-to-action to download Bumble, which is a popular dating app. The segment being targeted likely includes young adults and millennials who are comfortable with using technology to meet new people and form connections.

I believe this ad is an undifferentiated (mass market) ad because it appeals to a broad range of consumers who may be looking for romantic connections, regardless of their specific demographics or interests. However, the placement of the ad on social media platforms like Instagram and Facebook may indicate that it is specifically targeting users of these platforms.

Anyone who is willling to enter romantic relationships are part of the segment and especially people from 18-30 would be the major target market compared to other ages. Being in that bracket and looking for one , I am part of it.

In terms of effectiveness, the ad does a good job of highlighting the need for human connection and positioning Bumble as a solution to this need. The tagline is relatable and taps into a basic human desire for connection, while the call-to-action is clear and straightforward. Overall, I think this ad is likely to be effective in attracting users to download the Bumble app and potentially form new connections.

```python
!pip install nbconvert
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nbconvert in /usr/local/lib/python3.8/dist-packages (6.5.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (3.1.2)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (1.5.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (5.7.3)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (0.7.2)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-packages (from nbconvert) (0.7.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-packages (from nbconvert) (6.0.0)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (4.6.3)
Requirement already satisfied: lxml in /usr/local/lib/python3.8/dist-packages (from nbconvert) (4.9.2)
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (2.6.1)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (5.7.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.8/dist-packages (from nbconvert) (0.2.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from nbconvert) (23.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (2.1.2)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.8/dist-packages (from nbconvert) (1.2.1)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.8/dist-packages (from jupyter-core>=4.7->nbconvert) (3.0.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.8/dist-packages (from nbclient>=0.5.0->nbconvert) (6.1.12)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.8/dist-packages (from nbformat>=5.1->nbconvert) (2.16.3)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.8/dist-packages (from nbformat>=5.1->nbconvert) (4.3.3)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.8/dist-packages (from bleach->nbconvert) (1.15.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (22.2.0)
Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (5.12.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.19.3)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.8/dist-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.2)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.8/dist-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.8/dist-packages (from importlib-resources>=1.4.0->jsonschema>=2.6->nbformat>=5.1->nbconvert) (3.15.0)
```

```python
!jupyter nbconvert -- to html assignment2_aravindrao.ipynb
```

```
[NbConvertApp] WARNING | pattern 'to' matched no files
[NbConvertApp] WARNING | pattern 'html' matched no files
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.8/dist-packages/jupyter_core/application.py", line 277, in launch_instance
    return super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.8/dist-packages/nbconvert/nbconvertapp.py", line 423, in start
    self.convert_notebooks()
  File "/usr/local/lib/python3.8/dist-packages/nbconvert/nbconvertapp.py", line 585, in convert_notebooks
    raise ValueError(
ValueError: Please specify an output format with '--to <format>'.
The following formats are available: ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
```

```python
jupyter nbconvert --to html notebook.ipynb
```

```
[NbConvertApp] WARNING | pattern 'to' matched no files
[NbConvertApp] WARNING | pattern 'html' matched no files
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.8/dist-packages/jupyter_core/application.py", line 277, in launch_instance
    return super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.8/dist-packages/nbconvert/nbconvertapp.py", line 423, in start
    self.convert_notebooks()
  File "/usr/local/lib/python3.8/dist-packages/nbconvert/nbconvertapp.py", line 585, in convert_notebooks
    raise ValueError(
ValueError: Please specify an output format with '--to <format>'.
The following formats are available: ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
```