Introduction

This project employs an unsupervised machine learning algorithm designed to cluster Airbnb rental properties in Chicago. The analysis leverages property-related features, encompassing continuous, categorical, and free-text data. Prior to training the final model, an Agglomerative clustering algorithm, additional NLP features were engineered from the raw text descriptions.

The data, freely available from Inside Airbnb (https://insideairbnb.com/get-the-data/), was last updated on March 11, 2025. This dataset includes 8,748 properties (rows) and 79 features (columns), with detailed feature descriptions provided in a data dictionary at: https://docs.google.com/spreadsheets/d/1iWCNJcSutYqpULSQHlNyGInUvHg2BoUGoNRIGa
usp=sharing

Inputs

In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from geopy.distance import geodesic
import re

import nltk

from Levenshtein import distance as levenshtein_distance
from rake_nltk import Rake
from collections import Counter
import ast

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski
from sklearn.decomposition import PCA
```

In [428…
```python
df = pd.read_csv('listings.csv')
```

EDA

Raw Data Exploration

In [429…
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8748 entries, 0 to 8747
Data columns (total 79 columns):
 #    Column                                       Non-Null Count  Dtype  
---   ------                                       --------------  -----  
 0    id                                           8748 non-null   int64  
 1    listing_url                                  8748 non-null   object 
 2    scrape_id                                    8748 non-null   int64  
 3    last_scraped                                 8748 non-null   object 
 4    source                                       8748 non-null   object 
 5    name                                         8748 non-null   object 
 6    description                                  8585 non-null   object 
 7    neighborhood_overview                        5099 non-null   object 
 8    picture_url                                  8748 non-null   object 
 9    host_id                                      8748 non-null   int64  
 10   host_url                                     8748 non-null   object 
 11   host_name                                    8748 non-null   object 
 12   host_since                                   8748 non-null   object 
 13   host_location                                7049 non-null   object 
 14   host_about                                   5723 non-null   object 
 15   host_response_time                           8032 non-null   object 
 16   host_response_rate                           8032 non-null   object 
 17   host_acceptance_rate                         8300 non-null   object 
 18   host_is_superhost                            8550 non-null   object 
 19   host_thumbnail_url                           8748 non-null   object 
 20   host_picture_url                             8748 non-null   object 
 21   host_neighbourhood                           8533 non-null   object 
 22   host_listings_count                          8748 non-null   int64  
 23   host_total_listings_count                    8748 non-null   int64  
 24   host_verifications                           8748 non-null   object 
 25   host_has_profile_pic                         8748 non-null   object 
 26   host_identity_verified                       8748 non-null   object 
 27   neighbourhood                                5100 non-null   object 
 28   neighbourhood_cleansed                       8748 non-null   object 
 29   neighbourhood_group_cleansed                 0 non-null      float64
 30   latitude                                     8748 non-null   float64
 31   longitude                                    8748 non-null   float64
 32   property_type                                8748 non-null   object 
 33   room_type                                    8748 non-null   object 
 34   accommodates                                 8748 non-null   int64  
 35   bathrooms                                    7756 non-null   float64
 36   bathrooms_text                               8735 non-null   object 
 37   bedrooms                                     8575 non-null   float64
 38   beds                                         7725 non-null   float64
 39   amenities                                    8748 non-null   object 
 40   price                                        7718 non-null   object 
```

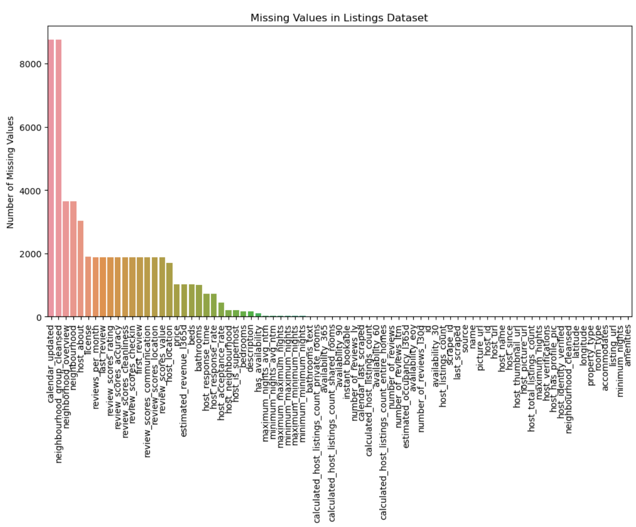
```
 41  minimum_nights                                8748 non-null   int64
 42  maximum_nights                                8748 non-null   int64
 43  minimum_minimum_nights                        8731 non-null   float64
 44  maximum_minimum_nights                        8731 non-null   float64
 45  minimum_maximum_nights                        8731 non-null   float64
 46  maximum_maximum_nights                        8731 non-null   float64
 47  minimum_nights_avg_ntm                        8731 non-null   float64
 48  maximum_nights_avg_ntm                        8731 non-null   float64
 49  calendar_updated                              0 non-null      float64
 50  has_availability                              8649 non-null   object
 51  availability_30                               8748 non-null   int64
 52  availability_60                               8748 non-null   int64
 53  availability_90                               8748 non-null   int64
 54  availability_365                              8748 non-null   int64
 55  calendar_last_scraped                         8748 non-null   object
 56  number_of_reviews                             8748 non-null   int64
 57  number_of_reviews_ltm                         8748 non-null   int64
 58  number_of_reviews_l30d                        8748 non-null   int64
 59  availability_eoy                              8748 non-null   int64
 60  number_of_reviews_ly                          8748 non-null   int64
 61  estimated_occupancy_l365d                     8748 non-null   int64
 62  estimated_revenue_l365d                       7718 non-null   float64
 63  first_review                                  6870 non-null   object
 64  last_review                                   6870 non-null   object
 65  review_scores_rating                          6870 non-null   float64
 66  review_scores_accuracy                        6870 non-null   float64
 67  review_scores_cleanliness                     6870 non-null   float64
 68  review_scores_checkin                         6870 non-null   float64
 69  review_scores_communication                   6870 non-null   float64
 70  review_scores_location                        6870 non-null   float64
 71  review_scores_value                           6870 non-null   float64
 72  license                                       6845 non-null   object
 73  instant_bookable                              8748 non-null   object
 74  calculated_host_listings_count                8748 non-null   int64
 75  calculated_host_listings_count_entire_homes   8748 non-null   int64
 76  calculated_host_listings_count_private_rooms  8748 non-null   int64
 77  calculated_host_listings_count_shared_rooms   8748 non-null   int64
 78  reviews_per_month                             6870 non-null   float64
dtypes: float64(22), int64(22), object(35)
memory usage: 5.3+ MB
```

In [430…

```python
plt.figure(figsize=(12, 6))
plt.title('Missing Values in Listings Dataset')
sns.barplot(df.isnull().sum().sort_values(ascending= False).reset_index(), x
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Number of Missing Values')
```

Out[430]:

```
Text(0, 0.5, 'Number of Missing Values')
```



In [431…

```python
(df.isnull().sum()>0).sum()/len(df.columns)
```

Out[431]:

```
0.45569620253164556
```

About 45% of the columns have at least 1 missing value. The data dictionary Google spreadsheet was downloaded as csv (named as listings_dictionary.csv) and cleaned for ease of reading. This data dictionary will be used to confirm data type and check if columns are required.

In [432…

```python
data_dict = pd.read_csv('listings_dictionary.csv', skiprows = 7).fillna('-')
```

In [433…

```python
data_dict.head()
```

Out[433]:

	Field	Type	Calculated	Description	Reference
0	id	integer	-	Airbnb's unique identifier for the listing	-
1	listing_url	text	y	-	-
2	scrape_id	bigint	y	Inside Airbnb "Scrape" this was part of	-
3	last_scraped	datetime	y	UTC. The date and time this listing was "scrap...	-
4	source	text	-	One of "neighbourhood search" or "previous scr...	-

In [434…

```
info_df = pd.concat([df.dtypes,df.isnull().sum()], axis = 1)
info_df = info_df.reset_index().rename({'index':'Field',0:'df dtype',1:'null
info_df['null_perc'] = info_df['null_perc']/len(df)
info_df = info_df.merge(data_dict, on = 'Field', how = 'inner')
```

In [435…

```
info_df.head()
```

Out[435]:

	Field	df dtype	null_perc	Type	Calculated	Description	Reference
0	id	int64	0.0	integer	-	Airbnb's unique identifier for the listing	-
1	listing_url	object	0.0	text	y	-	-
2	scrape_id	int64	0.0	bigint	y	Inside Airbnb "Scrape" this was part of	-
3	last_scraped	object	0.0	datetime	y	UTC. The date and time this listing was "scrap...	-
4	source	object	0.0	text	-	One of "neighbourhood search" or "previous scr...	-

Some columns need data type munging.

In [436…

```
info_df[info_df['null_perc']>0].sort_values('null_perc', ascending = False)
```

Out[436]:

	Field	df dtype	null_perc	Type	Calculated	Des
50	calendar_updated	float64	1.000000	date	-	
30	neighbourhood_group_cleansed	float64	1.000000	text	y	The neighbourhoo as geocoded usir
8	neighborhood_overview	object	0.417124	text	-	Host's descriptic neighb
28	neighbourhood	object	0.417010	text	-	
15	host_about	object	0.345793	text	-	Description a
69	license	object	0.217535	text	-	licence/permit/reg
75	reviews_per_month	float64	0.214678	numeric	y	The average nu reviews per montl
68	review_scores_value	float64	0.214678	-	-	
60	first_review	object	0.214678	date	y	The da first/oldes
63	review_scores_accuracy	float64	0.214678	-	-	
64	review_scores_cleanliness	float64	0.214678	-	-	
65	review_scores_checkin	float64	0.214678	-	-	
66	review_scores_communication	float64	0.214678	-	-	
67	review_scores_location	float64	0.214678	-	-	
61	last_review	object	0.214678	date	y	The da last/newes
62	review_scores_rating	float64	0.214678	-	-	
14	host_location	object	0.194216	text	-	The host's self r
41	price	object	0.117741	currency	-	daily price currency.\nNOT
39	beds	float64	0.116941	integer	-	The number c
36	bathrooms	float64	0.113397	numeric	-	The number of ba in th
17	host_response_rate	object	0.081847	-	-	
16	host_response_time	object	0.081847	-	-	
18	host_acceptance_rate	object	0.051212	-	-	That rate at whic accepts bookin
22	host_neighbourhood	object	0.024577	text	-	
19	host_is_superhost	object	0.022634	boolean [t=true; f=false]	-	
38	bedrooms	float64	0.019776	integer	-	The number of be
7	description	object	0.018633	text	-	Detailed descr th
51	has_availability	object	0.011317	boolean	-	[t=true;
49	maximum_nights_avg_ntm	float64	0.001943	numeric	y	the maximum_nig

						from the
48	minimum_nights_avg_ntm	float64	0.001943	numeric	y	the minimum_nig from the
47	maximum_maximum_nights	float64	0.001943	integer	y	the maximum_nig from the
46	minimum_maximum_nights	float64	0.001943	integer	y	the maximum_nig from th
44	minimum_minimum_nights	float64	0.001943	integer	y	the minimum_nig from th
45	maximum_minimum_nights	float64	0.001943	integer	y	the minimum_nig from the
37	bathrooms_text	object	0.001486	string	-	The number of ba in the listing.

Calendar updated and neighborhood group cleaned are both columns with all null values. Other columns with over 20% missing values are concerning and should be investigated further.

Data Preprocessing

Dropping Columns

Columns that do not contribute any information to our project scope are dropped prior to any further data cleaning:

- scrape_id, last_scraped, source, listing_url, calendar_updated, calendar_last_scraped
  - stationary values as the data does not go back further than one scrape
- name, picture_url, host_url, host_thumbnail_url, host_picture_url, host_id, host_location, host_since, host_about, host_response_time, `host_response_rate, host_acceptance_rate, host_is_superhost, host_total_listings_count, host_verifications,host_name host_has_profile_pic, host_identity_verified,calculated_host_listings_count, calculated_host_listings_count_entire_homes, calculated_host_listings_count_private_rooms, calculated_host_listings_count_shared_rooms, first_review, last_review`
  - irrelevant to the project
- host_neighborhood, neighborhood, neighbourhood_group_cleansed, host_listings_count, reviews_per_month, property_type, neighbourhood_cleansed, neighborhood_overview
  - other related or calculated column that provides pertinent information is included
  - Latitude and Longitude will be used to provide distance feature
- min_min, min_max, max_min, max*max, availability*, has_availability, availability_eoy, estimated_occupancy_l365d, estimated_revenue_l365d
  - looks into the future, and therefore, dropped to minimize data leakage

In [437…

```
drop_col = ['listing_url', 'scrape_id', 'last_scraped', 'source','calendar_l
             'name','picture_url','host_url','host_thumbnail_url', 'host_pi
             'host_response_time','host_response_rate', 'host_acceptance_ra
             'host_total_listings_count', 'host_verifications',
             'host_has_profile_pic', 'host_identity_verified',
             'host_neighbourhood', 'host_listings_count','neighbourhood','h
              'neighbourhood_group_cleansed','neighbourhood_cleansed','revi
             'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
             'maximum_minimum_nights', 'minimum_maximum_nights',
             'maximum_maximum_nights', 'minimum_nights_avg_ntm',
             'maximum_nights_avg_ntm','availability_30', 'availability_60',
             'availability_365','has_availability','availability_eoy','esti
             'calculated_host_listings_count',
             'calculated_host_listings_count_entire_homes',
             'calculated_host_listings_count_private_rooms',
             'calculated_host_listings_count_shared_rooms']
```

In [438…
```
df = df.drop(drop_col, axis = 1)
info_df = info_df[~info_df['Field'].isin(drop_col)]
```

In [439…
```
info_df[info_df['null_perc']>0].sort_values('null_perc', ascending = False)
```

Out[439]:

	Field	df dtype	null_perc	Type	Calculated	Desci
69	license	object	0.217535	text	-	licence/permit/regis n
62	review_scores_rating	float64	0.214678	-	-	
63	review_scores_accuracy	float64	0.214678	-	-	
64	review_scores_cleanliness	float64	0.214678	-	-	
65	review_scores_checkin	float64	0.214678	-	-	
66	review_scores_communication	float64	0.214678	-	-	
67	review_scores_location	float64	0.214678	-	-	
68	review_scores_value	float64	0.214678	-	-	
41	price	object	0.117741	currency	-	daily price i currency.\nNOTE
39	beds	float64	0.116941	integer	-	The number of
36	bathrooms	float64	0.113397	numeric	-	The number of bath in the
38	bedrooms	float64	0.019776	integer	-	The number of bec
7	description	object	0.018633	text	-	Detailed descrip the
37	bathrooms_text	object	0.001486	string	-	The number of bath in the listing. \r

Dropping Rows

It appears that some properties are new and are missing review information (about 21.5%). These properties will be removed prior to further analysis.

Additionally, properties that do not have a description will also be removed. It is a key feature we will be using NLP on, and there are less than 2% of properties without a description.

In [440…
```
df = df[(df['review_scores_rating'].notnull())&(df['description'].notnull())
```

Data Type Munging

In [441…
```
df['price'] = df['price'].str.replace(r'[$,]', '', regex = True).astype(floa
```

Cleaning bathrooms_text

In [442…
```
df['bathrooms_text'].str.split().str[-1].value_counts()
```

Out[442]:
```
bath         4357
baths        2449
Half-bath       3
half-bath       2
Name: bathrooms_text, dtype: int64
```

In [443…
```
df[df['bathrooms_text'].str.contains('share', na= False, case = False)]['bat
```

Out[443]:
```
bath         589
baths        325
half-bath      1
Name: bathrooms_text, dtype: int64
```

In [444…
```
df[df['bathrooms_text'].str.contains('half-bath', na = False, case = False)]
```

Out[444]:

	bathrooms	bathrooms_text
401	0.5	Shared half-bath
1927	NaN	Half-bath
5720	0.5	Half-bath
5914	0.5	Half-bath
6479	0.5	Private half-bath

In [445…
```
df['bathrooms_text'].str.split().str[0].unique()
```

Out[445]:
```
array(['1', '2', '1.5', '3', '2.5', '3.5', '11', nan, '4', '0', 'Shared',
       '4.5', '5', '11.5', '6.5', '5.5', '7', 'Half-bath', '6', '9.5',
       '7.5', '8', '8.5', '9', 'Private'], dtype=object)
```

In [446…
```
df['bathrooms_text'].str.split().str[1].unique()
```

Out[446]:
```
array(['shared', 'bath', 'baths', 'private', nan, 'half-bath'],
      dtype=object)
```

It appears there are shared baths. Check if there are other categories of bath rooms.

Assume if not listed as shared, it is private.

In [447…
```
df['bathrooms_shared'] = np.where(df['bathrooms_text'].str.contains('share',

df = df.drop('bathrooms_text', axis = 1)
```

Replacing license to a categorical variable

In [448…
```
pd.Series(np.where(df['license'].fillna('null').str.replace(r'[A-Za-z]', '',
         'Licensed',df['license'])).value_counts()
```

Out[448]:
```
Licensed
5872
City registration pending
178
32+ Days Listing
61
32+days Listing
27
City Registration Pending
7
Registered
4
Per city of chicago, no registration # is needed since this rental is 32 da
ys or more.       1
DOB-111617
1
City registration pending R19000048093
1
Applied for registration
1
Chicago registration number pending
1
2120298, 2120297
1
Registration number pending
1
Registration pending
1
c
1
Pending
1
PENDING
1
City registration permit pending
1
city registration pending
1
47-5611763
1
pending
1
dtype: int64
```

In [449…
```
pd.Series(np.where((df['license'].str.contains('32', na = False))&\
         (df['license'].str.contains('day', na = False, case = False)),'NA',
         np.where((df['license'].str.contains('pending', na = False, case =
         (df['license'].str.contains('applied', na = False, case = False)),
                 np.where((df['license'].str.contains(r'\d',regex = True)|\
                           (df['license'].str.contains('registered', na = Fa
```

Out[449]:
```
LICENSE    5879
nan         660
PEND        195
NA           89
dtype: int64
```

In [450…
```
df['license'] = np.where((df['license'].str.contains('32', na = False))&\
         (df['license'].str.contains('day', na = False, case = False)),'NA',
         np.where((df['license'].str.contains('pending', na = False, case =
         (df['license'].str.contains('applied', na = False, case = False)),
                 np.where((df['license'].str.contains(r'\d',regex = True)|\
                           (df['license'].str.contains('registered', na = Fa
```

Creating Geo-Features

- Distance from the Bean

In [451…
```
def calculate_to_loc(dataframe, fixed_point):
    return dataframe.apply(
        lambda row: geodesic((row['latitude'], row['longitude']), fixed_poin
        axis=1
    )
```

In [452…
```
df['latitude'].isnull().sum(), df['longitude'].isnull().sum()
```

Out[452]:
```
(0, 0)
```

In [453…
```
fixed_point = (41.892423, -87.634049) # Bean (Downtown Chicago)
fixed_lat = 41.892423
fixed_lon = -87.634049

df['km_DT'] = calculate_to_loc(df, fixed_point)
df['lat_diff'] = df['latitude'] - fixed_lat
df['lon_diff'] = df['longitude'] - fixed_lon
```

In [454…
```
df.drop(['latitude', 'longitude'], axis = 1, inplace = True)
```

Free Text Fields

- Description, amenities

In [455…
```
df.head().T
```

Out[455]:

	0	1	2	3	4
id	2384	7126	10945	28749	71930
description	Solo Hyde Park visitors are invited stay in th...	A very small studio in a wonderful neighborhood.	Beautiful first floor apartment in Historic Ol...	Located on a peaceful treelined street in ener...	A peacefu shared space i Chicago's Ukrainian..
room_type	Private room	Entire home/apt	Entire home/apt	Entire home/apt	Private room
accommodates	1	2	4	6	2
bathrooms	1.0	1.0	1.0	2.0	1.0
bedrooms	1.0	1.0	2.0	3.0	1.0
beds	1.0	1.0	2.0	3.0	1.0
amenities	["Host greets you", "Hot water kettle", "Carbo...	["Window AC unit", "Central heating", "Dishes ...	["Window AC unit", "Dishes and silverware", "H...	["Dishes and silverware", "TV with DVD player,...	["Dishes and silverware" "Dedicated workspace..
price	125.0	81.0	187.0	196.0	76.0
number_of_reviews	250	569	117	244	129
number_of_reviews_ltm	16	53	34	47	19
number_of_reviews_l30d	0	0	0	3	0
number_of_reviews_ly	20	52	36	42	20
review_scores_rating	4.99	4.72	4.72	4.82	4.89
review_scores_accuracy	4.98	4.85	4.83	4.87	4.93
review_scores_cleanliness	4.99	4.57	4.81	4.76	4.75
review_scores_checkin	4.99	4.91	4.83	4.94	4.96
review_scores_communication	4.98	4.88	4.87	4.88	4.95
review_scores_location	4.95	4.9	4.97	4.93	4.84
review_scores_value	4.94	4.76	4.72	4.72	4.84
license	LICENSE	LICENSE	LICENSE	LICENSE	LICENSE
instant_bookable	f	f	t	f	
bathrooms_shared	1	0	0	0	1
km_DT	12.2284	3.965647	2.222031	6.297014	3.781326
lat_diff	-0.104523	0.009237	0.019537	0.027127	0.003727
lon_diff	0.046249	-0.046161	-0.005761	-0.066641	-0.04529

Removing Emojis

In [456…

```
def contains_emoji(text):
    if pd.isna(text):
        return 0
    text = str(text)
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictogr
                               u"\U0001F680-\U0001F6FF"  # transport & map s
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002700-\U000027BF"  # Dingbats
                               u"\U0001F900-\U0001F9FF"  # Supplemental Symb
                               u"\U0001F700-\U0001F77F"  # Alchemical Symbol
                               u"\U0001F780-\U0001F7FF"  # Geometric Shapes
                               u"\U0001F800-\U0001F8FF"  # Supplemental Arro
                               u"\U0001F100-\U0001F1FF"  # Enclosed Alphanum
                               u"\U0001F200-\U0001F2FF"  # Enclosed Ideograp
                               u"\U0001F300-\U0001F5FF"  # Miscellaneous Sym
                               u"\U0001F600-\U0001F64F"  # Emoticons
                               u"\U0001F650-\U0001F67F"  # Ornamental Dingba
                               u"\U0001F680-\U0001F6FF"  # Transport and Map
                               u"\U0001F700-\U0001F77F"  # Alchemical Symbol
                               u"\U0001F780-\U0001F7FF"  # Geometric Shapes
                               u"\U0001F800-\U0001F8FF"  # Supplemental Arro
                               u"\U0001F900-\U0001F9FF"  # Supplemental Symb
                               u"\U0001FA00-\U0001FA6F"  # Chess Symbols
                               u"\U0001FA70-\U0001FAFF"  # Symbols and Picto
                               "]+", flags=re.UNICODE)
    return 1 if emoji_pattern.search(text) else 0

def remove_emojis(text):
    if pd.isna(text):
        return text
    text = str(text)
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictogr
                               u"\U0001F680-\U0001F6FF"  # transport & map s
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002700-\U000027BF"  # Dingbats
                               u"\U0001F900-\U0001F9FF"  # Supplemental Symb
                               u"\U0001F700-\U0001F77F"  # Alchemical Symbol
                               u"\U0001F780-\U0001F7FF"  # Geometric Shapes
                               u"\U0001F800-\U0001F8FF"  # Supplemental Arro
                               u"\U0001F100-\U0001F1FF"  # Enclosed Alphanum
                               u"\U0001F200-\U0001F2FF"  # Enclosed Ideograp
                               u"\U0001F300-\U0001F5FF"  # Miscellaneous Sym
                               u"\U0001F600-\U0001F64F"  # Emoticons
                               u"\U0001F650-\U0001F67F"  # Ornamental Dingba
                               u"\U0001F680-\U0001F6FF"  # Transport and Map
                               u"\U0001F700-\U0001F77F"  # Alchemical Symbol
                               u"\U0001F780-\U0001F7FF"  # Geometric Shapes
                               u"\U0001F800-\U0001F8FF"  # Supplemental Arro
                               u"\U0001F900-\U0001F9FF"  # Supplemental Symb
                               u"\U0001FA00-\U0001FA6F"  # Chess Symbols
```

```
                                u"\U0001FA70-\U0001FAFF"  # Symbols and Picto
                                "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

In [457…
```
df['emoji_description'] = df['description'].apply(contains_emoji).values
```

In [458…
```
df['description'] = df['description'].apply(remove_emojis)
```

Feature Engineering (NLP)

Key Phrase and Keyword Extraction

- Using brute fource (occurance) + RAKE algorithim to identify top keywords and key phrases.
- Uses levenshtein distance to keep "unique" terms.
- Vectorize description field.

In [459…
```
def preprocess_text(text):
    text = text.lower()
    # remove HTML tags
    text = re.sub(r'<[^>]+>', ' ', text)
    # remove special characters
    text = re.sub(r'[^\w\s]', ' ', text)
    # remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

stop = set(nltk.corpus.stopwords.words('english'))

# Extract keywords from text: using brute count
def extract_keywords_count(text):
    words = text.split()
    #filter keywords for at least > 4 charecters and not a stop word
    words = list(filter(lambda x: len(x) >= 4 and x not in stop, words))
    # only top 10 occurances
    keywords = [element[0] for element in Counter(words).most_common(10)]
    return keywords

# Getting the key phrases using RAKE Algorithim
def extract_key_phrase(text):
    key_phrases = Rake()
    key_phrases.extract_keywords_from_text(preprocess_text(text))
    ranked_phrases = key_phrases.get_ranked_phrases()
    ranked_phrases = list(dict.fromkeys(ranked_phrases))[:10]
    return ranked_phrases

# Unique between key words brute count + keyphrases RAKE
def merge_description(lst):
    if len(lst) <3:
        return lst
    filtered_list =[lst[0]]
    for item in lst[1:]:
        distance_score = 1-(levenshtein_distance(item.replace(' ',''), lst[0
        if distance_score <= .5:
            filtered_list.append(item)
    return filtered_list
```

In [460…
```
df['description_1'] = df['description'].map(preprocess_text).map(extract_key
df['description_2'] = df['description'].map(preprocess_text).map(extract_key
df['description_v'] = df.apply(lambda row: merge_description(list(set(row['d
df.drop(['description_1','description_2'], axis = 1, inplace = True)
```

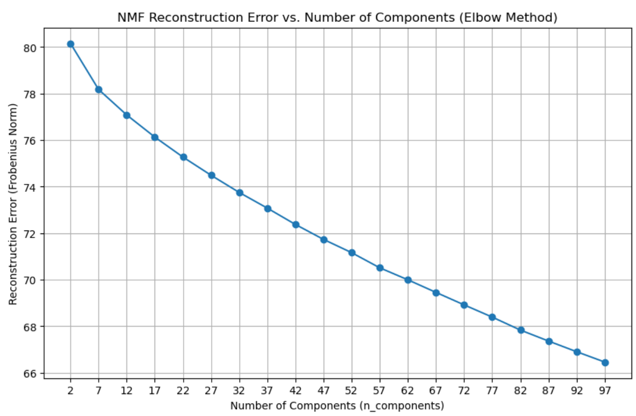
NMF to cluster the description keywords

In [461…
```
tfidf_vectorizer = TfidfVectorizer(max_df=0.85, min_df=2, stop_words='englis
tfidf_matrix = tfidf_vectorizer.fit_transform(df['description_v'])
```

Grid Search Number of Clusters

In [406…
```
reconstruction_errors = []
components_range = range(2, 100, 5)

for n_comp in components_range:
    nmf_model_eval = NMF(n_components=n_comp, random_state=1, init='nndsvda'
    nmf_model_eval.fit(tfidf_matrix)
    reconstruction_errors.append(nmf_model_eval.reconstruction_err_)
    print(f"  N_components: {n_comp}, Reconstruction Error: {nmf_model_eval.
plt.figure(figsize=(10, 6))
plt.plot(components_range, reconstruction_errors, marker='o', linestyle='-')
plt.title('NMF Reconstruction Error vs. Number of Components (Elbow Method)'
plt.xlabel('Number of Components (n_components)')
plt.ylabel('Reconstruction Error (Frobenius Norm)')
plt.xticks(list(components_range)) # Ensure all component numbers are shown
plt.grid(True)
plt.show()
```

```
  N_components: 2, Reconstruction Error: 80.1541
  N_components: 7, Reconstruction Error: 78.1745
  N_components: 12, Reconstruction Error: 77.0799
  N_components: 17, Reconstruction Error: 76.1293
  N_components: 22, Reconstruction Error: 75.2709
  N_components: 27, Reconstruction Error: 74.4900
  N_components: 32, Reconstruction Error: 73.7519
  N_components: 37, Reconstruction Error: 73.0771
  N_components: 42, Reconstruction Error: 72.3759
  N_components: 47, Reconstruction Error: 71.7331
  N_components: 52, Reconstruction Error: 71.1636
  N_components: 57, Reconstruction Error: 70.5129
  N_components: 62, Reconstruction Error: 69.9972
  N_components: 67, Reconstruction Error: 69.4567
  N_components: 72, Reconstruction Error: 68.9205
  N_components: 77, Reconstruction Error: 68.3953
  N_components: 82, Reconstruction Error: 67.8392
  N_components: 87, Reconstruction Error: 67.3672
  N_components: 92, Reconstruction Error: 66.9096
  N_components: 97, Reconstruction Error: 66.4651
```



While there is no clear "elbow", there's a significant improvement going from 2 to 12 components.The decrease from 12 to 17 is still noticeable (0.9506). After 17 components, the decreases become more consistent and smaller (generally below 0.8 and moving towards 0.4-0.5).

In [462…
```
n_cluster = 17
nmf = NMF(n_components=n_cluster, random_state=1, init='nndsvda', max_iter=5
nmf_W = nmf.fit_transform(tfidf_matrix) # Document-topic matrix
nmf_H = nmf.components_ # Topic-word matrix

def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print(f"Topic {topic_idx + 1}:")
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))
    print("\n")

# Get the feature names (words) from the TF-IDF vectorizer
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()

# Display the top words for each topic
num_top_words = 5 # Number of top words to display per topic
print(f"Top {num_top_words} words per topic:\n")
display_topics(nmf, tfidf_feature_names, num_top_words)
```

```
Top 5 words per topic:

Topic 1:
minutes away downtown chicago drive
Topic 2:
blueground home start living love
Topic 3:
centrally place located peaceful simple
Topic 4:
special rates message queen building
Topic 5:
room private shared bed living
Topic 6:
stylish space perfect experience vibrant
Topic 7:
walking chicago distance field attractions
Topic 8:
great place time comfortable family
Topic 9:
square logan blue line garden
Topic 10:
studio west away shops blue
Topic 11:
night river north level separate
Topic 12:
unit bedroom condo bath newly
Topic 13:
park lincoln wicker hyde grant
Topic 14:
access easy enjoy group perfectly
Topic 15:
walk minute min line wrigley
Topic 16:
free parking street quiet safe
Topic 17:
apartment bedroom fully furnished living
```

In [463…
```
topic_feature_names = [f'nmf_topic_{i}' for i in range(n_cluster)]

# Create a DataFrame from nmf_W with the new column names
nmf_df = pd.DataFrame(nmf_W, columns=topic_feature_names, index=df.index)

# Concatenate the original DataFrame with the new NMF features DataFrame
df = pd.concat([df, nmf_df], axis=1)
```

In [464…
```
df.drop(['description_v','description'], axis = 1, inplace = True)
```

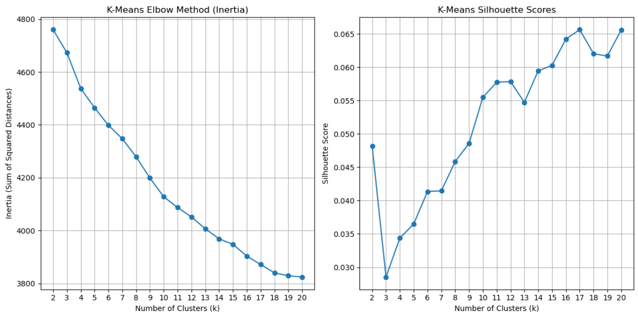
K-Means Clustering Amenities

In [465…
```
df['amenities'] = df['amenities'].apply(ast.literal_eval).apply(lambda x: '
tfidf_vectorizer = TfidfVectorizer(max_df=0.85, min_df=2, stop_words='englis
tfidf_matrix = tfidf_vectorizer.fit_transform(df['amenities'])
```

In [411…
```
inertia_values = []
silhouette_scores = []
components_range = range(2, 21)

for n_comp in components_range:
    kmeans_model = KMeans(n_clusters=n_comp, random_state=1, n_init='auto')
    # Fit the model to the TF-IDF matrix
    kmeans_model.fit(tfidf_matrix)
    # Store the inertia value
    inertia_values.append(kmeans_model.inertia_)
    # Predict clusters for the current K
    cluster_labels = kmeans_model.predict(tfidf_matrix)
    # Calculate silhouette score
    score = silhouette_score(tfidf_matrix, cluster_labels)
    silhouette_scores.append(score)
    print(f"  K: {n_comp}, Inertia: {kmeans_model.inertia_:.2f}, Silhouette
```

```
  K: 2, Inertia: 4760.26, Silhouette Score: 0.048
  K: 3, Inertia: 4673.09, Silhouette Score: 0.029
  K: 4, Inertia: 4537.48, Silhouette Score: 0.034
  K: 5, Inertia: 4464.52, Silhouette Score: 0.036
  K: 6, Inertia: 4398.31, Silhouette Score: 0.041
  K: 7, Inertia: 4347.70, Silhouette Score: 0.041
  K: 8, Inertia: 4279.05, Silhouette Score: 0.046
  K: 9, Inertia: 4198.69, Silhouette Score: 0.049
  K: 10, Inertia: 4127.57, Silhouette Score: 0.055
  K: 11, Inertia: 4087.27, Silhouette Score: 0.058
  K: 12, Inertia: 4051.29, Silhouette Score: 0.058
  K: 13, Inertia: 4006.76, Silhouette Score: 0.055
  K: 14, Inertia: 3968.52, Silhouette Score: 0.059
  K: 15, Inertia: 3947.68, Silhouette Score: 0.060
  K: 16, Inertia: 3903.22, Silhouette Score: 0.064
  K: 17, Inertia: 3871.57, Silhouette Score: 0.066
  K: 18, Inertia: 3839.72, Silhouette Score: 0.062
  K: 19, Inertia: 3829.00, Silhouette Score: 0.062
  K: 20, Inertia: 3824.01, Silhouette Score: 0.066
```

In [412…
```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(components_range, inertia_values, marker='o', linestyle='-')
plt.title('K-Means Elbow Method (Inertia)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(list(components_range))
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(components_range, silhouette_scores, marker='o', linestyle='-')
plt.title('K-Means Silhouette Scores')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.xticks(list(components_range))
plt.grid(True)

plt.tight_layout()
plt.show()
```



Elbow Method:

- K=2 to K=4: The inertia drops significantly (e.g., from 4760.26 to 4537.48, a drop of over 220 in two steps).
- K=4 to K=10: The drops continue but become somewhat less steep (e.g., from ~70 to ~80 per step of 1).
- K=10 to K=15: The reduction in inertia becomes noticeably smaller, hovering around 20-40 per step.
- K=15 to K=20: The drops become quite small, especially after K=18 (a drop of only 10.72) and K=19 (a drop of only 4.99).

Silhouette: The scores start low and generally increase, but the overall values remain quite low (all below 0.1). This is common for text data, but it suggests the clusters might not be extremely distinct or well-separated. The score reaches a local high at K=10 (0.055). It then continues to gradually climb, reaching its highest observed values at K=17 (0.066) and K=20 (0.066).

K=17 is a strong candidate because it falls within the region where the inertia curve starts to flatten, and it achieves one of the highest silhouette scores.

In [466…
```
kmeans_model = KMeans(n_clusters=17, random_state=1, n_init='auto')
cluster_assignments = kmeans_model.fit_predict(tfidf_matrix)
df['amenities_KMeans'] = cluster_assignments
```

In [467…
```
df.drop(['amenities'], axis = 1, inplace = True)
```

Feature Engineering (Encoding Categorical Variables)

In [472…
```
df = pd.get_dummies(df, columns = ['room_type','license','instant_bookable',
```

Scaling Continuous Data

- Min Max Scalar was used to keep features at 0-1 unit

In [473…
```
cont_cols = ['km_DT','lat_diff','lon_diff','bathrooms','bedrooms','beds','ac
 'number_of_reviews_l30d','number_of_reviews_ly','review_scores_rating',
       'review_scores_accuracy', 'review_scores_cleanliness',
       'review_scores_checkin', 'review_scores_communication',
       'review_scores_location', 'review_scores_value']
```

In [ ]:
```
scaler = MinMaxScaler()
continuous = scaler.fit_transform(df[cont_cols])
df[cont_cols] = continuous
```

In [480…
```
df = df.set_index('id')
```

In [ ]:
```
df.to_pickle('data.pkl')
df = pd.read_pickle('data.pkl')
```

Model Building

- Agglomerative Clustering
  - Does not require the number of clusters to be specified in advance
  - Bottoms up processing
  - Small data set does not get computationally expensive
  - Properties exhibit hierarchical relationships

In [490…
```
len(df.dropna()), len(df)
```

Out[490]:
```
(6077, 6823)
```

Since there are not many missing NaNs, we will drop rows that contain a null value.

In [491…
```
df = df.dropna()
```

Dendogram

In [503…
```
X = df.values
```

Euclidean distance is the most common and intuitive distance metric. It measures the "straight-line" distance between two points in a multi-dimensional space.

Since the features are all numerical and have been Min-Max scaled to a 0-1 range, Euclidean distance becomes very meaningful.

Scaling ensures that no single feature dominates the distance calculation merely because it has a larger range of values. It implies that proximity in this feature space is a direct measure of similarity.

Ward linkage tends to produce clusters that are roughly spherical, compact, and of similar size. It's generally robust and less susceptible to noise compared to other linkage methods like 'single' linkage. It tries to keep the clusters as homogeneous as possible by minimizing the increase in total within-cluster variance.

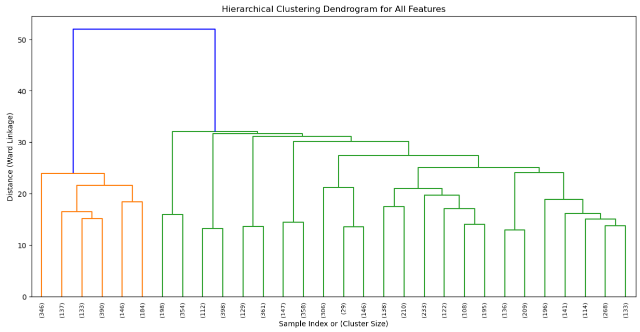
In [ ]:
```
Z = linkage(X, method='ward', metric='euclidean')

plt.figure(figsize=(15, 7))
plt.title('Hierarchical Clustering Dendrogram for All Features')
plt.xlabel('Sample Index or (Cluster Size)')
plt.ylabel('Distance (Ward Linkage)')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=30,  # show only the last 30 merges
    leaf_rotation=90,
    leaf_font_size=8,
    show_leaf_counts=True,
    above_threshold_color='blue',
)
```

Out[ ]:

```
{'icoord': [[25.0, 25.0, 35.0, 35.0],
  [15.0, 15.0, 30.0, 30.0],
  [45.0, 45.0, 55.0, 55.0],
  [22.5, 22.5, 50.0, 50.0],
  [5.0, 5.0, 36.25, 36.25],
  [65.0, 65.0, 75.0, 75.0],
  [85.0, 85.0, 95.0, 95.0],
  [105.0, 105.0, 115.0, 115.0],
  [125.0, 125.0, 135.0, 135.0],
  [155.0, 155.0, 165.0, 165.0],
  [145.0, 145.0, 160.0, 160.0],
  [175.0, 175.0, 185.0, 185.0],
  [215.0, 215.0, 225.0, 225.0],
  [205.0, 205.0, 220.0, 220.0],
  [195.0, 195.0, 212.5, 212.5],
  [180.0, 180.0, 203.75, 203.75],
  [235.0, 235.0, 245.0, 245.0],
  [285.0, 285.0, 295.0, 295.0],
  [275.0, 275.0, 290.0, 290.0],
  [265.0, 265.0, 282.5, 282.5],
  [255.0, 255.0, 273.75, 273.75],
  [240.0, 240.0, 264.375, 264.375],
  [191.875, 191.875, 252.1875, 252.1875],
  [152.5, 152.5, 222.03125, 222.03125],
  [130.0, 130.0, 187.265625, 187.265625],
  [110.0, 110.0, 158.6328125, 158.6328125],
  [90.0, 90.0, 134.31640625, 134.31640625],
  [70.0, 70.0, 112.158203125, 112.158203125],
  [20.625, 20.625, 91.0791015625, 91.0791015625]],
 'dcoord': [[0.0, 15.151337945036339, 15.151337945036339, 0.0],
  [0.0, 16.408342949409203, 16.408342949409203, 15.151337945036339],
  [0.0, 18.36196818367173, 18.36196818367173, 0.0],
  [16.408342949409203,
   21.570525473978876,
   21.570525473978876,
   18.36196818367173],
  [0.0, 23.950096881173756, 23.950096881173756, 21.570525473978876],
  [0.0, 15.992429147952588, 15.992429147952588, 0.0],
  [0.0, 13.212212382287257, 13.212212382287257, 0.0],
  [0.0, 13.655983565268999, 13.655983565268999, 0.0],
  [0.0, 14.44988033264926, 14.44988033264926, 0.0],
  [0.0, 13.515377619159258, 13.515377619159258, 0.0],
  [0.0, 21.22031430411815, 21.22031430411815, 13.515377619159258],
  [0.0, 17.43670841951672, 17.43670841951672, 0.0],
  [0.0, 13.995079906684452, 13.995079906684452, 0.0],
  [0.0, 17.036356426258845, 17.036356426258845, 13.995079906684452],
  [0.0, 19.736068534774237, 19.736068534774237, 17.036356426258845],
  [17.43670841951672,
   20.97872637659182,
   20.97872637659182,
   19.736068534774237],
  [0.0, 12.877101612671687, 12.877101612671687, 0.0],
  [0.0, 13.738732587920051, 13.738732587920051, 0.0],
  [0.0, 15.059311979356496, 15.059311979356496, 13.738732587920051],
  [0.0, 16.171825698172135, 16.171825698172135, 15.059311979356496],
  [0.0, 18.923958171748076, 18.923958171748076, 16.171825698172135],
  [12.877101612671687,
   24.023390578785133,
   24.023390578785133,
   18.923958171748076],
  [20.97872637659182,
   25.008186373223584,
   25.008186373223584,
   24.023390578785133],
  [21.22031430411815,
   27.35395968429988,
   27.35395968429988,
   25.008186373223584],
  [14.44988033264926,
   30.132718068034876,
   30.132718068034876,
   27.35395968429988],
  [13.655983565268999,
   31.142268355200095,
   31.142268355200095,
   30.132718068034876],
```

```
 [13.212212382287257,
  31.651684121774586,
  31.651684121774586,
  31.142268355200095],
 [15.992429147952588,
  32.081209853157695,
  32.081209853157695,
  31.651684121774586],
 [23.950096881173756,
  51.94864804868549,
  51.94864804868549,
  32.081209853157695]],
'ivl': ['(346)',
 '(137)',
 '(133)',
 '(390)',
 '(146)',
 '(184)',
 '(198)',
 '(354)',
 '(112)',
 '(398)',
 '(129)',
 '(361)',
 '(147)',
 '(358)',
 '(306)',
 '(29)',
 '(146)',
 '(138)',
 '(210)',
 '(233)',
 '(122)',
 '(108)',
 '(195)',
 '(136)',
 '(209)',
 '(196)',
 '(141)',
 '(114)',
 '(268)',
 '(133)'],
'leaves': [12121,
 12100,
 12098,
 12123,
 12110,
 12118,
 12020,
 12076,
 12001,
 12074,
 12051,
 12094,
 12014,
 12077,
 12120,
 12069,
 12122,
 12106,
 12114,
 12119,
 12111,
 11836,
 12113,
 12019,
 12034,
 12035,
 12090,
 12095,
 12062,
 12089],
'color_list': ['C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'blue'],
'leaves_color_list': ['C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
 'C2',
```

```
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2',
'C2']}
```



Potential Ks

- K = 3
  - This would separate the data into the large orange group on the left and two large green groups on the right. This is a very broad clustering.
- K = 5 or K = 6
  - Cutting around a distance of 25-30 seems to reveal a good level of distinctness for a moderate number of clusters.
- K = 8 to K = 10
  - More granular clusters, cutting around distance 15-20 could yield 8 to 10 clusters.

Base Model

In [ ]:
```
n_clusters = 5
base_model = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidea
base_model.fit(X)
labels = base_model.labels_
```

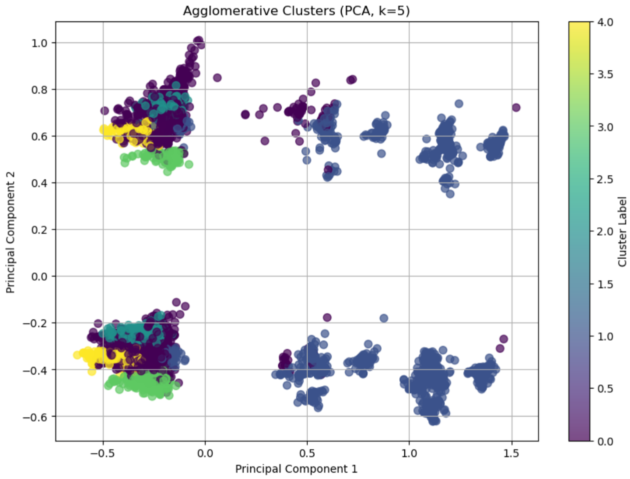
In [509…
```
silhouette_avg = silhouette_score(X, labels)
davies_bouldin_idx = davies_bouldin_score(X, labels)
calinski_harabasz_idx = calinski_harabasz_score(X, labels)

print(f"Cluster Evaluation for K = {n_clusters}:")
print(f"Silhouette Score: {silhouette_avg:.3f} (Higher is better, range -1 t
print(f"Davies-Bouldin Index: {davies_bouldin_idx:.3f} (Lower is better, min
print(f"Calinski-Harabasz Index: {calinski_harabasz_idx:.3f} (Higher is bett
```

```
Cluster Evaluation for K = 5:
Silhouette Score: 0.138 (Higher is better, range -1 to 1)
Davies-Bouldin Index: 1.888 (Lower is better, minimum 0)
Calinski-Harabasz Index: 558.849 (Higher is better)
```

The Silhouette Score of 0.138 is positive, indicating that data points are, on average, more similar to their own cluster than to others, its relatively low value suggests the clusters are not very distinct or well-separated, implying some overlap. This is further supported by the Davies-Bouldin Index of 1.888, which, being above 1, suggests that the within-cluster variance is large compared to the separation between clusters, pointing to less defined boundaries. The Calinski-Harabasz Index of 558.849 is a positive value, indicating some cluster structure.

Visualizing Clusters

In [510…
```
pca = PCA(n_components=2, random_state=1)
pca_components = pca.fit_transform(X)

plt.figure(figsize=(10, 7))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=labels, cmap='viri
plt.title(f'Agglomerative Clusters (PCA, k={n_clusters})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster Label')
plt.grid(True)
```



Cluster Separation and Distribution:

Vertical Separation by PC2: There's a very clear and strong separation along Principal Component 2 (the y-axis). The clusters tend to group into two distinct horizontal bands of -0.2 to -0.6 and 0.4 to 1.0.

The yellow and green clusters are quite compact and visually well-separated from the large dark purple and blue clusters. The teal clusters are also moderately well-seperated.

- Within the top band, we see distinct yellow, green, and a teal cluster.
- Similarly, within the bottom band, we see another set of yellow, green, and a teal cluster.

GridSearch N Clusters

In [512…
```
n_clusters_to_test = [3, 5, 6, 8, 9, 10]

results = []

for n_clusters in n_clusters_to_test:
    model = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidean
    labels = model.fit_predict(X)
    silhouette_avg = silhouette_score(X, labels)
    davies_bouldin_idx = davies_bouldin_score(X, labels)
    calinski_harabasz_idx = calinski_harabasz_score(X, labels)
    # Store results
    results.append({
        'n_clusters': n_clusters,
        'silhouette_score': silhouette_avg,
        'davies_bouldin_index': davies_bouldin_idx,
        'calinski_harabasz_index': calinski_harabasz_idx
    })

results_df = pd.DataFrame(results)
```

In [513…
```
results_df
```

Out[513]:

	n_clusters	silhouette_score	davies_bouldin_index	calinski_harabasz_index
0	3	0.078284	2.029176	648.675525
1	5	0.138328	1.888306	558.849108
2	6	0.168462	1.878533	550.503589
3	8	0.196052	2.048123	524.217370
4	9	0.206415	1.951481	514.261610
5	10	0.213882	1.965713	510.894984

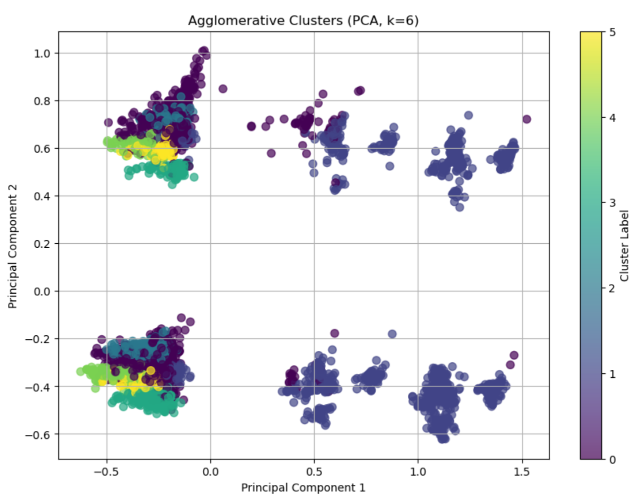
Trade-off at K=6:

This gives us the lowest Davies-Bouldin Index (1.878), indicating the best balance of compactness and separation. The Silhouette Score is also reasonably good for this dataset (0.168), although not the absolute highest. The Calinski-Harabasz index is still relatively high compared to higher K values, suggesting it retains some overall cluster density.

```
In [514…
n_clusters = 6
base_model = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidea
base_model.fit(X)
labels = base_model.labels_

silhouette_avg = silhouette_score(X, labels)
davies_bouldin_idx = davies_bouldin_score(X, labels)
calinski_harabasz_idx = calinski_harabasz_score(X, labels)

print(f"Cluster Evaluation for K = {n_clusters}:")
print(f"Silhouette Score: {silhouette_avg:.3f} (Higher is better, range -1 t
print(f"Davies-Bouldin Index: {davies_bouldin_idx:.3f} (Lower is better, min
print(f"Calinski-Harabasz Index: {calinski_harabasz_idx:.3f} (Higher is bett
```

```
Cluster Evaluation for K = 6:
Silhouette Score: 0.168 (Higher is better, range -1 to 1)
Davies-Bouldin Index: 1.879 (Lower is better, minimum 0)
Calinski-Harabasz Index: 550.504 (Higher is better)
```

```
In [515…
pca = PCA(n_components=2, random_state=1)
pca_components = pca.fit_transform(X)

plt.figure(figsize=(10, 7))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=labels, cmap='viri
plt.title(f'Agglomerative Clusters (PCA, k={n_clusters})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster Label')
plt.grid(True)
```



The k=6 plot reveals a slightly finer granularity on the left side, where the yellow and green clusters appear to differentiate further, indicating the additional cluster has likely refined the groupings within this dense region.

This re-segmentation on the left contrasts with the largely unchanged distribution of the dominant right-side cluster across both visualizations.