University of Glasgow | School of Computing Science

BSc (Hons) Computing Science

CSC3005 Data Analytics

Recommender System - REPORT

AY2020/2021

P2: Group 1

Group Members (Student ID)	Crystal Toh Yi Shan (1901851)
	The Nu Win (1901854)
	Wu JiaJie (1901828)
	Teng Zheng Yu (1901888)

Abstract

Recently with Covid-19, people have been told to stay home when possible and isolate themselves from others and to reduce social interactions. As such everyone is feeling lonelier than ever during this pandemic, and people are trying to find ways to find company. One thing more people did now, compared to pre-Covid-19 was to purchase or adopt pet dogs to keep them company. As such, dog adoption and sales have actually gone up drastically during this pandemic.

However given the circumstances of the pandemic, the situation is unstable and rules can change quickly affecting how businesses can operate and where people can go. Hence visiting a pet shop in some cases might not be a feasible solution to those who are looking for a pet dog.

Thus, we propose an online solution of a dog recommender system which will assist users with deciding and looking into the type of dog breeds that will suit the buyer's needs, requirements, and lifestyle choices based within a Singapore context.

Hence, we will discuss our solution of implementing the system using content-based filtering, and other data analytics methods and algorithms.

Table of Contents

Abstract **1**

Table of Contents **2**

Contribution Page **3**

Chapter 1: Introduction **4**

Chapter 2: Algorithms **5**

- 2.1 Data Pre-processing 5
- 2.2 Data Mining Algorithms 9
  - 2.2.1 K-means Clustering 10
  - 2.2.2 Pseudocode for Data Mining Algorithms 11
- 2.3 Recommendation Generation 11
  - 2.3.1 Pseudocode for Recommendation Generation 12
- 2.4 Result Analysis 13

Chapter 3: Conclusion **15**

- Conclusion 15
- Future Work 15

Source Code **16**

References **29**

Contribution Page

	Crystal Toh Yi Shan	The Nu Win	Teng Zheng Yu	Wu Jia Jie
Project				
Data Pre-processing	x			x
Data Mining Algorithm		x	x	
Data Visualization	x	x	x	x
Data Analysis	x	x	x	x
Report				
Chapter 1	x		x	
Chapter 2	x	x	x	x
Chapter 3		x		x

Chapter 1: Introduction

A recommendation system (RS) is a tool that helps users to discover relevant contents based on their preferences or requirements. There are several approaches to such a system, but the major approaches are Content-based Filtering (CBF) and Collaborative Filtering (CF). The former recommends the relevant items to users by collecting user's preferences whereas the latter recommends based on a set of users who are most similar to the active user.

An online survey conducted in 2021 January showed that 3 in 5 people owned a pet in Asia, and dogs are ranked 1st as the most owned pets in most Asian countries, including Singapore [1]. A news article published by The Strait Times in 2020 also mentioned that there was an increase in adopting and fostering pets during Covid-19 Pandemic in Singapore [2]. Currently, there are a few dog breed recommendation systems in the market, but they have some limitations. They only recommend one dog breed based on the user's preferences. Therefore, the user does not have freedom in choosing the dog breeds.

The objective of this study is to implement a dog recommendation system that can find the related dogs to users based on the user's feature preference of the dog. Our study is focused on the content-based recommendation system because content-based recommendation can be used to find the related dogs for new users given the user's query unlike the collaborative filtering recommendation which cannot handle when new users come in. Moreover, choosing a pet to adopt is purely based on the owner's interests and content-based provides user independence ability. Therefore, the content-based recommendation method is the perfect suit for our dog recommender system.

Chapter 2: Algorithms

2.1 Data Pre-processing

The data collected consists of over 198 breeds of dogs, with 40 features that differentiates one breed of dog from the other. The steps taken during data pre-processing are as follows:

1) Import required libraries
2) Import dataset CSV file into dataframe
3) Get information about the dataset (features, datatype, number of rows and columns)
4) Drop irrelevant features (overall features of same categories, cold weather)
5) Find if there are any duplicated data
6) Find if there are any missing data
7) Drop one dog breed because its 9 out of 31 features are missing
8) Explore the variation between “max_lifespan vs min_lifespan” and “max_weight and in_weight”

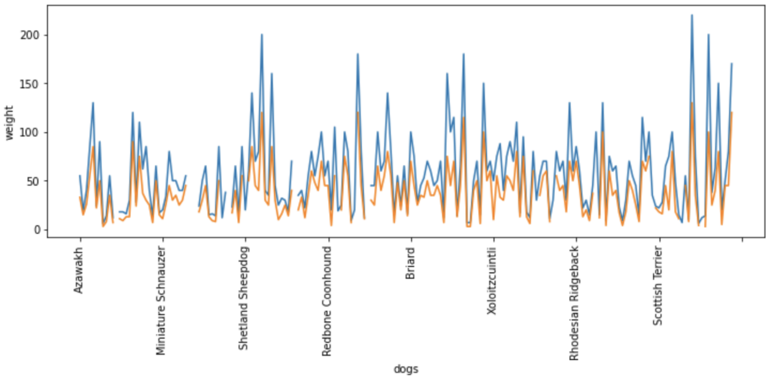

Figure 1: Max and min lifespan VS dog breeds

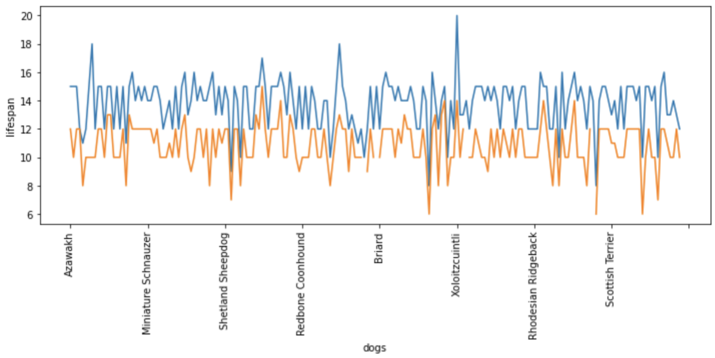

Figure 2: Max and Min weight vs dog breeds

From figure x and x, maximum and minimum values of each feature are similar and thus the range of weight and lifespan are not useful for analysis. Therefore, they are dropped.

9) Perform correlation to do more feature selections
To do this, we use heatmap to compare all the features in pairs.

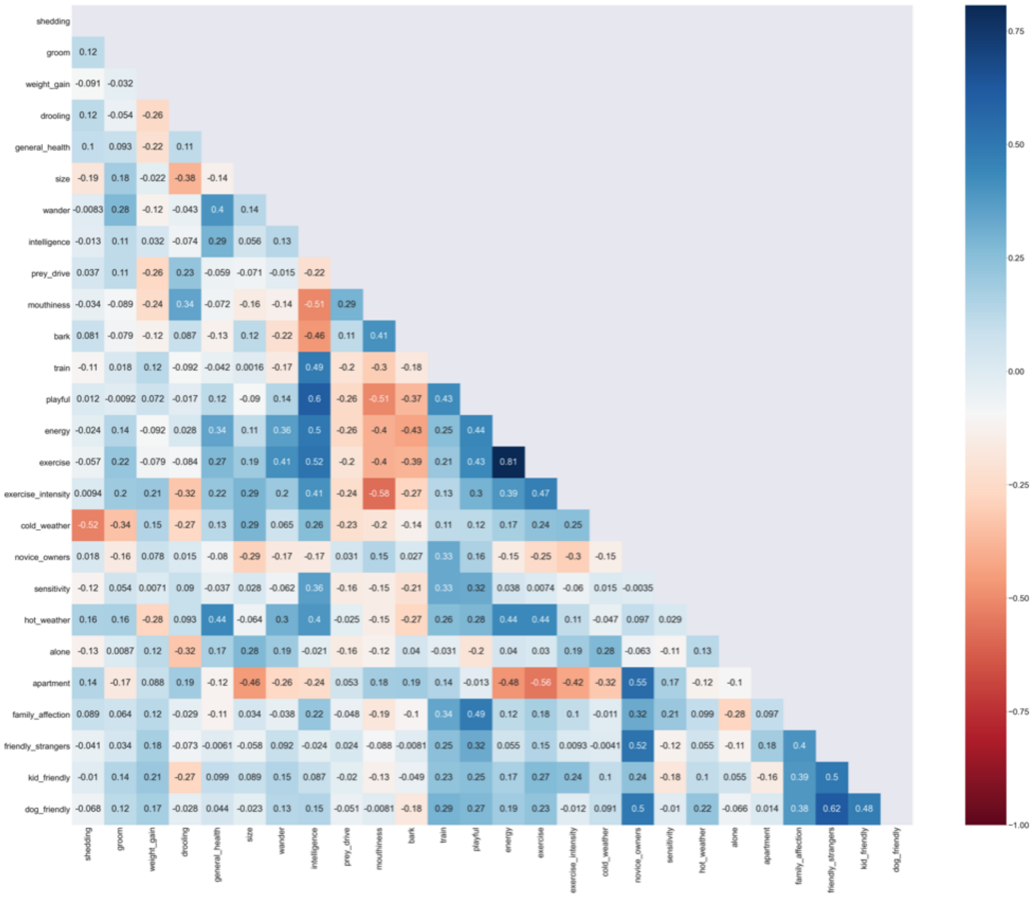

Figure 3: Heatmap of all features correlation

From figure x, it is observed that exercise and energy are highly correlated with a value of 0.81 and friendly to other dogs and friendly to other strangers has a weak correlation value of 0.61. Moreover, to verify the correlation results, a bubble chart with an additional frequency parameter is used which is displayed as the size of the dot.

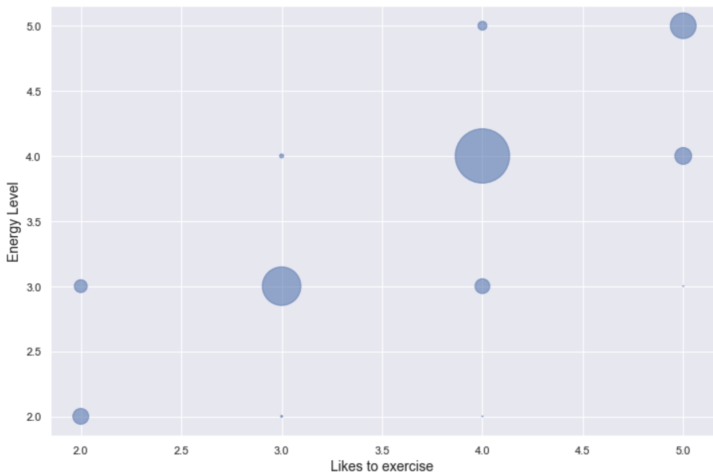

Figure 4: Energy Level VS Likes to Exercise Bubble Chart

As shown in figure x, it is verified that energy level and likes to exercise indeed are correlated. Therefore, feature exercise is dropped.

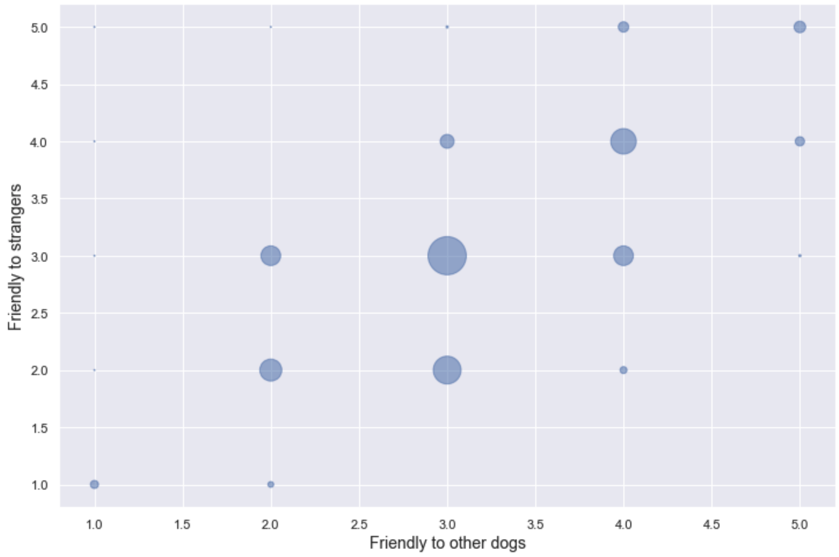

Figure 5: Friendly to strangers VS friendly to other dogs bubble chart

Figure x verifies that the correlation value we get from the heatmap is accurate since it is observed that two features, ‘friendly to strangers’ and ‘friendly to other dogs’, are not highly correlated.

10) After dropping the irrelevant and redundant features, principal component analysis (PCA) is applied to reduce the dimensions of the dataset while retaining the same information. Cumulative explained variance ratio graph is used to visualize and calculate how many eigenvectors are needed to capture the 70% of the original data.

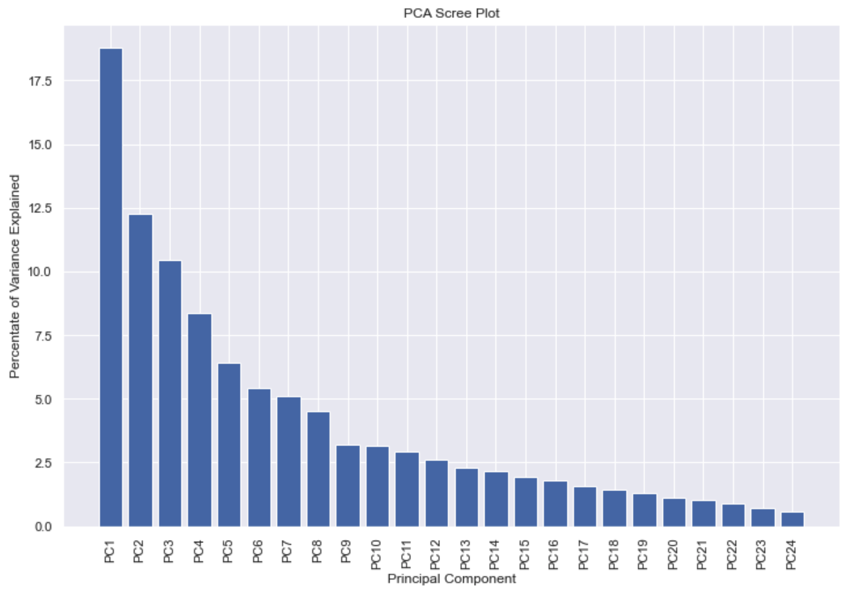

Figure 6: Variance of every PC

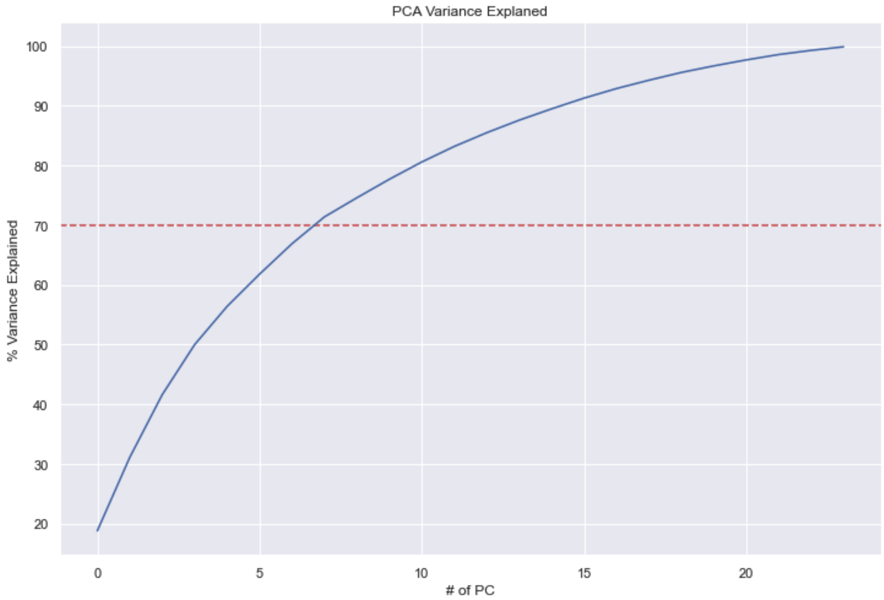

Figure 7: Number of PC VS percentage of variance explained

From the figure x above, it is observed that 8 eigenvectors are needed to capture 70% of the dataset features.

The minimum total variance explained can be widely different for scenario to scenario. In our case we have found that the total variance from 70% onwards does not lose much of it’s information. Furthermore, inversely, the number of features that are being reduced is high in comparison, which makes this a reasonable tradeoff.

Furthermore, we explore the weight of each feature in every eigenvector by using heatmap and sort the values.

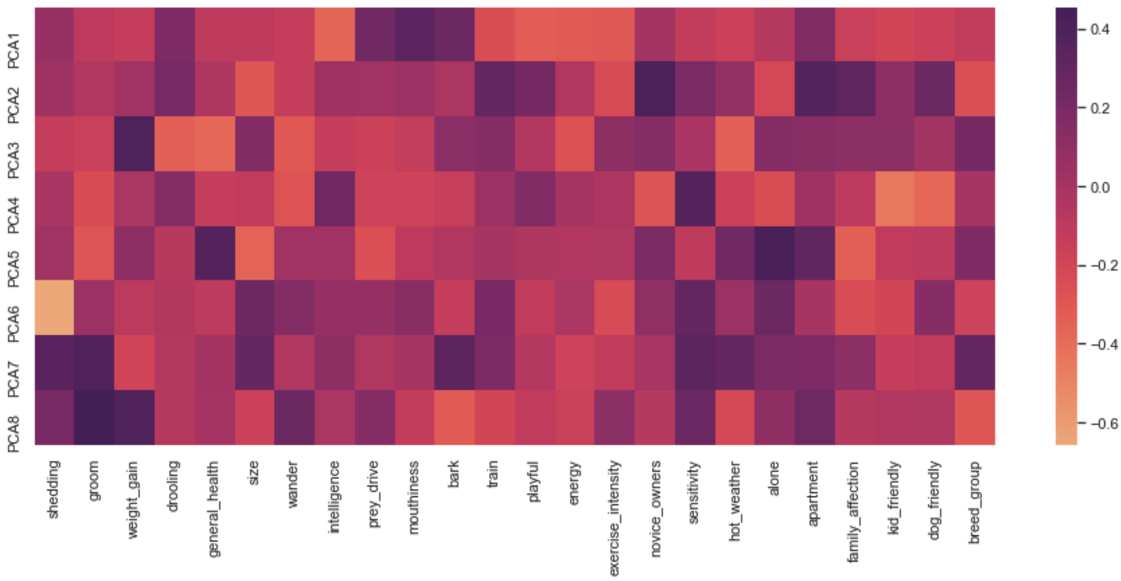

Figure 8: Heatmap of PCA VS features

	1st Max	2nd Max	3rd Max
PC-1	mouthiness	bark	prey_drive
PC-2	novice_owners	apartment	family_affection
PC-3	weight_gain	breed_group	size
PC-4	sensitivity	intelligence	playful
PC-5	alone	general_health	apartment
PC-6	sensitivity	alone	size
PC-7	groom	shedding	bark
PC-8	groom	weight_gain	wander

Figure :9 Weight of top contributed features in each PC

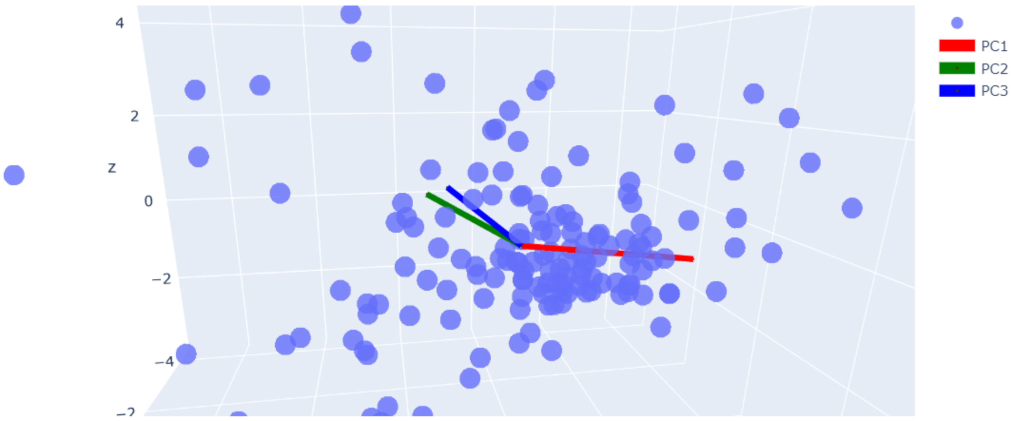

Figure 10: 3D plot of PC1 VS PC2 VS PC3

From the figure x, it is observed that PC1 is the largest eigenvector that describes most of the variance.

2.2 Data Mining Algorithms

There are four types of data mining tasks, namely, clustering, predictive modelling, association rules and anomaly detection. In our approach, we narrow down the options to clustering and classification since they are the most suitable methods to find the similarity metric. Classification is supervised and used for data which has predefined classes while clustering is unsupervised and used when no predefined classes are available and then, group the similar data into the same cluster. In our dataset, we have 198 unique dog breeds. Therefore, if classification were applied to data mining approach, the limitations were there would be 198 labels and when new data comes in, the classification would not handle as well as clustering. Therefore, the clustering method is chosen as a data mining method.

Anomaly detection is irrelevant to our dataset as each data entry and sample refers to a unique dog breed. Hence no data should be treated as an anomaly unless it's features are all lopsided, which none exists within our dataset.

2.2.1 K-means Clustering

K-Means clustering is a clustering algorithm based on distances and centroids. First we calculate the sum of squared errors against the number of clusters and plot it to a graph. By identifying the knee point, we will have a good estimate of the number of clusters to generate using the algorithm.

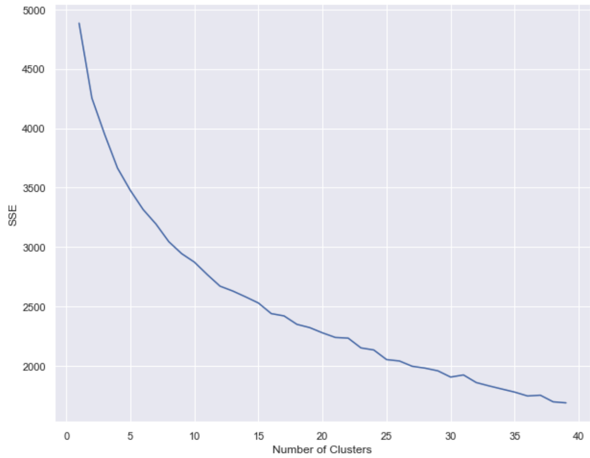

Figure 11: Plot of SSE for each number of clusters

Once we fit the dataset into the algorithm, the algorithm will run for a set amount of iterations or until the cluster centroids do not change anymore. Once that is done we will have a list of clusters which explains what clusters each sample falls into. We then append these clusters to the original dataset and arrange them accordingly based on cluster ID.

family_affection	kid_friendly	dog_friendly	Cluster ID
5	3	2	0
5	1	3	0
4	2	2	0
5	3	2	0
5	4	3	0
...	...	...	...
5	5	5	12
5	3	5	12
5	5	3	12
5	4	5	12
4	5	3	12

Figure 12:
Appending Cluster ID to each sample

2.2.2 Pseudocode for Data Mining Algorithms

START

SET k_means cluster to 13 and iteration to 50
CALL k_means.fit(the PCA data's dataframe object)
SET labels = generated k means labels
SET clusterID dataframe with cluster ID
SET dataset_cluster and CALL copy of dataset
SET dataset_cluster['Cluster ID'] = labels
SET dataset_cluster.index = list with names of all dog breeds
SET dataset_cluster and sort the cluster ID in ascending order
DISPLAY dataset_cluster.columns

INITIALIZE C0 as a list containing all dog breeds in that cluster
INITIALIZE C1 as a list containing all dog breeds in that cluster
INITIALIZE C2 as a list containing all dog breeds in that cluster
INITIALIZE C3 as a list containing all dog breeds in that cluster
INITIALIZE C4 as a list containing all dog breeds in that cluster
INITIALIZE C5 as a list containing all dog breeds in that cluster
INITIALIZE C6 as a list containing all dog breeds in that cluster
INITIALIZE C7 as a list containing all dog breeds in that cluster
INITIALIZE C8 as a list containing all dog breeds in that cluster
INITIALIZE C9 as a list containing all dog breeds in that cluster
INITIALIZE C10 as a list containing all dog breeds in that cluster
INITIALIZE C11 as a list containing all dog breeds in that cluster
INITIALIZE C12 as a list containing all dog breeds in that cluster
SET cluster_array = C0 to 12

END

2.3 Recommendation Generation

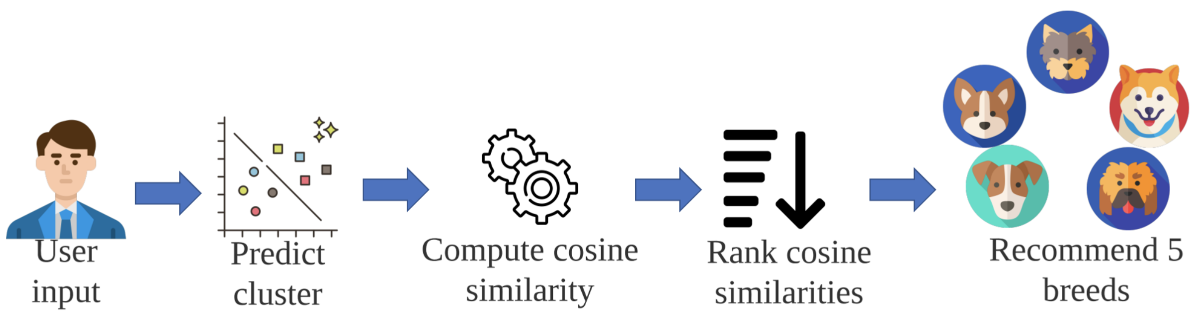

Figure 13: Overall recommendation generation process

Firstly, we take the user's query on preferred features of the dog. In the second step, we fit and transform the user's query to predict which cluster user's query belongs to. In the next step, compute the similarity metrics between the user's query and the dog breeds in the same cluster.

The representation of user's input features and the dog breeds features are constructed to select the most relevant 5 dog breeds to users. To achieve this, we are using cosine similarity to compute 'user preference vectors, Vuser ' and every 'dog breeds, Vbreed' in the same cluster. The formula to calculate cosine similarity is as follows:

$$Similarity = \cos(\theta) = \frac{V_{user} \cdot V_{breed}}{||V_{user}||.\,||V_{breed}||}$$

After computing the cosine similarity between Vuser and every Vbreed, a recommendation list will be generated with the ranking in ascending value. In cosine similarity, the angle between two vectors are computed and thus, the smaller the value (angle), the more similar to the user query. Therefore, we will recommend the 5 dog breeds which have the lowest cosine similarity values.

2.3.1 Pseudocode for Recommendation Generation

```
START
FUNCTION input_is_valid():
        TRY
                Get user input
                IF user input is more than 0 and less than 5
                        RETURN True
                ELSE
                        PRINT “Input not valid, please enter input again”
                        RETURN False
                END IF
        EXCEPT
                PRINT “Input not valid, please enter input again”
                RETURN false

FOR every question in the question list
        SET next_qns to False
        WHILE next_qns is False
                INPUT user_input for every questions
                SET user_input to user_input
                IF input_is_valid(user_in):
                        APPEND user_in to list_of_user_input
                        SET next_qns as True
                END IF
        END WHILE
END FOR
```

```
PRINT list_of_user_input

SET list_of_user_input and reshape it to (-1,1)
SET user_input and scale it
SET user_input and reshape it to (1,-1)

SET the number of eigenvectors  to 8
FIT the dataframe with the eigenvector
TRANSFORM user_input to pca

FIT k.means clustering for user_pca
PREDICT the user_pca
SET the user_input array to chosen_cluster

SET a new empty dataframe
APPEND no index into the dataframe
SET recommendation_list

FOR every row in dataframe
   i_Values = newDF.loc[row.name].values

   COMPUTE dot_product of list_of_user_input and i row in dataframe
   COMPUTE cosine_similarity between user_input and i row in dataframe
   SET result = row_name and cosine similarity value
   APPEND the result into recommendation_list
END FOR

SORT the value in recommendation_list in ascending order
PRINT the top 5 lowest value and their index name

SET labels = ['Breed', 'Cosine Similarity']
SET a new result_dataframe and append the recommendation list
PRINT top 5 dog breed names

END
```

2.4 Result Analysis

```
Hi, I will be recommending 5 dog breeds to you today.
For each question, enter a number from 1 to 5.
1 = Strongly disagree. 5 = Strongly agree.
1. I do not mind a dog that sheds its fur often: 3
2. I prefer a dog that is easy to groom: 5
3. I do not mind a dog that can gain weight easily: 3
4. I do not mind my dog drooling: 4
5. A dog that is resistant to illnesses is important: 2
6. I prefer a big dog: 4
7. I prefer an intelligent dog bred for jobs that require decision: 4
8. I prefer a dog that wanders around on it's own: 3
9. I prefer a dog that chases and hunts for small prey: 4
10. I do not mind a dog that likes chewing on things: 3
11. I prefer a dog that barks: 3
12. I want a dog that is easy to train: 4
13. I prefer a playful dog: 4
14. I prefer a dog with high energy and stamina: 3
15. I like taking my dog out for walks: 3
16. I have little to no experience raising dogs: 3
17. My home is generally quiet without loud sounds or distractions: 4
18. I would like to bring my dog out and excercise on a hot sunny day: 4
19. I prefer a dog that is able to be by itself and not crave attention: 3
20. I prefer my dog to be calm indoors and polite with strangers: 3
21. I prefer a dog that is affectionate with my family members: 4
22. The dog has to be friendly with small kids & children: 3
23. I would like my dog to be friendly and not dominate other dogs: 3
Inputs: [3, 5, 3, 4, 2, 4, 4, 3, 4, 3, 3, 4, 4, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3]
According to your desired dog features, Top 5 Breeds to consider are as follows:
    Chinese Crested
         Chihuahua
 Italian Greyhound
    Xoloitzcuintli
   Toy Fox Terrier
```

Figure 14: User Interface

We alter the value by 1 for Family_affection and check if the user's query belongs to cluster 0 as it should be. The recommended dogs are from cluster 0 as well.

```
print("The user input falls into cluster:",ans)
chosen_cluster
```

```
The user input falls into cluster: 0
```

	shedding	groom	weight_gain	drooling	general_health	size	wander	intelligence	prey_drive	mouthiness	...
Rhodesian Ridgeback	3	5	3	4	2	4	4	3	4	3	...
Whippet	4	5	1	5	3	3	4	4	5	3	...
Bedlington Terrier	5	2	3	5	3	2	4	3	5	4	...
Ibizan Hound	5	5	1	5	4	3	4	3	5	3	...
Chinook	2	3	3	5	3	4	1	4	4	4	...
Xoloitzcuintli	5	5	3	5	5	3	5	5	5	3	...
Scottish Deerhound	4	4	2	3	2	5	4	3	5	3	...
Chihuahua	4	5	3	5	2	1	2	3	5	4	...
Italian Greyhound	4	5	1	5	2	1	4	3	5	4	...
Canaan Dog	3	3	2	5	4	3	2	4	3	3	...
Toy Fox Terrier	4	5	2	4	2	1	4	4	5	2	...
Chinese Crested	5	4	2	5	2	1	1	3	3	4	...
Saluki	4	4	1	5	4	4	5	3	5	4	...
Greyhound	3	5	2	5	3	4	4	3	5	4	...

14 rows × 23 columns

Figure 15: Cluster 0 dataframe

Chapter 3: Conclusion

Conclusion

We have seen the different steps taken from pre-processing, to data-mining and post processing with the results of recommending dog breeds to the user. Each step along the way was important and crucial in reaching the final goal. Pre-processing prepared the data in a way that was usable for the context in our problem statement. It also helps with visualization, which helps us greatly in determining steps to take and giving insight into our data. Data mining then allowed us to make use of our dataset and test around different algorithms to see what works and how it works. K-means is a tool that fits into our context and was hence utilized. And we further utilized cosine-similarity in order to determine the closest data samples with the input that was taken from the user. This way, we can tell which of the data points, or in our case the dog breeds are closest to what the user has entered based on the query. Lastly, post-processing closes everything by validating if our output aligns with what is expected. This is also necessary as this analysis tells us if the results are reliable to a certain degree. In post processing we were also able to see how the data has been clustered by the system. Since the dataset contains a high number of features, and the number of total possible combinations for all features is quite high, it can be difficult to know how the groups are clustered.

Future Work

For future works we have decided to have a system with a more interactive user interface. As the context of this work is simply to integrate a system with the appropriate algorithms for recommending dog breeds to users, we have looked over user interface as a necessity.
Furthermore, since this is the first iteration of this project, we have decided not to go with a user based collaborative filtering system. This would need users input as part of our database, and should this be implemented, we plan to roll this out and validate it through AB testing. Furthermore, we plan to increase the number of dog breeds within our system's dataset in order to make the system more robust and complete.

Source Code

```
# import required libraries
import pandas as pd
import numpy as np
import seaborn as sns

import plotly.express as px
import plotly.graph_objects as go
import plotly.graph_objs as go

%matplotlib inline
import matplotlib.pyplot as plt

import scipy.stats as stats
from scipy.stats import chi2_contingency

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_classif
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import cluster

from pca import pca as pca_2
from sklearn import cluster
from numpy import dot
from numpy.linalg import norm
import numpy as np
from sklearn import preprocessing
from pandas import DataFrame

# load the dataset
dataset = pd.read_csv('dogs.csv', na_values='?', index_col='|')
print(dataset.shape)

header =
['url','shedding','overall_health','groom','weight_gain','drooling','general_health','size','wander',
'intelligence',
```

```
'overall_trainability','prey_drive','mouthiness','bark','train','playful','energy','exercise','overall_e
xerciseneeds',

'exercise_intensity','cold_weather','novice_owners','sensitivity','overall_adaptability','hot_weat
her','alone',

'apartment','family_affection','friendly_strangers','overall_friendly','kid_friendly','dog_friendly','
breed_group',

'max_lifespan','min_lifespan','max_weight','min_weight','min_height','max_height','shoulder_
height']
dataset.index.names = ['breed']
dataset.columns = header
dataset.head(5)
dataset.info()
dataset['min_lifespan']= pd.to_numeric(dataset['min_lifespan'],errors='coerce')
dataset['min_weight']= pd.to_numeric(dataset['min_weight'],errors='coerce')

# drop overall_features - does not define a feature
dataset =
dataset.drop(['overall_health','overall_trainability','overall_exerciseneeds','overall_adaptabilit
y',
                    'overall_friendly','url','min_height','max_height','shoulder_height'], axis = 1)
dataset.info()
# locate rows of duplicate data
dups = dataset.duplicated().sum()
print('The amount of duplicated data is:',dups)
columnStatistics = pd.DataFrame(dataset.max(axis=0))
columnStatistics.columns = ['MaxValues']

columnStatistics['MinValues'] = dataset.min(axis=0)
uniqueCounts = pd.DataFrame(columnStatistics.index)
uniqueCounts.set_index(0, inplace=True)
uniqueCounts['UniqueValues'] = np.nan
for col in dataset:
   uniqueCounts.loc[col]['UniqueValues'] = dataset[col].nunique()
columnStatistics['UniqueValues'] = uniqueCounts['UniqueValues']

columnStatistics
# likert scale from a scale of 1 to 5, no zero min val
for i in dataset.columns:
   num_missing = (dataset[[i]].isnull()).sum()
   perc = num_missing/dataset.shape[0]*100
   print('> %s, Missing: %d (%.1f%%)' % (i,num_missing,perc))

# inpute missing values
bool_series = pd.isnull(dataset["weight_gain"])
```

```
dataset[bool_series]
dataset = dataset.drop(labels='Korean Jindo Dog')
dataset.loc[dataset.index == 'Korean Jindo Dog']

index_list = dataset.index.tolist()
prey_drive = dataset['prey_drive'].value_counts().sort_index()
fig, ax = plt.subplots(figsize=(12, 4))
ax = prey_drive.plot(kind='bar',width=1.0)
ax.set(xlabel = "Rating of Prey Drive",
    ylabel = "Fequency")
plt.show()
print('The median of prey_drive rating is:',dataset["prey_drive"].median())
print('The mean of prey_drive rating is:',round(dataset["prey_drive"].mean(),0))

# Replace the missing values with mean/median
# Since it is an ordinal data, a median is more suitable
dataset["prey_drive"].fillna(dataset['prey_drive'].mean(), inplace = True)
dataset.info()
# format fields
dataset = dataset.astype({header[4]: 'int64', header[6]: 'int64', header[7]: 'int64', header[8]:
'int64',
            header[11]: 'int64', header[12]: 'int64', header[13]: 'int64', header[14]: 'int64',
            header[15]: 'int64', header[19]: 'int64', header[33]: 'int64', header[34]: float,
            header[36]: float})
dataset.info()
weight_max = dataset['max_weight']
weight_min = dataset['min_weight']
weight_max.plot.line(figsize=(12,4))
weight_min.plot.line()
plt.ylabel("weight")
plt.xlabel("dogs")
plt.xticks(rotation=90)
plt.show()

lifespan_max = dataset['max_lifespan']
lifespan_min = dataset['min_lifespan']
lifespan_max.plot.line(figsize=(12,4))
lifespan_min.plot.line()
plt.ylabel("lifespan")
plt.xlabel("dogs")
plt.xticks(rotation=90)
plt.show()
# no interesting analysis can be made from the maximum and minimum besides the range
# if (max-min)/2 != mean of the weight

dataset = dataset.drop(['max_weight','min_weight'], axis = 1)
dataset = dataset.drop(['max_lifespan','min_lifespan'], axis = 1)
dataset.info()
```

```
corr = dataset.corr()
sns.set(font_scale=4)
mask = np.triu(np.ones_like(corr))
plt.figure(figsize=(100, 80))
sns.heatmap(corr, annot=True, cmap='RdBu', vmin=-1, mask=mask)
sns.set(font_scale=1)
dataset

bubble_df = pd.DataFrame({'count' : dataset.groupby(['energy',
'exercise']).size()}).reset_index()
plt.figure(figsize=(12, 8))
plt.scatter(x=bubble_df['exercise'].values, y=bubble_df['energy'].values,
            alpha=0.5,
            s = bubble_df['count'].values **2)
plt.xlabel('Likes to exercise', size=14)
plt.ylabel('Energy Level', size=14)
dataset = dataset.drop(columns='exercise')
dataset = dataset.drop(columns='cold_weather')
dataset.info()
bubble_df = pd.DataFrame({'count' : dataset.groupby(['dog_friendly',
'friendly_strangers']).size()}).reset_index()
plt.figure(figsize=(12, 8))
plt.scatter(x=bubble_df['dog_friendly'].values, y=bubble_df['friendly_strangers'].values,
            alpha=0.5,
            s = bubble_df['count'].values **2)
plt.xlabel('Friendly to other dogs', size=14)
plt.ylabel('Friendly to strangers', size=14)
dataset = dataset.drop(columns='friendly_strangers')
bubble_df = pd.DataFrame({'count' : dataset.groupby(['playful',
'intelligence']).size()}).reset_index()
plt.figure(figsize=(12, 8))
plt.scatter(x=bubble_df['playful'].values, y=bubble_df['intelligence'].values,
            alpha=0.5,
            s = bubble_df['count'].values **2)
plt.xlabel('Playful', size=14)
plt.ylabel('Intelligent', size=14)
bubble_df = pd.DataFrame({'count' : dataset.groupby(['energy',
'intelligence']).size()}).reset_index()
x = np.array(bubble_df['energy'].values)
y = np.array(bubble_df['intelligence'].values)
plt.figure(figsize=(12, 8))

plt.scatter(x, y, alpha=0.5,s = bubble_df['count'].values **2)
plt.xlabel('Energy', size=14)
plt.ylabel('Intelligent', size=14)
dataset = dataset.drop(['breed_group'], axis = 1)
x = StandardScaler().fit_transform(dataset)
# standardization
```

```
x = pd.DataFrame(x, columns =dataset.columns.values)
print(x.shape)
x.round(2).head()

pca_fitting_dataset = x.copy()
pca_fitting_dataset
print(len(dataset.columns))
pca = PCA(n_components= len(dataset.columns)) #covariance matrix
pca.fit(x)
print(x.shape)

percent_variance = np.round(pca.explained_variance_ratio_* 100, decimals =2)
columns = ['PC1', 'PC2', 'PC3', 'PC4','PC5', 'PC6', 'PC7', 'PC8','PC9', 'PC10', 'PC11',
'PC12','PC13', 'PC14', 'PC15',
        'PC16','PC17', 'PC18', 'PC19', 'PC20','PC21', 'PC22', 'PC23']
plt.figure(figsize=(12, 8))
plt.bar(x= range(1,24), height=percent_variance, tick_label=columns)
plt.ylabel('Percentate of Variance Explained')
plt.xlabel('Principal Component')
plt.xticks(rotation=90)
plt.title('PCA Scree Plot')
plt.show()
print(percent_variance)

# scree plot
plt.figure(figsize=(12, 8))
plt.xlabel('# of Features')
plt.ylabel('Eigenvalues')
plt.title('PCA Eigenvalues')
plt.ylim(0,max(pca.explained_variance_))
plt.style.context('seaborn-whitegrid')
plt.axhline(y=1, color='r', linestyle='--')
plt.plot(pca.explained_variance_)
plt.show()
plt.figure(figsize=(12, 8))
plt.title('PCA Variance Explaned')
plt.xlabel('# of PC')
plt.ylabel('% Variance Explained')
plt.axhline(y=70, color='r', linestyle='--')

plt.plot(np.cumsum(np.round(pca.explained_variance_ratio_,decimals=3)*100))
plt.show()
#PCA1 is at 0 in xscale
pca = PCA(n_components= 8) #covariance matrix
pca.fit(x)
x_pca = pca.transform(x)
print(x_pca.shape)
```

```
x_pca_df = pd.DataFrame(x_pca, columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8'])
x_pca_df.head()
# this shows the covariance matrix and it shows that PC2 is positively related to PC1 and so
on
plt.figure(figsize=(16,6))
ax = sns.heatmap(pca.components_[0:8],
            cmap="flare",
            yticklabels=[ "PCA"+str(x) for x in range(1,9)],
            xticklabels=list(dataset.columns))

components = pd.DataFrame(pca.components_, columns=x.columns)
components.rename(index= lambda x:'PC-' + str(x+1), inplace=True)

# Top 3 positive contributors
pd.DataFrame(components.columns.values[np.argsort(-components.values,axis=1)[:,:3]],
        index=components.index, columns=['1st Max', '2nd Max','3rd Max'])

# number of features = 26
# number of samples = 158
df = pd.DataFrame(dataset)
# compute the mean of each feature mean[0] == mean of shedding feature
mean = df.mean()
x_minus_mean = pd.DataFrame(df - mean).to_numpy()

print(x_minus_mean.shape)
# V1/PC1 : (x-mean)*V1/PC1 etc
print(pca.components_.T.shape)
pca.components_.shape

pca.components_.shape


data = go.Scatter3d(
  x = x_pca[:,0],
  y = x_pca[:,1],
  z = x_pca[:,2],
  name = "",
  mode='markers',
  marker=dict(
    size=10,
    opacity=0.8
)

)

dc_1 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,:1])[0][0]],
          y = [0,np.matmul(x_minus_mean,pca.components_.T[:,:1])[1][0]],
          z = [0,np.matmul(x_minus_mean,pca.components_.T[:,:1])[2][0]],
```

```
                marker = dict( size = 1,
                           color ="red"),
                line = dict( color = "red",
                        width = 10),
                name = "PC1"
                )
dc_2 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,1:2])[0][0]],
                y = [0,np.matmul(x_minus_mean,pca.components_.T[:,1:2])[1][0]],
                z = [0,np.matmul(x_minus_mean,pca.components_.T[:,1:2])[2][0]],
                marker = dict( size = 1,
                          color = "rgb(84,48,5)"),
                line = dict( color = "green",
                        width = 10),
                name = "PC2"
                )
dc_3 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,2:3])[0][0]],
                y = [0,np.matmul(x_minus_mean,pca.components_.T[:,2:3])[1][0]],
                z = [0,np.matmul(x_minus_mean,pca.components_.T[:,2:3])[2][0]],
                marker = dict( size = 1,
                          color = "rgb(84,48,5)"),
                line = dict( color = "blue",
                        width = 10),
                name = "PC3"
               )

data = [data,dc_1,dc_2,dc_3]
layout = go.Layout(
   xaxis=dict(
      titlefont=dict(
        family='Courier New, monospace',
        size=18,
        color='#7f7f7f'
      )
   )
)
fig = go.Figure(data=data, layout=layout)
fig.show()

data = go.Scatter3d(
   x=x_pca[:,3],
   y = x_pca[:,4],
   z = x_pca[:,5],
   name = "data",
   mode='markers',
   marker=dict(
      size=10,
      opacity=0.8
)
```

```
)

dc_1 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,4:5])[0][0]],
                y = [0,np.matmul(x_minus_mean,pca.components_.T[:,4:5])[1][0]],
                z = [0,np.matmul(x_minus_mean,pca.components_.T[:,4:5])[2][0]],
                marker = dict( size = 1,
                           color = "rgb(84,48,5)"),
                line = dict( color = "red",
                        width = 10),
                name = "PC4"
                )
dc_2 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,5:6])[0][0]],
                y = [0,np.matmul(x_minus_mean,pca.components_.T[:,5:6])[1][0]],
                z = [0,np.matmul(x_minus_mean,pca.components_.T[:,5:6])[2][0]],
                marker = dict( size = 1,
                          color = "rgb(84,48,5)"),
                line = dict( color = "green",
                        width = 10),
                name = "PC5"
                )
dc_3 = go.Scatter3d( x = [0,np.matmul(x_minus_mean,pca.components_.T[:,6:7])[0][0]],
                y = [0,np.matmul(x_minus_mean,pca.components_.T[:,6:7])[1][0]],
                z = [0,np.matmul(x_minus_mean,pca.components_.T[:,6:7])[2][0]],
                marker = dict( size = 1,
                          color = "rgb(84,48,5)"),
                line = dict( color = "blue",
                        width = 10),
                name = "PC6"
             )

data = [data,dc_1,dc_2,dc_3]
layout = go.Layout(
   xaxis=dict(
      title='PC1',
      titlefont=dict(
        family='Courier New, monospace',
        size=18,
        color='#7f7f7f'
      )
   )
)
fig = go.Figure(data=data, layout=layout)
fig.show()

# reduce the data towards the PCs
model = pca_2(n_components=8)
# Fit transform
```

```
x.columns = pd.RangeIndex(x.columns.size)
x.columns  = x.columns.astype(str)
results = model.fit_transform(x)
# Plot explained variance
#fig, ax = model.plot()
fig, ax = model.scatter(legend=False)
# Make biplot with the number of features
fig, ax = model.biplot(n_feat=23, legend=False)

# Convert X_pca_2 into dataframe
pca_df = pd.DataFrame(x_pca, columns=['PCA1', 'PCA2', 'PCA3', 'PCA4', 'PCA5', 'PCA6',
'PC7', 'PC8'])
pca_df

# dis = distance.cdist(X_pca_2, X_pca_2, 'euclidean')
# print(dis)

# Make plot outputs appear and be stored within the notebook
%matplotlib inline

plt.rcParams["figure.figsize"] = (10,8)

numClusters = range(1,40)
SSE = []
for k in numClusters:
    k_means = cluster.KMeans(n_clusters=k)
    k_means.fit(dataset)
    SSE.append(k_means.inertia_) # Sum of squared distances of samples to their closest
cluster center

plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.plot(numClusters, SSE)

k_means = cluster.KMeans(n_clusters=13, max_iter=50, random_state=1)
k_means.fit(pca_df)
to_cluster_pca_df = pca_df
labels = k_means.labels_
print('labels:', labels)
clusterID = pd.DataFrame(labels, columns=['Cluster ID'])
# to_cluster_pca_df['Cluster ID'] = labels
# sorted_pca_df = to_cluster_pca_df.sort_values(['Cluster ID'], ascending = True)
# sorted_pca_df
clusterID

# Centroids
centroids = k_means.cluster_centers_
centroids_df= pd.DataFrame(centroids,columns=pca_df.columns)
```

```
centroids_df

# Grouping original dataset via label
dataset_cluster = dataset.copy()
dataset_cluster['Cluster ID'] = labels
dataset_cluster.index = index_list
dataset_cluster = dataset_cluster.sort_values(['Cluster ID'], ascending = True)
dataset_cluster.columns

# Create 13 cluster df
c0 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 0]
c1 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 1]
c2 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 2]
c3 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 3]
c4 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 4]
c5 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 5]
c6 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 6]
c7 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 7]
c8 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 8]
c9 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 9]
c10 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 10]
c11 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 11]
c12 = dataset_cluster.loc[dataset_cluster['Cluster ID'] == 12]

# Dropping Label before PCA
c0 = c0.drop(['Cluster ID'], axis = 1)
c1 = c1.drop(['Cluster ID'], axis = 1)
c2 = c2.drop(['Cluster ID'], axis = 1)
c3 = c3.drop(['Cluster ID'], axis = 1)
c4 = c4.drop(['Cluster ID'], axis = 1)
c5 = c5.drop(['Cluster ID'], axis = 1)
c6 = c6.drop(['Cluster ID'], axis = 1)
c7 = c7.drop(['Cluster ID'], axis = 1)
c8 = c8.drop(['Cluster ID'], axis = 1)
c9 = c9.drop(['Cluster ID'], axis = 1)
c10 = c10.drop(['Cluster ID'], axis = 1)
c11 = c11.drop(['Cluster ID'], axis = 1)
c12 = c12.drop(['Cluster ID'], axis = 1)

# Store clusters into an array
clus_arr = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12]

#printing cluster 0 dataframe
eg1=clus_arr[0].iloc[[0]]
print(eg1.values.tolist())
clus_arr[0]

#print cluster 9 data frame
```

```
eg2=clus_arr[9].iloc[[0]]
print(eg2.values.tolist())
clus_arr[9]

print(dataset.columns)
#these are the feautures in the dataframe
# 'shedding', 'groom', 'weight_gain', 'drooling', 'general_health',
#       'size', 'wander', 'intelligence', 'prey_drive', 'mouthiness', 'bark',
#       'train', 'playful', 'energy', 'exercise_intensity', 'novice_owners',
#       'sensitivity', 'hot_weather', 'alone', 'apartment', 'family_affection',
#       'kid_friendly', 'dog_friendly'

list_of_qn=["1. I do not mind a dog that sheds its fur often: ",
            "2. I prefer a dog that is easy to groom: ",
            "3. I do not mind a dog that can gain weight easily: ",
            "4. I do not mind my dog drooling: ",
            "5. A dog that is resistant to illnesses is important: ",
            "6. I prefer a big dog: ",
            "7. I prefer an intelligent dog bred for jobs that require decision: ",
            "8. I prefer a dog that wanders around on it's own: ",
            "9. I prefer a dog that chases and hunts for small prey: ",
            "10. I do not mind a dog that likes chewing on things: ",
            "11. I prefer a dog that barks: ",
            "12. I want a dog that is easy to train: ",
            "13. I prefer a playful dog: ",
            "14. I prefer a dog with high energy and stamina: ",
            "15. I like taking my dog out for walks: ",
            "16. I have little to no experience raising dogs: ",
            "17. My home is generally quiet without loud sounds or distractions: ",
            "18. I would like to bring my dog out and excercise on a hot sunny day: ",
            "19. I prefer a dog that is able to be by itself and not crave attention: ",
            "20. I prefer my dog to be calm indoors and polite with strangers: ",
            "21. I prefer a dog that is affectionate with my family members: ",
            "22. The dog has to be friendly with small kids & children: ",
            "23. I would like my dog to be friendly and not dominate other dogs: "
          ]

#check if user input is valid
def input_is_valid(user_input):
  try:
    test = int(user_input)
    if test > 0 and test <= 5:
      return True
    else:
      print("Input not valid, enter input again.")
      return False
  except:
    print("Input not valid, enter input again.")
```

```
        return False

print("Hi, I will be recommending 5 dog breeds to you today.")
print("For each question, enter a number from 1 to 5.")
print("1 = Strongly disagree. 5 = Strongly agree.")

list_of_user_input=[]

for n in range(len(list_of_qn)):
    next_qns = False
    while next_qns == False:
        user_in = input(list_of_qn[n])
        user_in = int(user_in)
        if input_is_valid(user_in):
            list_of_user_input.append(int(user_in))
            next_qns = True

print("Inputs:", list_of_user_input)

user_input = np.array(list_of_user_input).reshape(-1, 1)
user_input = preprocessing.scale(user_input)
user_input = user_input.reshape(1, -1)

# Fit and transform user inputs
pca = PCA(n_components= 8)
pca.fit(pca_fitting_dataset)
user_pca = pca.transform(user_input)

k_means.fit(pca_df)
ans = k_means.predict(user_pca)
ans = ans[0]
chosen_cluster = clus_arr[ans]

newDF = pd.DataFrame() #creates a new dataframe that's empty
newDF = newDF.append(chosen_cluster, ignore_index = False) #chosen cluster is Cluster
recommendation_list = [] #to print the results from cosine formula

for i,row in newDF.iterrows():
    #gets the result from the dataframe without header and index
    #loc is used to return the result from each row of the dataset using label
    #row.name is to get the label of each row from the dataset
    i_Values = newDF.loc[row.name].values
    #returns dot product of two arraws - list_of_ans and
    dot_Result = dot(list_of_user_input,i_Values)
    cos_sim = dot_Result / (norm(list_of_user_input)*norm(i_Values))
    result = (row.name,cos_sim)
    recommendation_list.append(result)
```

```
recommendation_list.sort(key=lambda x:x[1])
print(recommendation_list[:5])

labels = ['Breed', 'Cosine Similarity']
result_df = DataFrame(recommendation_list[:5],columns=['Breed', 'Cosine Similarity'])
print("According to your desired dog features, Top 5 Breeds to consider are as follows:
\n",result_df['Breed'].to_string(index=False))
```

References

[1] *Pet ownership in Asia*. Rakuten Insight. (2021, February 27). https://insight.rakuten.com/pet-ownership-in-asia/.

[2] Khoo, H. (2021, April 10). *More people in Singapore interested in adopting or fostering pets during Covid-19 pandemic*. The Straits Times. https://www.straitstimes.com/lifestyle/home-design/more-interested-in-adopting-or-fostering-pets-during-covid-19-pandemic-as-they.