King Saud University
College of Computer and Information Sciences
Department of Information Technology

IT326- Project **First Semester 1445-1446**

Final Report

	NAME	ID
GROUP#	*Aroub Alsalihi*	*444200560*
	Sarah Aljablai	444201613
	Danah Binsaeed	444201175
	Sarah alruwaitea	444200758
	Atheer bin Badie	444200894

TABLE OF CONTENTS

Pet adoption

1.PROBLEM

There are many pets in the world that want to pick up from the shelters. but not all are equally likely to find homes. Some pets are quickly adopted, while others remain in shelters for extended periods. This brings up an important question: **what factors influence the likelihood of pet adoption in shelters?** Understanding these factors can help shelters improve adoption rates, prioritize resources, and give more pets a chance at finding a loving home.

2.DATA MINING TASK

In this project we applied Data mining task to predict the likelihood of adoption for a given pet using classification and clustering
For classification, the model will predict a pet's likelihood of adoption using features like type, breed, age, size, and health. The target attribute will be "AdoptionLikelihood."
For clustering, pets will be grouped based on similar characteristics, excluding adoption likelihood. This will reveal patterns and traits that influence adoption trends and provide insights to improve rates.

3.DATA

The source : https://www.kaggle.com/datasets/rabieelkharoua/predict-pet-adoption-status-dataset
Number of objects : 2007
Number of attributes: 13
Class Label : AdoptionLikelihood

Attributes description

- ***Attributes Type :***
- PetID: Nominal , Unique identifier for each pet.
- PetType: Nominal , Type of pet (e.g., Dog, Cat, Bird, Rabbit).
- Breed: Nominal , Specific breed of the pet.
- AgeMonths: Numeric (Ratio) , Age of the pet in months.
- Color: Nominal , Color of the pet.
- Size: Ordinal , Size category of the pet (Small, Medium, Large).
- WeightKg: Numeric (Ratio) , Weight of the pet in kilograms.
- Vaccinated: Binary , Vaccination status of the pet (0 - Not vaccinated, 1 - Vaccinated).
- HealthCondition: Binary , Health condition of the pet (0 - Healthy, 1 - Medical condition).
- TimeInShelterDays: Numeric (Ratio) , Duration the pet has been in the shelter (days).
- AdoptionFee: Numeric (Ratio) , Adoption fee charged for the pet (in dollars).
- PreviousOwner: Binary , Whether the pet had a previous owner (0 - No, 1 - Yes).
- AdoptionLikelihood (class labels): Binary , Likelihood of the pet being adopted (0 - Unlikely, 1 - Likely).

df1.info()

```
RangeIndex: 2007 entries, 0 to 2006
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   PetID               2007 non-null   int64
 1   PetType             2007 non-null   object
 2   Breed               2007 non-null   object
 3   AgeMonths           2007 non-null   int64
 4   Color               2007 non-null   object
 5   Size                2007 non-null   object
 6   WeightKg            2007 non-null   float64
 7   Vaccinated          2007 non-null   int64
 8   HealthCondition     2007 non-null   int64
 9   TimeInShelterDays   2007 non-null   int64
 10  AdoptionFee         2007 non-null   int64
 11  PreviousOwner       2007 non-null   int64
 12  AdoptionLikelihood  2007 non-null   int64
dtypes: float64(1), int64(8), object(4)
```

- **Missing values**

```
print ("Missing value")
print(df1.isna().sum())
Missing value
PetID                 0
PetType               0
Breed                 0
AgeMonths             0
Color                 0
Size                  0
WeightKg              0
Vaccinated            0
HealthCondition       0
TimeInShelterDays     0
AdoptionFee           0
PreviousOwner         0
AdoptionLikelihood    0
dtype: int64
```

No missing values

- **statistical measures for numeric attributes as five number summary**

Using ***.describe() function*** in python provides a quick overview of the dataset, including measures of central tendency, dispersion, and count of data points.

General Observations:

- **Count**: Each column has 2007 entries, meaning the dataset has no missing values.

Pet adoption

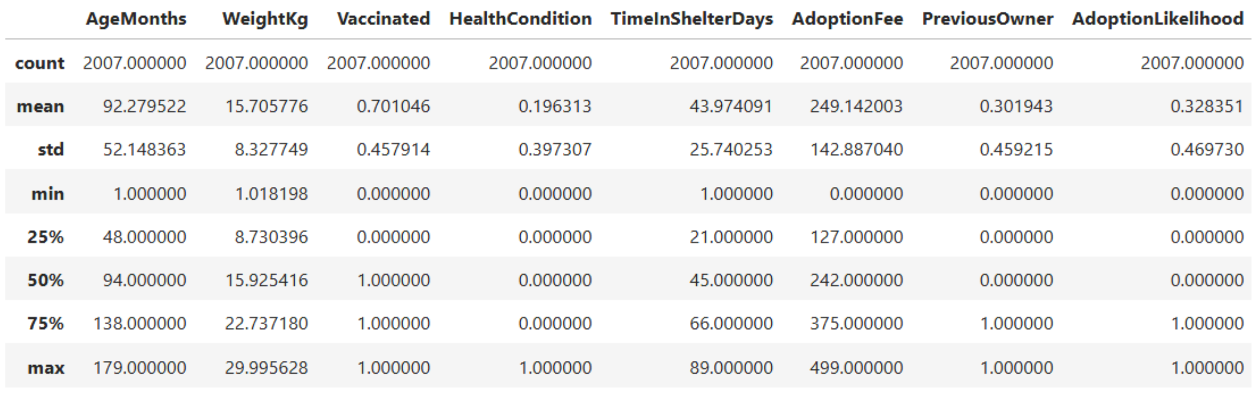
- **Averages (Mean)**:
    - Animals are, on average, 92.28 months old (approximately 7.7 years).
    - Their average weight is 15.71 kg.
    - 70% of the animals are vaccinated.
    - Only 19.6% of the animals have reported health issues.
    - Animals stay in the shelter for an average of 44 days.
    - The typical adoption fee is $249.
    - About 30.2% of animals had a previous owner.
    - The average likelihood of adoption is 32.8%.

- **Standard Deviation**:
    - Age varies widely, with a standard deviation of 52.15 months.
    - Weight also shows considerable variation, with a standard deviation of 8.33 kg.
    - Adoption fees have a wide range, with a standard deviation of $142.89.
    - Shelter time varies by around 25.74 days, suggesting some animals stay much longer than others.
- **Minimum and Maximum Values**:
    - Ages range from 1 month to 179 months (almost 15 years).
    - Weights range from 1.02 kg to nearly 30 kg.
    - Adoption fees range from $0 to $499.
    - Shelter time spans from 1 day to a maximum of 89 days.

- **Percentiles**:
    - **Age**:
        - 25% of animals are younger than 48 months (4 years).
        - 50% (median) are younger than 94 months (7.8 years).
        - 75% are younger than 138 months (11.5 years).
    - **Weight**:
        - 25% weigh less than 8.73 kg.
        - 50% weigh less than 15.93 kg.
        - 75% weigh less than 22.74 kg.
    - **Shelter Time**:
        - Half of the animals stay 45 days or less.
        - 75% stay 66 days or less, but a few stay much longer.
- **Binary Columns**:
    - 70% of animals are vaccinated.
    - Only 19.6% have health conditions.
    - 30% had a previous owner.
    - 33% are likely to be adopted.

In summary, most animals are middle-aged, medium-weight, healthy, and stay in the shelter for a little over a month. Many are vaccinated, and about a third have a higher chance of being adopted.

df2.describe()



	AgeMonths	WeightKg	Vaccinated	HealthCondition	TimeInShelterDays	AdoptionFee	PreviousOwner	AdoptionLikelihood
count	2007.000000	2007.000000	2007.000000	2007.000000	2007.000000	2007.000000	2007.000000	2007.000000
mean	92.279522	15.705776	0.701046	0.196313	43.974091	249.142003	0.301943	0.328351
std	52.148363	8.327749	0.457914	0.397307	25.740253	142.887040	0.459215	0.469730
min	1.000000	1.018198	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	48.000000	8.730396	0.000000	0.000000	21.000000	127.000000	0.000000	0.000000
50%	94.000000	15.925416	1.000000	0.000000	45.000000	242.000000	0.000000	0.000000
75%	138.000000	22.737180	1.000000	0.000000	66.000000	375.000000	1.000000	1.000000
max	179.000000	29.995628	1.000000	1.000000	89.000000	499.000000	1.000000	1.000000

- **Variance**

var() measures how spread out the values in a dataset are, and it is a common statistic used to analyze data variability , as the variance increase its indicate that the values are more spread out far from the mean and lot of variation. , where the decrease variance indicates that the values are close to the mean and suggests uniformity or little variation.

1. **Age Variance (2719.45)**:
   - The wide age range shows a diverse population of animals, from very young to senior.
2. **Weight Variance (69.35)**:
   - The significant variance in weight highlights a variety of animal sizes, from small breeds to large ones.
3. **Vaccinated Variance (0.2097)**:
   - The low variance indicates most animals have a similar vaccination status, with a majority being vaccinated.
4. **Health Condition Variance (0.1579)**:
   - There is some variation in health conditions, though most animals are likely healthy.
5. **Time in Shelter Variance (662.56)**:
   - A high variance suggests some animals are adopted quickly, while others remain in the shelter for much longer periods.
6. **Adoption Fee Variance (20416.70)**:
   - The large spread in fees reflects varying costs, possibly influenced by the type of animal, its characteristics, or shelter policies.
7. **Previous Owner Variance (0.2109)**:
   - The variation shows a mix of animals with and without prior ownership, pointing to different backgrounds.

8. **Adoption Likelihood Variance (0.2286)**:
   - The moderate variance suggests differences in how likely animals are to be adopted, likely influenced by factors such as age, size, and health.

Overall Observations:

- The dataset highlights a wide variety of animals based on their age, size, and histories.
- Most animals have consistent vaccination statuses, which is encouraging for potential adopters.
- The length of shelter stays and adoption fees show significant variation, indicating that while some animals find homes quickly, others take longer to be adopted.

```
var_data = df ["AgeMonths"].var ()
print ("AgeMonths veriance = ",var_data)
```

```
AgeMonths veriance =  2719.4517389535395
```

```
var_data = df ["WeightKg"].var ()
print ("WeightKg veriance = ",var_data)
```

```
WeightKg veriance =  69.35140643666254
```

```
var_data = df ["Vaccinated"].var ()
```

Pet adoption

```
print ("Vaccinatedvar veriancer =",  var_data)
```

```
Vaccinatedvar veriancer = 0.20968484680487
```

```
var_data = df ["HealthCondition"].var ()
print ("HealthCondition veriance= " ,var_data)
```

```
HealthCondition veriance=  0.15785279934982935
```

```
var_data = df ["TimeInShelterDays"].var ()
print ("TimeInShelterDays veriance=",var_data)
```

```
TimeInShelterDays veriance= 662.5606444244752
```

```
var_data = df ["AdoptionFee"].var ()
print ("AdoptionFee veriance = ",var_data)
```

```
AdoptionFee veriance =  20416.706146135595
```

```
var_data = df ["PreviousOwner"].var ()
print ("PreviousOwner veriance = ",var_data)
```

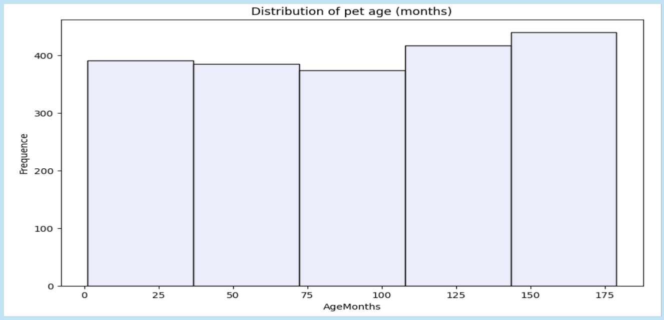
```
PreviousOwner veriance =  0.21087857503722912
```

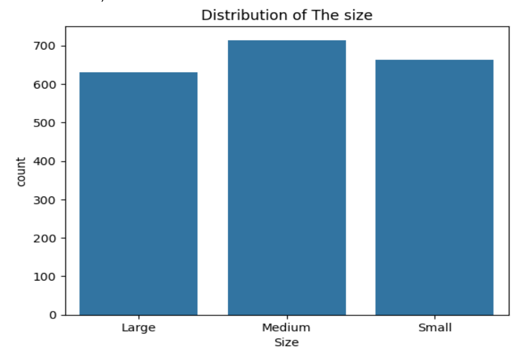
```
var_data = df ["AdoptionLikelihood"].var ()
print ("AdoptionLikelihood veriance = ",var_data)
```

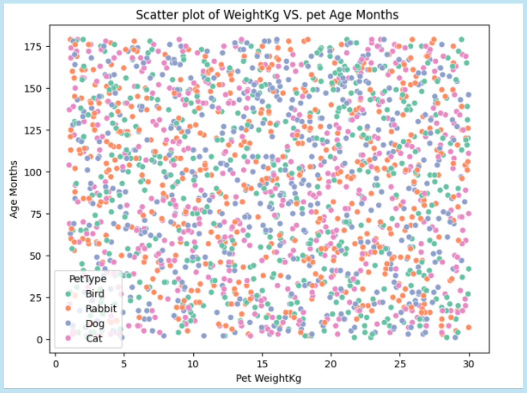
```
AdoptionLikelihood veriance =  0.22064648108489274
```

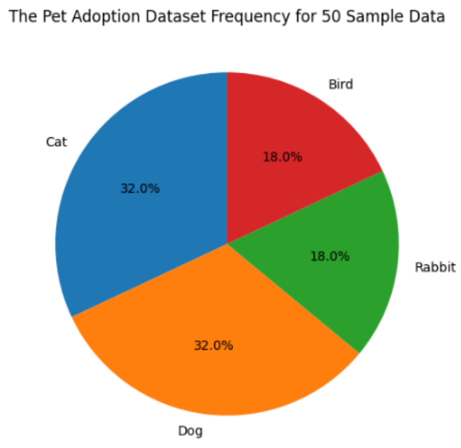
Pet adoption

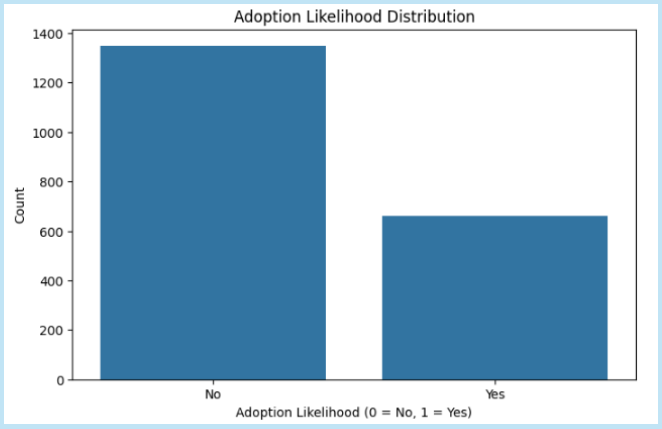
Graph Name	Graph Picture	Graph description

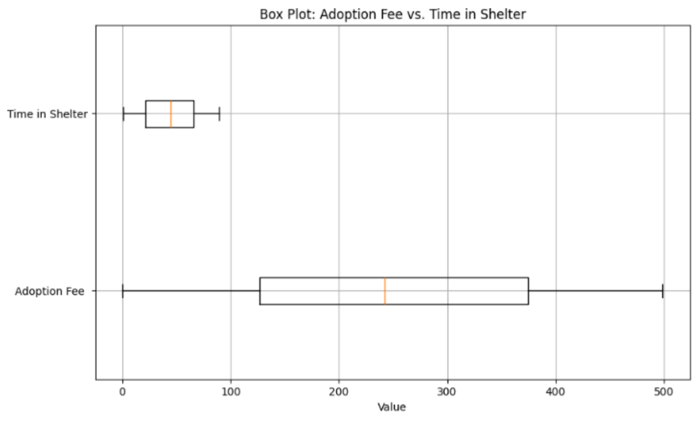
Histogram of AgeMonths	`#Histogram of AgeMonths`
	`plt.figure(figsize=(10,6))`
	`sns.histplot(df1['AgeMonths'],bins=5,edgecolor='black',color='lavender')`
	`plt.title('Distribution of pet age (months)')`
	`plt.xlabel('AgeMonths')`
	`plt.ylabel('Frequence')`
	`plt.show()`
	
		The histogram shows that the dataset contains pets of all age groups, with a relatively even distribution from very young to senior animals. This even spread suggests the dataset captures a diverse range of animals in terms of age. However, due to the wide range of values (0 to 175 months), preprocessing steps like normalization or age binning may be required to simplify analysis or ensure consistency in machine learning models.

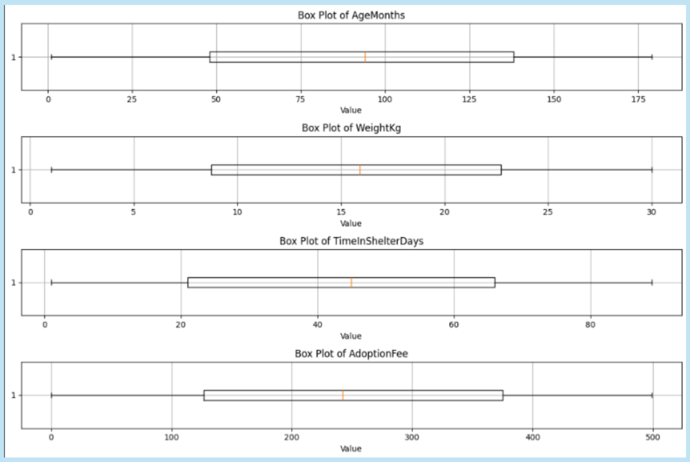
Bar plot for petType	`# Bar chart`
	`sns.countplot(x='Size', data=df)`
	`plt.title('Distribution of The size')`
	`plt.show()`
	
		The dataset provides a balanced representation of all size categories, with Medium-sized animals being slightly more common. This balance indicates that size is unlikely to introduce bias into analysis but may still need encoding during preprocessing for machine learning purposes.

Scatter plot of WeightKg VS. pet Age Months	`# SCATTER PLOT:`
	`plt.figure(figsize=(8,6))`
	`sns.scatterplot(x= 'WeightKg' , y='AgeMonths' , data = df  , hue='PetType' , palette='Set2' )`
	`plt.title('Scatter plot of WeightKg VS. pet Age Months ')`
	`plt.xlabel('Pet WeightKg')`
	`plt.ylabel('Age Months')`
	`plt.show()`
	
		The scatter plot highlights the diversity in pet types, weights, and ages. While there is no strong correlation between weight and age, the pet type appears to influence these attributes. Further preprocessing, such as encoding and scaling, will be necessary to make this data suitable for deeper analysis or modeling.

pie chart of random sample petType	
	```
	# Take a random sample of 50 rows
	data_sample = df.sample(n=50, random_state=1)

	# Calculate the frequency of each Pet Type in the sample
	PetType_frequency = data_sample['PetType'].value_counts(normalize=True) * 100

	# Plot the pie chart
	plt.figure(figsize=(6, 6))
	PetType_frequency.plot.pie(autopct='%1.1f%%', startangle=90)
	plt.title('The Pet Adoption Dataset Frequency for 50 Sample Data')
	plt.ylabel('')
	plt.show()
	```
	 
		This distribution suggests that cats and dogs might have a larger presence in the overall dataset, which could reflect their popularity as pets. To prepare the dataset for analysis or modeling, we will use encoding techniques, such as one-hot encoding, to convert the "PetType" categorical variable into a numerical format. This will ensure the pet types are properly represented and can be utilized in machine learning models.

Bar plot for Adoption Likelihood (classlabel)	
	```
	# Bar plot for Adoption Likelihood
	adoption_counts = df1['AdoptionLikelihood'].value_counts()
	plt.figure(figsize=(8, 5))
	sns.barplot(x=adoption_counts.index, y=adoption_counts.values)
	plt.title('Adoption Likelihood Distribution')
	plt.xlabel('Adoption Likelihood (0 = No, 1 = Yes)')
	plt.ylabel('Count')
	plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
	plt.show()
	```
	 
		The bar plot highlights that most pets fall into the "unlikely to be adopted" category, while fewer are "likely to be adopted." This imbalance should be carefully considered during analysis and model training. Using techniques such as adjusting class weights or focusing on evaluation metrics like precision and recall can help ensure accurate predictions without needing to balance the data.

box plots for 'AdoptionFee' and 'TimeInShelterDays'	`#  box plots for 'AdoptionFee' and 'TimeInShelterDays'`
	`plt.figure(figsize=(10, 6))`
	`plt.boxplot([df['AdoptionFee'], df['TimeInShelterDays']], vert=False, labels=['Adoption Fee ', 'Time in Shelter'])`
	`plt.title('Box Plot: Adoption Fee vs. Time in Shelter')`
	`plt.xlabel('Value')`
	`plt.grid(True)`
	`plt.show()`
	
		The box plot shows that adoption fees have a wide range, while shelter times are more consistent. Both attributes are clean and lack significant outliers, but they may require scaling for modeling purposes. Additionally, adoption fees seem to have more variability, potentially playing a more significant role in influencing adoption outcomes

Box plots for every numeric columns	`# Filter out the 'PetID' column along with binary columns`
	`non_binary_columns_without_id = [`
	`    col for col in numeric_columns`
	`    if df[col].nunique() > 2 and col != 'PetID']`
	`# individual box plots for non-binary numeric columns excluding 'PetID'`
	`plt.figure(figsize=(12, len(non_binary_columns_without_id) * 2))`
	
	`for i, column in enumerate(non_binary_columns_without_id, 1):`
	`    plt.subplot(len(non_binary_columns_without_id), 1, i)`
	`    plt.boxplot(df[column], vert=False)`
	`    plt.title(f'Box Plot of {column}')`
	`    plt.xlabel('Value')`
	`    plt.grid(True)`
	`plt.tight_layout()`
	`plt.show()`
	
		The box plots reveal that most features (**AgeMonths**, **WeightKg**, **TimeInShelterDays**, and **AdoptionFee**) are clean and have no significant outliers, as there are no points beyond the whiskers. These distributions are suitable for further analysis, though scaling or normalization may be needed due to differing ranges.

4.DATA PREPROCCESSING

We chose to apply data preprocessing because it is a crucial step to ensure the quality and reliability of the mining outcome. We know that our Raw data might contains inconsistencies, missing values, or noise that needed to apply data preprocessing, and to improve the overall data quality

DATA CLEANING

1-Missing values

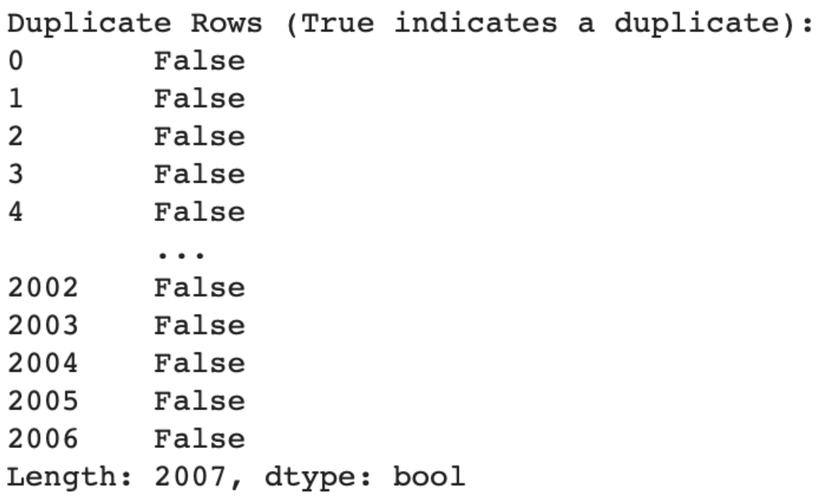
```
print ("Missing value")

print(df1.isna().sum())
```

```
Missing value
PetID              0
PetType            0
Breed              0
AgeMonths          0
Color              0
Size               0
WeightKg           0
Vaccinated         0
HealthCondition    0
TimeInShelterDays  0
AdoptionFee        0
PreviousOwner      0
AdoptionLikelihood 0
dtype: int64
```

Description: Missing values are a prevalent issue in datasets that can occur for many reasons, These missing values, if not handled properly, can lead to biased analysis. hence it’s important to check if we had any to handle them. But we were lucky for not having any missing value

2-duplicate rows:

```
duplicates = df1.duplicated()
print("Duplicate Rows (True indicates a duplicate):")
print(duplicates)
```



```
Duplicate Rows (True indicates a duplicate):
0       False
1       False
2       False
3       False
4       False
        ...
2002    False
2003    False
2004    False
2005    False
2006    False
Length: 2007, dtype: bool
```

Pet adoption

Description : Duplicate rows in a dataset occur when two or more rows contain identical values across all columns. Thus to ensure that no redundancy that will impact the efficiency. but for this one we were lucky for not having any duplicate to handle.

3-Outliers

```
age_array = df1['AgeMonths'].to_numpy()
z_score = zscore(age_array) t
hreshold = 2
outliers = [age_array[i] for i, z in enumerate(z_score) if abs(z) > threshold]
print("Outliers using Z-Score in AgeMonths column:")
print(outliers)
#Calculate Z-scores for a AdoptionFee column
AdoptionFee_array = df1['AdoptionFee'].to_numpy()
 z_score = zscore(AdoptionFee_array)
 threshold = 2
outliers = [AdoptionFee_array[i] for i, z in enumerate(z_score) if abs(z) > threshold] print("Outliers using Z-Score in AdoptionFee column:")
print(outliers)
# Calculate Z-scores for a TimeInShelterDays column
TimeInShelterDays_array=df1['TimeInShelterDays'].to_numpy()
z_score = zscore(TimeInShelterDays_array)
threshold = 2
outliers = [TimeInShelterDays_array[i] for i, z in enumerate(z_score) if abs(z) > threshold] print("Outliers using Z-Score in TimeInShelterDays column:")
print(outliers)                                                          # Calculate Z-scores for a WeightKg column
WeightKg_array = df1['WeightKg'].to_numpy() z_score = zscore(WeightKg_array)
threshold = 2
outliers = [WeightKg_array[i] for i, z in enumerate(z_score) if abs(z) > threshold]
print("Outliers using Z-Score in WeightKg column:")
print(outliers)
```

```
Outliers using Z-Score in AgeMonths column:
[]
Outliers using Z-Score in AdoptionFee column:
[]
Outliers using Z-Score in TimeInShelterDays column:
[]
Outliers using Z-Score in WeightKg column:
[]
```

Description: Outliers are data points that deviate significantly from the dataset, that either offering insights or distorting analysis and model performance. But as we can see there are no outlier to apply any techniques to handle it

Now after ensuring there are no data needed to clean we will move on to the transformation

DATA TRANSFORMATION

Data transformation involves converting raw data into a more suitable format or structure for analysis or modeling. This step is essential for improving Data uniformity, and enhanced clarity.

1-Normalization:

```
column_to_normilaize=[ 'WeightKg','AdoptionFee' ]

data_to_normalize = df1[column_to_normilaize]

minmax_scaler = MinMaxScaler()

normalized_data_minmax = minmax_scaler.fit_transform(data_to_normalize) df1[column_to_normilaize] = normalized_data_minmax
```

Pet adoption

```
print("Min-Max scaled data")

print(df1)
```

```
Min-Max scaled data
     PetID PetType            Breed  AgeMonths   Color    Size  WeightKg  \
0      500    Bird         Parakeet        131  Orange   Large  0.138783
1      501  Rabbit           Rabbit         73   White   Large  0.520009
2      502     Dog  Golden Retriever       136  Orange  Medium  0.036514
3      503    Bird         Parakeet         97   White   Small  0.080105
4      504  Rabbit           Rabbit        123    Gray   Large  0.672244
...    ...     ...              ...        ...     ...     ...       ...
2002  2502     Dog           Poodle         72  Orange   Small  0.897969
2003  2503  Rabbit           Rabbit        124   Brown   Small  0.127988
2004  2504  Rabbit           Rabbit        113  Orange   Small  0.025551
2005  2505     Dog         Labrador         12    Gray   Large  0.688239
2006  2506  Rabbit           Rabbit        126   White  Medium  0.603973

     Vaccinated  HealthCondition  TimeInShelterDays  AdoptionFee  \
0             1                0                 27     0.280561
1             0                0                  8     0.470942
2             0                0                 85     0.771543
3             0                0                 61     0.434870
4             0                0                 28     0.028056
...         ...              ...                ...          ...
2002          1                0                 66     0.052104
2003          1                1                 59     0.300601
2004          1                0                 68     0.605210
2005          1                0                 59     0.957916
2006          1                0                 10     0.535070

     PreviousOwner  AdoptionLikelihood
0                0                   0
1                0                   0
2                0                   0
3                1                   0
4                1                   0
...            ...                 ...
2002             1                   1
2003             0                   0
2004             0                   0
2005             0                   0
2006             1                   0

[2007 rows x 13 columns]
```

Description: Normalization is a data preprocessing technique used to scale numerical features to a standard range. By normalizing **WeightKg** and **AdoptionFee**, we ensure that different features can be compared by making it fall in the range[0,1] using MinMaxScaler method

2-Discretization

```
column_to_discretize = ['AgeMonths', 'TimeInShelterDays']
age_bins = [0, 6, 12, 24, 60, 120, 180]
age_labels = ['1-6 months', '7-12 months', '13-24 months', '25-60 months', '61-120 months', '121-180 months']
df1['AgeMonths'] = pd.cut(df1['AgeMonths'], bins=age_bins, labels=age_labels, right=True)
# For TimeInShelterDays
shelter_bins = [0, 10, 30, 60, 90]
```

Pet adoption

```
shelter_labels = ['Very Short Stay', 'Short Stay', 'Medium Stay', 'Long Stay']
df1['TimeInShelterDays'] = pd.cut(df1['TimeInShelterDays'], bins=shelter_bins, labels=shelter_labels, right=False)
 # Display the updated DataFrame
 print(df1[['AgeMonths', 'TimeInShelterDays']])
       AgeMonths TimeInShelterDays
0    121-180 months       Short Stay
1     61-120 months   Very Short Stay
2    121-180 months        Long Stay
3     61-120 months       Long Stay
4    121-180 months       Short Stay
...          ...             ...
2002   61-120 months        Long Stay
2003  121-180 months      Medium Stay
2004   61-120 months        Long Stay
2005     7-12 months      Medium Stay
2006  121-180 months       Short Stay

[2007 rows x 2 columns]
```

Description:

Discretization is the process of converting continuous numerical data into discrete categories or intervals (bins). discretizing this variable helps to categorize pets into **age** and **TimeInShelterDays** groups and shelter duration ranges. This is beneficial for improving model performance by reducing the complexity of inputs, which can help algorithms find patterns more efficiently.

3-Encoding

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from scipy import stats
 le = LabelEncoder()
df1['PetType']=le.fit_transform(df1['PetType'])
 df1['Size']=le.fit_transform(df1['Size'])
df1['Breed']=le.fit_transform(df1['Breed']) print(df1)

    PetID  PetType  Breed     AgeMonths   Color  Size  WeightKg  \
0     500       0     2  121-180 months  Orange     0  0.138783
1     501       3     5   61-120 months   White     0  0.520009
2     502       2     0  121-180 months  Orange     1  0.036514
3     503       0     2   61-120 months   White     2  0.080105
4     504       3     5  121-180 months    Gray     0  0.672244
...   ...     ...   ...             ...     ...   ...       ...
2002  2502       2     4   61-120 months  Orange     2  0.897969
2003  2503       3     5  121-180 months   Brown     2  0.127988
2004  2504       3     5   61-120 months  Orange     2  0.025551
2005  2505       2     1     7-12 months    Gray     0  0.688239
2006  2506       3     5  121-180 months   White     1  0.603973

     Vaccinated  HealthCondition TimeInShelterDays  AdoptionFee  \
0             1                0        Short Stay     0.280561
1             0                0   Very Short Stay     0.470942
2             0                0         Long Stay     0.771543
3             0                0         Long Stay     0.434870
4             0                0        Short Stay     0.028056
...         ...              ...               ...          ...
2002          1                0         Long Stay     0.052104
2003          1                1       Medium Stay     0.300601
2004          1                0         Long Stay     0.605210
```

```
2005       1         0     Medium Stay    0.957916
2006       1         0      Short Stay    0.535070

     PreviousOwner  AdoptionLikelihood
0             0             0
1             0             0
2             0             0
3             1             0
4             1             0
...         ...           ...
2002          1             1
2003          0             0
2004          0             0
2005          0             0
2006          1             0

[2007 rows x 13 columns]
```

Description:Encoding is a data transformation technique used to convert categorical data into a numerical format, it ensures that categorical data is both compatible with models and retains its meaningfulness in numerical analysis.
PetType: types like "Dog," "Cat," etc. Encoding allows the model to understand these categories as distinct groups.

Size: Similarly, sizes like "Small," "Medium," and "Large" are converted into numerical values, enabling the model to analyze their impact on adoption likelihood.

Breed: With many different breeds, encoding helps represent this information numerically.

Without encoding, these categorical variables would be ignored or improperly handled, severely limiting the model's ability to learn from the data and produce reliable results.

FEATURE SELECTION

1-Filter FS Method using univarince metho

How It Works: Evaluates features independently using statistical tests.

```
X = df2.drop(columns=['AdoptionLikelihood'])
y = df2['AdoptionLikelihood']
X = pd.get_dummies(X, drop_first=True) #converted to a numerical format using one-hot encoding, which is necessary for the model.

# Split the data into training and testing sets
#70% of the data is used for training , 30% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000, random_state=42) #allows the model more iterations to find the best solution if needed. for random state its the most papular used


num_features_to_select = 4 # select 4 top feature , It fits the model to the training data and starts eliminating less important features.

# Initialize RFE for feature selection
```

```
selector = RFE(estimator=model, n_features_to_select=num_features_to_select)
selector.fit(X_train, y_train)

# Transform the data for selected features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)

# Train the model with selected features
model.fit(X_train_selected, y_train)

# Evaluate accuracy

y_pred = model.predict(X_test_selected)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print selected features and their ranking

print("Selected Features:", selected_features.tolist())
```

Conclusion: filter method successfully identified four important features that significantly contribute to the model's predictive performance, achieving an accuracy of 83%. This method is good for initial feature selection but does not consider feature interactions.

```
Selected Features:
Index(['AgeMonths', 'HealthCondition', 'Breed_Labrador', 'Size_Medium'], d
type='object')
Accuracy of the model with selected features: 0.8275290215588723
```

2-Wrapper method (recursive feature elimination)

How It Works: Trains a model, removes least important features iteratively.

```
X = df2.drop(columns=['AdoptionLikelihood'])
y = df2['AdoptionLikelihood']
X = pd.get_dummies(X, drop_first=True) #converted to a numerical format using
one-hot encoding, which is necessary for the model.

# Split the data into training and testing sets
#70% of the data is used for training , 30% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000, random_state=42) #allows the model more
iterations to find the best solution if needed. for random state its the most
papular used


num_features_to_select = 4 # select 4 top feature , It fits the model to the
training data and starts eliminating less important features.

# Initialize RFE for feature selection
selector = RFE(estimator=model, n_features_to_select=num_features_to_select)
```

```
selector.fit(X_train, y_train)

# Transform the data for selected features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)

# Train the model with selected features
model.fit(X_train_selected, y_train)

# Evaluate accuracy

y_pred = model.predict(X_test_selected)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print selected features and their ranking
print("Selected Features:", selected_features.tolist())
```

```
Accuracy: 0.89
Selected Features: ['AgeMonths', 'HealthCondition', 'Breed_Labrador', 'Size_Mediu
m']
```

Conclusion: it select the same selected features as the filter method but achieved a higher accuracy of 89%. This indicates that RFE is effective in identifying the most relevant features while considering their interactions.

3-Embedded FS Method L1 Regularization.

How It Works: Combines feature selection with model training (e.g., Lasso).

```
# Convert categorical variables to numerical to helps the model understand the
data
df2 = pd.get_dummies(df2, drop_first=True)

# Define features and target variable
x = df2.drop(columns=['AdoptionLikelihood'])
y = df2['AdoptionLikelihood']

# Split the data into training and testing sets
#70% of the data is used for training , 30% for testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)

# Standardize the features for equal weight
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Create and fit the Lasso model
model = Lasso(alpha=0.1)
model.fit(x_train_scaled, y_train)
```

```
# identifies which features were selected based on their coefficients
#Features with non-zero coefficients are considered important for the model.
selected_features = x.columns[model.coef_ != 0]
print("Selected Features:", selected_features)

# Make predictions to calculate the accuracy
y_pred = model.predict(x_test_scaled)
y_pred_binary = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_binary)
print("Accuracy with selected features:", accuracy)
```

```
Selected Features: Index(['Vaccinated', 'HealthCondition', 'Breed_Labrador', 'Siz
e_Medium'], dtype='object')
Accuracy with selected features: 0.857379767827529
```

Conclusion: The embedded method selected a slightly different set of features, including Vaccinated instead of AgeMonths. It achieved an accuracy of 86%, demonstrating that L1 regularization can effectively identify important features while maintaining strong predictive performance.

And after trying all these three features selection we ended choosing the Wrapper Method (RFE) since it gave the highest accuracy

CONTINGENCY_TABLE

```
#contingency_table

contingency_table = pd.crosstab(df1['PetType'], df1['Breed'])
print("Contingency Table:")
print(contingency_table)

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)
print("\nChi-Square Statistics:", chi2_stat)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:")
print(expected)

# Set alpha level
alpha = 0.05
print("\nAlpha Level:", alpha)

# Check if we reject the null hypothesis
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant association between PetType
and Breed.")
else:
    print("Fail to reject the null hypothesis: There is no significant association
between PetType and Breed.")
```

```
Contingency Table:
```

Pet adoption

```
Breed      0    1    2    3    4    5    6
PetType
0          0    0  487    0    0    0    0
1          0    0    0  252    0    0  253
2        162  193    0    0  167    0    0
3          0    0    0    0    0  493    0

Chi-Square Statistics: 6021.0
Degrees of Freedom: 18
Expected Frequencies:
[[ 39.30941704  46.83158944 118.17090184  61.14798206  40.52267065
  119.62680618  61.39063279]
 [ 40.76233184  48.56253114 122.53861485  63.40807175  42.0204285
  124.04833084  63.65969108]
 [ 42.13452915  50.19730942 126.66367713  65.5426009   43.43497758
  128.22421525  65.80269058]
 [ 39.79372197  47.40857    119.62680618  61.90134529  41.02192327
  121.10064773  62.14698555]]

Alpha Level: 0.05
Reject the null hypothesis: There is a significant association between Pet
Type and Breed.
```

During our correlation analysis, given the significant Chi-Square statistic, we observed a significant relationship between the variables PetType and Breed. and since every pet have one breed only unless the Dog and the Cat breeds have the approximately the same number. We decided to select only one of these correlated variables to reduce redundancy in our dataset and simplify our model, we will select the more relevant variable for our analysis"PetType".hence we remove the Breed column

Removing 'Breed' column

```python
print("Before dropping 'Breed' column:")
print(df1.head())

# Drop the 'Breed' column
df1.drop(columns=['Breed'], inplace=True)

# Display the first few rows of the dataset after dropping the column
print("\nAfter dropping 'Breed' column:")
print(df1.head())
```

```
Before dropping 'Breed' column:
   PetID  PetType  Breed       AgeMonths   Color  Size  WeightKg  Vaccinated  \
0    500        0      2  121-180 months  Orange     0  0.138783           1
1    501        3      5   61-120 months   White     0  0.520009           0
2    502        2      0  121-180 months  Orange     1  0.036514           0
3    503        0      2   61-120 months   White     2  0.080105           0
4    504        3      5  121-180 months    Gray     0  0.672244           0

   HealthCondition TimeInShelterDays  AdoptionFee  PreviousOwner  \
0                0        Short Stay     0.280561              0
1                0   Very Short Stay     0.470942              0
2                0         Long Stay     0.771543              0
3                0         Long Stay     0.434870              1
4                0        Short Stay     0.028056              1
```

Pet adoption
```
   AdoptionLikelihood
0                   0
1                   0
2                   0
3                   0
4                   0

After dropping 'Breed' column:
   PetID  PetType       AgeMonths   Color  Size  WeightKg  Vaccinated  \
0    500        0  121-180 months  Orange     0  0.138783           1
1    501        3   61-120 months   White     0  0.520009           0
2    502        2  121-180 months  Orange     1  0.036514           0
3    503        0   61-120 months   White     2  0.080105           0
4    504        3  121-180 months    Gray     0  0.672244           0

   HealthCondition TimeInShelterDays  AdoptionFee  PreviousOwner  \
0                0        Short Stay     0.280561              0
1                0   Very Short Stay     0.470942              0
2                0         Long Stay     0.771543              0
3                0         Long Stay     0.434870              1
4                0        Short Stay     0.028056              1

   AdoptionLikelihood
0                   0
1                   0
2                   0
3                   0
4                   0
```

5.DATA MINING TECHNIQUE

We utilized a dataset containing various features related to pets available for adoption,We employed both supervised and unsupervised learning techniques to analyze the data.

- **Classification**

  Supervised Learning: Classification
  For our classification task, we used a Decision Tree algorithm. This recursive method creates a tree structure where each leaf node represents a final decision regarding the likelihood of adoption. We trained our model to predict whether a pet is likely to be adopted based on features such as age, size, and health conditions.

  To prepare our data, we split the dataset into training and testing subsets. We experimented with three different training sizes: 70%, 60%, and 80%, and evaluated the model using two attribute selection measures—Information Gain (Entropy) and Gini Index. Model performance was assessed using accuracy and a confusion matrix, which summarizes key performance metrics like sensitivity, specificity, precision, and error
  Reasons for choosing partition
  -Reasons for Using 80% Training and 20% Testing
  • By allocating 80% of the dataset to training, the model has enough data to learn patterns and generalize well, particularly for larger datasets or complex models.
  • A 20% test set ensures the model's performance is evaluated on a substantial and diverse set of unseen data, providing reliable insights into its generalization capabilities
  -Reasons for Using 70% Training and 30% Testing

Pet adoption

70% Training: Provides enough data for the model to learn patterns effectively, especially for larger datasets, without overwhelming the algorithm with too much data or overfitting to the training set.
• 30% Testing: Ensures sufficient data is available to evaluate the model's performance on unseen data, which gives a reliable indication of how the model will generalize in real-world scenarios.
-Reasons for Using 60% Training and 40% Testing
The larger test set (40%) allows for a thorough and diverse evaluation of the model. In cases where the model needs to handle rare or edge cases, a bigger test set ensures that these scenarios are well-represented and the model's robustness is properly assessed.
Ensuring Reliable Performance Metrics
By having a substantial 40% of the data reserved for testing, I can ensure that the performance metrics (e.g., accuracy, precision, recall) are more reliable and reflective of how the model will perform in real-world situations.

Python Packages and Libraries Used:

1. **sklearn.tree.DecisionTreeClassifier**
   - **Purpose:** Used to create and train the decision tree model.
   - **Functions Used:**
     - DecisionTreeClassifier(): Creates the decision tree model.
     - plot_tree(): Visualizes the decision tree structure**.**
2. **sklearn.model_selection.train_test_split**
   - **Purpose:** Splits the dataset into training and testing sets to evaluate the model.
   - **Function Used:**
     - train_test_split(): Divides the dataset into specified proportions for training and testing.
3. **sklearn.metrics**
   - **Purpose:** Provides metrics to evaluate the model's performance.
   - **Functions Used:**
     - accuracy_score(): Calculates the model's accuracy.
     - confusion_matrix(): Computes the confusion matrix for evaluating classification performance.
     - ConfusionMatrixDisplay(): Visualizes the confusion matrix as a plot.
4. **sklearn.tree**
   - **Purpose:** Offers utilities for working with decision trees.
   - **Functions Used:**
     - plot_tree(): Generates a visualization of the trained decision tree.
5. **matplotlib.pyplot**
   - **Purpose:** Provides tools for plotting and visualizing data.
   - **Functions Used:**
     - plt.show(): Displays plots, including the decision tree and confusion matrix.

```python
#split dataset in features and target variable
fn=df2.keys().tolist()[:-1]
X=df2[fn]
y=df2['AdoptionLikelihood']
```

```python
from sklearn.preprocessing import LabelEncoder
encoder =LabelEncoder()
for col in X.select_dtypes('object').columns:
   X.loc[:,col]=encoder.fit_transform(X[col])
```

```
# Initialize classifiers with both Gini and Entropy criteria
dt_gini = DecisionTreeClassifier(criterion='gini', random_state=1)
dt_entropy = DecisionTreeClassifier(criterion='entropy', random_state=1)
```

```
Partition: 20.0 Test - 80.0 Train
# Split data for the partition size
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)

# Train and evaluate with Gini index
clf=dt_gini.fit(X_train, y_train)
y_pred_gini = dt_gini.predict(X_test)
accuracy_gini = accuracy_score(y_test, y_pred_gini)
# Error Rate
error_rate = 1 - accuracy_gini
cm_gini = confusion_matrix(y_test, y_pred_gini)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_gini[1, 1]
TN = cm_gini[0, 0]
FP = cm_gini[0, 1]
FN = cm_gini[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
# Precision
precision = TP / (TP + FP)
print("Gini Index:")
print("Accuracy:",accuracy_gini)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
print("Confusion Matrix:\n", cm_gini)
cn=df2['AdoptionLikelihood'].unique()
disp=ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test, display_labels=cn)
# Plot the decision tree Gini Index
plt.figure(figsize=(15, 15), dpi=600)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
plt.title("Decision Tree Gini Index")
plt.show()
# Train and evaluate with Entropy (Information Gain)
clf=dt_entropy.fit(X_train, y_train)
y_pred_entropy = dt_entropy.predict(X_test)
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
# Error Rate
error_rate = 1 - accuracy_entropy
```

```
cm_entropy = confusion_matrix(y_test, y_pred_entropy)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_entropy[1, 1]
TN = cm_entropy[0, 0]
FP = cm_entropy[0, 1]
FN = cm_entropy[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
# Precision
precision = TP / (TP + FP)
print("Entropy (Information Gain):")
print("Accuracy: ",accuracy_entropy)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
print("Confusion Matrix:\n", cm_entropy)
cn=df2['AdoptionLikelihood'].unique()
disp=ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test, display_labels=cn)
# Plot the decision Entropy tree
plt.figure(figsize=(15, 15), dpi=600)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
plt.title("Decision Tree Entropy")
plt.show()
```

Partition: 30.0 Test - 70.0 Train

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=1)
# Train and evaluate with Gini index
clf=dt_gini.fit(X_train, y_train)
y_pred_gini = dt_gini.predict(X_test)
accuracy_gini = accuracy_score(y_test, y_pred_gini)
# Error Rate
error_rate = 1 - accuracy_gini
cm_gini = confusion_matrix(y_test, y_pred_gini)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_gini[1, 1]
TN = cm_gini[0, 0]
FP = cm_gini[0, 1]
FN = cm_gini[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
```

```
# Precision
precision = TP / (TP + FP)
print("Gini Index:")
print("Accuracy:",accuracy_gini)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
print("Confusion Matrix:\n", cm_gini)
cn=df2['AdoptionLikelihood'].unique()
disp=ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test, display_labels=cn)
# Plot the decision tree Gini Index
plt.figure(figsize=(15, 15), dpi=600)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
plt.title("Decision Tree Gini Index")
plt.show()
# Train and evaluate with Entropy (Information Gain)
clf=dt_entropy.fit(X_train, y_train)
y_pred_entropy = dt_entropy.predict(X_test)
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
# Error Rate
error_rate = 1 - accuracy_entropy
cm_entropy = confusion_matrix(y_test, y_pred_entropy)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_entropy[1, 1]
TN = cm_entropy[0, 0]
FP = cm_entropy[0, 1]
FN = cm_entropy[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
# Precision
precision = TP / (TP + FP)
print("Entropy (Information Gain):")
print("Accuracy: ",accuracy_entropy)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
print("Confusion Matrix:\n", cm_entropy)
cn=df2['AdoptionLikelihood'].unique()
disp=ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test, display_labels=cn)
# Plot the decision Entropy tree
plt.figure(figsize=(15, 15), dpi=600)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
```

```
plt.title("Decision Tree Entropy")
plt.show()
```

Partition: 40.0 Test - 60.0 Train

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)
# Train and evaluate with Gini index
clf=dt_gini.fit(X_train, y_train)
y_pred_gini = dt_gini.predict(X_test)
accuracy_gini = accuracy_score(y_test, y_pred_gini)
# Error Rate
error_rate = 1 - accuracy_gini
cm_gini = confusion_matrix(y_test, y_pred_gini)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_gini[1, 1]
TN = cm_gini[0, 0]
FP = cm_gini[0, 1]
FN = cm_gini[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
# Precision
precision = TP / (TP + FP)
print("Gini Index:")
print("Accuracy:",accuracy_gini)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
print("Confusion Matrix:\n", cm_gini)
cn=df2['AdoptionLikelihood'].unique()
disp=ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test, display_labels=cn)
# Plot the decision tree Gini Index
plt.figure(figsize=(15, 15), dpi=600)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
plt.title("Decision Tree Gini Index")
plt.show()
# Train and evaluate with Entropy (Information Gain)
clf=dt_entropy.fit(X_train, y_train)
y_pred_entropy = dt_entropy.predict(X_test)
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
# Error Rate
error_rate = 1 - accuracy_entropy
cm_entropy = confusion_matrix(y_test, y_pred_entropy)
# Sensitivity (Recall) and Specificity  require TN, FP, FN, TP :
TP = cm_entropy[1, 1]
```

Pet adoption

```
TN = cm_entropy[0, 0]
FP = cm_entropy[0, 1]
FN = cm_entropy[1, 0]
# Sensitivity (Recall )
sensitivity = TP / (TP + FN)
# Specificity
specificity = TN / (TN + FP)
# Precision
precision = TP / (TP + FP)
print("Entropy (Information Gain):")
print("Accuracy: ",accuracy_entropy)
print("Error Rate:", error_rate)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
# Plot the decision Entropy tree
plt.figure(figsize=(20, 20), dpi=700)
class_names = df2['AdoptionLikelihood'].astype(str).unique()
plot_tree(clf, filled=True, feature_names=fn, class_names=class_names)
plt.title("Decision Tree Entropy")
plt.show()
```

Pet adoption

- **Clustering**
  Clustering Technique Used: K-Means
  Overview of K-Means:
  K-Means is a widely used clustering algorithm that partitions the dataset into k distinct clusters.
  The algorithm minimizes the within-cluster sum of squares (WCSS) to form compact and well-separated clusters.
  Each cluster is represented by its centroid, and data points are assigned to the nearest centroid.
  We use it to It is computationally efficient and can handle large datasets, making it suitable for this dataset's size.

Python Packages and Libraries Used:

1. **scikit-learn**:
   - **Purpose**:
     - Used to perform K-Means clustering and evaluate the clustering results.
   - **Functions Used**:
     - KMeans: For clustering implementation.
     - silhouette_score: For evaluating the clustering quality.
2. **matplotlib**:

- **Purpose**:
  - For visualizing the clustering results and evaluation metrics such as the Elbow Method and Silhouette Scores.
- **Functions Used**:
  - plot: To visualize the WCSS vs. kk (Elbow Method) and Silhouette Scores for different kk.
  - scatter: For plotting clusters in reduced-dimensional space.

3. **yellowbrick**:

- **Purpose**:
  - To visualize the Silhouette analysis for different numbers of clusters.
- **Functions Used**:
  - SilhouetteVisualizer: For plotting the silhouette score visualization.

4. **pandas and numpy**:

- **Purpose**:
  - For data manipulation and preprocessing before applying the clustering algorithm.

5. **scikit-learn.preprocessing**:

- **Purpose**:

- For data standardization and normalization, ensuring all features contribute equally to the clustering process.

Evaluation Methods:

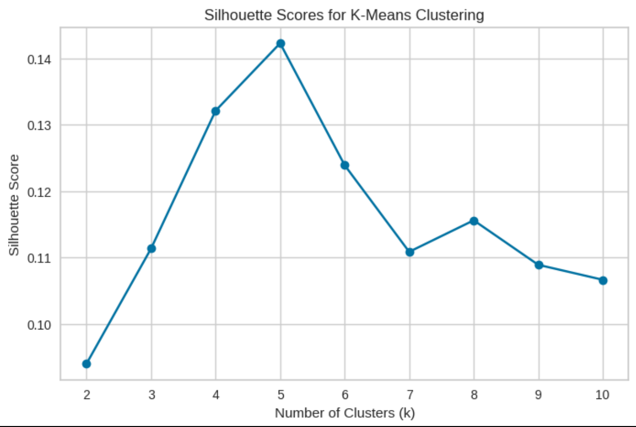
1. **Elbow Method**:
   - Used to determine the optimal number of clusters by plotting WCSS against kk.
   - The "elbow point" indicates where increasing k yields diminishing returns.
2. **Silhouette Score**:
   - Measures how well-separated the clusters are, with higher scores indicating better-defined clusters.

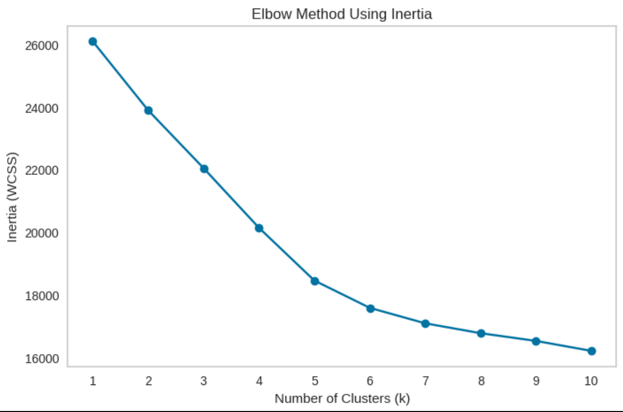
```
   import pandas as pd
   from sklearn.cluster import KMeans
   from sklearn.preprocessing import StandardScaler

   import matplotlib.pyplot as plt


features = df2.drop(columns=['AdoptionLikelihood'], axis=1)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
pet_scaled = pd.DataFrame(scaled_features , columns=features.columns)
print("Scaled DataFrame:")
print(pet_scaled)
```

```
from sklearn.metrics import silhouette_score
inertia_values = []
k_range = range(1, 11)  # Test for k from 1 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)  # Use your normalized features
    inertia_values.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia_values, marker='o')
plt.title("Elbow Method Using Inertia")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia (WCSS)")
plt.xticks(k_range)
plt.grid()
plt.show()



silhouette_scores = []
k_range = range(2, 11)  # Start from 2 clusters (k=1 is meaningless for silhouette)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(scaled_features)
    score = silhouette_score(scaled_features, labels)
    silhouette_scores.append(score)
```

Pet adoption

```
plt.figure(figsize=(8, 5))
plt.plot(k_range, silhouette_scores, marker='o')
plt.title("Silhouette Scores for K-Means Clustering")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")

plt.show()
```

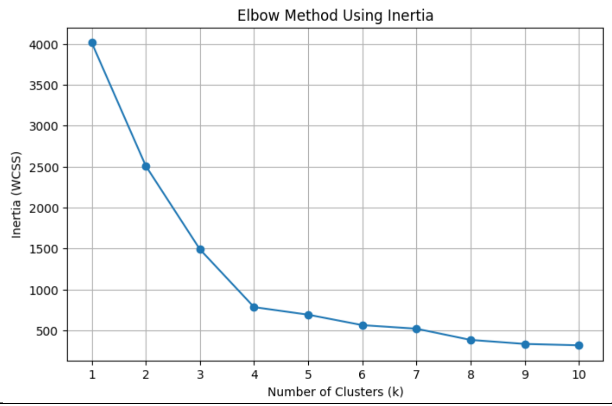




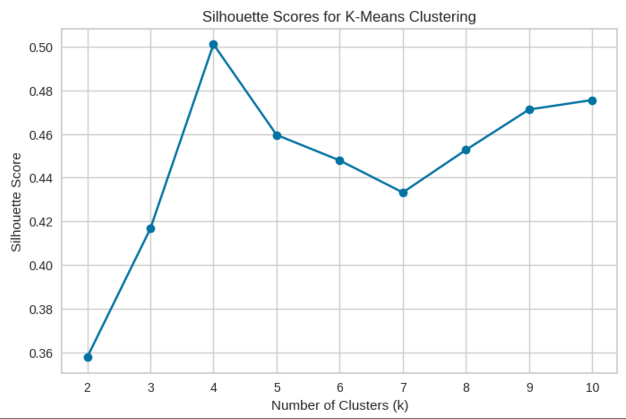
Pet adoption

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

features = df2[['AgeMonths',  'WeightKg']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
pet_scaled = pd.DataFrame(scaled_features , columns=features.columns)
print("Scaled DataFrame:")
print(pet_scaled)
```

```
inertia_values = []
k_range = range(1, 11)  # Test for k from 1 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)  # Use your normalized features
    inertia_values.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia_values, marker='o')
plt.title("Elbow Method Using Inertia")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia (WCSS)")
plt.xticks(k_range)
plt.grid()
plt.show()
```



```
from sklearn.metrics import silhouette_score

silhouette_scores = []
k_range = range(2, 11)  # Start from 2 clusters (k=1 is meaningless for silhouette)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(scaled_features)
    score = silhouette_score(scaled_features, labels)
    silhouette_scores.append(score)

plt.figure(figsize=(8, 5))
plt.plot(k_range, silhouette_scores, marker='o')
plt.title("Silhouette Scores for K-Means
Clustering")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Silhouette Score")
plt.show()
```



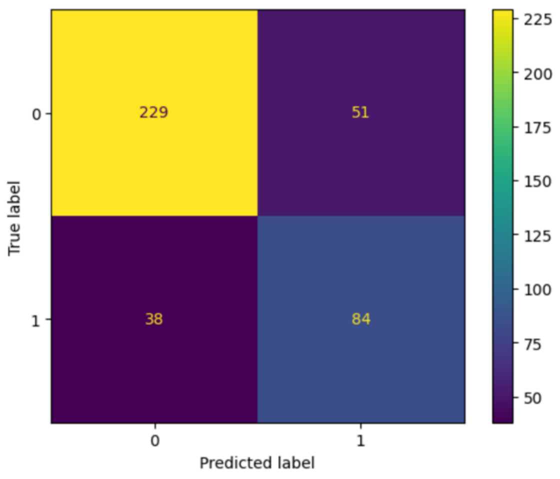
6.EVALUATION AND COMPARSION

classification:

Gini index:

- 80 % training set 20% testing set:

Confusion Matrix:



The confusion matrix represents the performance measurement of the classification model. In this case, we have a 2x2 confusion matrix where each row represents the actual classification, and each column represents the predicted classification. From the results, it appears:

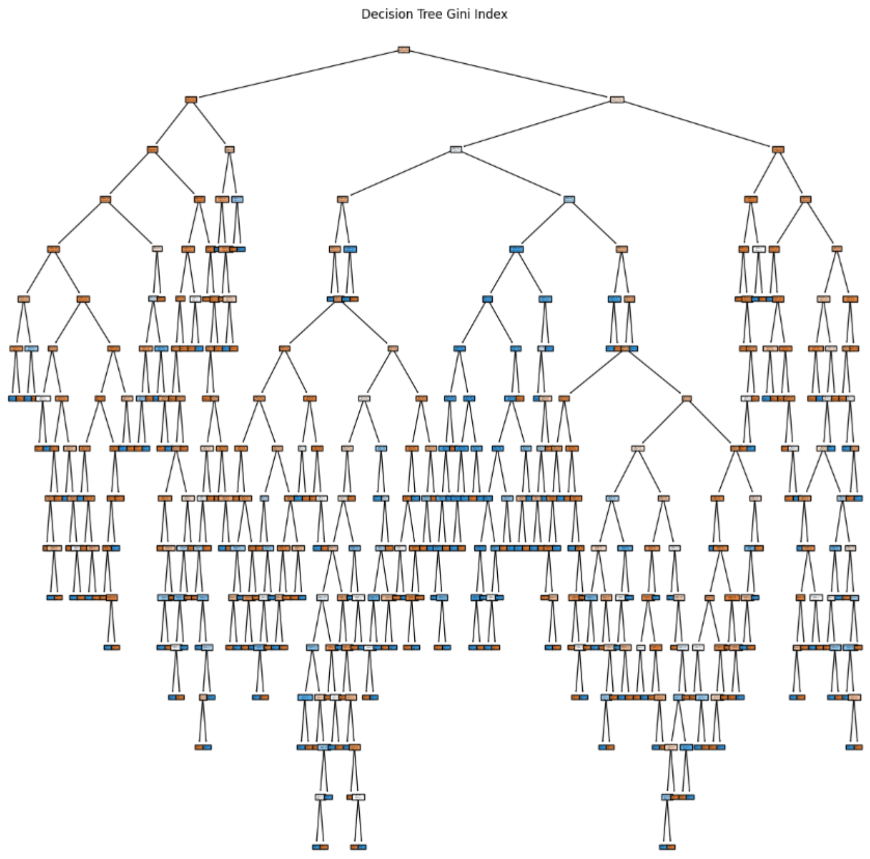
The top-left cell: (229) represents the number of instances of the negative class correctly classified as negative (TN).

The top-right cell: (51) represents the number of instances of the negative class incorrectly classified as positive (FP).

The bottom-left cell: (38) represents the number of instances of the positive class incorrectly classified as negative (FN).

The bottom-right cell: (84) represents the number of instances of the positive class correctly classified as positive (TP)

Decision tree:

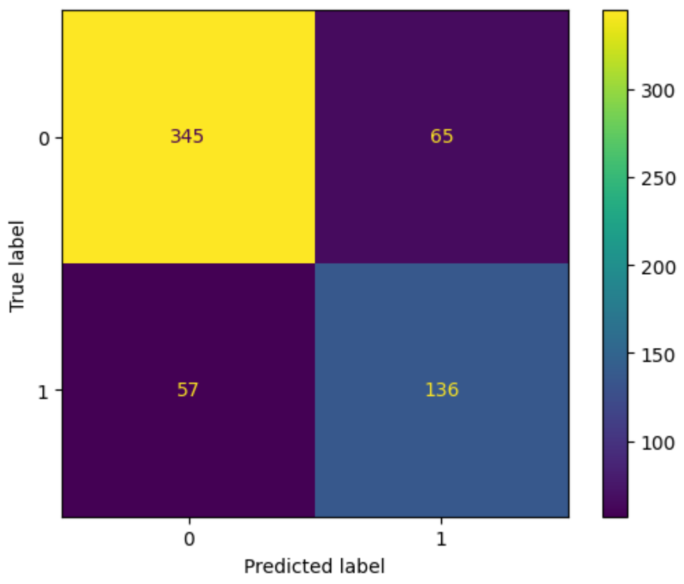


In this tree, the splitting process begins with the criterion of Vaccinated, where samples are segregated based on their Vaccinated values. The selection of features at each node is determined by their Gini Index. Following the split on Vaccinated, the tree considers the AgeMonths, dividing samples accordingly. Subsequently, HealthCondition is examined, leading to further division of samples. This splitting procedure persists for each attribute, guided by their respective values at each level, until reaching the leaf nodes. These leaf nodes act as terminal points, providing the final classification (whether it likely adopted or Unlikely) based on the path followed through the tree

Pet adoption

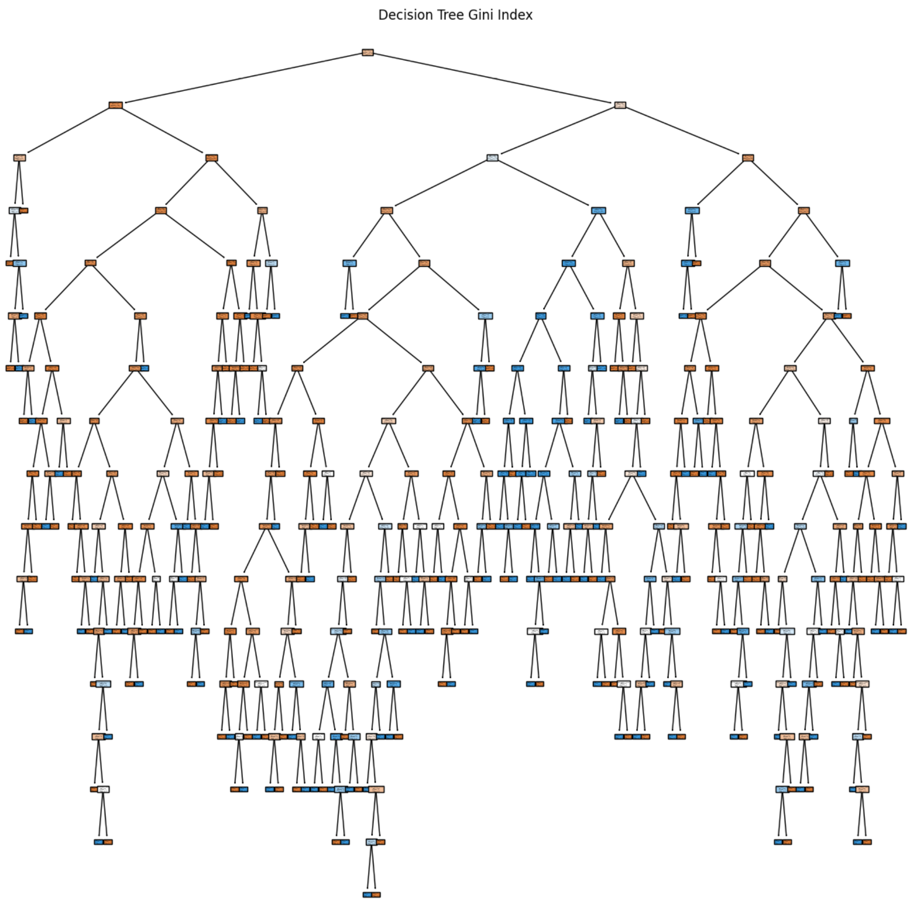
- 70% training set 30% testing set:

Confusion Matrix:



As we shown we used confusion matrix to summarizes the basic measures for performance evaluation This matrix is a 2x2 array where:
-(345) is true negatives (TN) means actual negatives correctly identified by the model.
-(65) is false positives (FP) means actual negatives incorrectly identified as positives.
-(57) is false negatives (FN) means actual positives incorrectly identified as negatives.
-(136) is true positives (TP) means actual positives correctly identified by the model.

Decision tree:

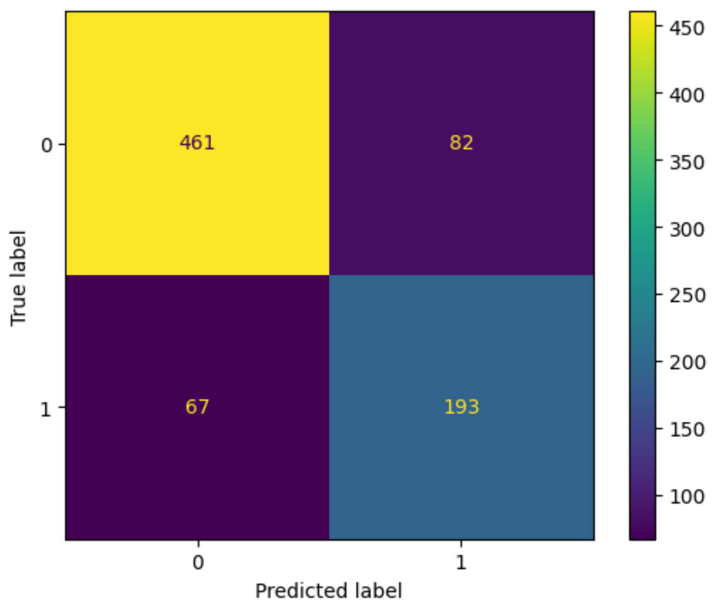


The decision tree model for predicting pet being adopted (Vaccinated) as the initial splitting criterion. If Vaccinated is less than or equal to 0.5, the model further splits the data based on features such as size, HealthCondition, AgeMonths, and others. The terminal nodes (leaf nodes) in the decision tree provide the final predicted outcome or class label, where class 1 represents pet likely being adopted class 2 represents Unlikely

Pet adoption

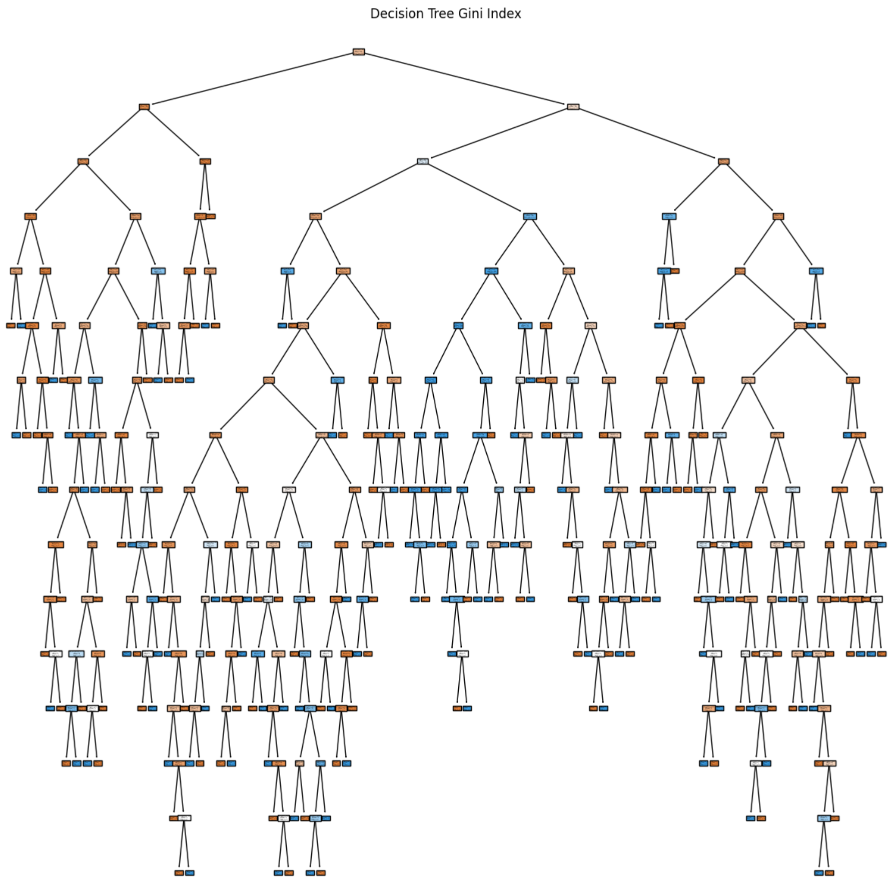
60% training set 40% testing set:

Confusion Matrix:



As we shown we used confusion matrix to summarizes the basic measures for performance evaluation This matrix is a 2x2 array where:
-(461) is true negatives (TN) means actual negatives correctly identified by the model.
-(82) is false positives (FP) means actual negatives incorrectly identified as positives.
-(67) is false negatives (FN) means actual positives incorrectly identified as negatives.
-(193) is true positives (TP) means actual positives correctly identified by the model

Decision tree:



As we shown the tree starts with the root node in this data is "Vaccinated" This means the tree branches out from the root node based on the 'Vaccinated' value , The tree makes several splits based on the feature values, and these are represented by the decision nodes. The features at each node are selected based on Gini Index. And also we have leaf Node this is the terminal nodes that provide the final classification based on the path taken through the tree.A leaf node will also provide the class label ('1' or '0')

Pet adoption

Gini index Comparison:

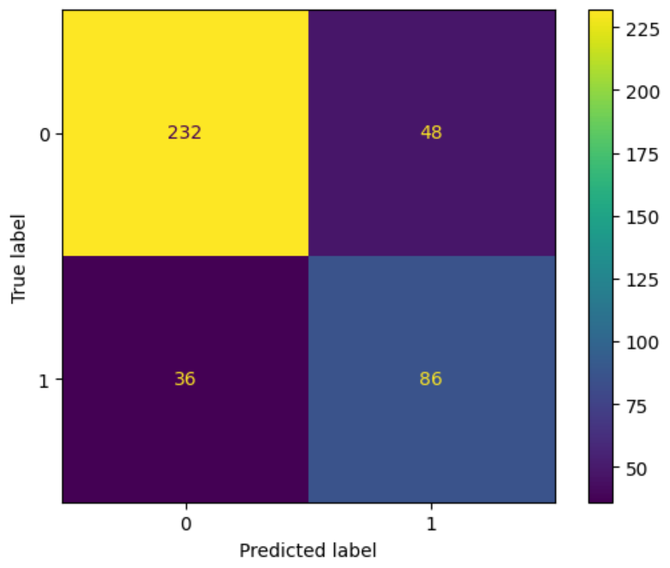
	80 % training set 20% testing set:	**70% training set 30% testing set:**	**60% training set 40% testing set:**
Accuracy	0.7786069651741293	0.7976782752902156	0.8144458281444583
Sensitivity	0.6885245901639344	0.7046632124352331	0.7423076923076923
Specificity	0.8178571428571428	0.8414634146341463	0.848987108655617
Precision	0.6222222222222222	0.6766169154228856	0.7018181818181818
Error Rate	0.2213930348258707	0.20232172470978438	0.18555417185554168

Pet adoption

Entropy (Information Gain):

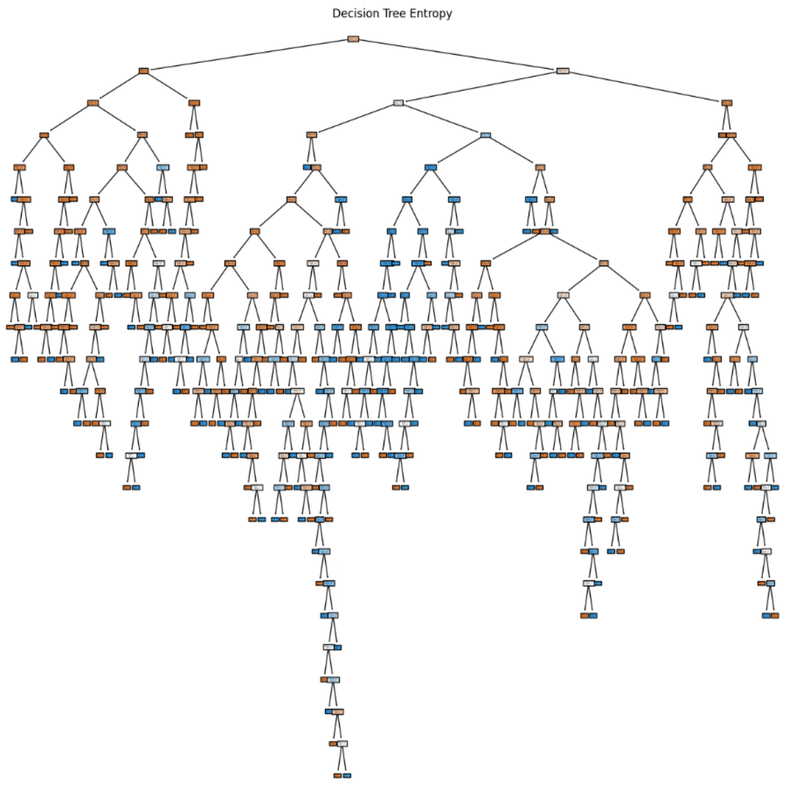
- 80 % training set 20% testing set:

Confusion Matrix:



As we shown we used confusion matrix to summarizes the basic measures for performance evaluation This matrix is a 2x2 array where:
-(232) is true negatives (TN) means actual negatives correctly identified by the model.
-(48) is false positives (FP) means actual negatives incorrectly identified as positives.
-(36) is false negatives (FN) means actual positives incorrectly identified as negatives.
-(86) is true positives (TP) means actual positives correctly identified by the model.

Decion tree:

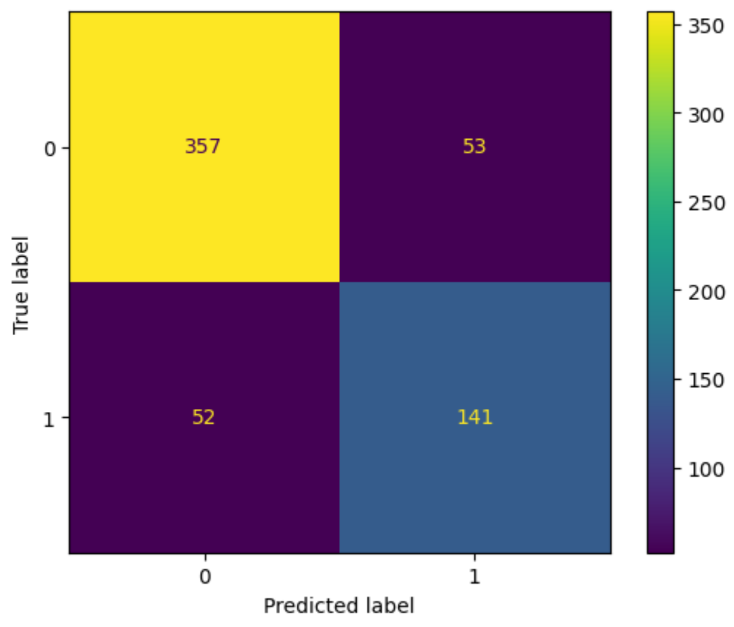


As we shown the tree starts with the root node in this data is "Vaccinated" This means the tree branches out from the root node based on the 'Vaccinated' value , The tree makes several splits based on the feature values, and these are represented by the decision nodes. The features at each node are selected based on entropy .
And also we have leaf Node this is the terminal nodes that provide the final classification based on the path taken through the tree.A leaf node will also provide the class label ('1' or '0')

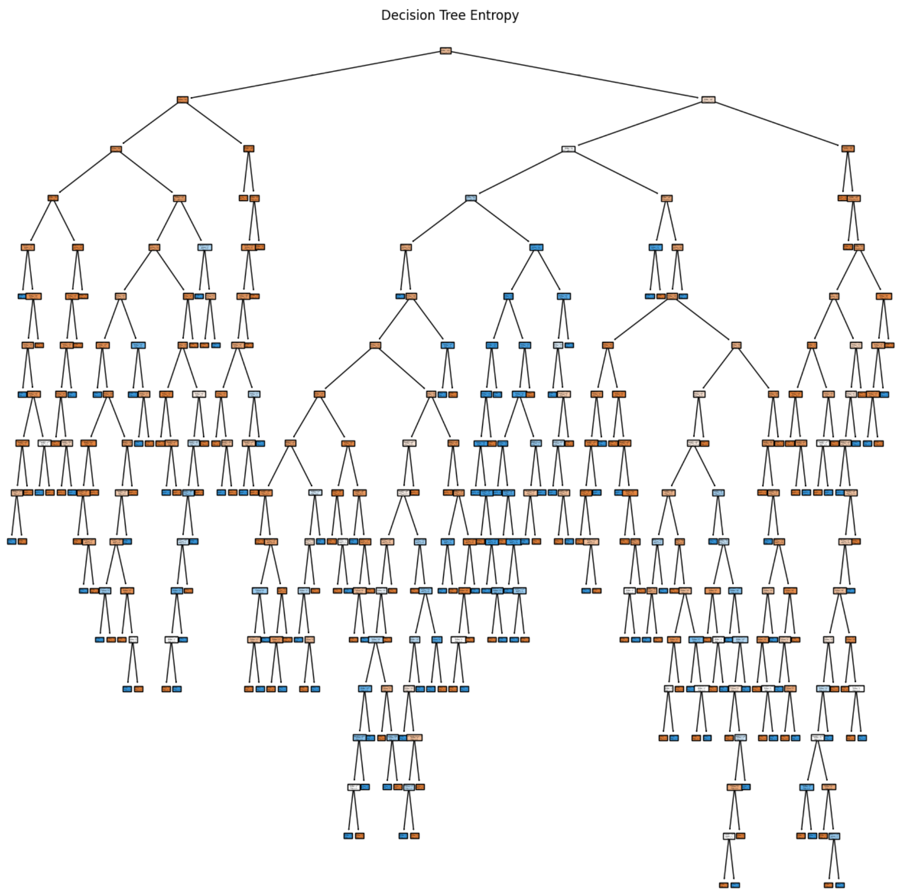
Pet adoption

- 70% training set 30% testing set:

Confusion Matrix:



As we shown we used confusion matrix to summarizes the basic measures for performance evaluation This matrix is a 2x2 array where:
-(357) is true negatives (TN) means actual negatives correctly identified by the model.
-(53) is false positives (FP) means actual negatives incorrectly identified as positives.
-(52) is false negatives (FN) means actual positives incorrectly identified as negatives.
-(141) is true positives (TP) means actual positives correctly identified by the model.

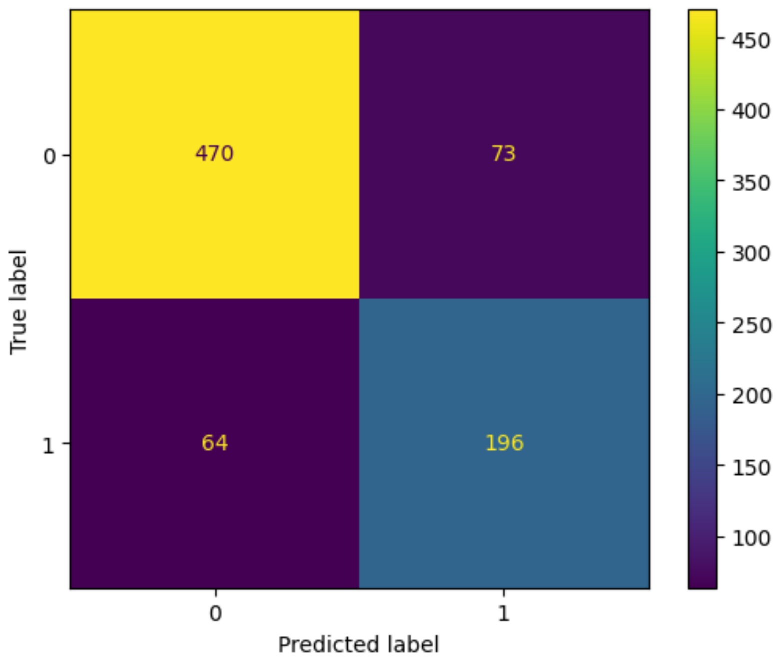


In this tree, the splitting process begins with the criterion of Vaccinated , where samples are segregated based on their Vaccinated values. The selection of features at each node is determined by their entropy values. Following the split on Vaccinated, the tree considers the size , dividing samples accordingly. Subsequently, the PetType is examined, leading to further division of samples. This splitting procedure persists for each attribute, guided by their respective values at each level, until reaching the leaf nodes. These leaf nodes act as terminal points, providing the final classification whether it likely adopted or Unlikely, based on the path followed through the tree.

Pet adoption

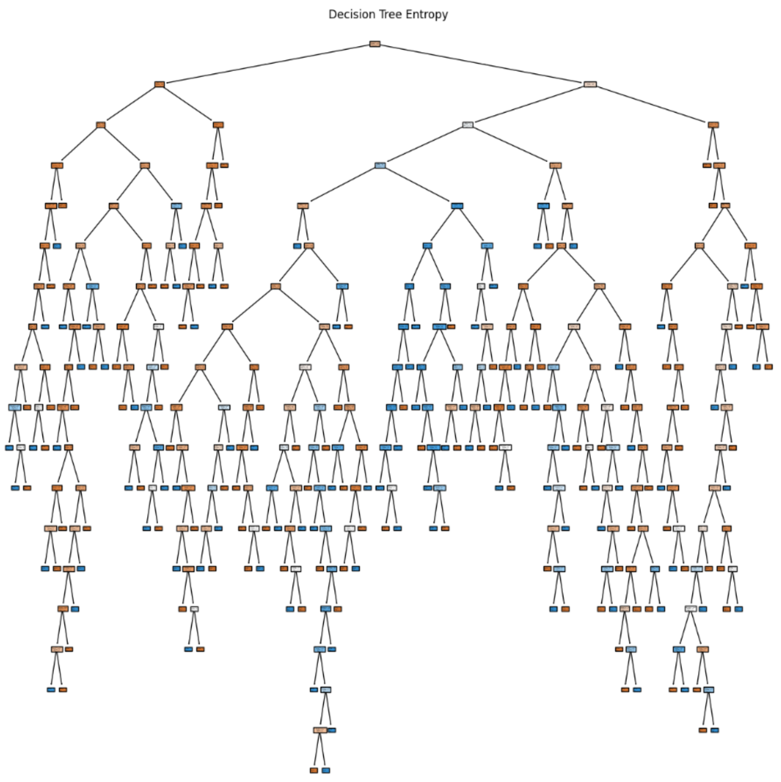
- 60% training set 40% testing set:

Confusion Matrix:



As we shown we used confusion matrix to summarizes the basic measures for performance evaluation This matrix is a 2x2 array where:
-(470) is true negatives (TN) means actual negatives correctly identified by the model.
-(73) is false positives (FP) means actual negatives incorrectly identified as positives.
-(64) is false negatives (FN) means actual positives incorrectly identified as negatives.
-(196) is true positives (TP) means actual positives correctly identified by the model.

Decision tree:



The decision tree for predicting is built on the importance of Vaccinated and further splits on features such as size , HealthCondition, WeightKg, AdoptionFee, AgeMonths, and PetType. The tree's structure reveals complex decision pathways based on combinations of these features. Terminal nodes provide the final predicted outcome (1 for pet being Likely adopted, 2 for Unlikely)based on the feature values and their importance. The depth and complexity of the decision tree demonstrate the diverse factors considered by the model in assessing Likelihood of the pet being adopted . Understanding the decision tree provides valuable insights into the model's inner workings and its ability to predict Likelihood of the pet being adopted

Entropy (Information Gain) Comparison:

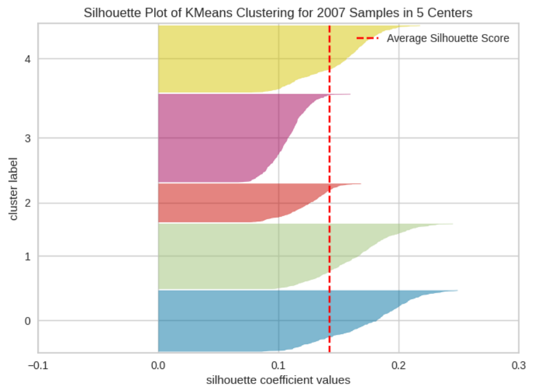
	80 % training set 20% testing set:	**70% training set 30% testing set:**	**60% training set 40% testing set:**
Accuracy	0.7910447761194029	0.8258706467661692	0.8293897882938979
Sensitivity	0.7049180327868853	0.7305699481865285	0.7538461538461538
Specificity	0.8285714285714286	0.8707317073170732	0.8655616942909761
Precision	0.6417910447761194	0.7268041237113402	0.7286245353159851
Error Rate	0.20895522388059706	0.17412935323383083	0.17061021170610213

After comparing the performance metrics from the Gini Index and Entropy (Information Gain) tables, it is evident that the Entropy criterion with a 60% training set and 40% testing set provides the most consistent and optimal results across all key metrics. This setup achieves the highest accuracy (0.8293), specificity (0.8655), and precision (0.7286), while also having the lowest error rate (0.1706). This means that using Entropy as the criterion for Decision Tree classification is likely to give better results with this type of data.

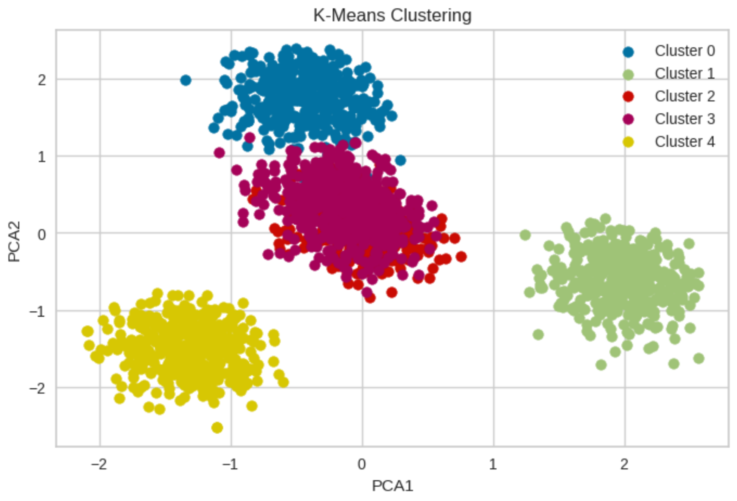
Pet adoption

Clustering (results with low silhouette score):

```
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5, random_state=42)
visualizer = SilhouetteVisualizer(kmeans, colors="yellowbrick")
visualizer.fit(pet_scaled)
print("The average silhouette score is:",
visualizer.silhouette_score_)
visualizer.show()
```



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_features = pca.fit_transform(scaled_features)
df2['PCA1'] = pca_features[:, 0]
df2['PCA2'] = pca_features[:, 1]

plt.figure(figsize=(8, 5))
for cluster in range(optimal_k):
    cluster_data = df2[df2['Cluster'] == cluster]
    plt.scatter(cluster_data['PCA1'], cluster_data['PCA2'], label=f'Cluster {cluster}')

plt.title("K-Means Clustering")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.legend()
plt.show()
```



Pet adoption

	K = 3	K = 4	K = 5
Average Silhouette width	0.11	0.13	0.14
Total within-cluster sum of square	22000	20000	18000

Discussion & comparison

Elbow Method (Inertia vs Number of Clusters): The Elbow method looks at the total within-cluster sum of squares (inertia) and helps identify the point where adding more clusters does not significantly improve the fit.
From the results:
K=3: Inertia approximately 22000
K=4: Inertia decreases to approximately 20000
K=5: Inertia further reduces to approximately 18000
The inertia steadily decreases as the number of clusters K increases. This is expected since increasing K reduces the distance between data points and their cluster centers. The rate of decrease slows noticeably after K=5. This suggests that adding more clusters beyond K=5 does not significantly improve the compactness of the clusters.
Silhouette Score vs Number of Clusters: The Silhouette coefficient measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.
The silhouette score peaks at K=5 , indicating that the clusters are best separated at this point.
For K=4 and K=6 , the silhouette score is lower, showing less well-defined clusters. Beyond K=5, the score decreases significantly, indicating that the additional clusters result in worse separation or overly fragmented clusters.
Conclusion:
Optimal K: Based on the silhouette score, K=5 is the most suitable choice, as it provides the highest score, indicating well-defined and separated clusters.
Support from the Elbow Method: the inertia plot supports K=5 as a reasonable point where the rate of inertia reduction begins to slow.
The combination of the Elbow method and the Silhouette coefficient suggests that K=5 is the best choice. It balances the compactness (inertia) of the clusters with their separation (silhouette score), ensuring that the clusters are well-defined without unnecessary complexity. K=5 is the optimal number of clusters, balancing compactness (inertia) and separation (silhouette score).

Silhouette Score: A Silhouette Score output of 0.14 which is closer to 0 indicates suboptimal clustering performance. This score is low, suggesting that the clusters are not well-defined and that there is significant overlap between them, or that the clustering is not very strong.

The silhouette plot visually confirms that the clustering at K=5 does not result in well-separated groups. The average silhouette score line (marked in red) is low, and there are many points close to or below this line, signifying poor assignment quality for these clusters.

Although the elbow method suggested that 5 clusters , the silhouette plot highlights that this choice does not necessarily lead to well-defined or meaningful clusters.

Pet adoption

The visual results show that it's important to use different ways to check clustering results to make sure the chosen number of clusters actually fits the data well.

Visualizing the Clusters:

The plot shows five well-separated clusters (clusters 0 to 4) in the two-dimensional space "PCA", indicating that the K-means algorithm successfully identified distinct groupings within the data.

As we mentioned before in the silhouette score result that might be an overlapping, also from the plot above we can see that Clusters like Cluster 3 and Cluster 2 appear more densely packed, suggesting that data points in these clusters may have more similar feature values. In contrast, Cluster 1 appears somewhat more spread out, indicating a bit more variability within that group. (or might be due to the dimensions).

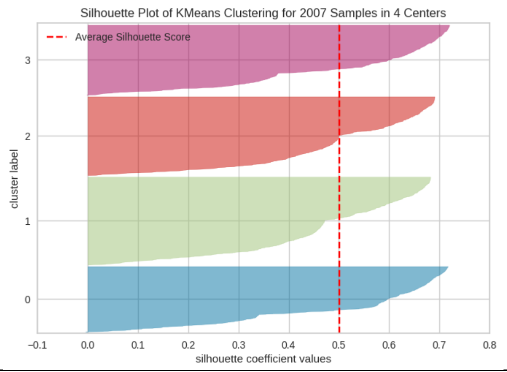
Cluster 0, Cluster 2, and Cluster 3 are relatively close to each other compared to Cluster 1 and Cluster 4, which are more isolated. This proximity could imply that the clusters close together share some characteristics, while the isolated clusters represent unique groups.

These clusters could represent distinct groups based on features like age, size, time in shelter, and other features. The isolated cluster (Cluster 1) may represent a unique subset of pets that have distinct characteristics (such as larger or smaller values in a key feature), which separate them from the rest.

In summary, the K-means clustering has effectively grouped the data into distinct clusters with meaningful separations, and the visualization using PCA provides a clear view of these groupings. The relatively isolated clusters indicate unique data groups, while closely located clusters may share **overlapping characteristics.**

Clustering (using other features combination for better results):

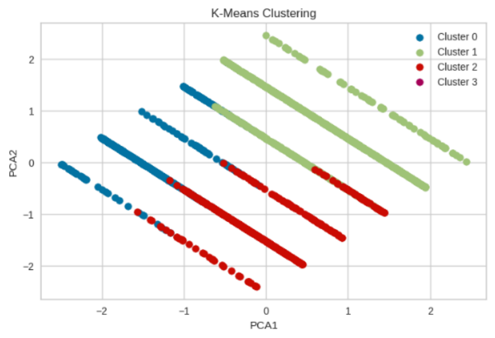
```
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, random_state=42)
visualizer = SilhouetteVisualizer(kmeans, colors="yellowbrick")
visualizer.fit(pet_scaled)
print("The average silhouette score is:",
visualizer.silhouette_score_)
visualizer.show()
```



```
from sklearn.decomposition import PCA
optimal_k = 4

pca = PCA(n_components=2)
pca_features = pca.fit_transform(scaled_features)
df['PCA1'] = pca_features[:, 0]
df['PCA2'] = pca_features[:, 1]
```

Pet adoption

```
plt.figure(figsize=(8, 5))
for cluster in range(optimal_k):
    cluster_data = df[df['Cluster'] == cluster]
    plt.scatter(cluster_data['PCA1'],
cluster_data['PCA2'], label=f'Cluster {cluster}')

plt.title("K-Means Clustering")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.legend()
plt.show()
```



Pet adoption

	K = 2	K = 4	K = 10
Average Silhouette width	0.36	0.50	0.47
Total within-cluster sum of square	2500	Approximately 800	Approximately 200

comparing and discussing different K numbers

From the Elbow Method plot, we can observe that : As K increases from 1 to 10, the inertia decreases, indicating that the clusters are becoming more compact,since increasing K allows more centroids to reduce the sum of squared distances.

k = 2 (First significant drop) Inertia drops significantly between k=1 and k=2, indicating that adding a second cluster provides a considerable improvement in clustering the data.

k = 4 (Elbow point) The inertia begins to level off after k=4. we can considered this as the "elbow" point, where increasing k beyond 4 yields diminishing returns in terms of reducing inertia.

k = 10 (Minimal Decrease in Inertia) Beyond k=4, the inertia decreases only slightly for larger values of k, as seen at k=10. This indicates that the additional clusters don't improve the clustering much further.k=10, the clustering is overly fine-grained.Each cluster would likely have a small number of members,rather than truly meaningful groups.

in conclusion the optimal K number will be the elbow point which is **k = 4** .

for K=2

Silhouette Score: Approximately 0.36 (the lowest in the range). A low silhouette score indicates that the clustering is poor, because two clusters are insufficient to capture the natural structure of the data. Many data points are likely assigned to the wrong cluster, because 2 clusters may forces diverse data points into only two groups, oversimplifying the dataset. It creates overly broad clusters, leading to low-quality groupings.

---

for K=4 (the optimal k number):

Silhouette Score: Approximately 0.51 (the highest value). The best clustering performance in this range, indicating that clusters are well-separated and data points fit well within their groups. Data points within each cluster are relatively compact and distinct from those in other clusters. **in Conclusion:** k=4 is the optimal choice based on the silhouette score.

Pet adoption

for K=10:

Silhouette Score: Approximately 0.46 (comparable to the scores for k=3 and k=5). While the silhouette score suggests a reasonable clustering result, having ten clusters might introduce unnecessary complexity, clusters may be too fragmented, splitting naturally cohesive groups into smaller, less meaningful sub-groups.

At 0.5 (the highest silhouette score of k values),

indicating the best balance between compactness and separation of clusters. the clustering is:

Fairly compact: Points within the clusters are moderately close to each other.

Reasonably separated: Clusters are distinct but not entirely isolated, meaning some overlap might still occur.

A score of 0.5 suggests moderately well-defined clusters and distinct, Clusters are distinguishable but not perfectly separated — pets of similar ages and weights could share overlapping characteristics.

The optimaal value of k is likely a good fit for the data.

Visualizing the Clusters: There are four distinct clusters (labeled as Cluster 0, Cluster 1, Cluster 2, and Cluster 3), as determined by the k=4 clustering. The clusters appear to be well-separated in the PCA-reduced space. This indicates that the K-Means algorithm has successfully grouped similar data points into distinct clusters with minimal overlap in this lower-dimensional representation.
as we discussed in the silhouette score results and from the clustering visualisation plot above we can see that there is an overlapping. the solution might be adding more features or try other features combinations . Absolutely we tried both of this solutions but it results with a minimum silhouette score than the current features selected.
including additional features may cause cluster boundaries to become less distinct because unrelated features dilute the meaningful patterns in the data. This leads to decreasing intra-cluster cohesion, clusters overlapping due to increased dimensionality. By focusing on "AgeMonths" and "WeightKg", we are limiting the clustering to features that maximize separability, and provide Simpler model, better-defined clusters, and a Silhouette Score of 0.5, indicating moderate quality.These features likely show a clear separation in the dataset, minimizing ambiguity between clusters.

7.FINDINGS

Classification Findings

1. **Performance Comparison**:
   The Decision Tree Classifier was evaluated using both **Gini Index** and **Entropy (Information Gain)** as criteria.
   Results across different training-testing splits (80%-20%, 70%-30%, 60%-40%) consistently showed that Entropy outperformed Gini Index
   - **Under the Information Gain (Entropy) criterion:** the 60-40 split model demonstrates superior performance across all key metrics:
     Accuracy: With the highest accuracy of 0.829, the 60-40 split shows it generalizes slightly better than other configurations, making it the most reliable for overall predictions.
     Error Rate: The 60-40 split achieves the lowest error rate of 0.1706, reflecting fewer misclassifications compared to the 80-20 split, which has the highest error rate at 0.20.
     Sensitivity: A sensitivity of 0.753 indicates that the 60-40 split model is the most effective at correctly identifying positive instances, outperforming the other splits.
     In summary, the model trained with the 60-40 split strikes the best balance between sufficient training data and a diverse test set

   - **Under the different partitions results in GINI INDEX**:
   Among these partitioning, the model trained on the 60% training set and 40% testing achieved the highest accuracy (0.814), followed by the model trained on the 70% training set and 30% testing set (0.797), followed by the model trained on the 60% training set and 40% testing set with an accuracy of (0.778).
   For error rate, the model trained on the 80% training set and 20% testing set achieved the highest error rate (0.22), followed by the models trained on the 70% training set and 30% testing set (0.20), and the 60% training set and 40% testing set (0.18).
   For sensitivity, the model trained on the 60% training set and 40% testing set achieved the highest sensitivity (0.74), followed by the models trained on the 70% training set and 30% testing set (0.70), and the 80% training set and 20% testing set (0.68).
   In terms of specificity, the model trained on the 60% training set and 40% testing set obtained the highest specificity (0.848), followed by the model trained on the 70% training set and 30% testing set (0.841), and the model trained on the 80% training set and 20% testing set (0.817).
   In terms of precision, the model trained on the 60% training set and 40% testing set obtained the highest precision (0.70), followed by the model trained on the 70% training set and 30% testing set (0.67), and the model trained on the 80% training set and 20% testing set (0.62).
   In summary, the 60-40 model strikes a good balance between classification accuracy, specificity, and sensitivity, which is why it is considered the best based on the provided results.

   - **The best model between information gain and the Gini index:**

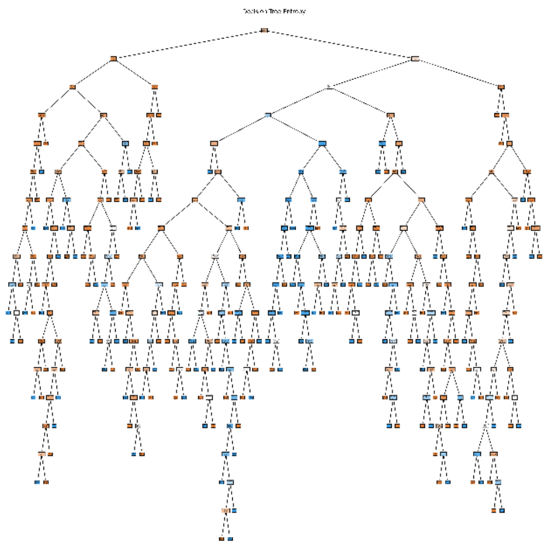
	Best Gini index (60-40)	**Best Entropy (60-40)**
Accuracy	0.8144458281444583	0.8293897882938979
Sensitivity	0.7423076923076923	0.7538461538461538
Specificity	0.848987108655617	0.8655616942909761
Precision	0.7018181818181818	0.7286245353159851
Error Rate	0.18555417185554168	0.17061021170610213

Pet adoption

Entropy: Performs better overall, especially in accuracy, sensitivity, and specificity, which are critical metrics for classification tasks.

Gini: While slightly less effective than Entropy, it still provides a strong performance, especially when dealing with imbalanced data

-This was the decision tree associated with this division:



The decision tree for predicting is built on the importance of Vaccinated and further splits on features such as size , HealthCondition, WeightKg, AdoptionFee, AgeMonths, and PetType. The tree's structure reveals complex decision pathways based on combinations of these features. Terminal nodes provide the final predicted outcome (1 for pet being Likely adopted, 2 for Unlikely) based on the feature values and their importance. The depth and complexity of the decision tree demonstrate the diverse factors considered by the model in assessing Likelihood of the pet being adopted. Understanding the decision tree provides valuable insights into the model's inner workings and its ability to predict Likelihood of the pet being adopted

2. **Confusion Matrix Analysis**:
   - The confusion matrices revealed balanced performance across class predictions, with Entropy producing fewer misclassifications than Gini.
   - Entropy better captured the relationships in the dataset, suggesting its suitability for the feature distributions.
3. **Conclusion**:
   - **Entropy is the preferred criterion** for this dataset, offering higher accuracy and better decision-making capacity compared to Gini Index.
   - The classifier's performance improves with larger training datasets, but there is a trade-off in generalization capability, particularly when training data is too large.
   - The result are interesting and The most interesting aspect is the identification of critical features like **Vaccinated**, **HealthCondition**, and **AdoptionFee**, which significantly impact adoption decisions. These findings can directly inform shelters or pet care organizations about the attributes to prioritize for increasing adoption rates. The results highlight that data mining not only aids in prediction but also uncovers meaningful patterns in the data.

Clustering Findings

Optimal Number of Clusters: k=4

1. **Evaluation Metrics:**
   - The clustering results were evaluated using two methods:
     - Elbow Method: Identified k=4 as the "elbow point," where adding more clusters yielded diminishing improvements in compactness (measured by Within-Cluster Sum of Squares, WCSS).
     - Silhouette Score: k=4 achieved the highest score of 0.51, indicating well-separated and compact clusters.
2. **Cluster Characteristics:**
   - The clusters were visualized using PCA for dimensionality reduction. Four distinct clusters were observed, with minimal overlap:
     - **Cluster 0**: Younger, smaller pets.
     - **Cluster 1**: Medium-sized pets at intermediate stages of development.
     - **Cluster 2**: Older or heavier pets, with more variability.
     - **Cluster 3:** Outliers or unique subsets with distinct characteristics.
3. **Comparison with k=5:**
   - While the Elbow Method initially suggested k=5 as a possible configuration, the Silhouette Score for k=5 was only 0.14, significantly lower than for k=4.
   - The visualization for k=5 showed overlapping clusters, reducing interpretability and meaningful group separation.

Metric	**K=4**	**K=5**
Silhouette Score	0.51 (Best)	0.14 (Poor)
Cluster Separation	Well-separated	Overlapping clusters
Cluster Compactness	Compact	Moderate overlap
Interpretability	Clear groups	Ambiguous overlaps

4. **Conclusion:**
   - Based on the evaluation metrics and visual analysis, k=4 was selected as the optimal number of clusters for our dataset.

Pet adoption

- This configuration offers the best balance between compactness and separation, providing meaningful and interpretable groupings of the data.
- Also the result are interesting , These clusters provide a fresh perspective, offering insights into distinct categories of pets that can be used for targeted strategies, such as marketing or tailored care. The evaluation metrics, especially the high Silhouette Score (0.51), confirm the meaningfulness of these clusters, making the findings not only accurate but also interpretable and practical.

Overall Insights: The combination of classification and clustering techniques has provided comprehensive insights into the dataset. Classification predicts outcomes effectively, while clustering reveals underlying structures in the data, offering a dual perspective for data-driven decision-making.

8.REFRENCES

Python Libraries Used Across All Phases

1. **General Data Manipulation and Analysis**:
   - [1] W. McKinney, "Data structures for statistical computing in Python," *Proceedings of the 9th Python in Science Conference*, 2010. Available: https://pandas.pydata.org/.
   - [2] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020. Available: https://numpy.org/.
2. **Data Visualization**:
   - [3] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. Available: https://matplotlib.org/.
   - [4] M. L. Waskom et al., "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. Available: https://seaborn.pydata.org/.
3. **Preprocessing and Feature Scaling**:
   - [5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. Available: https://scikit-learn.org/.

Phase 1: Problem, Dataset, and Exploration

1. **Data Exploration and Visualization**:
   - [6] A. Gelman, J. Hill, and M. Yajima, "Why we (usually) don't have to worry about multiple comparisons," *Journal of Research on Educational Effectiveness*, vol. 5, no. 2, pp. 189–211, 2012. [Used for understanding statistical measures in exploratory data analysis].

- [7] T. H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, 2nd ed. Springer, 2016. [Adapted for matplotlib/seaborn visualizations].

2. **Outlier Detection and Summary Statistics**:

- [8] D. C. Hoaglin, F. Mosteller, and J. W. Tukey, *Understanding Robust and Exploratory Data Analysis*. Wiley, 1983. [Five-number summary, boxplots].
- [9] J. Rousseeuw and M. Leroy, *Robust Regression and Outlier Detection*. Wiley, 1987. [Identifying and handling outliers].

Phase 2: Preprocessing

1. **Missing Data Handling**:

- [10] A. Gelman and J. Hill, *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006. [Strategies for imputing or handling missing data].

2. **Feature Selection and Dimensionality Reduction**:

- [11] I. Guyon and A. Elisseeff, "An introduction to feature extraction," in *Feature Extraction*, Springer, 2006, pp. 1–25.
- [12] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Springer, 1998.

3. **Scaling and Transformation**:

- [13] L. I. Smith, "A tutorial on principal components analysis," Cornell University, 2002. [PCA for feature scaling and reduction].

Phase 3: Classification and Clustering

1. **Classification Techniques**:

- [14] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [15] L. Breiman et al., *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

2. **Clustering Techniques**:

- [16] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, 2007, pp. 1027–1035.
- [17] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

3. **Evaluation Metrics**:

- [18] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. [Confusion matrix evaluation].
- [19] A. L. Barabási et al., "Silhouette width criterion for clustering," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2009, p. P02060, 2009.