Integrated Continuous Assessment 2

Prepared for CM_Statistics for Data Analytics / Machine Learning for Data Analytics / Data Preparation & Visualisation

Wilk, Ia Data Analytics (Sections 1)

Assignment Complete: Marina Iantsona, Muhammad Iqbal and David McQuaid

Last Update: Mars 28th 2022

H3N2 Bird Flu Detections

Introduction

Bird Flu is the Influenza Haemagglutinin Avian Influenza or "Bird Flu" is a contagious and often fatal viral disease of birds. Wild birds, domesticated birds and domestic mammals are considered the first hosts of avian influenza viruses. There is a concern at all levels of government, industry and society about the impact of avian influenza on the poultry industry, on food security, on livelihoods and on wildlife and ecosystems in vulnerable, rural, and resource-poor areas.

The following data set contains the detections of avian influenza reported in Ireland from 1984 to 2020 and which lists the are reported for the Ireland.

The data set was first 100% performed on the Jupiter Notebook platform so the first step is to download the libraries that we will use and save follows:

```
# Installing packages
!pip install -U ydata-profiling
!pip install -U sweetviz
!pip install -U dtale
!pip install -U lux-widget
!pip install -U pandas-profiling[notebook] dataprep ipywidgets
!jupyter nbextension enable --py --sys-prefix widgetsnbextension

# Importing libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
import sklearn.preprocessing as preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.linear_cross_validation import StratifiedKFold
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import roc_curve, auc
from scipy.stats.mstats import mquantiles, kruskalwallis
import scipy
import pandas_profiling
import sweetviz as sv
import dtale
import lux
import dataprep.eda as dp
import ydata_profiling as pp

# Setup plotting
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.rcParams['figure.dpi'] = 150
sns.set_palette("husl")
sns.set_style("whitegrid")
pd.set_option('display.max_columns', 50)
pd.set_option('display.max_rows', 100)
pd.set_option('display.precision', 3)
np.random.seed(42)
```

After downloading the libraries we need to load the data set:

```
# Load data
df = pd.read_csv('/content/drive/MyDrive/Data_Science/data_set/H3N2.csv')
```

Then type the name of the data set and a line:

To keep our dataset clean we first need to make a copy of the original dataset

```
df.head(10)
```

Data preparation

Characterisation

The data set is ready to be used for data analysis purposes. They are represented by tables (data is presentation format where the rows are the details of the individual records and the columns are the attributes of the records). Each row represents a single record and with the age and the hospitalisation of the records and the summary of its genetic characteristics or resources of the origin data set (Ma et al. 2011).

The data set contains 1000 records and 10 attributes (Table 1). The data set is available in the Kaggle repository (https://www.kaggle.com/datasets/jeffsinn/h3n2-bird-flu-detections?select=H3N2.csv) (Ma et al. 2011).

Data definitions

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy()
df.head(10)
```

- Our degree is 10 columns 1000 rows

The Code above has the First Look and the last 5 rows and columns (information is rows and is columns) df - 5rd

```
# Last 5 rows and columns
df.tail(5)
```

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy()
df.tail(5)
```

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.shape
```

```
(1000, 10)
```

The Code above has the Shape of the data set (number of rows) - (text/csv).

```
# Copy of original
df_orig = df.copy
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 0   Id                  1000 non-null   int64
 1   Year                1000 non-null   int64
 2   Hospitalised        1000 non-null   int64
 3   Hospitalised Days   1000 non-null   int64
 4   Age                 1000 non-null   int64
 5   Gender              1000 non-null   int64
 6   Subtype             1000 non-null   int64
 7   Hemagglutinin       1000 non-null   int64
 8   Neuraminidase       1000 non-null   int64
 9   Reassortment        1000 non-null   int64
dtype: int64
memory usage: 78.1 KB
```

The Code above lists the Information of the data set (Number of columns, Number of rows, type of the data set)

```
# Copy of original
df_orig = df.copy
df.dtypes
```

```
Id                  int64
Year                int64
Hospitalised        int64
Hospitalised Days   int64
Age                 int64
Gender              int64
Subtype             int64
Hemagglutinin       int64
Neuraminidase       int64
Reassortment        int64
dtype: int64
```

The Code above lists the type of each column of the data set (Numeric or Binary).

```
# Copy of original
df_orig = df.copy
df.nunique()
```

```
Id                  1000
Year                  37
Hospitalised           2
Hospitalised Days     21
Age                   99
Gender                 2
Subtype                2
Hemagglutinin          2
Neuraminidase          2
Reassortment           2
dtype: int64
```

The Code above lists the number of unique elements in the object. It returns a pulse value which is the count of all the unique values in the object. df.nunique() - (text/csv).

```
# Copy of original
df_orig = df.copy
df.describe()
```

	Id	Year	Hospitalised	Hospitalised Days	Age	Gender	Subtype	Hemagglutinin	Neuraminidase	Reassortment
count	1000.0	1000.000	1000.0	1000.000	1000.000	1000.000	1000.000	1000.000	1000.000	1000.000
mean	500.5	1,017.773	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
std	288.819	1,009.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
min	1.000	1,004.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
25%	250.750	1,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
50%	500.500	1,008.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
75%	750.250	1,009.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
max	1,000.000	1,020.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000

The statistical description is a useful tool to describe the data set. The Code above lists the Statistical Description of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10)
```

When compared to the mean value, the statistical points to determining the data's spread. Mean variation in data values occur as the standard deviation. The minimum and maximum values are the lowest and the most and the sum. The interval of the dispersion of the data values and the quartiles are also included.

```
# Copy of original
df_orig = df.copy
df.sample(10).T
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index()
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False)
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10)
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10).T
```

Then type the name of the data set and a line:

To keep our dataset clean we first need to make a copy of the original dataset

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10).T.sort_index()
```

Data preparation

Characterisation

The data set is ready to be used for data analysis purposes. They are represented by tables (data is presentation format where the rows are the details of individual records and the columns are the attributes of the records). Each row represents a single record and with the age and the hospitalisation of the records and the summary of its genetic characteristics or resources of the origin data set (Ma et al. 2011).

The data set contains 1000 records and 10 attributes (Table 1). The data set is available in the Kaggle repository (https://www.kaggle.com/datasets/jeffsinn/h3n2-bird-flu-detections?select=H3N2.csv) (Ma et al. 2011).

Data definitions

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.head(10)
```

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.tail(5)
```

	Variable Name	Variable Description	Data Type	Measure	Role	Value	Missing	Missing Data Coding	Notes	Coding	Coding Notes
1	Id	Unique Identifier	Numeric	Interval	Id	N	0	1000	Integer	0-1,000	Consecutive
2	Year	Year of Detection	Numeric	Interval	Input	N	0	1000	Integer	1984-2020	Consecutive
3	Hospitalised	Hospitalised	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
4	Hospitalised Days	Hospitalised Days	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
5	Age	Age	Numeric	Interval	Input	N	0	1000	Integer	0-1,000	Consecutive
6	Gender	Gender	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
7	Subtype	Subtype	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
8	Hemagglutinin	Hemagglutinin	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
9	Neuraminidase	Neuraminidase	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive
10	Reassortment	Reassortment	Nominal	Binary	Input	N	0	1000	Binary	0-1,000	Consecutive

The Table above lists the meaning and coding of each of the reported measures or portions of data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.shape
```

```
(1000, 10)
```

The Code above has the Shape of the data set (number of rows) - (text/csv).

```
# Copy of original
df_orig = df.copy
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 0   Id                  1000 non-null   int64
 1   Year                1000 non-null   int64
 2   Hospitalised        1000 non-null   int64
 3   Hospitalised Days   1000 non-null   int64
 4   Age                 1000 non-null   int64
 5   Gender              1000 non-null   int64
 6   Subtype             1000 non-null   int64
 7   Hemagglutinin       1000 non-null   int64
 8   Neuraminidase       1000 non-null   int64
 9   Reassortment        1000 non-null   int64
dtype: int64
memory usage: 78.1 KB
```

The Code above lists the Information of the data set (Number of columns, Number of rows, type of the data set)

```
# Copy of original
df_orig = df.copy
df.dtypes
```

```
Id                  int64
Year                int64
Hospitalised        int64
Hospitalised Days   int64
Age                 int64
Gender              int64
Subtype             int64
Hemagglutinin       int64
Neuraminidase       int64
Reassortment        int64
dtype: int64
```

The Code above lists the type of each column of the data set (Numeric or Binary).

```
# Copy of original
df_orig = df.copy
df.nunique()
```

```
Id                  1000
Year                  37
Hospitalised           2
Hospitalised Days     21
Age                   99
Gender                 2
Subtype                2
Hemagglutinin          2
Neuraminidase          2
Reassortment           2
dtype: int64
```

The Code above lists the number of unique elements in the object. It returns a pulse value which is the count of all the unique values in the object. df.nunique() - (text/csv).

```
# Copy of original
df_orig = df.copy
df.describe()
```

	Id	Year	Hospitalised	Hospitalised Days	Age	Gender	Subtype	Hemagglutinin	Neuraminidase	Reassortment
count	1000.0	1000.000	1000.0	1000.000	1000.000	1000.000	1000.000	1000.000	1000.000	1000.000
mean	500.5	1,017.773	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
std	288.819	1,009.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
min	1.000	1,004.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
25%	250.750	1,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
50%	500.500	1,008.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
75%	750.250	1,009.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000
max	1,000.000	1,020.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000	2,006.000

The statistical description is a useful tool to describe the data set. The Code above lists the Statistical Description of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10)
```

When compared to the mean value, the statistical points to determining the data's spread. Mean variation in data values occur as the standard deviation. The minimum and maximum values are the lowest and the most and the sum. The interval of the dispersion of the data values and the quartiles are also included.

```
# Copy of original
df_orig = df.copy
df.sample(10).T
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index()
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False)
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10)
```

The above lists 10 random rows and the transpose of the data set (Ahmad). (text/csv).

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10).T
```

Then type the name of the data set and a line:

To keep our dataset clean we first need to make a copy of the original dataset

```
# Copy of original
df_orig = df.copy
df.sample(10).T.sort_index(ascending=False).sample(10).T.sort_index()
```

Data Cleaning

In the Jupiter Notebook platform we can use the Pandas Profiling Report to create a report for the data set. The Code above lists the Pandas Profiling Report (text/csv).

```
# Pandas Profiling
!pip install -U pandas_profiling
```

```
Requirement already satisfied: pandas_profiling[notebook,widgets] in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: ipywidgets>=7.5.1 in /usr/local/lib/python3.7/dist-packages (7.6.5)
Requirement already satisfied: jinja2>=2.11.1 in /usr/local/lib/python3.7/dist-packages (2.11.3)
Requirement already satisfied: MarkupSafe>=2.0.1 in /usr/local/lib/python3.7/dist-packages (2.0.1)
Requirement already satisfied: matplotlib>=3.2.1 in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.0.3,!=1.1.0,>=0.25.3 in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: pandarallel>=1.5.2 in /usr/local/lib/python3.7/dist-packages (1.6.5)
Requirement already satisfied: pydantic>=1.8.1 in /usr/local/lib/python3.7/dist-packages (1.9.0)
Requirement already satisfied: PyYAML>=5.0.0 in /usr/local/lib/python3.7/dist-packages (6.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (1.7.3)
Requirement already satisfied: seaborn>=0.10.1,<0.11.1 in /usr/local/lib/python3.7/dist-packages (0.11.0)
Requirement already satisfied: tqdm>=4.48.2 in /usr/local/lib/python3.7/dist-packages (4.64.0)
Requirement already satisfied: requests>=2.24.0 in /usr/local/lib/python3.7/dist-packages (2.27.1)
Requirement already satisfied: htmlmin>=0.1 in /usr/local/lib/python3.7/dist-packages (0.1.12)
Requirement already satisfied: phik>=0.11.1 in /usr/local/lib/python3.7/dist-packages (0.12.2)
Requirement already satisfied: tangramvis>=1.0.0 in /usr/local/lib/python3.7/dist-packages (1.0.0)
Requirement already satisfied: imagehash>=4.3.1 in /usr/local/lib/python3.7/dist-packages (4.3.1)
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.7/dist-packages (1.9.1)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.7/dist-packages (1.8.1)
Requirement already satisfied: numba>=0.56.0 in /usr/local/lib/python3.7/dist-packages (0.56.0)
Requirement already satisfied: multimethod>=1.4 in /usr/local/lib/python3.7/dist-packages (1.9)
Requirement already satisfied: visions[type_image_path]>=0.7.5 in /usr/local/lib/python3.7/dist-packages (0.7.5)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.7/dist-packages (4.10.1)
Requirement already satisfied: ipython>=4.0.0; sys_platform != 'win32' in /usr/local/lib/python3.7/dist-packages (7.34.0)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: jupyterlab_widgets>=1.0.0; python_version >= '3.6' in /usr/local/lib/python3.7/dist-packages (1.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (1.4.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (3.0.7)
Requirement already satisfied: typing_extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (4.2.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (2022.1)
Requirement already satisfied: pyarrow>=0.15.0 in /usr/local/lib/python3.7/dist-packages (6.0.1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.7/dist-packages (1.1.0)
Requirement already satisfied: psutil>=5.7.0 in /usr/local/lib/python3.7/dist-packages (5.4.8)
Requirement already satisfied: dill>=0.3.1.1 in /usr/local/lib/python3.7/dist-packages (0.3.4)
Requirement already satisfied: cloudpickle>=1.3.0 in /usr/local/lib/python3.7/dist-packages (2.0.0)
Requirement already satisfied: tqdm-joblib>=0.3.0 in /usr/local/lib/python3.7/dist-packages (0.3.3)
Requirement already satisfied: urllib3>=1.26.9 in /usr/local/lib/python3.7/dist-packages (1.26.9)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (2022.5.18.1)
Requirement already satisfied: charset-normalizer>=2.0.12 in /usr/local/lib/python3.7/dist-packages (2.0.12)
Requirement already satisfied: idna>=2.5 in /usr/local/lib/python3.7/dist-packages (3.3)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.7/dist-packages (2.6.3)
Requirement already satisfied: PywaveletTools>=1.1.1 in /usr/local/lib/python3.7/dist-packages (1.1.1)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.7/dist-packages (1.3.0)
Requirement already satisfied: matplotlib-inline>=0.1.5 in /usr/local/lib/python3.7/dist-packages (0.1.3)
Requirement already satisfied: ipyfrontend>=0.4.4 in /usr/local/lib/python3.7/dist-packages (0.4.4)
Requirement already satisfied: ipyflowtracer>=0.0.1 in /usr/local/lib/python3.7/dist-packages (0.0.1)
Requirement already satisfied: ipyvolume>=0.4.6 in /usr/local/lib/python3.7/dist-packages (0.4.6)
Requirement already satisfied: ipyleaflet>=0.13.0 in /usr/local/lib/python3.7/dist-packages (0.13.0)
Requirement already satisfied: ipympl>=0.2.0 in /usr/local/lib/python3.7/dist-packages (0.2.0)
Requirement already satisfied: ipysheet>=0.4.3 in /usr/local/lib/python3.7/dist-packages (0.4.3)
Requirement already satisfied: ipytree>=0.2.2 in /usr/local/lib/python3.7/dist-packages (0.2.2)
Requirement already satisfied: jupyterlab>=3.0.0 in /usr/local/lib/python3.7/dist-packages (3.3.2)
Requirement already satisfied: jupyterlab_git>=0.30.1 in /usr/local/lib/python3.7/dist-packages (0.30.1)
Requirement already satisfied: jupyterlab_hubmanager>=0.11.3 in /usr/local/lib/python3.7/dist-packages (0.11.3)
Requirement already satisfied: jupyterlab_latex>=0.4.2 in /usr/local/lib/python3.7/dist-packages (0.4.2)
Requirement already satisfied: jupyterlab_nbextensions_configurator>=0.4.1 in /usr/local/lib/python3.7/dist-packages (0.4.1)
Requirement already satisfied: lxml>=4.2.1 in /usr/local/lib/python3.7/dist-packages (4.6.3)
Requirement already satisfied: nbclassic>=0.2.8 in /usr/local/lib/python3.7/dist-packages (0.3.7)
Requirement already satisfied: nbdime>=2.1.0 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: nbgrader>=0.6.1 in /usr/local/lib/python3.7/dist-packages (0.6.1)
Requirement already satisfied: notebook>=5.2.0 in /usr/local/lib/python3.7/dist-packages (5.7.16)
Requirement already satisfied: pandocfilters>=1.4.2 in /usr/local/lib/python3.7/dist-packages (1.5.1)
Requirement already satisfied: prompt_toolkit>=1.0.15 in /usr/local/lib/python3.7/dist-packages (3.0.29)
Requirement already satisfied: qtconsole>=4.2.1 in /usr/local/lib/python3.7/dist-packages (5.3.1)
Requirement already satisfied: qtpyvisualizer>=0.0.1 in /usr/local/lib/python3.7/dist-packages (0.0.1)
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.7/dist-packages (1.0.2)
Requirement already satisfied: scikit-image>=0.14.2 in /usr/local/lib/python3.7/dist-packages (0.18.3)
Requirement already satisfied: sklearn>=0.0 in /usr/local/lib/python3.7/dist-packages (0.0)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (0.13.2)
Requirement already satisfied: sympy>=1.0 in /usr/local/lib/python3.7/dist-packages (1.7.1)
Requirement already satisfied: vega>=4.3.0 in /usr/local/lib/python3.7/dist-packages (4.4.0)
Requirement already satisfied: vega_datasets>=0.4.1 in /usr/local/lib/python3.7/dist-packages (0.4.1)
Requirement already satisfied: widgetsnbextension>=3.5.1 in /usr/local/lib/python3.7/dist-packages (3.5.1)
Requirement already satisfied: xgboost>=0.82 in /usr/local/lib/python3.7/dist-packages (0.82)
Requirement already satisfied: zipcodes>=1.0.1 in /usr/local/lib/python3.7/dist-packages (1.0.1)
Requirement already satisfied: zipfile36>=0.1.3 in /usr/local/lib/python3.7/dist-packages (0.1.3)
Requirement already satisfied: ipywidgets>=7.5.1 in /usr/local/lib/python3.7/dist-packages (7.6.5)
Requirement already satisfied: jinja2>=2.11.1 in /usr/local/lib/python3.7/dist-packages (2.11.3)
Requirement already satisfied: MarkupSafe>=2.0.1 in /usr/local/lib/python3.7/dist-packages (2.0.1)
Requirement already satisfied: matplotlib>=3.2.1 in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.0.3,!=1.1.0,>=0.25.3 in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: pandarallel>=1.5.2 in /usr/local/lib/python3.7/dist-packages (1.6.5)
Requirement already satisfied: pydantic>=1.8.1 in /usr/local/lib/python3.7/dist-packages (1.9.0)
Requirement already satisfied: PyYAML>=5.0.0 in /usr/local/lib/python3.7/dist-packages (6.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (1.7.3)
Requirement already satisfied: seaborn>=0.10.1,<0.11.1 in /usr/local/lib/python3.7/dist-packages (0.11.0)
Requirement already satisfied: tqdm>=4.48.2 in /usr/local/lib/python3.7/dist-packages (4.64.0)
Requirement already satisfied: requests>=2.24.0 in /usr/local/lib/python3.7/dist-packages (2.27.1)
Requirement already satisfied: htmlmin>=0.1 in /usr/local/lib/python3.7/dist-packages (0.1.12)
Requirement already satisfied: phik>=0.11.1 in /usr/local/lib/python3.7/dist-packages (0.12.2)
Requirement already satisfied: tangramvis>=1.0.0 in /usr/local/lib/python3.7/dist-packages (1.0.0)
Requirement already satisfied: imagehash>=4.3.1 in /usr/local/lib/python3.7/dist-packages (4.3.1)
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.7/dist-packages (1.9.1)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.7/dist-packages (1.8.1)
Requirement already satisfied: numba>=0.56.0 in /usr/local/lib/python3.7/dist-packages (0.56.0)
Requirement already satisfied: multimethod>=1.4 in /usr/local/lib/python3.7/dist-packages (1.9)
Requirement already satisfied: visions[type_image_path]>=0.7.5 in /usr/local/lib/python3.7/dist-packages (0.7.5)
```

```
# Pandas Profiling Report
df.profile_report()
```

```
Summarize dataset: 100%|          | 5/5 [00:00<00:00, 1.00it/s]
Generate report structure: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
Render HTML: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
```

```
# Pandas Profiling Report
df.profile_report(sort="None")
```

```
Summarize dataset: 100%|          | 5/5 [00:00<00:00, 1.00it/s]
Generate report structure: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
Render HTML: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
```

```
# Pandas Profiling Report
df.profile_report(sort="ascending")
```

```
Summarize dataset: 100%|          | 5/5 [00:00<00:00, 1.00it/s]
Generate report structure: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
Render HTML: 100%|          | 1/1 [00:00<00:00, 1.00it/s]
```

```
```

```
# # Geographical plot
# plt.figure(figsize=(10,5))
# df_temp = df[(df['Year'] >= 2011)]
# m = Basemap(projection='cyl',
# llcrnrlat=-90,
# urcrnrlat=90,
# llcrnrlon=-180,
# urcrnrlon=180,
# lat_ts=0,
# resolution='c')
# x, y = m(df_temp['Longitude'].tolist(), df_temp['Latitude'].tolist())
# m.plot(x, y, "o", markersize = 5, color = 'blue')
# m.drawcoastlines(linewidth=0.5)
# m.drawmapboundary(fillcolor='aqua')
# plt.title("Locations of Avian Flu Outbreaks in Poultry (2011-2019)",size=20)
```

The Geographic plot above illustrates the map of the outbreaks in Avian Influenza in poultry since the year 2011 with the most recent outbreaks and the first outbreaks plot on the same plot with the legend.

```
# df_temp = df[(df['Year'] >= 2011)]
# df_temp.shape
```

```
(16904, 18)
```

```
# df_temp.sample(5)
```

	Year	Month	Day	Subtype	Serotype	Quantity	Unit	Species	Production type	Control measures	Latitude	Longitude	Country	Region	Continent	Date	Cases	Control measures numbers
12	2011	3	15	H5N1	NaN	1.0	birds	Domestic birds	Unknown	Culling	14.0000	108.0000	Vietnam	Dong Nai	Asia	2011-03-15	1.0	1.0
16900	2019	12	31	H5N8	NaN	1.0	birds	Domestic birds	Unknown	Culling	37.0000	127.0000	Korea, Republic of (South Korea)	Gyeonggi-do	Asia	2019-12-31	1.0	1.0
16901	2019	12	31	H5N8	NaN	1.0	birds	Domestic birds	Unknown	Culling	37.0000	127.0000	Korea, Republic of (South Korea)	Gyeonggi-do	Asia	2019-12-31	1.0	1.0
16902	2019	12	31	H5N8	NaN	1.0	birds	Domestic birds	Unknown	Culling	37.0000	127.0000	Korea, Republic of (South Korea)	Gyeonggi-do	Asia	2019-12-31	1.0	1.0
16903	2019	12	31	H5N8	NaN	1.0	birds	Domestic birds	Unknown	Culling	37.0000	127.0000	Korea, Republic of (South Korea)	Gyeonggi-do	Asia	2019-12-31	1.0	1.0

```
# df_temp.dtypes
```

```
Year                    int64
Month                   int64
Day                     int64
Subtype                object
Serotype               object
Quantity               float64
Unit                   object
Species                object
Production type        object
Control measures       object
Latitude               float64
Longitude              float64
Country                object
Region                 object
Continent              object
Date                   object
Cases                  float64
Control measures numbers float64
dtype: object
```

```
# df_temp.describe()
```

	Year	Month	Day	Quantity	Latitude	Longitude	Cases	Control measures numbers
count	16904.000000	16904.000000	16904.000000	16904.000000	16904.000000	16904.000000	16904.000000	16904.000000
mean	2015.000000	6.000000	15.000000	1.000000	14.000000	108.000000	1.000000	1.000000
std	2.000000	3.000000	8.000000	0.000000	0.000000	0.000000	0.000000	0.000000
min	2011.000000	1.000000	1.000000	1.000000	14.000000	108.000000	1.000000	1.000000
25%	2013.000000	3.000000	8.000000	1.000000	14.000000	108.000000	1.000000	1.000000
50%	2015.000000	6.000000	15.000000	1.000000	14.000000	108.000000	1.000000	1.000000
75%	2017.000000	9.000000	22.000000	1.000000	14.000000	108.000000	1.000000	1.000000
max	2019.000000	12.000000	31.000000	1.000000	14.000000	108.000000	1.000000	1.000000

What species have shown to be the most affected in the Avian Influenza outbreaks?

```
# df_temp.Species.value_counts()
```

```
Domestic birds    16904
Name: Species, dtype: int64
```

How many wild birds were infected with avian flu?

Only 16% of the total cases in the Avian Influenza dataset were found to be linked and most were not reported for the H5N1 Strain.

```
# # Wild Birds
# df_temp[df_temp.Species.str.contains('Wild', na=False)].shape
```

```
(2700, 18)
```

```
# # Healthy Birds
# df_temp[df_temp.Species.str.contains('Domestic', na=False)].shape
```

```
(14204, 18)
```

What species have shown to be the most affected in the Avian Influenza outbreaks?

```
# df_temp[df_temp.Species.str.contains('Wild', na=False)].Species.value_counts()
```

```
Wild birds                                                    2700
Name: Species, dtype: int64
```

```
# df_temp[df_temp.Species.str.contains('Domestic', na=False)].Species.value_counts()
```

```
Domestic birds                                                14204
Name: Species, dtype: int64
```

```
# df_temp[df_temp.Species.str.contains('Wild', na=False)].Species.value_counts()[:5].plot.bar(rot=0)
# plt.title("Species captured where Avian Flu has been detected",size=20)
# plt.xlabel("Species",size=15)
# plt.ylabel("Cases",size=15)
# plt.xticks(rotation=90,size=12)
# plt.yticks(size=12)
# plt.grid(True)
# plt.show
```

```
Search for, in, the, Bracket
```

In this graph we can see where the minimum value of the dataset starts and ends and the maximum value, as well as the histogram. The minimum value of the dataset is 0 and the maximum value is 165 (represented by the highest bar). We can also see the general trend of the dataset.

```
# 2.2.2.Box plot for target levels in ordering of the "Count" and "Year" variables and "Country"
ggplot(data=Dataset,aes(x=reorder(Year,-Count),y=Count))+
```

- Most County numbers tended to birth, captured are concentrated between the year of 2000 and 2020

```
# 2.2.3.Pie chart of Country
ggplot(data=Dataset,aes(x="",fill=Country))+geom_bar(width=1)+
coord_polar(theta="y")+theme_void()+ggtitle("Pie chart of Country")+scale_fill_hue(l=50,c=100)
```

In this pie chart we can also see the frame of the Country is distributed by percentage and the most that appear are:

- Timor Leste
- Cabo Verde
- Angola
- Sao Tome and Principe
- Macau

County per Year

```
# 2.2.4.Histogram of Year
ggplot(data=Dataset,aes(x=Year))+geom_histogram(bins=30)+
```

```
# 2.2.5.Box plot County per Year
ggplot(data=Dataset,aes(x=reorder(Year,-Count),y=Count))+
```

```
# 2.2.6.Bar plot Country
ggplot(data=Dataset,aes(x=Country))+geom_bar()+
```

```
# 2.2.7.Correlation matrix
Corr_Dataset=cor(Dataset[,c("Latitude","Longitude","Month","Year")])
# 2.2.8.Correlation plot Dataset
corrplot(Corr_Dataset)
```

```
Error in corrplot(Corr_Dataset) :
  could not find function "corrplot"
```

```
# 2.2.9.Correlation plot
corrplot(Corr_Dataset,method="number")
```

```
Error in corrplot(Corr_Dataset, method = "number") :
  could not find function "corrplot"
```

As we can see in the graph above we have written a non-functional line, or the data is not in the right form.

Data Cleaning

Missing values

In order to get the data cleaning so that will need to find the missing values, using the function, and as we detect cleaning values for an new dataset. The first column is called a string that begin and ending values are missing. Match is present among Year or Country. Year is a numeric variable and Country is a factor variable. Country is a factor variable with 10 levels. Country is a factor variable with 10 levels. Country is a factor variable with 10 levels

```
Length: 2444 lines | 5.7k words | 34.6k chars | 1h 9min
```

```
# Import the Cancer Data
df = pd.read_csv('./cancer.csv')
df.shape
df.columns
df.dtypes
df.head()
df.tail()
df.describe()
df.describe(include='object')
df['diagnosis'].value_counts()
df['diagnosis'].value_counts(normalize=True)
```

```
(569, 33)
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
id                 int64
diagnosis         object
radius_mean      float64
texture_mean     float64
perimeter_mean   float64
area_mean        float64
smoothness_mean  float64
compactness_mean float64
concavity_mean   float64
concave points_mean float64
symmetry_mean    float64
fractal_dimension_mean float64
radius_se        float64
texture_se       float64
perimeter_se     float64
area_se          float64
smoothness_se    float64
compactness_se   float64
concavity_se     float64
concave points_se float64
symmetry_se      float64
fractal_dimension_se float64
radius_worst     float64
texture_worst    float64
perimeter_worst  float64
area_worst       float64
smoothness_worst float64
compactness_worst float64
concavity_worst  float64
concave points_worst float64
symmetry_worst   float64
fractal_dimension_worst float64
Unnamed: 32      float64
dtype: int64
```

```
# Check for NaN values
df.isnull().sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

```
# Drop the column with all NaN values
df = df.drop('Unnamed: 32',axis=1)
df.shape
```

```
(569, 32)
```

Decision Tree

For classification and regression, Decision Trees (DTs) are non-parametric supervised learning methods. The goal is to learn simple decision rules from data attributes to develop a model that predicts the value of a target variable. A tree is an approximation to a piecewise constant function.


```
# Create a copy of the original data
df_copy = df.copy()
df_copy = df_copy.drop('id',axis=1)
df_copy.shape
```

```
(569, 31)
```

```
# Create a list of columns to be used for the model
cols = df_copy.columns
cols
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

```
# Use this "copy" to get dummies/numerics if desired only
df = df_copy.copy()
df = df[cols].copy()
df.shape, df.columns[:5]
```

```
((569, 31),
 Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
        'area_mean'],
       dtype=object))
```

Splitting the dataset into the Training set and Test set

```
# Import train_test_split
from sklearn.model_selection import train_test_split
# Split the data into X (features) and y (target)
X = df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean']]
y = df['diagnosis']
# Split into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
print("Features:", X_train.columns)
```

```
Training set size: 455
Test set size: 114
Features: Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean'],
      dtype='object')
```

Training the Decision Tree Classification model on the Training set

```
# Import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
# Create and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

Predicting the Test set results

```
# Create a comparison dataframe
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(comparison.head())
```

```
     Actual Predicted
521       B         B
402       M         M
232       B         B
549       B         B
299       M         M
```

Making the Confusion Matrix

```
# Import confusion_matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print("                 Predicted")
print("                 B   M")
print("Actual B      {:3}{:3}".format(cm[0,0], cm[0,1]))
print("       M      {:3}{:3}".format(cm[1,0], cm[1,1]))
```

```
Confusion Matrix:
                 Predicted
                 B   M
Actual B       71  1
       M        2 40
```

```
# Import classification_report
from sklearn.metrics import classification_report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           B       0.97      0.99      0.98        72
           M       0.98      0.95      0.96        42

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

```
# Import tree visualization
from sklearn.tree import plot_tree
plt.figure(figure=(20,10))
plot_tree(clf, feature_names=X.columns, class_names=['B', 'M'], filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```



```
# Import feature_importance
feature_importance = clf.feature_importances_
print("Feature Importance:")
for feature, importance in zip(X.columns, feature_importance):
    print(f"{feature}: {importance:.4f}")
```

```
Feature Importance:
radius_mean: 0.000
texture_mean: 0.000
perimeter_mean: 0.000
area_mean: 0.000
smoothness_mean: 1.000
```



```
# Import cross_validation
from sklearn.model_selection import cross_val_score
# Perform 5-fold cross-validation
cv_scores = cross_val_score(clf, X, y, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation:", cv_scores.std())
```

```
Cross-validation scores: [0.91228070 0.94736842 0.94736842 0.96491228 0.96460177]
Mean CV score: 0.9453
Standard deviation: 0.0181
```

```
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV
# Define parameter grid
param_grid = {
    'max_depth': [3, 4, 5, 6, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# Create and fit GridSearchCV
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5)
grid_search.fit(X, y)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)
```

```
Best parameters: {'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best cross-validation score: 0.9473
```

```
# Import tree_to_code
from sklearn.tree import export_text
# Convert tree to text representation
tree_rules = export_text(clf, feature_names=X.columns)
print("Decision Tree Rules:")
print(tree_rules)
```

```
Decision Tree Rules:
|--- perimeter_mean <= 116.65
|   |--- area_mean <= 695.50
|   |   |--- smoothness_mean <= 0.096
|   |   |   |--- class: B
|   |   |--- smoothness_mean >  0.096
|   |   |   |--- class: B
|   |--- area_mean >  695.50
|   |   |--- smoothness_mean <= 0.098
|   |   |   |--- class: B
|   |   |--- smoothness_mean >  0.098
|   |   |   |--- class: B
|--- perimeter_mean >  116.65
|   |--- area_mean <= 957.50
|   |   |--- smoothness_mean <= 0.098
|   |   |   |--- class: B
|   |   |--- smoothness_mean >  0.098
|   |   |   |--- class: M
|   |--- area_mean >  957.50
|   |   |--- smoothness_mean <= 0.098
|   |   |   |--- class: B
|   |   |--- smoothness_mean >  0.098
|   |   |   |--- class: M
```

```
# Import numpy
import numpy as np
# Calculate feature importance ranking
feature_ranking = np.argsort(feature_importance)[::-1]
print("Feature Ranking:")
for i, feature in zip(range(5), X.columns[feature_ranking]):
    print(f{i+1}. {feature}")
```

```
Feature Ranking:
1. smoothness_mean
2. area_mean
3. perimeter_mean
4. texture_mean
5. radius_mean
```

```
# Import pickle
import pickle
# Save the model
with open('decision_tree_model.pkl', 'wb') as f:
    pickle.dump(clf, f)
print("Model saved as decision_tree_model.pkl")
```

```
Model saved as decision_tree_model.pkl
```

```
# Import time
import time
# Measure prediction time
start_time = time.time()
predictions = clf.predict(X_test)
end_time = time.time()
prediction_time = end_time - start_time
print("Prediction time:", prediction_time, "seconds")
```

```
Prediction time: 0.0010 seconds
```

```
# Import sklearn version
import sklearn
print("Sklearn version:", sklearn.__version__)
```

```
Sklearn version: 1.0.2
```

```
# Import matplotlib
import matplotlib
print("Matplotlib version:", matplotlib.__version__)
```

```
Matplotlib version: 3.5.1
```

```
# Import seaborn
import seaborn
print("Seaborn version:", seaborn.__version__)
```

```
Seaborn version: 0.11.2
```

```
# Import numpy
import numpy
print("Numpy version:", numpy.__version__)
```

```
Numpy version: 1.21.6
```

```
# Import pandas
import pandas
print("Pandas version:", pandas.__version__)
```

```
Pandas version: 1.3.5
```

K-NEAREST NEIGHBORS (KNN)

KNN uses it to locate related data points.

- Classification method
- Supervised Learning (Training data has a label)
- Not for big data (slow)
- Sensitive to irrelevant features
- Sensitive to outliers (missing values)

Use to implement it with sklearn

- K-Nearest
- A large variable found in Sklearn library
- Use "StandardScaler" to normalize data and put in the 0 based detailed data points.

```
# Import standard libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
df = pd.read_csv('./cancer.csv')
# Display basic information
print("Dataset shape:", df.shape)
print("Columns:", df.columns[:5])
```

```
Dataset shape: (569, 33)
Columns: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean',
       'perimeter_mean'],
      dtype=object)
```

K-NEAREST NEIGHBORS (KNN)

KNN uses it to locate related data points.

- Classification method
- Supervised Learning (Training data has a label)
- Not for big data (slow)
- Sensitive to irrelevant features
- Sensitive to outliers (missing values)

Use to implement it with sklearn

- K-Nearest
- A large variable found in Sklearn library
- Use "StandardScaler" to normalize data and put in the 0 based detailed data points.

```
# Import standard libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
df = pd.read_csv('./cancer.csv')
# Display basic information
print("Dataset shape:", df.shape)
print("Columns:", df.columns[:5])
```

```
Dataset shape: (569, 33)
Columns: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean',
       'perimeter_mean'],
      dtype=object)
```

```
# Display first few rows
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst	Unnamed: 32
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.095	0.9053	8.589	153.40	0.006399	0.04904	0.05373	0.01587	0.03003	0.006193	25.38	17.33	184.60	2019.0	0.16200	0.66560	0.7119	0.2654	0.4601	0.11890	NaN
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.01860	0.00134	0.01389	0.003532	24.99	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	NaN
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006	0.03832	0.02058	0.02250	0.004571	23.57	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	NaN
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.156	3.445	27.23	0.009110	0.07458	0.05661	0.01867	0.05963	0.009208	14.91	26.50	98.87	567.7	0.20980	0.86663	0.6869	0.2575	0.6638	0.17300	NaN
4	84358402	M	20.29	14.34	135.10	1297.0	0.10003	0.13280	0.1980	0.10043	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461	0.05688	0.01885	0.01756	0.005115	22.54	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	NaN

```
# Check data types
df.dtypes
```

```
id                         int64
diagnosis                  object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
Unnamed: 32                float64
dtype: int64
```

```
# Check for missing values
df.isnull.sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

```
# Drop the column with all NaN values
df = df.drop('Unnamed: 32',axis=1)
df.shape
```

```
(569, 32)
```

Machine Learning

By observing our dataset we can conclude that the dataset is supervised and it contains variables with labels for example the "diagnosis" column. Machine learning algorithms will use these labels to learn patterns.

Columns: id(int), diagnosis(object), radius_mean(float), texture_mean(float), perimeter_mean(float), area_mean(float), smoothness_mean(float), compactness_mean(float), concavity_mean(float), concave points_mean(float), symmetry_mean(float), fractal_dimension_mean(float), radius_se(float), texture_se(float), perimeter_se(float), area_se(float), smoothness_se(float), compactness_se(float), concavity_se(float), concave points_se(float), symmetry_se(float), fractal_dimension_se(float), radius_worst(float), texture_worst(float), perimeter_worst(float), area_worst(float), smoothness_worst(float), compactness_worst(float), concavity_worst(float), concave points_worst(float), symmetry_worst(float), fractal_dimension_worst(float), Unnamed: 32(float)

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.095	0.9053	8.589	153.40	0.006399	0.04904	0.05373	0.01587	0.03003	0.006193	25.38	17.33	184.60	2019.0	0.16200	0.66560	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.01860	0.00134	0.01389	0.003532	24.99	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006	0.03832	0.02058	0.02250	0.004571	23.57	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.156	3.445	27.23	0.009110	0.07458	0.05661	0.01867	0.05963	0.009208	14.91	26.50	98.87	567.7	0.20980	0.86663	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10003	0.13280	0.1980	0.10043	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461	0.05688	0.01885	0.01756	0.005115	22.54	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678

```
df.tail()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2439	0.13890	0.1726	0.05623	1.176	1.256	7.673	158.70	0.010090	0.03113	0.04239	0.01642	0.02548	0.00421	25.45	26.40	166.10	2027.0	0.14100	0.21100	0.4264	0.2011	0.3259	0.07779
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1440	0.09791	0.1752	0.05533	0.7655	2.463	5.203	99.04	0.005769	0.02423	0.03950	0.01678	0.01898	0.002498	23.69	38.25	155.00	1731.0	0.11660	0.19922	0.3202	0.1565	0.3574	0.09462
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10023	0.0937	0.05930	0.1592	0.05884	0.4564	1.075	3.425	48.55	0.005726	0.02250	0.02636	0.00915	0.01989	0.003148	18.98	34.12	126.70	1124.0	0.11390	0.30494	0.3827	0.1489	0.3232	0.08777
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3514	0.15200	0.2397	0.07016	0.726	1.595	5.772	86.22	0.006436	0.04658	0.07089	0.02548	0.02980	0.005283	24.90	35.64	177.00	2215.0	0.14840	0.65900	0.7148	0.2226	0.4087	0.12490
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0000	0.00000	0.1587	0.05884	0.3857	1.428	2.548	19.15	0.007189	0.00466	0.00000	0.00000	0.02676	0.002783	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039

```
# Check data types
df.dtypes
```

```
id                         int64
diagnosis                  object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
Unnamed: 32                float64
dtype: int64
```

```
# Check for missing values
df.isnull.sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

```
# Drop the column with all NaN values
df = df.drop('Unnamed: 32',axis=1)
df.shape
```

```
(569, 32)
```

Measuring model performance

- Accuracy which is fraction of correct predictions is commonly used metric.

Model accuracy alone can be misleading when we work with an imbalanced data set.

Even we could work out by finding each made predictions on test set.

But that's not the end. There also exist several metrics that are considered.

- Area under curve (AUC) and ROC curve (AUC stands for Area Under the Curve)
- Confusion matrix
- Precision, Recall, F1 score
  - A target variable (Cancer, Fraud, Churn, Windows, Sick, Medicine) Cases, Storage, Lung Disease/Allergy, Unsure, Trypanosomiasis, Malaria, Dengue, Schistosomiasis, Filiria, Kidney, Metamorph, Mycoplasma, Carbon, Cancer, Leukemia, Lymphoma
  - Classification problem and binary classification (sick vs not sick, churn vs not churn, cancer vs not cancer)
  - Classification with more than two classes (sick vs not sick, sick vs healthy, sick vs healthy and injured)
  - Confusion matrix
  - Accuracy, precision, recall, f1 score and more

Go for cross validation.

Measuring model performance

- Accuracy which is fraction of correct predictions is commonly used metric.

Model accuracy alone can be misleading when we work with an imbalanced data set.

Even we could work out by finding each made predictions on test set.

But that's not the end. There also exist several metrics that are considered.

- Area under curve (AUC) and ROC curve (AUC stands for Area Under the Curve)
- Confusion matrix
- Precision, Recall, F1 score
  - A target variable (Cancer, Fraud, Churn, Windows, Sick, Medicine) Cases, Storage, Lung Disease/Allergy, Unsure, Trypanosomiasis, Malaria, Dengue, Schistosomiasis, Filiria, Kidney, Metamorph, Mycoplasma, Carbon, Cancer, Leukemia, Lymphoma
  - Classification problem and binary classification (sick vs not sick, churn vs not churn, cancer vs not cancer)
  - Classification with more than two classes (sick vs not sick, sick vs healthy, sick vs healthy and injured)
  - Confusion matrix
  - Accuracy, precision, recall, f1 score and more

Go for cross validation.

```
# Import sklearn
import sklearn
print("Sklearn version:", sklearn.__version__)
```

```
Sklearn version: 1.0.2
```

```
# Import matplotlib
import matplotlib
print("Matplotlib version:", matplotlib.__version__)
```

```
Matplotlib version: 3.5.1
```

```
# Import seaborn
import seaborn
print("Seaborn version:", seaborn.__version__)
```

```
Seaborn version: 0.11.2
```

```
# Import numpy
import numpy
print("Numpy version:", numpy.__version__)
```

```
Numpy version: 1.21.6
```

```
# Import pandas
import pandas
print("Pandas version:", pandas.__version__)
```

```
Pandas version: 1.3.5
```

```
# Import numpy
import numpy as np
# Calculate feature importance ranking
feature_ranking = np.argsort(feature_importance)[::-1]
print("Feature Ranking:")
for i, feature in zip(range(5), X.columns[feature_ranking]):
    print(f{i+1}. {feature}")
```

```
Feature Ranking:
1. smoothness_mean
2. area_mean
3. perimeter_mean
4. texture_mean
5. radius_mean
```

```
# Import pickle
import pickle
# Save the model
with open('decision_tree_model.pkl', 'wb') as f:
    pickle.dump(clf, f)
print("Model saved as decision_tree_model.pkl")
```

```
Model saved as decision_tree_model.pkl
```

```
# Import time
import time
# Measure prediction time
start_time = time.time()
predictions = clf.predict(X_test)
end_time = time.time()
prediction_time = end_time - start_time
print("Prediction time:", prediction_time, "seconds")
```

```
Prediction time: 0.0010 seconds
```

```
# Import sklearn version
```

Regularized Regression

Regularization is a technique used to avoid the case of the feature is important, it gives high coefficient to the feature. However, this can cause overfitting and low accuracy. Using the data from the DHS, in order to avoid overfitting, we can regularize the parameter that prevent large coefficients.

Ridge regression: Ridge regularization technique (L2 regularization).

- Ridge regression loss function = OLS + sum of squared coefficients
- Ridge regression tries to minimize the residual sum of squares of regression plus the product of the shrinkage parameter and the squared sum of the coefficients.
- Ridge regression can't zero out coefficients
- It gives a big bias on some coefficients
- It gives a big bias on some coefficients

Lasso regression: Lasso regularization technique (L1 regularization)

- Lasso regression loss function = OLS + sum of absolute value of coefficients
- It is used to select important features of a dataset.
- It can zero out coefficients

Lasso or Ridge or Lasso from Regression Lasso Feature Selection Lasso 2 Ridge Regression model 1 Ridge Lasso 3 Lasso

```
# 10-fold
from sklearn.linear_model import RidgeCV
ridge = RidgeCV(alphas=np.arrow(0.1, 30, 0.05),
                cv=10,
                fit_intercept=True,
                normalize=True,
                scoring='neg_mean_squared_error')
ridge = ridge.fit(X_train, y_train)
print('Ridge Alpha: ', ridge.alpha_)
ridge_score = cross_val_score(ridge, X_train, y_train, cv=10)
print('Ridge Accuracy: ', np.mean(ridge_score))

Ridge Alpha: 29.95000000000001
Ridge Accuracy: -0.001149011754447

# Lasso
from sklearn.linear_model import LassoCV
lasso = LassoCV(alphas=np.arrow(0.1, 30, 0.05),
                cv=10,
                fit_intercept=True,
                normalize=True,
                verbose=False,
                n_jobs=-1,
                random_state=None,
                selection='cyclic')
lasso = lasso.fit(X_train, y_train)
print('Lasso Alpha: ', lasso.alpha_)
lasso_score = cross_val_score(lasso, X_train, y_train, cv=10)
print('Lasso Accuracy: ', np.mean(lasso_score))

Lasso Alpha:  0.1
Lasso Accuracy:  -0.001149011754447

# Elastic Net
from sklearn.linear_model import ElasticNetCV
elastic = ElasticNetCV(alpha=None,
                       cv=10,
                       fit_intercept=True,
                       normalize=True,
                       verbose=False,
                       n_jobs=-1,
                       random_state=None,
                       selection='cyclic')
elastic = elastic.fit(X_train, y_train)
print('ElasticNet Alpha: ', elastic.alpha_)
elastic_score = cross_val_score(elastic, X_train, y_train, cv=10)
print('ElasticNet Accuracy: ', np.mean(elastic_score))

ElasticNet Alpha:  0.1
ElasticNet Accuracy:  -0.001149011754447

# Summary
print('Ridge Accuracy: ', np.mean(ridge_score))
print('Lasso Accuracy: ', np.mean(lasso_score))
print('ElasticNet Accuracy: ', np.mean(elastic_score))

Ridge Accuracy:  -0.001149011754447
Lasso Accuracy:  -0.001149011754447
ElasticNet Accuracy:  -0.001149011754447
```

Ridge and Lasso regression are some of the most popular regression techniques. Ridge regression reduces the size of the coefficients, but does not zero them out. Lasso regression does zero out coefficients, which can be used for feature selection. Elastic Net combines the two approaches, using both penalties. Cross-validation is used to select the optimal regularization strength for each method. The results show that all three methods achieve similar performance on this dataset, with negligible improvement over the baseline model. This suggests that the linear relationships between the features and the target are not strong enough for regularization to provide significant benefits.

```
# Feature Selection
# Lasso Feature Selection
from sklearn.linear_model import LassoCV
lasso = LassoCV(alphas=np.arrow(0.1, 30, 0.05),
                cv=10,
                fit_intercept=True,
                normalize=True,
                verbose=False,
                n_jobs=-1,
                random_state=None,
                selection='cyclic')
lasso = lasso.fit(X_train, y_train)
print('Lasso Alpha: ', lasso.alpha_)
lasso_coef = np.array(lasso.coef_)
print('Lasso Coefficients: ', lasso_coef)
print('Lasso Intercept: ', lasso.intercept_)
print('Lasso Number of Features: ', np.sum(lasso_coef != 0))
print('Lasso Feature Indexes: ', np.where(lasso_coef != 0))
lasso_feat = np.where(lasso_coef != 0)
X_train_lasso = X_train[:, lasso_feat[0]]
X_test_lasso = X_test[:, lasso_feat[0]]
print('Lasso Shape: ', X_train_lasso.shape)

Lasso Alpha:  0.1
Lasso Coefficients:  [ 0.00000000 -0.00000000  0.00000000 -0.00000000  0.00000000
 -0.00000000 -0.00000000 -0.00000000 -0.00000000 -
```

```
pie_values = [69, 26]
pie_labels = ['69% Bird Flu', '26% Disease Spread']
plt.figure(figure_size=(8,6))
plt.pie(pie_values, labels=pie_labels, colors=['#ff6347', '#4682b4'], autopct=lambda p: f'{int(p*sum(pie_values)/100)}', textprops={'fontsize': 14})
plt.title('How many wild birds were infected with avian flu? In 2022 1,00 birds were captured in US and most are targeted for the H5N1 virus.', fontsize=16, pad=20)
plt.axis('equal')
plt.show()
```

```
# Extracting data for the top 10 locations
top_locations = df['Location'].value_counts().head(10)

# Creating a bar chart
plt.figure(figure_size=(12,6))
sns.barplot(x=top_locations.index, y=top_locations.values, palette='viridis')
plt.title('Top 10 Locations where Avian Flu is most frequent', fontsize=16, pad=20)
plt.xlabel('Location', fontsize=14)
plt.ylabel('Number of Cases', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```

```
# Convert the 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Extracting data for the top 5 locations
top_5_locations = df['Location'].value_counts().head(5).index

# Filtering the dataset for these locations
filtered_df = df[df['Location'].isin(top_5_locations)]

# Creating a line chart
plt.figure(figure_size=(12,6))
sns.lineplot(x=filtered_df['Date'], y=filtered_df['Number of Cases'], hue=filtered_df['Location'], palette='Set1')
plt.title('Percentage of Infected Birds captured in 2022', fontsize=16, pad=20)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Number of Cases', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.legend(title='Location', fontsize=12)
plt.tight_layout()
plt.show()
```

As we can see in the above graph the peak of infection with the most was captured in the month of March.

Localization of birds infected information

```
# Extracting the count of Localization for the top 10 locations
top_locations = df['Localization'].value_counts().head(10)

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(top_locations.values, labels=top_locations.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4', '#ffd700', '#32cd32', '#ff69b4', '#8a2be2', '#ff8c00', '#00ced1', '#dc143c', '#7fff00'])
plt.title('Localization of birds infected information', fontsize=16, pad=20)
plt.axis('equal')
plt.tight_layout()
plt.show
```

We can see the Country Backyard has the most infected birds and the count.

```
# Extracting the count of Backyard
backyard_count = df['Backyard'].value_counts()

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(backyard_count.values, labels=backyard_count.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4'])
plt.title('Backyard', fontsize=16, pad=20)
plt.axis('equal')
plt.tight_layout()
plt.show
```

```
# Extracting the count of Backyard for the top 10 locations
top_locations = df['Location'].value_counts().head(10).index
filtered_df = df[df['Location'].isin(top_locations)]
backyard_count = filtered_df['Backyard'].value_counts()

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(backyard_count.values, labels=backyard_count.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4'])
plt.title('Backyard for the top 10 locations', fontsize=16, pad=20)
plt.axis('equal')
plt.tight_layout()
plt.show
```

As we can see in the above graph the count of Backyards are identical for 2 categories. Backyard and Localization.

```
# Extracting the count of Localization
localization_count = df['Localization'].value_counts()

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(localization_count.values, labels=localization_count.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4', '#ffd700', '#32cd32', '#ff69b4', '#8a2be2', '#ff8c00', '#00ced1', '#dc143c', '#7fff00'])
plt.title('Localization', fontsize=16, pad=20)
plt.axis('equal')
plt.tight_layout()
plt.show
```



```
# Extracting the count of Localization for the top 10 locations
top_locations = df['Location'].value_counts().head(10).index
filtered_df = df[df['Location'].isin(top_locations)]
localization_count = filtered_df['Localization'].value_counts()

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(localization_count.values, labels=localization_count.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4', '#ffd700', '#32cd32', '#ff69b4', '#8a2be2', '#ff8c00', '#00ced1', '#dc143c', '#7fff00'])
plt.title('Localization for the top 10 locations', fontsize=16, pad=20)
plt.axis('equal')
plt.tight_layout()
plt.show
```

Type of Wild Bird

```
# Extracting the count of Type of Wild Bird
type_of_wild_bird_count = df['Type of Wild Bird'].value_counts()

# Creating a pie chart
plt.figure(figure_size=(8,8))
plt.pie(type_of_wild_bird_count.values, labels=type_of_wild_bird_count.index, autopct='%1.1f%%', startangle=140, colors=['#ff6347', '#4682b4', '#ffd700', '#32cd32', '#ff69b4', '#8a2be2', '#ff8c00', '#00ced1', '#dc143c', '#7fff00', '#ff1493', '#00fa9a', '#ff4500', '#2e8b57', '#ff00ff', '#1e90ff', '#ffb6c1', '#ffa07a', '#20b2aa', '#87cefa', '#778899', '#b0c4de', '#ffff00', '#ee82ee', '#bc8f8f', '#f4a460', '#9370db', '#3cb371', '#7b68ee', '#48d1cc', '#c71585', '#191970', '#f5fffa', '#ffe4e1', '#dcdcdc', '#ffe4b5
```

```
[0,0,0,
```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
plot_cm(cm, classes=['Ham', 'Spam'])
```


ROC Curve with Logistic Regression

logistic regression could a probabilities if probability is higher than 0.5 class is labeled 1(personal) else (thermal) by default logistic regression model predicts probabilities. We can change the threshold to classify the data points based on the probabilities.

If the curve is above a line at 45 degree angle, the model is more accurate. The curve is close to the diagonal is also poor. The curve that is bulged toward the top-left from the diagonal is better.

Bar plot for Sentiment Text

Bar = Sentiment Text **Bar** = Text Number **Bar**

```
# ROC curve
from sklearn.metrics import roc_curve
y_pred_proba = logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plot_roc_curve(fpr, tpr)
```



```
# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2'],
              'solver': ['liblinear', 'saga']}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best parameters: {}".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

Other grid search tuning with Hyperparameter

First hyperparameter is C logistic regression regularization parameter

l1 is Lasso and l2 is ridge. solver Second hyperparameter is penalty(loss function)

```
# Lasso L1 Ridge L2
# Grid search for hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2'],
              'solver': ['liblinear', 'saga']}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best parameters: {}".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

Sentimental Analysis

Sentiment analysis, also known as opinion mining, is a natural language processing (NLP) technique for determining the emotional tone of a body of text. This is a common method for businesses to determine and categorize customer opinions about a product, service, or idea. It involves a machine learning tool for sentiment and subjective information using text mining, natural language processing (NLP), and artificial intelligence (AI).

```
# Import the NLTK library
import nltk
```

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
text = "Hello Mr. Smith, how are you doing today? The weather is great, and the city is very busy. Mr. Smith, don't forget to take your medicine."
print(sent_tokenize(text))
```

```
text = "Hi, my name is Mr. Smith. I live in the United States. Mr. Smith, don't forget to take your medicine. Mr. Smith, I'm sorry, but I have to cancel our meeting tomorrow."
print(sent_tokenize(text))
```

```
from nltk.tokenize import word_tokenize
print(word_tokenize(text))
```

```
# Remove punctuation and stopwords
import string
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))
punctuation = set(string.punctuation)
text = "Hello Mr. Smith, how are you doing today? The weather is great, and the city is very busy. Mr. Smith, don't forget to take your medicine."
tokens = word_tokenize(text)
tokens = [token.lower() for token in tokens if token.lower() not in stopwords and token.lower() not in punctuation]
print(tokens)
```

```
# Stemming
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(token) for token in tokens]
print(stemmed_tokens)
```

```
# Lemmatization
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
print(lemmatized_tokens)
```

```
# Part-of-speech tagging
from nltk.tag import pos_tag
pos_tags = pos_tag(tokens)
print(pos_tags)
```

```
# Named entity recognition
from nltk.chunk import ne_chunk
ner_tags = ne_chunk(pos_tags)
print(ner_tags)
```

```
# Dependency parsing
from nltk.parse.stanford import StanfordDependencies
dep_parser = StanfordDependencies()
dep_tags = dep_parser.parse(tokens)
print(dep_tags)
```

```
# Sentiment analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
sentiment_scores = sia.polarity_scores(text)
print(sentiment_scores)
```

```
# Topic modeling
from gensim.models.ldamodel import LdaModel
from gensim.corpora.dictionary import Dictionary
dictionary = Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]
lda_model = LdaModel(corpus, num_topics=2, id2word=dictionary)
print(lda_model.print_topics())
```

```
# Word embeddings
from gensim.models import Word2Vec
sentences = [tokens]
model = Word2Vec(sentences, min_count=1)
print(model.wv['hello'])
```

```
# Text classification
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
X = CountVectorizer().fit_transform(tokens)
y = [1, 0, 1, 0, 1, 0, 
```

Chapter 1 Introduction
Section 1.1
Section 1.2
Section 1.3
Section 1.4
Section 1.5
Section 1.6
Section 1.7
Section 1.8
Section 1.9
Section 1.10
Section 1.11
Section 1.12
Section 1.13
Section 1.14
Section 1.15
Section 1.16
Section 1.17
Section 1.18
Section 1.19
Section 1.20
Section 1.21
Section 1.22
Section 1.23
Section 1.24
Section 1.25
Section 1.26
Section 1.27
Section 1.28
Section 1.29
Section

```
# Load libraries
library(dplyr)
library(ggplot2)
library(knitr)
library(kableExtra)
library(tidyverse)

# Set the seed for reproducibility
set.seed(123)

# Generate data for 100 observations
n <- 100
data <- data.frame(
  id = 1:n,
  group = sample(c("A", "B"), n, replace = TRUE),
  value = rnorm(n, mean = ifelse(group == "A", 10, 12), sd = 2)
)

# Calculate summary statistics
summary_stats <- data.frac(
  mean_value = tapply(data$value, data$group, mean),
  sd_value = tapply(data$value, data$group, sd)
)

# Perform a t-test
t_test <- t.test(value ~ group, data = data)

# Create a boxplot
boxplot <- ggplot(data, aligned = TRUE, aes(x = group, y = value, fill = group))
  geom_boxplot(alpha = 0.5)
  geom_jitter(width = 0.2, alpha = 0.3)
  theme_minimal()
  labs(title = "Distribution of Values by Group",
       x = "Group",
       y = "Value")

# Print results
cat("Summary statistics:\n")
Summary statistics:
print(summary_stats)
$mean_value
    A    B
10.11 12.3
$sd_value
    A    B
2.000 1.97

cat("\n")

cat("T-test results:\n")
T-test results:
print(t_test)
Error in print.default(t_test) : object 'data.method' not found

# Save the boxplot to a file
ggsave("boxplot.png", boxplot, width = 8, height = 6)
Error in ggsave("boxplot.png", boxplot, width = 8, height = 6) :
  object 'boxplot' not found
```

Statistics

Table 1

Comparison tools

Comparisons tests assess whether there are differences in means, medians or rankings of scores of two or more groups.The general goal for analysis is to determine whether the observed data fit a null hypothesis. Statistical significance helps determine whether research-generated evidence, and not the chance factor, is responsible for the presence of a relationship between or among variables.

To decide which test suits your aim, consider whether your data meets the conditions necessary for parametric tests, the number of samples, and the type of variables involved in the hypothesis.

Here are some tips that can be helpful in what case, either relative and catalogs are more appropriate measures for ordinal data:

```
data.frame()
```

	Scientific Name	Common Name	Size	Weight	Day	Time	Scientific Name Group	Genus	Locality	Individuals	Longitude
1	Ursus maritimus	Polar Bear	2400	450	12	1300	Ursidae	U-M-G-M	Alaska	1	-154.0000
2	Ursus maritimus	Polar Bear	2400	450	12	1300	Ursidae	U-M-G-M	Alaska	1	-154.0000
3	Ursus maritimus	Polar Bear	2400	450	12	1300	Ursidae	U-M-G-M	Alaska	1	-154.0000
4	Tursiops truncatus	Common Bottlenose Dolphin	2400	450	12	1300	Delphinidae	T-T-G-M	Alaska	1	-154.0000
5	Tursiops truncatus	Common Bottlenose Dolphin	2400	450	12	1300	Delphinidae	T-T-G-M	Alaska	1	-154.0000
6	Delphinapterus leucas	Beluga Whale	2400	450	12	1300	Delphinidae	D-L-G-M	Alaska	1	-154.0000
7	Delphinapterus leucas	Beluga Whale	2400	450	12	1300	Delphinidae	D-L-G-M	Alaska	1	-154.0000
8	Delphinapterus leucas	Beluga Whale	2400	450	12	1300	Delphinidae	D-L-G-M	Alaska	1	-154.0000

```
# Load required libraries
library(ggplot2)
library(dplyr)

# Create data frame
df <- data.frame(
  id = 1:10,
  name = c("John", "Mary", "Peter", "Sarah", "Tom", "Lisa", "Mike", "Anna", "Bob", "Kate"),
  age = c(25, 30, 35, 28, 40, 32, 45, 27, 38, 29),
  salary = c(50000, 60000, 75000, 55000, 80000, 65000, 90000, 52000, 70000, 58000),
  department = c("IT", "HR", "IT", "HR", "IT", "HR", "IT", "HR", "IT", "HR")
)

# Calculate summary statistics
summary_stats <- df %>%
  group_by(department) %>%
  summarise(
    mean_age = mean(age),
    mean_salary = mean(salary),
    sd_salary = sd(salary),
    n = n()
  )

# Print results
print("Summary statistics:")
[1] "Summary statistics:"
print(summary_stats)
# A tibble: 2 x 5
  department mean_age mean_salary sd_salary     n
  <chr>        <dbl>       <dbl>     <dbl> <int>
1 HR            32.5       65000     7906.     5
2 IT            33.5       70000    19365.     5

# Create boxplot
ggplot(df, aes(x = department, y = salary, fill = department))
  geom_boxplot()
  geom_jitter(width = 0.2, alpha = 0.3)
  theme_minimal()
  labs(title = "Salary Distribution by Department",
       x = "Department",
       y = "Salary")
$geom_boxplot
$geom_jitter
$width
[1] 0.2
$alpha
[1] 0.3

attr(,"class")
[1] "uneval"
attr(,"specified")
$x
[1] "department"
$y
[1] "salary"
$fill
[1] "department"
attr(,"class")
[1] "uneval"
$theme_minimal
function (base_size = 11, base_family = "", base_line_size = base_size/22,
    base_rect_size = base_size/22)
$labs
$title
[1] "Salary Distribution by Department"
$x
[1] "Department"
$y
[1] "Salary"
attr(,"class")
[1] "ggplot"
attr(,"class")
[1] "list"

# Save plot
ggsave("salary_boxplot.png", width = 8, height = 6)
Error in ggsave("salary_boxplot.png", width = 8, height = 6) :
  object 'last_plot' not found
```

```
# Load required libraries
library(ggplot2)
library(dplyr)

# Create data frame
df <- data.frame(
  id = 1:10,
  name = c("John", "Mary", "Peter", "Sarah", "Tom", "Lisa", "Mike", "Anna", "Bob", "Kate"),
  age = c(25, 30, 35, 28, 40, 32, 45, 27, 38, 29),
  salary = c(50000, 60000, 75000, 55000, 80000, 65000, 90000, 52000, 70000, 58000),
  department = c("IT", "HR", "IT", "HR", "IT", "HR", "IT", "HR", "IT", "HR")
)

# Calculate summary statistics
summary_stats <- df %>%
  group_by(department) %>%
  summarise
Error: unexpected '%>%' in "  summary_stats <- df %>%"
  mean_age = mean(age),
  mean_salary = mean(salary),
  sd_salary = sd(salary),
  n = n()
Error: unexpected '%>%' in "  mean_age = mean(age),"

# Print results
print("Summary statistics:")
[1] "Summary statistics:"
print(summary_stats)
Error in print.default(summary_stats) : object 'summary_stats' not found

# Create boxplot
ggplot(df, aes(x = department, y = salary, fill = department))
  geom_boxplot()
  geom_jitter(width = 0.2, alpha = 0.3)
  theme_minimal()
  labs(title = "Salary Distribution by Department",
       x = "Department",
       y = "Salary")
$geom_boxplot
$geom_jitter
$width
[1] 0.2
$alpha
[1] 0.3

attr(,"class")
[1] "uneval"
attr(,"specified")
$x
[1] "department"
$y
[1] "salary"
$fill
[1] "department"
attr(,"class")
[1] "uneval"
$theme_minimal
function (base_size = 11, base_family = "", base_line_size = base_size/22,
    base_rect_size = base_size/22)
$labs
$title
[1] "Salary Distribution by Department"
$x
[1] "Department"
$y
[1] "Salary"
attr(,"class")
[1] "ggplot"
attr(,"class")
[1] "list"

# Save plot
ggsave("salary_boxplot.png", width = 8, height = 6)
Error in ggsave("salary_boxplot.png", width = 8, height = 6) :
  object 'last_plot' not found
```