Malsha Madhurangi_st20267780_CL HDCSE 25 89_Computational Intelligence assignment.docx

by Pathiraja Arachchillage Malsha Madhurangi

Submission date: 23-Mar-2024 01:06PM (UTC+0000)

Submission ID: 227670761

File name: 120029_Pathiraja_Arachchillage_Malsha_Madhurangi_Malsha_Madhurangi_st20267780_CL_HDCSE_25_89_Computational_Intelligence_assignm_2103900901.docx (5.32M)

Word count: 5007

Character count: 27953

Table of content

Mini Project

a. Introduction to the concept of deep learning

Deep learning is a method of artificial intelligence that usually refers to the way the human brain stores and stores data of a person. Just as the human brain contains millions of neurons to receive information, deep learning networks, artificial networks, and many artificial neurons work together in the computer. The deep learning models here can generate predictions, recognize complex patterns in images, sounds, and data. Deep learning methods can be used to automatically perform tasks that require human intervention, such as transcribing a few sounds, describing images. Deep learning models are defined as computer files that scientists have trained to perform an algorithm or specific task. Deep learning models are mainly used for data analysis. And a deep learning network can be divided into three parts as input layer, hidden layer and output layer. The input layer is formed using several nodes containing the data and the output layer is based on the two outputs 'yes' and 'no'. At the hidden layer, information is processed at different levels and behavior is adapted. For example, when a human is given an image of an unknown animal that needs to be classified, it compares the size, number of legs, and eye position of the unknown animal with a known animal. If the animal has cat eyes, the man guesses that it is a wild cat. In the same way, a deep learning algorithm while classifying an animal image considers each hidden layer as a different feature of the animal and tries to classify it correctly. Deep learning algorithms make machine learning methods more efficient. This is called supervised learning and it requires a large enough data set with widely varying data to increase the accuracy of the results. For example, if a training set for cat color discrimination contains more images of black cats, black cats will be correctly identified. But white cats are not recognized and thus white cat images need to be trained again in a machine learning model to recognize white cats. (Amazon Web Services, Inc., 2024)

Considering the history of deep learning, the concept of artificial neurons was proposed by Warren McCulloch and Walter Pitts in the 1940s. It is the foundation of deep learning. Here, a mathematical model based on neuron activity was created. Mcp-neuron is called In 1957, Frank Rosenblatt introduced the perceptron. In the late 1960s, deep learning was suppressed as there was little interest in binary neurons. Again, in the year 1986, with the introduction of the backpropagation algorithm by Geoffrey Hinton, David Rumelhart and Ronald Williams, deep learning made its way back to prominence. Backpropagation is known as the father of deep learning. LeCun improved the LeNet-5 architecture in the late 1990s. In 1999, Nvidia created the world's first GPU. In early 2006, deep learning models were developed on GPUs, and deep learning was used in speech recognition in the early 2010s. And in the year 2013, neural networks started to be used in computers. Thus, the demand for deep learning has increased and it has become an indispensable technology nowadays. (Sundaray, n.d.)

Year	Contributor	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand the brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of the neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced the MCP Model, which is considered as the ancestor of the Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced the Hebbian Learning Rule, which lays the foundation of modern neural network
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
		introduced Deep Belief Networks, also introduced layer-
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks
2012+		Deep Learning Revolution started

Table 1 — Brief History of Machine Learning (Wang & Raj, 2017)

Deep learning architecture is also used to solve complex tasks. Discusses 6 architectural techniques used in everyday life. Namely RNN, LSTM, GRU, CNN, DBN and DSN. RNN or Recurrent Neural Network Speech Synthesis helps to maintain memory of past input, solve problems. Also, two types of RNNs can be identified as bidirectional RNNs and deep RNNs. In bidirectional RNNs, past and future information can be obtained simultaneously. And in deep RNNs, more hierarchies can be added to the DL model. LSTM or Long Short Term Memory is a type of RNN and text compression is used for tasks such as speech recognition, handwriting recognition, gesture recognition and video captioning. GRU is a type of LSTM and is used for small and sparse data sets. They show good performance. CNN or Convolutional Neural Networks are used for video recognition, video processing, video analysis and NLP. , DBN or Deep Belief Network is a multi-layer network and is used in pattern recognition and NLP. In DSN or Deep Stacking Network, a group of single deep networks contributes to improving the training problem and provides the ability to learn complex functions and classifications. Here, simple architectures such as autoencoders can be identified, which help to reduce dimensionality and naturally detect anomalies. Finally deep learning architecture is the same idea but there are different methods. Nowadays, with the development of technology, deep learning for various fields, artificial intelligence (Artificial intelligence) makes learning tasks more efficient and easier. (Lisowski, n.d.)

b. A Literature review

Using Artificial Intelligence to Predict Probability of Survival and Need for Surgery in Horses with Acute Colic

The purpose of this study is to explore the use of machine learning algorithms in the process of predicting the need for surgery in horses with acute colic, using the basic chemical data obtained. This study was written by Mohammad A. Frywan A. and Samih M. By Abutarbush b. Acute colic is one of the conditions most horses face. Here the effects of gastrointestinal colic run the gamut from harmless spasmodic colic to the risk of strangulation. Abdominal pain in horses can be caused by factors such as deep ulceration, ischemia, intestinal distention, and stress on the cerebral root. However, in medicine, five main criteria are used to decide whether to undergo surgery. The five criteria were findings on transrectal palpation, presence of nasogastric reflux, cardiovascular and systemic status, Pain severity (as measured by non-responsiveness to analgesics) and abdominal outcomes can be indicated. A small percentage of horses typically require surgery, 4%-10%, and rarely the final diagnosis is made without laparotomy.

With the new development of technology in recent years, with the expansion of subjects like artificial intelligence and machine learning, image analysis, processing data sets, identifying various features, helping to make accurate and quick decisions, and identifying obscure and hidden relationships for humans provide many advantages.

Machine learning, a subset of Ai, synthesizes mathematical models using relevant training data. The training data obtained as these samples act as the trigger for setting the model parameters. Also, Ai can provide very quick and accurate responses through image inspection. An example is veterinary robot Sofie. Ai provides advantages such as face recognition, reproduction, and quick decisions, and for example, it was able to achieve a high accuracy of 98.7% by using a deep learning method, Convolutional neural networks, to count reticulocytes in cat blood images. This kind of process saves time and money in the absence of expensive equipment. Valletta et al. also reviewed machine learning algorithms in animal behavior studies, where 3 study results are discussed. The three outcomes are identifying and counting wildlife, discovering jack social structures, and providing quail eggs from mixed clutches to the egg-laying female. Accelerometric data has been used for this study. Also, elasmobranch studies of juvenile lemon shark behavior in the Bahamas reported a high F-measure of 88%, providing good empirical evidence of shark behavior and explaining scientific observations.

The primary objective of this study was to explore the ability of the ML algorithm to predict the need for surgery and probability of survival in horses based on baseline clinical data. Survival here refers to a horse recovering and being discharged from the hospital. This provides more data on disease progression than clinical data predicts. Also, machine learning algorithms were able to predict the need for surgery and the probability of survival in acute colic disease in horses with 76% and 85% accuracy. Also, the first step in the system

architecture is to implement an automated computer-based or smart-mobile tablet-based system application that can be easily used by clinicians to facilitate decision-making. By designing such an application, appropriate interfaces can be created to process data, provide appropriate outputs during prediction, and have multiple capabilities. Here, the results obtained using 4 algorithms using 285 clinical reports are summarized.

Finally, a model based on an Ai model was developed, predicting the clinical characteristics of horses using an ML algorithm. It uses Ai technology and ML algorithms to predict the probability of survival as well as the need for surgery in horses suffering from epigastric disease. Adaptation to a user-friendly interface is an important step as ML algorithms are optimized accordingly as datasets continue to expand. (Journal of Equine Veterinary Science, n.d.)

c. Exploratory data analysis (EDA)

Below are the imported libraries to perform all the tools and functions required to perform this prediction.

```
# Data manipulation and preprocessing
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

# Statistical tests
from scipy.stats import chi2_contingency

# Machine learning models and metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, f1_score, confusion_matrix

# Deep learning libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Model persistence
import pickle
import joblib
# Data visualization
import klib
import matplotlib.pyplot as plt
import seaborn as sns
```

Then train data set, test data set and sample submission data set are entered into the project to start the prediction process. Here "traindata" refers to the "traindata set" used in training the model and "testdata" refers to the test data set. "Submissiondata" means the data set representing the sample submission.

```
traindata = pd.read_csv('/content/drive/MyDrive/playground-series-s3e22/train.csv')
testdata  = pd.read_csv('/content/drive/MyDrive/playground-series-s3e22/test.csv')
submissiondata = pd.read_csv('/content/drive/MyDrive/playground-series-s3e22/sample_submission.csv')
```

Here, an identification is made if there is a missing value in the traindata data set. It is observed that there is no missing value.

traindata

	id	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	...	packed_cell_volume	total_protein	abdomo
0	0	yes	adult	530001	38.1	132.0	24.0	cool	reduced	dark_cyanotic	...	57.0	8.5	s
1	1	yes	adult	533836	37.5	88.0	12.0	cool	normal	pale_cyanotic	...	33.0	64.0	s
2	2	yes	adult	529812	38.3	120.0	28.0	cool	reduced	pale_pink	...	37.0	6.4	s
3	3	yes	adult	5262541	37.1	72.0	30.0	cold	reduced	pale_pink	...	53.0	7.0	
4	4	no	adult	5299629	38.0	52.0	48.0	normal	normal	normal_pink	...	47.0	7.3	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1230	1230	yes	adult	535246	38.5	129.0	48.0	cool	reduced	pale_pink	...	57.0	66.0	s
1231	1231	yes	adult	528570	37.5	60.0	50.0	cool	reduced	pale_cyanotic	...	35.0	6.4	s
1232	1232	yes	young	529685	37.5	84.0	40.0	normal	reduced	normal_pink	...	40.0	5.9	
1233	1233	yes	adult	534784	38.1	70.0	16.0	normal	reduced	bright_red	...	58.0	74.0	
1234	1234	yes	adult	528548	38.1	54.0	36.0	normal	normal	pale_pink	...	45.0	6.0	

1235 rows × 29 columns

```
klib.missingval_plot(traindata)
```

```
No missing values found in the dataset.
```

Here the unnecessary columns in the data set are removed. Here the id and hospital number columns are removed.

```
traindata.drop(['id','hospital_number'],axis =1, inplace = True)
testdata.drop (['id','hospital_number'],axis =1, inplace = True)
```

Related to viewing the structure of each variable in the dataset and its data types. This is done using the info() function and this code is used for both train and test datasets.

```
print(traindata.info())
print(testdata.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1235 entries, 0 to 1234
Data columns (total 27 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   surgery                 1235 non-null   object
 1   age                     1235 non-null   object
 2   rectal_temp             1235 non-null   float64
 3   pulse                   1235 non-null   float64
 4   respiratory_rate        1235 non-null   float64
 5   temp_of_extremities     1235 non-null   object
 6   peripheral_pulse        1235 non-null   object
 7   mucous_membrane         1235 non-null   object
 8   capillary_refill_time   1235 non-null   object
 9   pain                    1235 non-null   object
 10  peristalsis             1235 non-null   object
 11  abdominal_distention    1235 non-null   object
 12  nasogastric_tube        1235 non-null   object
 13  nasogastric_reflux      1235 non-null   object
 14  nasogastric_reflux_ph   1235 non-null   float64
 15  rectal_exam_feces       1235 non-null   object
 16  abdomen                 1235 non-null   object
 17  packed_cell_volume      1235 non-null   float64
 18  total_protein           1235 non-null   float64
 19  abdomo_appearance       1235 non-null   object
 6   peripheral_pulse        1235 non-null   object
 7   mucous_membrane         1235 non-null   object
 8   capillary_refill_time   1235 non-null   object
 9   pain                    1235 non-null   object
 10  peristalsis             1235 non-null   object
 11  abdominal_distention    1235 non-null   object
 12  nasogastric_tube        1235 non-null   object
 13  nasogastric_reflux      1235 non-null   object
 14  nasogastric_reflux_ph   1235 non-null   float64
 15  rectal_exam_feces       1235 non-null   object
 16  abdomen                 1235 non-null   object
 17  packed_cell_volume      1235 non-null   float64
 18  total_protein           1235 non-null   float64
 19  abdomo_appearance       1235 non-null   object
 20  abdomo_protein          1235 non-null   float64
 21  surgical_lesion         1235 non-null   object
 22  lesion_1                1235 non-null   int64
 23  lesion_2                1235 non-null   int64
 24  lesion_3                1235 non-null   int64
 25  cp_data                 1235 non-null   object
 26  outcome                 1235 non-null   object
dtypes: float64(7), int64(3), object(17)
memory usage: 260.6+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
```

Here, lesion_1, lesion_2, lesion_3 columns are considered.

These are categorical variables. Hence conversion to object or category type variables is required.

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

def convert_int_to_object(data, columns):
    for column in columns:
        if data[column].dtype == 'int64':
            data[column] = data[column].astype('object')


convert_int_to_object(traindata, lesion_columns)
convert_int_to_object(testdata, lesion_columns)
```

Here Categorical columns like Object, category are stored in a new data frame.

```
categorical_columns = traindata.select_dtypes(include=['object', 'category']).columns.tolist()
traindata[categorical_columns]
```

	surgery	age	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	peristalsis	abdominal_distention	nasogastric_tube	nasogastric_reflu
0	yes	adult	cool	reduced	dark_cyanotic	more_3_sec	depressed	absent	slight	slight	less_1_lite
1	yes	adult	cool	normal	pale_cyanotic	more_3_sec	mild_pain	absent	moderate	none	more_1_lite
2	yes	adult	cool	reduced	pale_pink	less_3_sec	extreme_pain	hypomotile	moderate	slight	none
3	yes	adult	cold	reduced	pale_pink	more_3_sec	mild_pain	hypomotile	moderate	slight	more_1_lite
4	no	adult	normal	normal	normal_pink	less_3_sec	alert	hypomotile	none	slight	less_1_lite
...	...	...	...	...	...	...	...	...	...	...	...
1230	yes	adult	cool	reduced	pale_pink	more_3_sec	depressed	absent	moderate	none	more_1_lite
1231	yes	adult	cool	reduced	pale_cyanotic	less_3_sec	mild_pain	hypomotile	slight	slight	none
1232	yes	young	normal	reduced	normal_pink	less_3_sec	mild_pain	hypomotile	slight	slight	none
1233	yes	adult	normal	reduced	bright_red	less_3_sec	mild_pain	hypomotile	slight	none	more_1_lite
1234	yes	adult	normal	normal	pale_pink	less_3_sec	mild_pain	absent	none	slight	none

1235 rows × 20 columns

Here quantitative columns like float64, int64, category are stored in a new data frame.

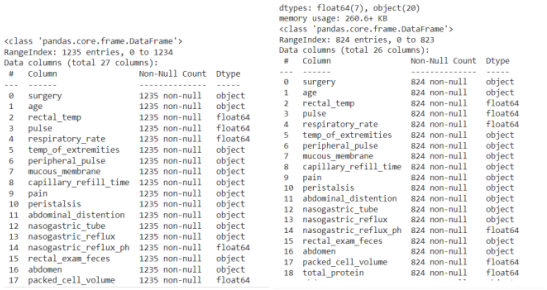
```
quantitative_columns = traindata.select_dtypes(include=['float64', 'int64', 'category']).columns.tolist()
traindata[quantitative_columns]
```

	rectal_temp	pulse	respiratory_rate	nasogastric_reflux_ph	packed_cell_volume	total_protein	abdomo_protein
0	38.1	132.0	24.0	6.5	57.0	8.5	3.4
1	37.5	88.0	12.0	2.0	33.0	64.0	2.0
2	38.3	120.0	28.0	3.5	37.0	6.4	3.4
3	37.1	72.0	30.0	2.0	53.0	7.0	3.9
4	38.0	52.0	48.0	7.0	47.0	7.3	2.6
...	...	...	...	...	...	...	...
1230	38.5	129.0	48.0	2.0	57.0	66.0	2.0
1231	37.5	60.0	50.0	3.0	35.0	6.4	3.6
1232	37.5	84.0	40.0	3.0	40.0	5.9	7.0
1233	38.1	70.0	16.0	2.0	58.0	74.0	2.0
1234	38.1	54.0	36.0	3.0	45.0	6.0	3.6

1235 rows × 7 columns

Then the train data and test data sets check the data type and structure with info().

```
traindata.info()
testdata.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1235 entries, 0 to 1234
Data columns (total 27 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   surgery                1235 non-null   object
 1   age                    1235 non-null   object
 2   rectal_temp            1235 non-null   float64
 3   pulse                  1235 non-null   float64
 4   respiratory_rate       1235 non-null   float64
 5   temp_of_extremities    1235 non-null   object
 6   peripheral_pulse       1235 non-null   object
 7   mucous_membrane        1235 non-null   object
 8   capillary_refill_time  1235 non-null   object
 9   pain                   1235 non-null   object
 10  peristalsis            1235 non-null   object
 11  abdominal_distention   1235 non-null   object
 12  nasogastric_tube       1235 non-null   object
 13  nasogastric_reflux     1235 non-null   object
 14  nasogastric_reflux_ph  1235 non-null   float64
 15  rectal_exam_feces      1235 non-null   object
 16  abdomen                1235 non-null   object
 17  packed_cell_volume     1235 non-null   float64
```



```
dtypes: float64(7), object(20)
memory usage: 260.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   surgery                824 non-null    object
 1   age                    824 non-null    object
 2   rectal_temp            824 non-null    float64
 3   pulse                  824 non-null    float64
 4   respiratory_rate       824 non-null    float64
 5   temp_of_extremities    824 non-null    object
 6   peripheral_pulse       824 non-null    object
 7   mucous_membrane        824 non-null    object
 8   capillary_refill_time  824 non-null    object
 9   pain                   824 non-null    object
 10  peristalsis            824 non-null    object
 11  abdominal_distention   824 non-null    object
 12  nasogastric_tube       824 non-null    object
 13  nasogastric_reflux     824 non-null    object
 14  nasogastric_reflux_ph  824 non-null    float64
 15  rectal_exam_feces      824 non-null    object
 16  abdomen                824 non-null    object
 17  packed_cell_volume     824 non-null    float64
 18  total_protein          824 non-null    float64
```

Copies data stored in traindata to df.

```
df = traindata
```

df is used to distribute the classification values of the data set and iterate over the training data and print the unique values in the corresponding columns of the test data.

```
categorical_columns = df.select_dtypes(include=['object']).columns

for column in categorical_columns[:-1]:
    print(column, testdata[column].unique())
```

```
surgery ['no' 'yes']
age ['adult' 'young']
temp_of_extremities ['normal' 'cool' 'None' 'cold' 'warm']
peripheral_pulse ['normal' 'reduced' 'None' 'absent' 'increased']
mucous_membrane ['normal_pink' 'bright_pink' 'bright_red' 'pale_cyanotic' 'dark_cyanotic'
 'pale_pink' 'None']
capillary_refill_time ['less_3_sec' 'more_3_sec' 'None' '3']
pain ['mild_pain' 'depressed' 'severe_pain' 'extreme_pain' 'None' 'moderate'
 'alert']
peristalsis ['hypomotile' 'absent' 'hypermotile' 'None' 'normal']
abdominal_distention ['slight' 'moderate' 'severe' 'none' 'None']
nasogastric_tube ['none' 'slight' 'None' 'significant']
nasogastric_reflux ['none' 'more_1_liter' 'less_1_liter' 'None']
rectal_exam_feces ['normal' 'decreased' 'absent' 'None' 'increased']
abdomen ['distend_small' 'distend_large' 'None' 'other' 'firm' 'normal']
abdomo_appearance ['clear' 'serosanguious' 'cloudy' 'None']
surgical_lesion ['no' 'yes']
lesion_1 [0 2208 2205 1400 2207 2209 4205 2124 3205 2112 2206 3111 5400 31110 400
 8400 2322 5000 12208 2305 2111 37 7209 300 5124 7111 5206 3207 2113 4300
 11400 7400 5205 3209 3113 3133 6111 4209 6112 6209 5111 8300 3025 4207
 4206 11124 2300 9000 11300 3300 5209 21110 4124 9400]
lesion_2 [0 3111 1400 4300]
lesion_3 [0]
cp_data ['no' 'yes']
```

Univariate analysis

This field defines a function of categorical variables to perform univariate analysis.

```
def univariate_categorical_analysis(data, column, ax):
    sns.countplot(data=data, x=column, ax=ax)
    ax.set_title(f'Distribution of {column}')
    ax.set_xlabel(column)
    ax.set_ylabel('Count')
    ax.tick_params(axis='x', rotation=45)
```

This code is used to define a function of numerical variables to perform a univariate analysis.

```
def univariate_numeric_analysis(data, column, ax):
    sns.histplot(data=data, x=column, kde=True, ax=ax)
    ax.set_title(f'Distribution of {column}')
    ax.set_xlabel(column)
    ax.set_ylabel('Frequency')

num_rows = 5
num_cols = 5
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))
```

Univariate analysis is used to perform categorical variables.

```
categorical_columns = df.select_dtypes(include=['object', 'category']).columns
for i, column in enumerate(categorical_columns):
    row = i // num_cols
    col = i % num_cols
    univariate_categorical_analysis(df, column, ax=axes[row, col])
```

This code removes unused subparts.

```
for i in range(len(categorical_columns), num_rows*num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axes[row, col])
```

This code is used to set the layout.

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

This sets up a subplot grid for numerical variables.

```
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 10))
```

Here univariate analysis is done for numerical variables.

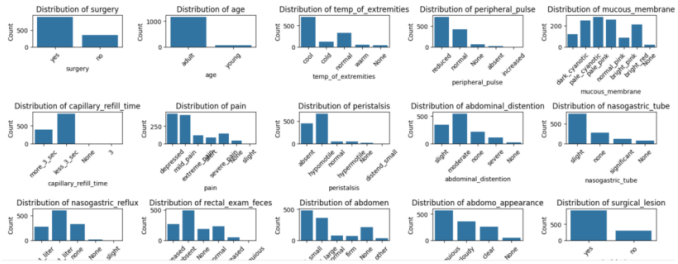
```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
for i, column in enumerate(numeric_columns):
    row = i // num_cols
    col = i % num_cols
    univariate_numeric_analysis(df, column, ax=axes[row, col])
```

This code removes unused subparts.

```
for i in range(len(numeric_columns), num_rows*num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axes[row, col])
```

The layout is adjusted.

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



Analysis of variance class balance using outcomes of dependent variables. There is no need to balance the data set as it appears to be correct.

```
df['outcome'].value_counts()
```

```
lived         574
died          410
euthanized    251
Name: outcome, dtype: int64
```

Bivariate analysis

For bivariate analysis a function is used to define the categorical variables with 'outcome'.

```
def bivariate_categorical_outcome_analysis(data, column, ax):
    sns.countplot(data=data, x=column, hue='outcome', ax=ax)
    ax.set_title(f'{column} vs Outcome')
    ax.set_xlabel(column)
    ax.set_ylabel('Count')
    ax.legend(title='Outcome')
```

Used to define a function by numeric variables with 'outcome' for bivariate analysis.

```
def bivariate_numeric_outcome_analysis(data, column, ax):
    sns.boxplot(data=data, x='outcome', y=column, ax=ax)
    ax.set_title(f'{column} vs Outcome')
    ax.set_xlabel('Outcome')
    ax.set_ylabel(column)
```

Used to set the subplot grid for categorical variables with 'results'.

```
num_cols = 3
num_rows = (len(df.select_dtypes(include=['object', 'category']).columns) - 1) // num_cols + 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 6*num_rows))
```

Bivariate analysis is performed for categorical variables with 'outcome'.

```
categorical_columns = df.select_dtypes(include=['object', 'category']).columns
for i, column in enumerate(categorical_columns):
    if column != 'outcome':
        row = i // num_cols
        col = i % num_cols
        bivariate_categorical_outcome_analysis(df, column, ax=axes[row, col])
```

Removes unused partitions.

```
for i in range(len(categorical_columns)-1, num_rows*num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axes[row, col])
```

The layout is adjusted.

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

Sets the subplot grid for numerical variables with 'results'.

```
num_cols = 3
num_rows = (len(df.select_dtypes(include=['float64', 'int64']).columns)) // num_cols + 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 6*num_rows))
```

Binary analysis is performed for numeric variables with 'outcome'.

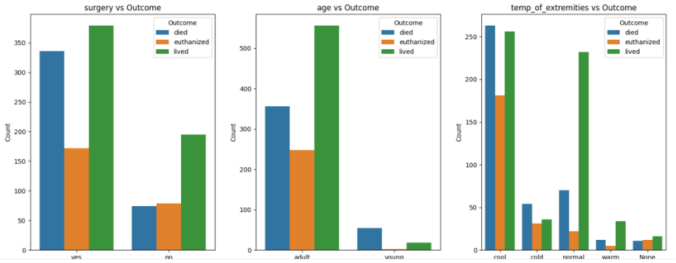
```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
for i, column in enumerate(numeric_columns):
    row = i // num_cols
    col = i % num_cols
    bivariate_numeric_outcome_analysis(df, column, ax=axes[row, col])
```

Removes unused partitions.

```
for i in range(len(numeric_columns), num_rows*num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axes[row, col])
```

The layout is adjusted.

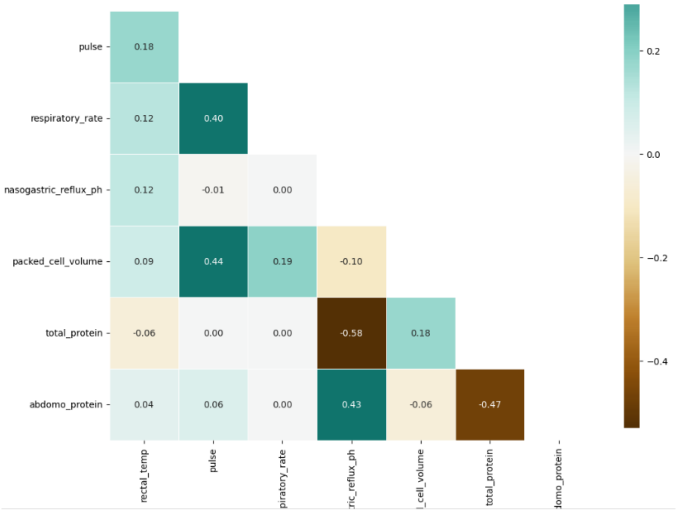
```
plt.tight_layout()
plt.show()
```



Multivariate analysis

It helps to understand the interdependencies between the variables in the data set and helps to generate a correlation plot for df.

```
klib.corr_plot(df)
```



There is a considerable negative correlation between total_protein variable and nasogastic_reflux_ph variables. And also there is a positive correlation between abdomo_protein and nasogastic_reflux_ph variables. Also there can be seen some negative correlation of total_protein with abdomo_protein. Respiratory rate and pulse variables have a positive correlation of 0.4. So when doing advance analysis or model building we need to focus of those correlation variables. if needed we must omit the correlated variables.

Test of association for categorical variables

The a library and a tool used for this are as follows.

```
import pandas as pd
from scipy.stats import chi2_contingency
```

Loading the data set.

```
data = df
```

Drops rows with missing values.

```
data.dropna(inplace=True)
```

Only categorical columns are selected here.

```
categorical_cols = data.select_dtypes(include=['object']).columns
```

Here an empty DataFrame is created to store the p values.

```
p_values = pd.DataFrame(index=categorical_cols, columns=categorical_cols)
```

The p-value is calculated for all pairs of categorical variables.

```
for col1 in categorical_cols:
    for col2 in categorical_cols:
        if col1 != col2:
            contingency_table = pd.crosstab(data[col1], data[col2])
            _, p_value, _, _ = chi2_contingency(contingency_table)
            p_values.loc[col1, col2] = p_value
```

Performs the task of highlighting significant p-values.

```
def highlight_significant(val):
    color = 'background-color: pink' if float(val) < 0.05 else ''
    return color
```

The p-values are formatted into the DataFrame.

```
formatted_p_values = p_values.style.applymap(highlight_significant)
```

Displayed the formatted p-value.

```
print("p-values with significant cells highlighted:")
formatted_p_values
```

p-values with significant cells highlighted:

	surgery	age	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	peristalsis	abdominal_distention	nasogastric_tube	nas
surgery	nan	0.078940	0.000000	0.000000	0.000000	0.080710	0.000000	0.000000	0.000000	0.207844	
age	0.078940	nan	0.097053	0.264076	0.000044	0.072287	0.001872	0.675392	0.003019	0.295050	
temp_of_extremities	0.000000	0.097053	nan	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
peripheral_pulse	0.000000	0.264076	0.000000	nan	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
mucous_membrane	0.000000	0.000044	0.000000	0.000000	nan	0.000000	0.000000	0.000000	0.000000	0.000000	
capillary_refill_time	0.080710	0.072287	0.000000	0.000000	0.000000	nan	0.000000	0.000000	0.000000	0.000000	
pain	0.000000	0.001872	0.000000	0.000000	0.000000	0.000000	nan	0.000000	0.000000	0.000000	
peristalsis	0.000000	0.675392	0.000000	0.000000	0.000000	0.000000	0.000000	nan	0.000000	0.000001	
abdominal_distention	0.000000	0.003019	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	nan	0.000000	
nasogastric_tube	0.207844	0.295050	0.000000	0.000000	0.000000	0.000000	0.000000	0.000001	0.000000	nan	
nasogastric_reflux	0.002891	0.773488	0.000160	0.003326	0.000002	0.023879	0.000000	0.000000	0.000043	0.000000	
rectal_exam_feces	0.000000	0.006813	0.000000	0.000000	0.000000	0.000055	0.000000	0.000000	0.000000	0.000104	
abdomen	0.000000	0.000000	0.000000	0.000000	0.000000	0.000030	0.000000	0.000000	0.000000	0.000006	
abdomo_appearance	0.000000	0.192151	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.131700	

d. System architecture

A complete machine learning model is used to predict the health outcomes of horses according to the developed methods. Here, based on several factors such as the horse's age, heart rate, breathing rate, pain level, abdominal division, the outcome of the horse is indicated as lived, euthanized and died. By analyzing all the factors provided in the data set, it is possible to easily determine the outcome of the horses with the results obtained from the data path trained by the model. Here data is collected based on all the factors given in the data set. The collected data is processed into training validation datasets which are separated for training and selection for the system. The model uses a user-friendly interface to make it easier for users to use the trained data. This allows users to enter horse health results and obtain horse outcome predictions using a clear interface. This process can be simply described as the architecture of a system.

How your application differs from other existing applications

According to the health results of the horses, decisions are made based on factors such as the behavior patterns of the horses and the appearance of the horses when considering the prediction of their outcome (lived, euthanized and died). Systems often focus on health factors such as age, pain level, respiratory rate, pulse, abdominal distension, etc. For example, a veterinarian monitors the horse's health to determine the horse's results. Looking at horse weight, observing horse behavior patterns can be another real-world application used by horse riders as well as horse owners. Although we focus on some of the same characteristics for the health status of the horses used for the advanced model in predicting the outcome of the horses from the existing applications, the methodology used in doing this is different. Because large-scale operations require a large number of people and manpower. Hence, the use of data sets obtained from real observations, training machine learning algorithms provides a sustainable solution. Using this advanced algorithm, it is possible to predict the results of horses, so it does not require direct human intervention. Therefore, this has become a very good and efficient alternative for predicting the results of horses in the real world.

Machine learning techniques

In order to calculate the outcome of horses according to the health results of horses, several machine learning methods are analyzed using training data sets and then they are compared with each model and the best model is selected and used. This project mainly focuses on predicting an outcome such as Outcome (died, authorized, lived), so it falls under natural regression functions. Therefore, 6 methods are selected to obtain optimal results from the training data. All these ensemble methods are selected by machine learning techniques.

- Ensemble techniques

The reliability and accuracy of data using equine health outcomes is enhanced by the inclusion of cohort approaches. This involves repeatedly applying the same algorithm or combining several different algorithms among techniques to create a model that performs more accurately. The main goal of doing this is to minimize errors and reduce overfitting. Ensemble approaches are used to combine the outputs of many models and they perform well. It is important for specific tasks such as outcome prediction. Following are the various grouping techniques used in the development of the project.

That is

- SVM – Support vector machine
- Logistic model
- Random forest model
- Random forest - Reduced model
- XGBoost model
- Neutral network - sq model

e. Full model evaluation

This model is concerned with predicting the correct outcome of horses based on their health status. As mentioned above EDA is implemented to analyze the data set used in training the model to get the final desired result in predicting the results of the horses. After that analysis, the best parameters to proceed with the training process are determined.

Advanced analysis - ML model building

To build Machine Learning, the data in the Train dataset has to be re-viewed.

```
data
```

	surgery	age	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	...	packed_cell_volume	total_pr
0	yes	adult	38.1	132.0	24.0	cool	reduced	dark_cyanotic	more_3_sec	depressed	...	57.0	
1	yes	adult	37.5	88.0	12.0	cool	normal	pale_cyanotic	more_3_sec	mild_pain	...	33.0	
2	yes	adult	38.3	120.0	28.0	cool	reduced	pale_pink	less_3_sec	extreme_pain	...	37.0	
3	yes	adult	37.1	72.0	30.0	cold	reduced	pale_pink	more_3_sec	mild_pain	...	53.0	
4	no	adult	38.0	52.0	48.0	normal	normal	normal_pink	less_3_sec	alert	...	47.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1230	yes	adult	38.5	129.0	48.0	cool	reduced	pale_pink	more_3_sec	depressed	...	57.0	
1231	yes	adult	37.5	60.0	50.0	cool	reduced	pale_cyanotic	less_3_sec	mild_pain	...	35.0	
1232	yes	young	37.5	84.0	40.0	normal	reduced	normal_pink	less_3_sec	mild_pain	...	40.0	
1233	yes	adult	38.1	70.0	16.0	normal	reduced	bright_red	less_3_sec	mild_pain	...	58.0	
1234	yes	adult	38.1	54.0	36.0	normal	normal	pale_pink	less_3_sec	mild_pain	...	45.0	

1235 rows × 27 columns

Model evaluation for best model

This code calculates the accuracy of machine learning models, and F1 score metrics. Here, 6 different regression models were proposed to predict the target variables. Namely SVM, Logistic, Random Forest, Random Forest-reduced, XGBOOST and Neural Network-sq. Here all these models are trained on datasets ready for evaluation. The XGBoost model is colored green as the best model.

```
accuracy = [accuracy_svm, accuracy_logistic, accuracy_rf, accuracy_rf_reduced_model, accuracy_xgb, accuracy_nm]
f1_score = [f1_svm, f1_logistic, f1_rf, f1_rf_reduced_model, f1_xgb, f1_nm]

Models = ["SVM", "Logistic", "Random Forest", "Random Forest - Reduced", "XGBoost", "Neural Network-sq"]

model_table = pd.DataFrame({"Model": Models, "Accuracy": accuracy, "F1-Score": f1_score})

# Find the index of the row with the maximum F1-score
best_model_index = model_table["F1-Score"].idxmax()

# Create a new DataFrame with the best model highlighted
highlighted_model_table = model_table.style.apply(lambda x: ["background: lightgreen" if x.name == best_model_index else "" for i in x], axis=1)

highlighted_model_table
```

	Model	Accuracy	F1-Score
0	SVM	0.627530	0.629472
1	Logistic	0.635628	0.636034
2	Random Forest	0.700405	0.702133
3	Random Forest - Reduced	0.672065	0.674336
4	XGBoost	0.704453	0.706219
5	Neural Network-sq	0.603239	0.602966

Implementation details

SVM model

- with hyperparameter optimization

Here first, in describing the SVM model, the model has been predicted with hyperparameter optimization and without hyperparameter optimization. First let's consider model training with hyperparameter optimization. First the relevant dataset is loaded and then the missing values are removed. Then convert categorical variables to numeric variables using label Encoder. Then the data from numerical variables are mapped to feature and target variables. Then split into training and test datasets. That division then defines the parametric grid. Then instantiate it using the GridSearchCV object. It then fits the GridSearchCV object to the training data. After fitting, the best parameters and best estimates are calculated. Then the best estimator is trained on the full training set. It makes predictions on the test set using that best estimate. Then the model is evaluated. It creates a confusing matrix. And then finally plot the heat map.

```
# Load the dataset
data = df

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly'],
    'gamma': ['scale', 'auto']
}
```

```
# Instantiate the GridSearchCV object
grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator_svm = grid_search.best_estimator_

# Train the best estimator on the full training set
best_estimator_svm.fit(X_train, y_train)

# Make predictions on the testing set using the best estimator
y_pred = best_estimator_svm.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred)
f1_svm = f1_score(y_test, y_pred, average='weighted')
```

```
print("Best Parameters:", best_params)
print("Accuracy:", accuracy_svm)
print("F1-Score:", f1_svm)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Here, when hyperparameter optimization is done, the accuracy of that model is reduced, so it has been commented in the code.

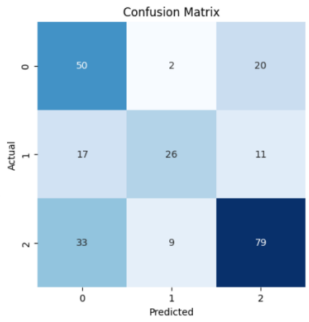
- Without hyper parameter optimization

Here the SVM model performs model prediction without hyperparameter optimization. First the relevant dataset is loaded and then the missing values are removed. Then convert categorical variables to numeric variables using label Encoder. Then the data from numerical variables are mapped to feature and target variables. Then split into training and test datasets. Then the SVM model is trained. Predictions are then made on the test set. The model is then trained. After training, a confusion matrix is formed. The plots then form the heat map.

```python
# Load the dataset
data = df

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = svm_model.predict(X_test)
```

```python
# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred)
f1_svm = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_svm)
print("F1-Score:",f1_svm)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.6275303643724697
F1-Score: 0.6294717114790555
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.69      0.58        72
           1       0.70      0.48      0.57        54
           2       0.72      0.65      0.68       121

    accuracy                           0.63       247
   macro avg       0.64      0.61      0.61       247
weighted avg       0.65      0.63      0.63       247
```



Logistic model

- with hyper parameter optimization

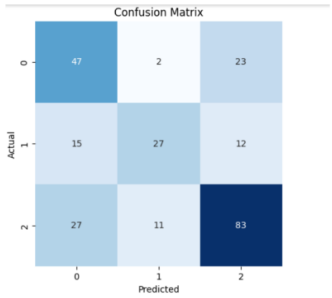
Second, let's consider how Logistic model performs one hyper parameter optimization. First the relevant dataset is loaded and then the missing values are removed. Then other features and target variables are considered. That is, X considers all the remaining columns of the data set except the outcome one, and only the outcome column is considered by y. Here x predicts y based on features. Then the classification columns are identified. Then one-hot encoding is applied to the classification columns. This converts categorical variables into numerical forms. Then split into training and test datasets. Then after that division defines the parametric grid. Then instantiate it using the GridSearchCV object. It then fits the GridSearchCV object to the training data. After fitting, the best parameters and best estimates are calculated. Then the best estimator is trained on the full training set. It makes predictions on the test set using that best estimate. Then the model is evaluated. It creates a confusing matrix. And then finally plot the heat map.

```
# Load the dataset
data = df

# Drop rows with missing values
data.dropna(inplace=True)

# Separate features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Identify categorical columns
categorical_columns = X.select_dtypes(include=['object']).columns

# Apply one-hot encoding to categorical columns
transformer = ColumnTransformer(transformers=[('onehot', OneHotEncoder(), categorical_columns)], remainder='passthrough')
X_encoded = transformer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)


# Define the parameter grid
param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}
```

```python
# Instantiate the GridSearchCV object
grid_search = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator_logistic = grid_search.best_estimator_

# Train the best estimator on the full training set
best_estimator_logistic.fit(X_train, y_train)

# Make predictions on the testing set using the best estimator
y_pred = best_estimator_logistic.predict(X_test)

# Evaluate the model
accuracy_logistic = accuracy_score(y_test, y_pred)
f1_logistic = f1_score(y_test, y_pred, average='weighted')

print("Best Parameters:", best_params)
print("Accuracy:", accuracy_logistic)
print("F1-Score:", f1_logistic)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```python
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

	precision	recall	f1-score	support
0	0.53	0.65	0.58	72
1	0.68	0.50	0.57	54
2	0.70	0.69	0.69	121
accuracy			0.64	247
macro avg	0.64	0.61	0.62	247
weighted avg	0.65	0.64	0.64	247



Random Forest model

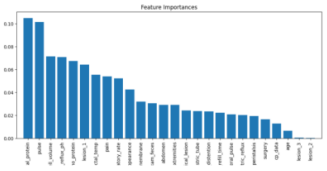
Then let's consider how to make prediction for a random forest model. First the relevant dataset is loaded and then the missing values are removed. Then convert categorical variables to numeric variables using labelEncoder. Then the data from numerical variables are mapped

to feature and target variables. Then split into training and test datasets. Then the Random forest model is trained. Predictions are then made on the test set. Then consider the feature importance. This arranges the features in descending order of importance. Then the significance of the plot feature is considered with the help of Accuracy and F1 score. A confusion matrix is then formed. The plots then form the heat map.

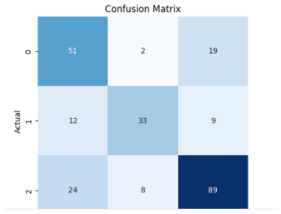
```
# Load the dataset
data = df.copy()

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders_rf = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders_rf[column] = LabelEncoder()
    data[column] = label_encoders_rf[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = rf_model.predict(X_test)
```

```
# Get feature importances
importances = rf_model.feature_importances_
features = X.columns

# Sort feature importances in descending order
indices = importances.argsort()[::-1]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), [features[i] for i in indices], rotation=90)
plt.xlim([-1, X.shape[1]])
plt.tight_layout()
plt.show()

accuracy_rf = accuracy_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_rf)
print("F1-Score:",f1_rf)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



	precision	recall	f1-score	support
0	0.59	0.71	0.64	72
1	0.77	0.61	0.68	54
2	0.76	0.74	0.75	121
accuracy			0.70	247
macro avg	0.70	0.68	0.69	247
weighted avg	0.71	0.70	0.70	247



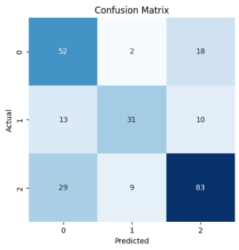
Random forest - Reduced model

Here, let's consider the reduced model used for feature selection for random forests with the limit calculated as 0.02. First the data is partitioned into features and target variables. Then the Random forest model is trained. Predictions are then made on the test set. A confusion matrix is then formed. The plots then form the heat map.

```
selected_features = features[importances > 0.02]

# Split data into features and target variable
X = data[selected_features]
y = data['outcome']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest model
rf_model_reduced = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_reduced.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = rf_model_reduced.predict(X_test)


accuracy_rf_reduced_model = accuracy_score(y_test, y_pred)
f1_rf_reduced_model = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_rf_reduced_model)
print("F1-Score:",f1_rf_reduced_model)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.6720647773279352
F1-Score: 0.6743361131248434
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.72      0.63        72
           1       0.74      0.57      0.65        54
           2       0.75      0.69      0.72       121

    accuracy                           0.67       247
   macro avg       0.68      0.66      0.66       247
weighted avg       0.69      0.67      0.67       247
```



XGBoost model

- with hyper parameter optimization

Here first, in describing the XGBoost model, the model has been predicted with hyperparameter optimization and without hyperparameter optimization. First let's consider model training with hyperparameter optimization. After first installing the XGBoost model, the related dataset is loaded and then the missing values are removed. Then convert categorical variables to numeric variables using labelEncoder. Then the data from numerical variables are mapped to feature and target variables. Then split into training and test datasets. That division then defines the parametric grid. Then instantiate it using the GridSearchCV object. It then fits the GridSearchCV object to the training data. After fitting, the best parameters and best estimates are calculated. Then the best estimator is trained on the full training set. It makes predictions on the test set using that best estimate. Then the model is evaluated. It creates a confusing matrix. And then finally plot the heat map.

```
# Load the dataset
data = df

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders_xgb = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders_xgb[column] = LabelEncoder()
    data[column] = label_encoders_xgb[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter grid
param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 200, 300],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2]

}
```

```
# Instantiate the GridSearchCV object
grid_search = GridSearchCV(XGBClassifier(objective='multi:softmax', num_class=len(set(y)), random_state=42),
                           param_grid,
                           cv=5,
                           scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator_xgb = grid_search.best_estimator_

# Train the best estimator on the full training set
best_estimator_xgb.fit(X_train, y_train)

# Make predictions on the testing set using the best estimator
y_pred = best_estimator_xgb.predict(X_test)

# Evaluate the model
accuracy_xgb = accuracy_score(y_test, y_pred)
f1_xgb = f1_score(y_test, y_pred, average='weighted')
```

```
print("Best Parameters:", best_params)
print("Accuracy:", accuracy_xgb)
print("F1-Score:", f1_xgb)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Here, when hyperparameter optimization is done, the accuracy of that model is reduced, so it has been commented in the code.

- without hyper parameter optimization

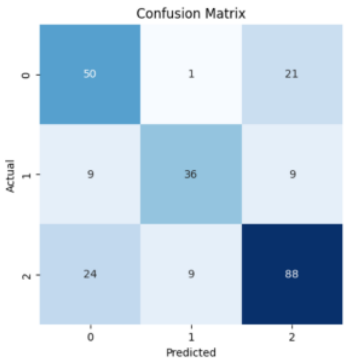
Here the XGBoost model performs model prediction without hyperparameter optimization. First the relevant dataset is loaded and then the missing values are removed. Then convert categorical variables to numeric variables using labelEncoder. Then the data from numerical variables are mapped to feature and target variables. Then split into training and test datasets. Then XGBoost performs model training. Predictions are then made on the test set. A confusion matrix is then formed. The plots then form the heat map.

```
# Load the dataset
data = df

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders_xgb = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders_xgb[column] = LabelEncoder()
    data[column] = label_encoders_xgb[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the XGBoost model
xgb_model = XGBClassifier(objective='multi:softmax', num_class=len(set(y)), random_state=42)
xgb_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = xgb_model.predict(X_test)
```

```
accuracy_xgb = accuracy_score(y_test, y_pred)
f1_xgb = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_xgb)
print("F1-Score:",f1_xgb)
print("Classification Report:")
print(classification_report(y_test, y_pred))


# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Identifying missing values, evaluating data types in different columns, is useful for understanding the structure, and provides a description of the x function. It includes non-null count and data type one.

```
Accuracy: 0.7044534412955465
F1-Score: 0.7062194956019678
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.69      0.65        72
           1       0.78      0.67      0.72        54
           2       0.75      0.73      0.74       121

    accuracy                           0.70       247
   macro avg       0.71      0.70      0.70       247
weighted avg       0.71      0.70      0.71       247
```




```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1235 entries, 0 to 1234
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   surgery                1235 non-null   int64
 1   age                    1235 non-null   int64
 2   rectal_temp            1235 non-null   float64
 3   pulse                  1235 non-null   float64
 4   respiratory_rate       1235 non-null   float64
 5   temp_of_extremities    1235 non-null   int64
 6   peripheral_pulse       1235 non-null   int64
 7   mucous_membrane        1235 non-null   int64
 8   capillary_refill_time  1235 non-null   int64
 9   pain                   1235 non-null   int64
 10  peristalsis            1235 non-null   int64
 11  abdominal_distention   1235 non-null   int64
 12  nasogastric_tube       1235 non-null   int64
 13  nasogastric_reflux     1235 non-null   int64
 14  nasogastric_reflux_ph  1235 non-null   float64
 15  rectal_exam_feces      1235 non-null   int64
 16  abdomen                1235 non-null   int64
 17  packed_cell_volume     1235 non-null   float64
 18  total_protein          1235 non-null   float64
 19  abdomo_appearance      1235 non-null   int64
 20  abdomo_protein         1235 non-null   float64
 21  surgical_lesion        1235 non-null   int64
 22  lesion_1               1235 non-null   int64
 23  lesion_2               1235 non-null   int64
```

The dataset provides a description of the non-null count and data type of the data in the test dataset.

```
testdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   surgery                824 non-null    object
 1   age                    824 non-null    object
 2   rectal_temp            824 non-null    float64
 3   pulse                  824 non-null    float64
 4   respiratory_rate       824 non-null    float64
 5   temp_of_extremities    824 non-null    object
 6   peripheral_pulse       824 non-null    object
 7   mucous_membrane        824 non-null    object
 8   capillary_refill_time  824 non-null    object
 9   pain                   824 non-null    object
 10  peristalsis            824 non-null    object
 11  abdominal_distention   824 non-null    object
 12  nasogastric_tube       824 non-null    object
 13  nasogastric_reflux     824 non-null    object
 14  nasogastric_reflux_ph  824 non-null    float64
 15  rectal_exam_feces      824 non-null    object
 16  abdomen                824 non-null    object
 17  packed_cell_volume     824 non-null    float64
 18  total_protein          824 non-null    float64
 19  abdomo_appearance      824 non-null    object
 20  abdomo_protein         824 non-null    float64
 21  surgical_lesion        824 non-null    object
 22  lesion_1               824 non-null    object
 23  lesion_2               824 non-null    object
 24  lesion_3               824 non-null    object
 25  cp_data                824 non-null    object
dtypes: float64(7), object(19)
memory usage: 167.5+ KB
```

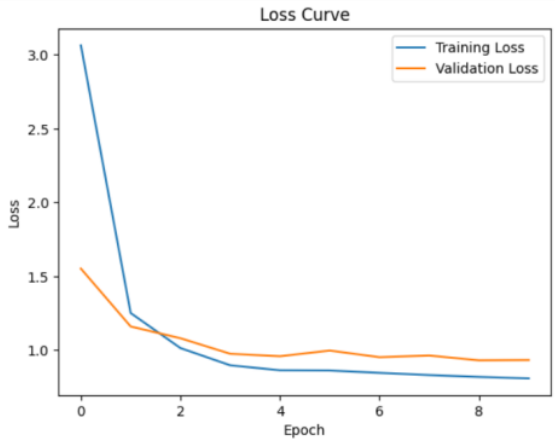
Neural Network model

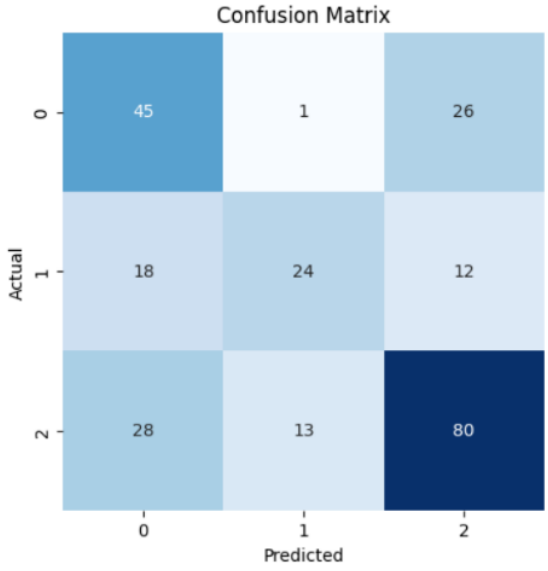
Here, when describing the neutral network model, the relevant dataset is first loaded and then the missing values are removed. Then convert categorical variables to numeric variables using labelEncoder. Here the target variables are converted to classification. Then split into training and test datasets. Then the Natural network model is trained. Then compile the model. The model is then trained. Then the model is evaluated. It then makes predictions on the test set. It calculates one F1 score. A confusion matrix is then formed. The plots then form the heat map.

```
# Load the dataset
data = df.copy()

# Drop rows with missing values
data.dropna(inplace=True)

# Convert categorical variables to numerical using LabelEncoder
label_encoders_nn = {}
for column in data.select_dtypes(include=['object']).columns:
    label_encoders_nn[column] = LabelEncoder()
    data[column] = label_encoders_nn[column].fit_transform(data[column])

# Split data into features and target variable
X = data.drop('outcome', axis=1)
y = data['outcome']

# Convert target variable to categorical
y_categorical = to_categorical(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42)

# Build the neural network model
NNmodel = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(len(set(y)), activation='softmax')
])
```

```
# Compile the model
NNmodel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = NNmodel.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.1)

# Plot the loss curve
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.legend()
plt.show()

# Evaluate the model
_, accuracy_nm = NNmodel.evaluate(X_test, y_test)
print("Accuracy:", accuracy_nm)

# Make predictions on the testing set
y_pred = NNmodel.predict(X_test)
y_pred_classes = [round(pred.argmax()) for pred in y_pred]
```

```python
# Calculate F1 score
f1_nm = classification_report(y_test.argmax(axis=1), y_pred_classes, output_dict=True)['weighted avg']['f1-score']
print("F1 Score:", f1_nm)

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test.argmax(axis=1), y_pred_classes)

# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```





Saving the model

This code is used to encode numerical values of categorical features.

```
label_encoders
```

```
{'surgery': LabelEncoder(),
 'age': LabelEncoder(),
 'temp_of_extremities': LabelEncoder(),
 'peripheral_pulse': LabelEncoder(),
 'mucous_membrane': LabelEncoder(),
 'capillary_refill_time': LabelEncoder(),
 'pain': LabelEncoder(),
 'peristalsis': LabelEncoder(),
 'abdominal_distention': LabelEncoder(),
 'nasogastric_tube': LabelEncoder(),
 'nasogastric_reflux': LabelEncoder(),
 'rectal_exam_feces': LabelEncoder(),
 'abdomen': LabelEncoder(),
 'abdomo_appearance': LabelEncoder(),
 'surgical_lesion': LabelEncoder(),
 'lesion_1': LabelEncoder(),
 'lesion_2': LabelEncoder(),
 'lesion_3': LabelEncoder(),
 'cp_data': LabelEncoder(),
 'outcome': LabelEncoder()}
```

This code is used to compress the label code and compress the trained model to a file.

Accordingly, label_encoders and XGBoost model are compressed successfully.

```
# Save the label encoders
joblib.dump(label_encoders, "label_encoders.pkl")
print("label_encoders saved successfully as", "encoded")


# Save the trained model to a file
filename = 'xgboost_model.pkl'
pickle.dump(xgb_model, open(filename, 'wb'))

print("XGBoost model saved successfully as", filename)
```

```
label_encoders saved successfully as encoded
XGBoost model saved successfully as xgboost_model.pkl
```

Making predictions for new test data

- getting predictions using XGBoost model

Here first the data type of the column is converted from int64 to object data type. and then encodes categorical variable labels. Each classification column is then stored in a label code. Predictions are then made based on the preprocessed test data. The numerical predictions are then mapped to their original labels. The numerical predictions are then converted to their corresponding labels. Then printing occurs as the original prediction.

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])

# Make predictions on the preprocessed testdata
predictions = xgb_model.predict(testdata)
#predictions = best_estimator_xgb.predict(testdata)


# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in predictions]
```

```
# Print the original predictions
print(original_predictions_labels)
```

```
['lived', 'lived', 'lived', 'euthanized', 'lived', 'died', 'lived', 'died', 'lived', 'lived', 'died', 'lived', 'lived', 'lived', 'lived', 'lived', 'lived', 'died', 'died', 'died', 'di
```

Preparing submission dataset

This code generates or updates the result column of the presentation data according to the results predicted by the machine learning models.

```
submissiondata['outcome'] = original_predictions_labels
```

In this code, the content of the submission data is directed to a CSV file called submission.csv.

```
submissiondata.to_csv('submission.csv')
submissiondata
```

	id	outcome
0	1235	lived
1	1236	lived
2	1237	lived
3	1238	euthanized
4	1239	lived
...	...	...
819	2054	died
820	2055	euthanized
821	2056	lived
822	2057	lived
823	2058	lived

824 rows × 2 columns

Getting predictions using models

Here first the data type of the column is converted from int64 to object data type. and then encodes categorical variable labels. Each classification column is then stored in a label code. Predictions are then made based on the preprocessed test data. The numerical predictions are then mapped to their original labels. The numerical predictions are then converted to their corresponding labels. Then the original forecast is printed as submission-svm.csv, submission-logistic.csv, submission-randomf.csv, submission-randomreduced.csv.

- Getting predictions using SVM model

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])

# Make predictions on the preprocessed testdata
predictions = svm_model.predict(testdata)
#predictions = best_estimator_svm.predict(testdata)

# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in predictions]

# Print the original predictions
print(original_predictions_labels)
```

```
submissiondata['outcome'] = original_predictions_labels
submissiondata.to_csv('submission-svm.csv')
```

```
['lived', 'lived', 'lived', 'euthanized', 'lived', 'died', 'died', 'died', 'died', 'lived', 'died', 'lived', 'euthanized', 'euthanized', 'died', 'lived', 'lived', 'died', 'euthanized
```

- Getting predictions using Logistic model

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])

# Make predictions on the preprocessed testdata
predictions = best_estimator_logistic.predict(testdata)

# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in predictions]

# Print the original predictions
print(original_predictions_labels)


submissiondata['outcome'] = original_predictions_labels
submissiondata.to_csv('submission-logistic.csv')
```

```
['lived', 'lived', 'lived', 'euthanized', 'lived', 'died', 'lived', 'died', 'died', 'lived', 'lived', 'lived', 'euthanized', 'euthanized', 'died', 'lived', 'lived', 'died', 'euthanize
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

- Getting predictions using Random Forest model

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])

# Make predictions on the preprocessed testdata
predictions = rf_model.predict(testdata)

# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in predictions]

# Print the original predictions
print(original_predictions_labels)
```

```
submissiondata['outcome'] = original_predictions_labels
submissiondata.to_csv('submission-randomf.csv')
```

```
['lived', 'died', 'lived', 'euthanized', 'lived', 'died', 'died', 'died', 'lived', 'lived', 'died', 'lived', 'euthanized', 'lived', 'died', 'lived', 'lived', 'died', 'died', 'died', '
```

- Getting predictions using Random Forest reduced model

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])
X = data[selected_features]
# Make predictions on the preprocessed testdata
predictions = rf_model_reduced.predict(testdata[selected_features])

# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in predictions]

# Print the original predictions
print(original_predictions_labels)
```

```
submissiondata['outcome'] = original_predictions_labels
submissiondata.to_csv('submission-randomfreduced.csv')
```

```
['lived', 'died', 'lived', 'euthanized', 'lived', 'died', 'died', 'died', 'lived', 'lived', 'lived', 'lived', 'lived', 'lived', 'died', 'lived', 'lived', 'died', 'died', 'died', 'died
```

- Getting predictions using Neural Network model

```
lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

for column in lesion_columns:
    if testdata[column].dtype == 'int64':
        testdata[column] = testdata[column].astype('object')

# Label encode categorical variables
label_encoders = {}  # Store label encoders for each categorical column
categorical_columns = testdata.select_dtypes(include=['object']).columns
for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    testdata[column] = label_encoders[column].fit_transform(testdata[column])

y_pred = NNmodel.predict(testdata)
y_pred_classes = [round(pred.argmax()) for pred in y_pred]

# Map numerical predictions to their original labels
prediction_labels = {0: 'died', 1: 'euthanized', 2: 'lived'}

# Convert numerical predictions to their corresponding labels
original_predictions_labels = [prediction_labels[prediction] for prediction in y_pred_classes]

# Print the original predictions
print(original_predictions_labels)


submissiondata['outcome'] = original_predictions_labels
submissiondata.to_csv('submission-NeuralN.csv')
```

```
26/26 [==============================] - 0s 1ms/step
['lived', 'died', 'lived', 'euthanized', 'lived', 'died', 'died', 'died', 'lived', 'lived', 'lived', 'lived', 'lived', 'euthanized', 'died', 'lived', 'lived', 'died', 'lived', 'lived',
```

Here the numerical variables lesion_1,lesion_2 and lesion_3 are associated with the XGBoost model, which makes the web application easy to build and has a significant accuracy of 76%.

Here data loading and preprocessing are done first and missing values columns are removed. Then the numerical forms for the categorical variables are converted and the labels are coded. XGBClassifier is then trained on the training data and predicts on the test set and generates a classification report showing precision, recall and F1 score and support. Then XGBClassifier and labelEncoder, trained by pickle and joblib, compress a disc. Accordingly, create a file as label_encoders_new.pkl and xgboost_model_new.pkl.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, f1_score,confusion_matrix


trainset = pd.read_csv('D:\Horses\train.csv')
trainset.columns

trainset = trainset.drop(['id','hospital_number'], axis =1)

trainset.info()
```

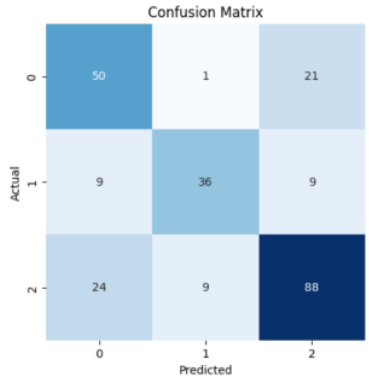
```
# ============================================================================
# lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']
#
# def convert_int_to_object(data, columns):
#     for column in columns:
#         if data[column].dtype == 'int64':
#             data[column] = data[column].astype('object')
#
# # Applying the function to traindata
# convert_int_to_object(trainset, lesion_columns)
# ============================================================================

trainset.isnull().sum()

trainset.dropna(inplace = True)

for i in trainset.select_dtypes(include=['object']).columns:
    print(i , ": ",trainset[i].unique())

# Convert categorical variables to numerical using LabelEncoder
label_encoders = {}
for column in trainset.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    trainset[column] = label_encoders[column].fit_transform(trainset[column])
```

```
# Split data into features and target variable
X = trainset.drop('outcome', axis=1)
y = trainset['outcome']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the XGBoost model
xgb_model = XGBClassifier(objective='multi:softmax', num_class=len(set(y)), random_state=42)
xgb_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = xgb_model.predict(X_test)

accuracy_xgb = accuracy_score(y_test, y_pred)
f1_xgb = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy_xgb)
print("F1-Score:",f1_xgb)
print("Classification Report:")
print(classification_report(y_test, y_pred))


# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```python
# Plot heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()


import pickle
import joblib
# Save the label encoders
joblib.dump(label_encoders, "label_encoders_new.pkl")
print("label_encoders saved successfully as", "label_encoders_new")


# Save the trained model to a file
filename = 'xgboost_model_new.pkl'
pickle.dump(xgb_model, open(filename, 'wb'))

print("XGBoost model saved successfully as", filename)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1235 entries, 0 to 1234
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   surgery               1235 non-null   object
 1   age                   1235 non-null   object
 2   rectal_temp           1235 non-null   float64
 3   pulse                 1235 non-null   float64
 4   respiratory_rate      1235 non-null   float64
 5   temp_of_extremities   1235 non-null   object
 6   peripheral_pulse      1235 non-null   object
 7   mucous_membrane       1235 non-null   object
 8   capillary_refill_time 1235 non-null   object
 9   pain                  1235 non-null   object
 10  peristalsis           1235 non-null   object
 11  abdominal_distention  1235 non-null   object
 12  nasogastric_tube      1235 non-null   object
 13  nasogastric_reflux    1235 non-null   object
 14  nasogastric_reflux_ph 1235 non-null   float64
 15  rectal_exam_feces     1235 non-null   object
 16  abdomen               1235 non-null   object
 17  packed_cell_volume    1235 non-null   float64
```

	precision	recall	f1-score	support
0	0.60	0.69	0.65	72
1	0.78	0.67	0.72	54
2	0.75	0.73	0.74	121
accuracy			0.70	247
macro avg	0.71	0.70	0.70	247
weighted avg	0.71	0.70	0.71	247




```
label_encoders saved successfully as label_encoders_new
XGBoost model saved successfully as xgboost_model_new.pkl
```

f. Conclusion of the final model

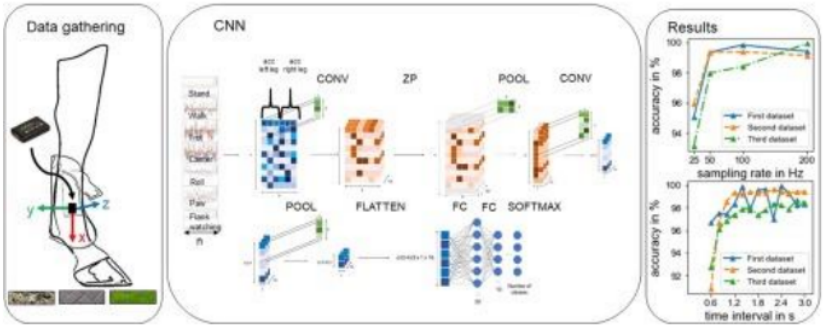
Throughout this prepared report, a number of different machine learning regression models have been evaluated to predict the outcome (died, authorized, lived) of horses using horse health status. From pre-processing processes as well as data exploration to model training and evaluation, it is concluded that the XGBoost model is the best model according to the comparisons based on the factors of Accuracy and F1 score related to each model. In addition to the XGBoost model, when compared with the other selected models, the main metric for predicting the outcome of the horses, the XGBoost model has an average relatively low absolute value of 0.70454. Therefore, another thing that can be concluded from this is that the essential health factors such as age, pain level, pulse, abdominal division to predict the outcome of the horses, this prediction is based on the results obtained from different models for predictions made from test and sample submission data sets. Shows correct evidence for.

Success of using deep learning techniques in the predict health outcomes of horses

Automatic Horse Activity Recognition by Convolutional Neural Networks Using Accelerometer Data

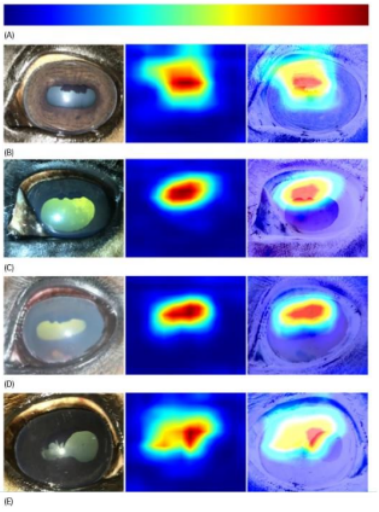
According to this report, observational behavior of horses is expected to provide important information about the health of horses. Here, the CNN model is expected to be used for tasks such as image classification that automatically extracts features using acceleration data to identify the behavior patterns of horses. Here, 7 horse activities were proposed to be classified based on deep learning approaches to identify horse activities. For the accuracy of horse activity detection, 6 test datasets were compiled based on the effect of the russified data sampling frequency, ground type and time series length factors, and 3 types of datasets are used to evaluate the model for the final conclusion. And the effect of a subject on the recognition accuracy is validated on a model with 50Hz sensor data processed at different sizes. Accordingly, at a sampling rate of 25Hz and a time interval of 2.1 seconds, 99% accuracy, at 69Hz and 2.4 seconds, four blind horse behaviors with the same accuracy, a sampling rate of 200Hz and 25Hz reduced accuracy by 4.75% and an accuracy of 0.6 seconds from 3 seconds. Decreases by 5.27%. Also the data normalization capability is valid on 50Hz

data. This leads to a mean classification accuracy of 97.84% and 96.10% in ten-fold validation between a lame horse and a pony. And here it shows a 0.24% decrease in the accuracy of using the data from the baseline with one sensitivity. According to this report, it can be concluded that 3 test data sets can achieve 99% accuracy using CNN model with sampling rate of 2.1-time interval and 25 Hz. (Computers and Electronics in Agriculture, n.d.)



Artificial intelligence highlights equine uveitis as a tool for differentiating equine eye diseases

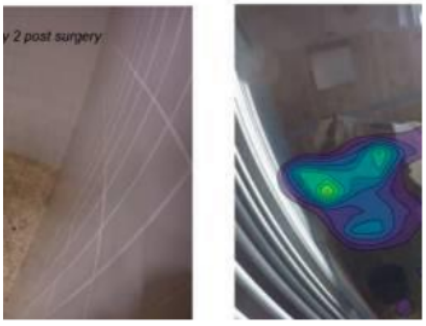
Nowadays, with the advancement of technology, the trend of experimenting in making new tools has increased. According to this report, the implementation of a mobile tool related to the eye diagnosis of uveitis, which can even lead to blindness in horses, and the visual classification system considers whether the horse exhibits uveitis or other eye diseases. In this way, the eyes affected by uveitis can be detected early and the eyes can be protected. Here, 2346 images of horses' eyes were trained by CNN or Convolutional Neural Network model and those images were increased to 9384 features. 261 unmodified images are used for the performance of the network being trained. This draws attention to less important areas of the horse's eye. It shows 99.82% accuracy on trained data and 96.66% accuracy on validated data in distinguishing equine eye disease from uveitis, other eye diseases, and healthy across 3 outcomes. (Anna May, n.d.)



Development and validation of an automated video tracking model for stable horses

This report examines whether horses are in good or poor health by observing their behavior. In this report it has been decided that an automated video tracking model for horses should be investigated. The primary objective is to improve the ability to automatically identify horses, assess horse pain and well-being. The basic analysis done here provides the opportunity to build an algorithm to automatically observe the behavior patterns of horses. Here the video analysis process has been done by labeling 21342 video files with a duration of 11.52 minutes. Here, Convolutional neural network Loopy has been used to automatically predict the video parts, and by considering the nose, withers and tail as the main points, the results have been obtained that there is a sensitivity of more than 80% and an error rate of 7%.

(Nuray kil, n.d.)

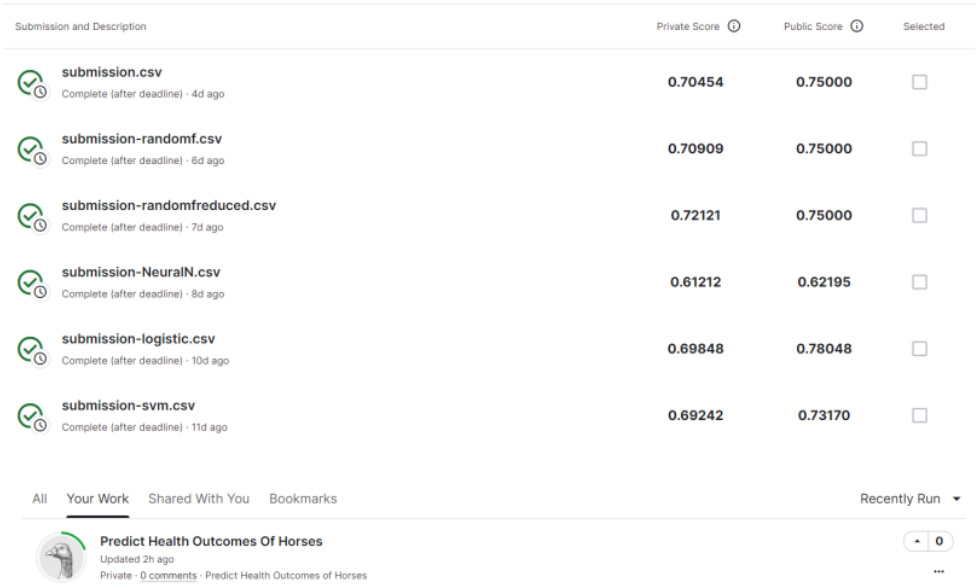


Appendix

Kaggle Competition - Predict Health outcomes of Horses

Competition Link - https://www.kaggle.com/competitions/playground-series-s3e22/overview

Screenshots related to the competition.



Software Artifact - Health outcomes of horses predictor

```python
import pandas as pd
import streamlit as st
import joblib
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder

# Load the trained model
xgb_model = joblib.load('C:\\Users\\acer\\OneDrive\\Desktop\\Horses\\xgboost_model_new.pkl')

# Load label encoders
with open('C:\\Users\\acer\\OneDrive\\Desktop\\Horses\\label_encoders_new.pkl', 'rb') as f:
    label_encoders = joblib.load(f)


# Remove 'outcome' label encoder
if 'outcome' in label_encoders:
    del label_encoders['outcome']


# Function to preprocess input data
def preprocess_input(data):
    # Perform label encoding for categorical variables
    for column, encoder in label_encoders.items():
        print(column, encoder)
        data[column] = encoder.transform(data[column])
    return data

# Function to make predictions
def predict_outcome(model, data):
    preprocessed_data = preprocess_input(data)
    prediction = model.predict(preprocessed_data)
    return prediction
```

```python
def convert_int_to_object(data, columns):
    for column in columns:
        if data[column].dtype == 'int64':
            data[column] = data[column].astype('object')

def reorder_columns(input_data):
    # Ensure columns are in the same order as during training

    expected_columns = [
    'surgery', 'age', 'rectal_temp', 'pulse', 'respiratory_rate',
    'temp_of_extremities', 'peripheral_pulse', 'mucous_membrane',
    'capillary_refill_time', 'pain', 'peristalsis', 'abdominal_distention',
    'nasogastric_tube', 'nasogastric_reflux', 'nasogastric_reflux_ph',
    'rectal_exam_feces', 'abdomen', 'packed_cell_volume', 'total_protein',
    'abdomo_appearance', 'abdomo_protein', 'surgical_lesion', 'lesion_1',
    'lesion_2', 'lesion_3', 'cp_data']

    return input_data.reindex(columns=expected_columns)


# Convert numerical prediction to text
def get_outcome_text(prediction_value):
    outcome_mapping = {0: 'Died', 1: 'Euthanized', 2: 'Lived'}
    return outcome_mapping.get(prediction_value, 'Unknown')

# Streamlit web app
def main():

    titleside = """
    <style>
    [data-testid="stMarkdownContainer"] {
        text-align: center;


    }
```

```
}

</style>
"""

st.markdown(titleside, unsafe_allow_html=True)
st.sidebar.title('Horse Health Prediction')



buttonside = """
<style>
[data-testid="stButton"]{
    padding-left: 100px;
}
</style>
"""

st.markdown(buttonside, unsafe_allow_html=True)




page_bg_img = """
<style>
[data-testid="stAppViewContainer"]{
    background-image: url("https://wallpapers.com/images/featured/horse-h3azzzaaorg8c9ay.jpg");
    background-size: cover;}


</style>
```

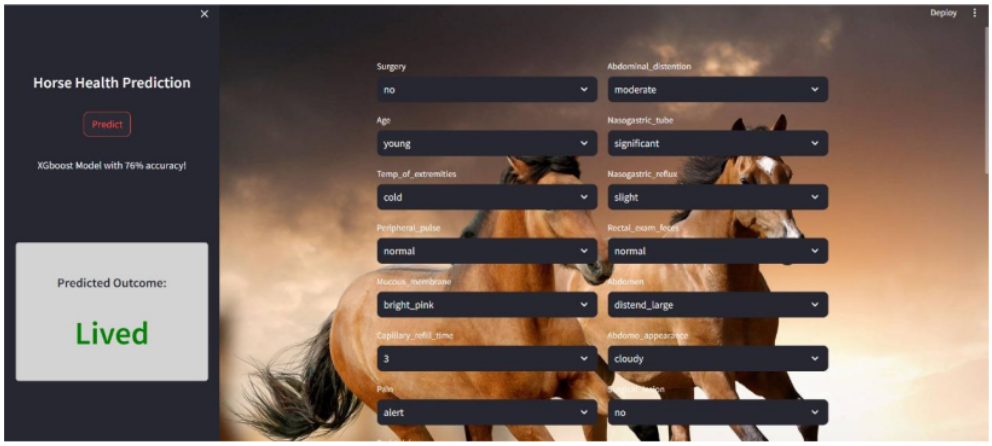
```
"""
st.markdown(page_bg_img, unsafe_allow_html=True)



headerbg = """
<style>
[data-testid="stHeader"]{
    background-image: url("https://wallpapers.com/images/featured/horse-h3azzzaaorg8c9ay.jpg");
    background-size: cover;}


</style>



"""
st.markdown(headerbg, unsafe_allow_html=True)


container = """
<style>
[data-testid="stAppViewBlockContainer"]{

    padding: 0rem 1rem 10rem;

</style>



"""
st.markdown(container, unsafe_allow_html=True)

sidet = """
<style>
```

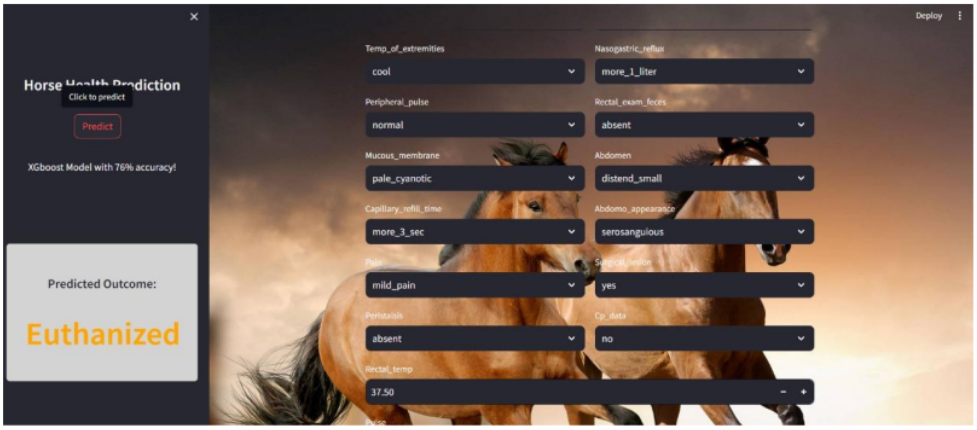
```
"""
st.markdown(container, unsafe_allow_html=True)

sidet = """
<style>
[data-testid="stSidebarUserContent"]{
    font-size: 2px;


}
</style>
"""
st.markdown(sidet, unsafe_allow_html=True)


# Load DataFrame
df = pd.read_csv('C:\\Users\\acer\\OneDrive\\Desktop\\Horses\\train.csv')

df.dropna(inplace = True)

#lesion_columns = ['lesion_1', 'lesion_2', 'lesion_3']

# Remove unnecessary columns
df = df.drop(['id', 'hospital_number','outcome'], axis=1)

# Applying the function to traindata
#convert_int_to_object(df, lesion_columns)

# Create dropdowns for categorical variables
categorical_vars = df.select_dtypes(include=['object']).columns
categorical_values = {col: df[col].unique().tolist() for col in categorical_vars}
```

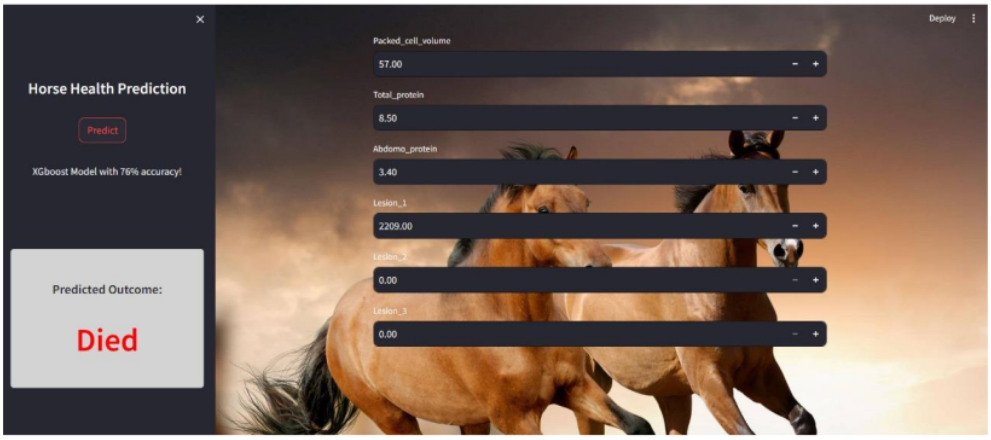
```
# Split categorical values into two halves
half_len = len(categorical_values) // 2
categorical_values_left = dict(list(categorical_values.items())[:half_len])
categorical_values_right = dict(list(categorical_values.items())[half_len:])


user_input = {}
left_column, right_column = st.columns(2)

with left_column:

    for feature, values in categorical_values_left.items():
        user_input[feature] = st.selectbox(feature.capitalize(), values)


with right_column:

    for feature, values in categorical_values_right.items():
        user_input[feature] = st.selectbox(feature.capitalize(), values)


# Add numerical input fields
numerical_vars = df.select_dtypes(include=['float64','int64']).columns
for feature in numerical_vars:
    user_input[feature] = st.number_input(feature.capitalize(), min_value=0.0)

st.sidebar.markdown("""
    <style>
        .centered-button {
            display: block;
            margin: 0 auto;
            width: 5%; /* Adjust the width as needed */
            text-align: center;
        }
    </style>
```

```
        }
    </style>
""", unsafe_allow_html=True)


if st.sidebar.button('Predict', key='predict_button', help='Click to predict',):
    # Convert user input to DataFrame
    input_data = pd.DataFrame([user_input])

    # Reorder columns
    input_data = reorder_columns(input_data)

    # Make prediction
    prediction = predict_outcome(xgb_model, input_data)


    # Convert numerical prediction to text
    predicted_outcome = get_outcome_text(prediction[0])
    # Style the predicted outcome text
    # Define colors based on predicted outcome
    outcome_colors = {'Died': 'red', 'Euthanized': 'orange', 'Lived': 'green'}
    outcome_color = outcome_colors.get(predicted_outcome, 'black')

    # Style the predicted outcome text with color
    predicted_outcome_html = f"<div style='position: absolute; top: 150px; left: 50%; transform: translateX(-50%); text-align:center; background-color: #D3D3D3; padd

    # Display styled predicted outcome
    st.sidebar.markdown(predicted_outcome_html, unsafe_allow_html=True)

    st.sidebar.write('XGboost Model with 76% accuracy!')


if __name__ == '__main__':
    main()
```

```
if __name__ == '__main__':
    main()
```




Horse Health Prediction
Click to predict
Predict
XGboost Model with 76% accuracy!
Predicted Outcome:
Euthanized
Deploy
Temp_of_extremities
cool
Nasogastric_reflux
more_1_liter
Peripheral_pulse
normal
Rectal_exam_feces
absent
Mucous_membrane
pale_cyanotic
Abdomen
distend_small
Capillary_refill_time
more_3_sec
Abdomo_appearance
serosanguious
Pain
mild_pain
Surgical_lesion
yes
Peristalsis
absent
Cp_data
no
Rectal_temp
37.50


Horse Health Prediction
Predict
XGboost Model with 76% accuracy!
Predicted Outcome:
Died
Deploy
Packed_cell_volume
57.00
Total_protein
8.50
Abdomo_protein
3.40
Lesion_1
2209.00
Lesion_2
0.00
Lesion_3
0.00

Malsha Madhurangi_st20267780_CL HDCSE 25 89_Computational Intelligence assignment.docx

ORIGINALITY REPORT

1%	1%	0%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eudl.eu Internet Source	<1%
2	www.researching.cn Internet Source	<1%
3	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1%
4	bpb-us-w2.wpmucdn.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		