



AY 2024/25 SEMESTER 1

SC4000 Machine Learning

Project Report

By Group 12

Name	Matric No.
Wong Zi Lun	U2240220D
Javen Ong Jing Hian	U2240610E
Thejesvi Ramesh	U2240554H
Loh Jyn Ern Daryl	U2240990B
Florian Goering	U2220036A

Table of Contents

1. Specific contributions of each group member	3
2. Introduction	3
2.1. Competition Description:	3
2.2. Dataset Description:	3
2.3. Exploratory Data Analysis	4
3. Data Preprocessing	9
3.1. Addressing duplicate records	9
3.2. Addressing missing unique categorical values	9
3.3. Addressing undefined or missing values	11
3.4. Transforming variables 'lesion_1', 'lesion_2' and 'lesion_3'	11
3.3. Imputation of missing values	12
3.4. Data Transformation	12
4. Proposed Model	13
5. Experimental study to demonstrate why your solution is effective	13
5.1. Exploring Model Stacking Configurations	13
1. Base Model Selection	13
2. Meta-Model Selection	14
3. Hyperparameter Tuning	14
5.2. Results of Stacking Experiments	14
5.3. Key Observations	15
5.4. Advantages of Stacking	15
6. Evaluation score, ranking position, and relative ranking of results for the competition	16
7. Conclusion	17
7.1. Importance of pre-processing data well	17
7.2. Variety of models available	17
7.3. Parameter choice	17
7.4. Limitations	18
8. References	19

1. Specific contributions of each group member

Name	Contributions
Wong Zi Lun	Data pre-processing Model selection and evaluation Project report
Javen Ong Jing Hian	Model selection and evaluation Project report Video
Thejesvi Ramesh	Data pre-processing Project report Video
Loh Jyn Ern Daryl	Data pre-processing Project report
Florian Goering	Model selection and evaluation Project report Video

2. Introduction

2.1. Competition Description:

We chose to participate in the Kaggle competition, “Predict Health Outcomes of Horses”. The competition required us to utilize machine learning techniques and data science skills to build a model that could accurately predict the health outcomes of horses, given a dataset containing various medical indicators of the horses.

Our model had to be able to accurately predict whether a horse lived, died or was euthanized, and we received a final score based on our predictions evaluated on a micro-averaged F1-Score between the predicted and actual test values.

2.2. Dataset Description:

We were provided 4 files:

- train.csv - the training dataset, containing 1235 records and 29 variables
- test.csv - the test dataset, containing 824 records and 28 variables
- sample_submission.csv - a sample submission file in the correct format
- horse.csv - the original dataset, containing 299 records and 28 variables

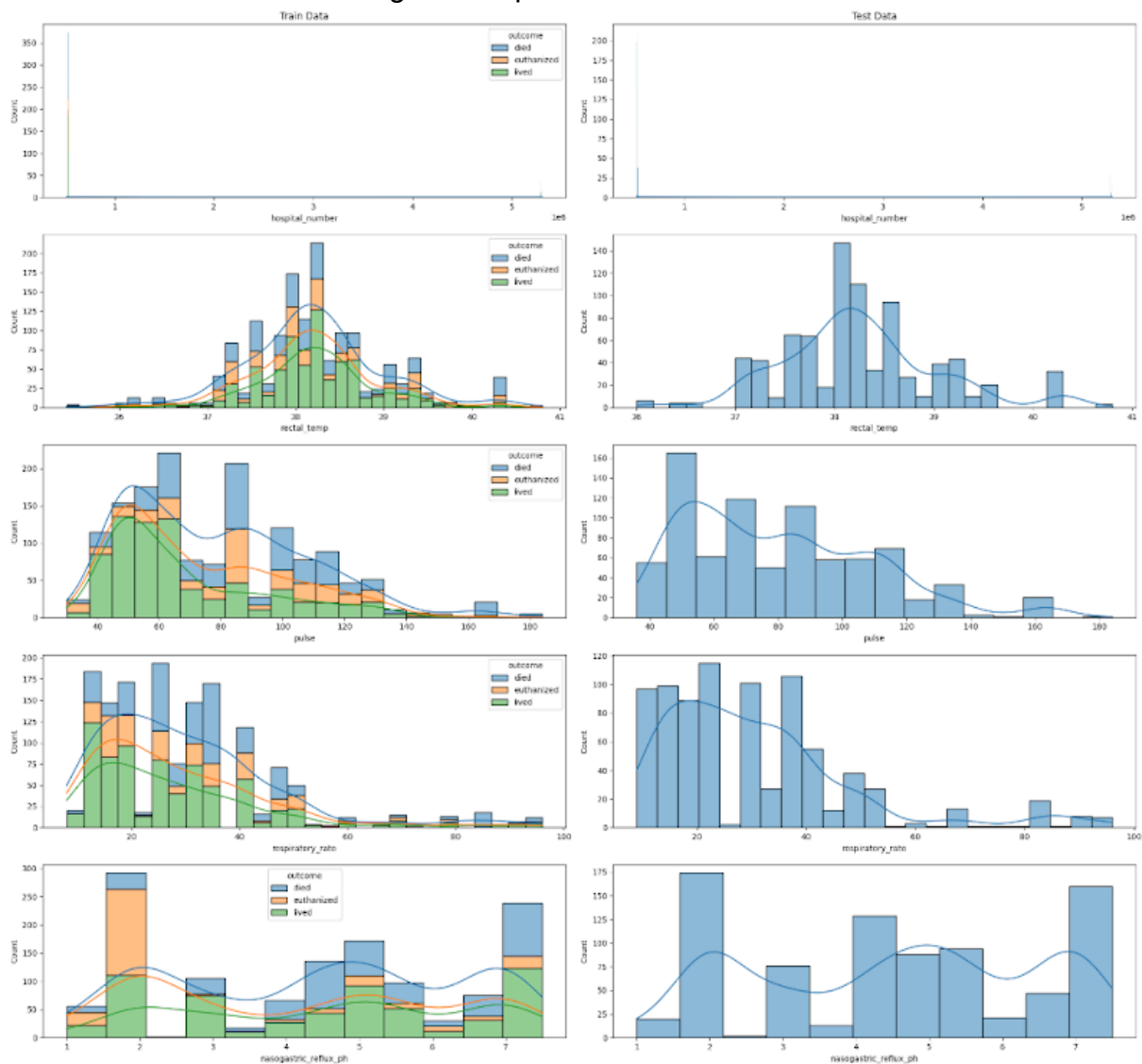
It was mentioned that the datasets provided (train.csv and test.csv) were generated from a deep learning model trained on a portion of the Horse Survival Dataset (horse.csv).

We decided to use both the training data and original data in training our model to improve generalization and reduce overfitting by exposing the model to a more diverse set of examples. The original dataset also serves as additional validation, providing an unbiased evaluation of the model's performance and aiding with tuning model parameters.

The new concatenated training dataset formed from combining train.csv and horse.csv contains 1534 records and 28 variables, excluding the 'id' variable in train.csv that will be irrelevant in training our model.

2.3. Exploratory Data Analysis

To first gain an understanding of the variables we will be working with, we opted to plot the distributions of our numerical and categorical variables against our target variable, 'outcome'. This was done using the matplotlib and seaborn libraries.



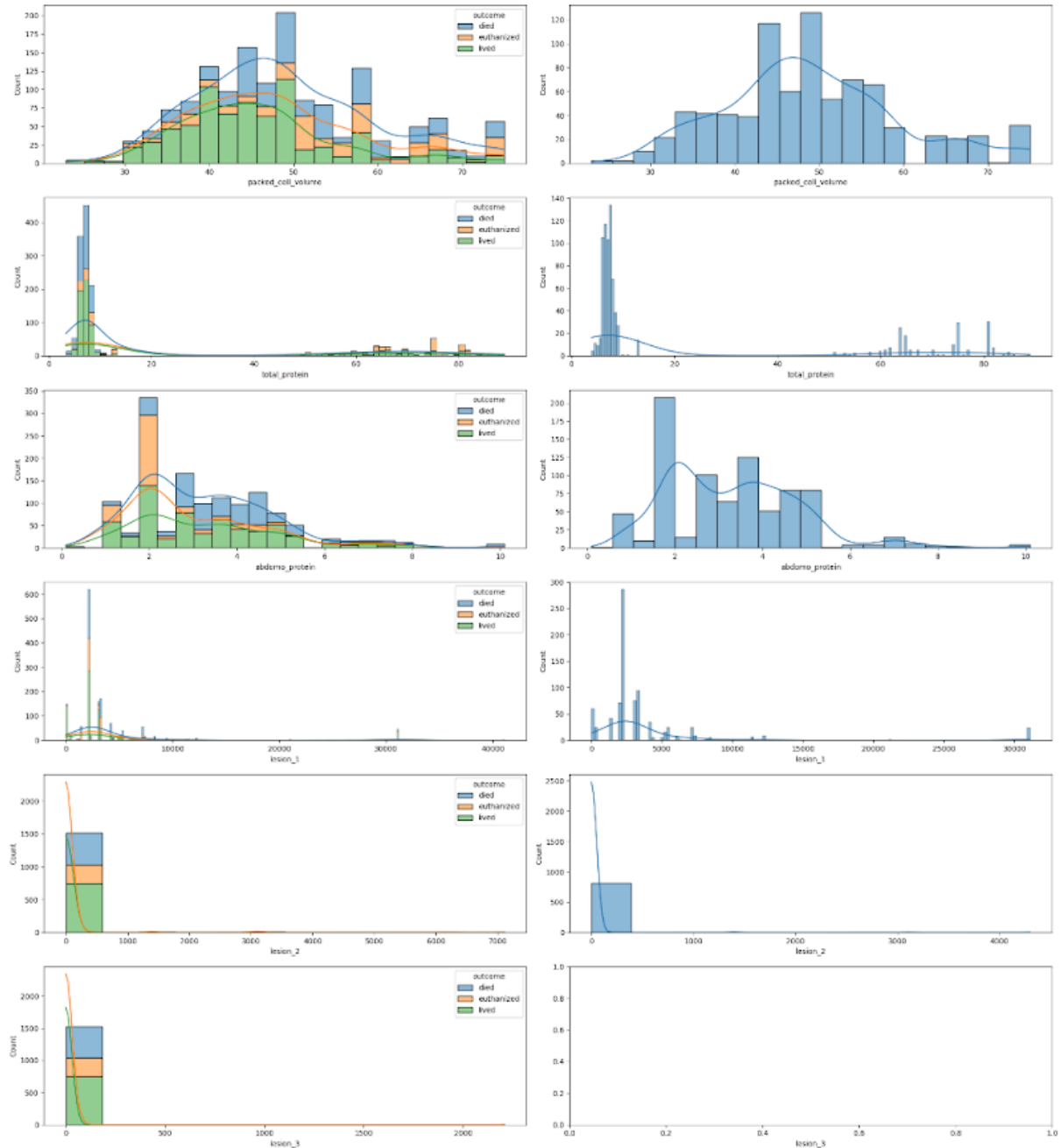


Figure 1: Distribution of numerical variables

From our plot of numerical variables above, we can observe that the distributions of our train and test data are quite similar. However, there are 2 issues with this data:

1. The records of the variable 'lesion_3' in test data are empty
2. The variable 'hospital_number' should be a categorical variable



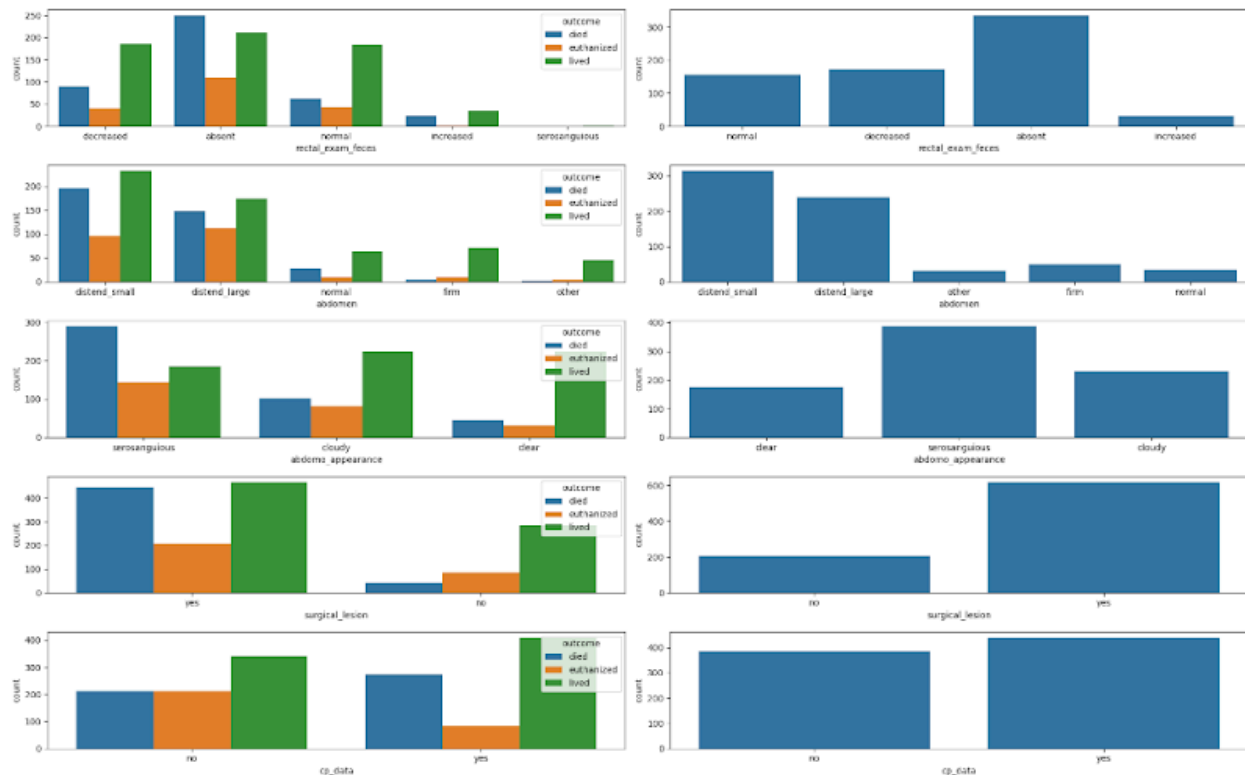


Figure 2: Distribution of categorical variables

From our plot of categorical variables above, we can observe some problems with the data:

- The values 'slight' and 'moderate' under the variable 'pain' only appears in the train and test data respectively
- The value 'distend_small' under the variable 'peristalsis' only appears in the train data
- The value 'slight' under the variable 'nasogastric_discharge' only appears in the train data
- The value 'serosanguinous' under the variable 'rectal_exam_feces' only appears in the train data

We will also check the train and test data for duplicate and missing values.

```
# Check for duplicate data in train_data_new
train_data_new[train_data_new.duplicated(keep=False)]
```

✓ 0.0s Python

	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	...	packed_cell_volume	total_protein
415	no	adult	530242	37.2	108.0	12.0	cool	reduced	pale_cyanotic	more_3_sec	...	52.0	8.2
621	yes	young	5290409	39.1	164.0	84.0	cold	normal	dark_cyanotic	more_3_sec	...	48.0	7.2
635	yes	adult	534787	38.0	36.0	16.0	cool	normal	normal_pink	less_3_sec	...	37.0	75.0
1238	yes	young	5290409	39.1	164.0	84.0	cold	normal	dark_cyanotic	more_3_sec	...	48.0	7.2
1266	no	adult	530242	37.2	108.0	12.0	cool	reduced	pale_cyanotic	more_3_sec	...	52.0	8.2
1510	yes	adult	534787	38.0	36.0	16.0	cool	normal	normal_pink	less_3_sec	...	37.0	75.0

6 rows × 28 columns

Figure 3: Duplicate records in train data

```
# Check for duplicate data in test_data
test_data[test_data.duplicated(keep=False)]
✓ 0.0s
```

surgery age hospital_number rectal_temp pulse respiratory_rate temp_of_extremities peripheral_pulse mucous_membrane capillary_refill_time ... abdomen packed_cell_volume total_protein

0 rows x 27 columns

Figure 4: Duplicate records in test data

There are also numerous missing values in our train and test data, so we will also need to decide on a strategy to deal with this later.

```
# Check for missing data in train_data_new
train_data_new.isnull().sum()
✓ 0.0s
```

surgery	0
age	0
hospital_number	0
rectal_temp	60
pulse	24
respiratory_rate	58
temp_of_extremities	95
peripheral_pulse	129
mucous_membrane	68
capillary_refill_time	38
pain	99
peristalsis	64
abdominal_distention	79
nasogastric_tube	184
nasogastric_reflux	127
nasogastric_reflux_ph	246
rectal_exam_feces	292
abdomen	331
packed_cell_volume	29
total_protein	33
abdomo_appearance	213
abdomo_protein	198
surgical_lesion	0
lesion_1	0
lesion_2	0
lesion_3	0
cp_data	0
outcome	0

dtype: int64

```
# Check for missing data in test_data
test_data.isnull().sum()
✓ 0.0s
```

surgery	0
age	0
hospital_number	0
rectal_temp	0
pulse	0
respiratory_rate	0
temp_of_extremities	35
peripheral_pulse	47
mucous_membrane	13
capillary_refill_time	6
pain	29
peristalsis	19
abdominal_distention	22
nasogastric_tube	64
nasogastric_reflux	14
nasogastric_reflux_ph	0
rectal_exam_feces	125
abdomen	154
packed_cell_volume	0
total_protein	0
abdomo_appearance	31
abdomo_protein	0
surgical_lesion	0
lesion_1	0
lesion_2	0
lesion_3	0
cp_data	0

dtype: int64

Figure 5: Missing values in train (left) and test (right) data

We then generate an overview of our train and test data.

	nunique	count	naCount	%OfNa	dtype	count	mean	std	min	25%	50%	75%	max
surgery	2	1534	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	2	1534	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hospital_number	288	1534	0	0.000000	int64	1534.0	980469.409387	1.392849e+06	518476.0	528806.0	529840.0	534259.25	5305629.0
rectal_temp	43	1474	60	0.039113	float64	1474.0	38.196744	7.798830e-01	35.4	37.8	38.2	38.60	40.8
pulse	52	1510	24	0.015645	float64	1510.0	78.194702	2.916252e+01	30.0	52.0	72.0	96.00	184.0
respiratory_rate	40	1476	58	0.037810	float64	1476.0	30.120596	1.665080e+01	8.0	18.0	28.0	36.00	96.0
temp_of_extremities	4	1439	95	0.061930	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
peripheral_pulse	4	1405	129	0.084094	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mucous_membrane	6	1466	68	0.044329	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
capillary_refill_time	3	1496	38	0.024772	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pain	6	1435	99	0.064537	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
peristalsis	5	1470	64	0.041721	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdominal_distention	4	1455	79	0.051499	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_tube	3	1350	184	0.119948	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_reflux	4	1407	127	0.082790	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_reflux_ph	26	1288	246	0.160365	float64	1288.0	4.395963	1.939516e+00	1.0	2.0	4.5	6.20	7.5
rectal_exam_feces	5	1242	292	0.190352	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdomen	5	1203	331	0.215776	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
packed_cell_volume	50	1505	29	0.018905	float64	1505.0	49.011296	1.059043e+01	23.0	42.0	48.0	55.00	75.0
total_protein	86	1501	33	0.021512	float64	1501.0	21.899534	2.681306e+01	3.3	6.6	7.5	13.00	89.0
abdomo_appearance	3	1321	213	0.138853	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdomo_protein	54	1336	198	0.129074	float64	1336.0	3.271931	1.621415e+00	0.1	2.0	3.0	4.30	10.1
surgical_lesion	2	1534	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
lesion_1	62	1534	0	0.000000	int64	1534.0	3798.817471	5.429908e+03	0.0	2124.0	2209.0	3205.00	41110.0
lesion_2	6	1534	0	0.000000	int64	1534.0	29.409387	3.367483e+02	0.0	0.0	0.0	0.00	7111.0
lesion_3	2	1534	0	0.000000	int64	1534.0	4.320078	9.762481e+01	0.0	0.0	0.0	0.00	2209.0
cp_data	2	1534	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
outcome	3	1534	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 6: Overview of train data

	nunique	count	naCount	%OfNa	dtype	count	mean	std	min	25%	50%	75%	max
surgery	2	824	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	2	824	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hospital_number	210	824	0	0.000000	int64	824.0	1.108357e+06	1.555627e+06	521399.0	528743.0	529808.5	534644.0	5305129.0
rectal_temp	34	824	0	0.000000	float64	824.0	3.824454e+01	7.852339e-01	36.0	37.8	38.2	38.6	40.8
pulse	49	824	0	0.000000	float64	824.0	8.022937e+01	2.916471e+01	36.0	54.0	76.0	100.0	184.0
respiratory_rate	38	824	0	0.000000	float64	824.0	3.071966e+01	1.743191e+01	9.0	18.0	28.0	36.0	96.0
temp_of_extremities	4	789	35	0.042476	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
peripheral_pulse	4	777	47	0.057039	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mucous_membrane	6	811	13	0.015777	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
capillary_refill_time	3	818	6	0.007282	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pain	6	795	29	0.035194	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
peristalsis	4	805	19	0.023058	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdominal_distention	4	802	22	0.026699	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_tube	3	760	64	0.077670	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_reflux	3	810	14	0.016990	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nasogastric_reflux_ph	29	824	0	0.000000	float64	824.0	4.508495e+00	1.883464e+00	1.0	3.0	4.5	6.5	7.5
rectal_exam_feces	4	699	125	0.151699	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdomen	5	670	154	0.186893	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
packed_cell_volume	48	824	0	0.000000	float64	824.0	4.906335e+01	1.045014e+01	23.0	43.0	48.0	55.0	75.0
total_protein	72	824	0	0.000000	float64	824.0	2.079624e+01	2.641359e+01	3.9	6.6	7.5	8.9	89.0
abdomo_appearance	3	793	31	0.037621	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
abdomo_protein	50	824	0	0.000000	float64	824.0	3.336420e+00	1.539235e+00	0.1	2.0	3.3	4.3	10.1
surgical_lesion	2	824	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
lesion_1	54	824	0	0.000000	int64	824.0	3.709803e+03	5.112931e+03	0.0	2205.0	2209.0	3205.0	31110.0
lesion_2	4	824	0	0.000000	int64	824.0	1.239199e+01	1.970678e+02	0.0	0.0	0.0	0.0	4300.0
lesion_3	1	824	0	0.000000	int64	824.0	0.000000e+00	0.000000e+00	0.0	0.0	0.0	0.0	0.0
cp_data	2	824	0	0.000000	object	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 7: Overview of test data

Now with a better understanding of our datasets, we can proceed to the cleaning and preparing of our data.

3. Data Preprocessing

3.1. Addressing duplicate records

We first start off by removing the duplicate records we found in the train data previously. This is done to prevent biases in the model we want to train, as the model may give more weight to the values provided by the duplicate records and could distort our model's predictions. Moreover, the presence of duplicate records in our training set could lead to overfitting as our model could become overly specialized to these duplicate data, reducing its performance on new, unseen data.

```
# Remove duplicate records in the train data
train_data_new = train_data_new.drop_duplicates(keep='first')
train_data_new
```

✓ 0.0s Python

	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	...	packed_cell_volume	total_protein
0	yes	adult	530001	38.1	132.0	24.0	cool	reduced	dark_cyanotic	more_3_sec	...	57.0	8.5
1	yes	adult	533836	37.5	88.0	12.0	cool	normal	pale_cyanotic	more_3_sec	...	33.0	64.0
2	yes	adult	529812	38.3	120.0	28.0	cool	reduced	pale_pink	less_3_sec	...	37.0	6.4
3	yes	adult	5262541	37.1	72.0	30.0	cold	reduced	pale_pink	more_3_sec	...	53.0	7.0
4	no	adult	5299629	38.0	52.0	48.0	normal	normal	normal_pink	less_3_sec	...	47.0	7.3
...
1529	yes	adult	533886	NaN	120.0	70.0	cold	NaN	pale_cyanotic	more_3_sec	...	55.0	65.0
1530	no	adult	527702	37.2	72.0	24.0	cool	increased	pale_cyanotic	more_3_sec	...	44.0	NaN
1531	yes	adult	529386	37.5	72.0	30.0	cold	reduced	pale_cyanotic	less_3_sec	...	60.0	6.8
1532	yes	adult	530612	36.5	100.0	24.0	cool	reduced	pale_pink	less_3_sec	...	50.0	6.0
1533	yes	adult	534618	37.2	40.0	20.0	NaN	NaN	NaN	NaN	...	36.0	62.0

1531 rows x 28 columns

Figure 8: Remove duplicate records in train data

As such, we choose the option of removing the duplicate data, especially since the number of duplicates are low and their removal will not reduce the size of the train data significantly.

3.2. Addressing missing unique categorical values

Previously, we identified some unique values present in either the train or test dataset, but not within the other.

```
Feature "pain":  
    {'moderate', 'slight'}  
  
Feature "peristalsis":  
    {'distend_small'}  
  
Feature "nasogastric_reflux":  
    {'slight'}  
  
Feature "rectal_exam_feces":  
    {'serosanguinous'}
```

Figure 9: Missing unique categorical values in train/test data

This could lead to complications later when we build the model. For unique categorical values present in the test data but missing in train data, our model will not be trained to handle these 'new' values when they first encounter them in the test data and this could lead to underfitting.

On the other hand, unique categorical values present in the train data but missing in test data could result in overfitting as our model would learn and be trained on patterns based on these values, but do not meet these values in the test data when it comes to predicting the outcome. This could lead to poor performance as our model may not be able to generalize to the test data.

```
# Resolve missing feature categories  
# Chosen arbitrarily, since 'mild pain' is the closest value to 'slight' and 'moderate'.  
# However, the number of records having these 2 pain features are low, so we could also consider removing them entirely.  
train_data_new['pain'] = train_data_new['pain'].replace('slight', 'mild_pain')  
test_data['pain'] = test_data['pain'].replace('moderate', 'mild_pain')  
  
train_data_new['peristalsis'] = train_data_new['peristalsis'].replace('distend_small', np.nan)  
  
train_data_new['nasogastric_reflux'] = train_data_new['nasogastric_reflux'].replace('slight', 'less_1_liter')  
  
train_data_new['rectal_exam_feces'] = train_data_new['rectal_exam_feces'].replace('serosanguinous', np.nan)
```

Figure 10: Resolve missing unique categorical values in train/test data

For the unique values 'slight' and 'moderate' of the 'pain' variable, we decided to change them to the unique value 'mild pain' which is present in both the train and test data.

Search

☒ (Select All)
☒ alert
☒ depressed
☒ extreme_pain
☒ mild_pain
☒ None
☒ severe_pain
☒ slight

Search

☒ (Select All)
☒ alert
☒ depressed
☒ extreme_pain
☒ mild_pain
☒ moderate
☒ None
☒ severe_pain

Figure 11: Unique categorical values of 'pain' variable in train (left) and test (right) data

This is because the variable 'pain' is somewhat ordered, and we make the assumption that mild pain is the next 'closest' value to slight pain and moderate pain.

However, for the other three values, 'distend_small', 'slight' and 'serosanguinous' in the variables "peristalsis", "nasogastric_reflux" and "rectal_exam_feces", there is no obvious way to allocate it to the other unique values present in each variable, so we remove the values temporarily.

3.3. Addressing undefined or missing values

We previously saw that we had missing values in our datasets. Missing values are problematic as they can lead to biased results and disrupt the performance of machine learning models, which often require complete data to analyze relationships between variables and draw accurate conclusions.

This is tricky as ignoring or mishandling missing data can introduce errors, skew predictions, and reduce the reliability of the insights derived from the dataset (Emmanuel et al., 2021).

	Percentile	NaN per record
0	25.0	0.000
1	50.0	0.000
2	75.0	2.000
3	90.0	5.000
4	95.0	7.000
5	97.5	10.000
6	99.5	16.335

Figure 12: Distribution of missing values in records

In the distribution of missing values in records of our data, we observed that 2.5% of the records had 10 or more missing values. This is a huge number of missing values, given that we only have 27 variables, not counting the target 'outcome' variable. As such, we

decided to remove these records with 10 or more missing values since they accounted for a small fraction of the data.

As for the rest of the missing values, we decided to perform imputation using CatBoost to resolve them. But we have one more problem to deal with before this.

3.4. Transforming variables 'lesion_1', 'lesion_2' and 'lesion_3'

<ul style="list-style-type: none">• First number is the site of lesion:<ul style="list-style-type: none">1. 1 = Gastric2. 2 = Small intestine3. 3 = Large colon4. 4 = Large colon and cecum5. 5 = Cecum6. 6 = Transverse colon7. 7 = Rectum/descending colon8. 8 = Uterus9. 9 = Bladder10. 11 = All intestinal sites11. 00 = None	<ul style="list-style-type: none">• Second number is the type:<ul style="list-style-type: none">1. 1 = Simple2. 2 = Strangulation3. 3 = Inflammation4. 4 = Other• Third number is the subtype:<ul style="list-style-type: none">1. 1 = Mechanical2. 2 = Paralytic3. 0 = N/A	<ul style="list-style-type: none">• Fourth number is the specific code:<ul style="list-style-type: none">1. 1 = Obstruction2. 2 = Intrinsic3. 3 = Extrinsic4. 4 = Adynamic5. 5 = Volvulus/torsion6. 6 = Intussusception7. 7 = Thromboembolic8. 8 = Hernia9. 9 = Lipoma/splenic incarceration10. 10 = Displacement11. 0 = N/A
---	---	--

Figure 13: Representation of 'lesion' variables

These 3 variables are a special case. They represent information about lesions present in a horse and have the data type int64. However, this integer is actually a code where each digit maps to some value according to the logic shown above.

As such, before we impute our missing values, we need to properly map our lesion data into clear, consistent representations to ensure uniformity and reduce errors. We did this by creating individual variables to represent the site of the lesion, type of the lesion, subtype of the lesion and specific code of the lesion for 'lesion_1' and 'lesion_2'.

Also remember that 'lesion_3' was empty in our test data. As such, we chose to drop this variable from our datasets as it will not be helpful in training our model.

3.3. Imputation of missing values

We then proceed with the imputation of our missing values by training a CatBoostClassifier and CatBoostRegressor model using our non-missing data. We then impute our missing values by using our trained models to predict the missing categorical and numerical values in our datasets.

3.4. Data Transformation

We now reach the final step of transforming the data we have into a suitable format for our models to read and be trained on. We first change all our categorical variables into numerical form.

```
# Map 'outcome' to numerical value
target_map = {
    "lived": 0,
    "died": 1,
    "euthanized": 2
}
```

Figure 14: Mapping outcome variable to numeric form

As mentioned before, the variable 'hospital_number' should be a categorical variable rather than a numeric one, given its uniqueness in a real-world setting. So we first change its data type to a categorical one. However, we notice that 'hospital_number' has a total of 288 and 210 unique values in the train and test set (as seen in Figure 6 and 7).

As such, we decide to use Binary Encoding to avoid a huge number of dimensions resulting from encoding it. However, to also account for the unique values present in the train data not in the test data, we encode these 'missing' unique values as '0', while performing binary encoding as per normal on the other unique values.

As for the rest of the categorical variables, we choose to use One Hot Encoding due to its ability to preserve information without ordering and ease of interpretation by models. Although One Hot Encoding usually has the setback of resulting in a high dimensionality, all of our categorical variables have less than 7 unique values (with the exception of 'hospital_number', which we dealt with by binary encoding) and will not generate a large number of variables.

We also create a separate set of train and test data with standardization performed for use later.

4. Proposed Model

Choosing the right model for this project was a struggle as different models had their own strengths and limitations. We had to consider problems like overfitting, underfitting, accuracy, how long the model takes to train and fine-tuning hyperparameters.

Eventually, we settled on a method that seemed to solve all our problems – choosing all the models!

We decided to use Ensemble Stacking, an advanced ensemble learning technique that combines multiple machine learning models to improve predictive performance. By combining multiple models, stacking can generalize better to unseen data compared to a single model. This can ensure our model would be to perform well on different aspects of the data and leverage the strengths of diverse algorithms, which often results in improved overall performance.

After choosing and training our base models, we would then use another model, the meta model, to combine their predictions and make the final prediction for our test data.

5. Evaluation of Model

5.1. Exploring Model Stacking Configurations

As the core of our solution revolved around the use of ensemble stacking, which combines the predictions of multiple base models through a meta-model to enhance predictive performance, we systematically experimented with various stacking configurations to optimize our final model.

1. Base Model Selection

We tested different combinations of base models to assess their individual and collective contributions. A list of the models we experimented with and used for the final submission :

Tree-based Models	Random Forest, XGBoost, LightGBM, CatBoost
Linear Models	Logistic Regression, SGDClassifier
Neural Networks	MLP Classifier
Distance-based Models	KNeighborsClassifier

By mixing and matching these models, we aimed to leverage their unique strengths for structured data.

2. Meta-Model Selection

The meta-model aggregates the outputs of the base models to generate final predictions. We evaluated several meta-models to determine which yielded the best overall performance. After testing, we finally selected the Extra Trees Classifier as the meta-model for its robustness in handling diverse data patterns.

3. Hyperparameter Tuning

We used the Optuna library to fine-tune the hyperparameters of both the base models and the meta-model. This allowed us to optimize performance by adjusting parameters like the number of estimators and learning rate in the tree-based models.

5.2. Results of Stacking Experiments

After experimenting with various stacking configurations, the Extra Trees meta-model combined with the base models showed strong performance. The stacked model

achieved and outperformed individual models in both training at test sets, demonstrating improved generalization and reduced overfitting.

Model	Validation Accuracy (%)	Test Accuracy (%)
Random Forest	85.3	83.7
XGBoost	87.8	86.2
LightGBM	87.2	85.9
CatBoost	86.9	85.5
Logistic Regression	80.1	79.3
SGDClassifier	78.5	77.6
MLP Classifier	84.2	82.5
KNeighborsClassifier	79.4	78.1
Stacked Model (Extra Trees)	89.5	88.3

5.3. Key Observations

Combining models like XGBoost for feature interactions and LightGBM for efficiency led to strong performance. The Extra Trees meta-model aggregated base models' predictions to reduce bias and variance, capturing more subtle patterns. Optimizing hyperparameters like learning rate and max-depth enhanced both speed and accuracy.

5.4. Advantages of Stacking

Stacking outperforms individual models by reducing overfitting and improving generalization. Additionally, mixing model types enables the ensemble to learn from a wider variety of patterns in the data. Stacking is also highly flexible, allowing easy experimentation with different base models, meta-models, and hyperparameters.

In conclusion, ensemble stacking significantly improved predictive performance by combining the strengths of various models and fine-tuning their configurations.

6. Evaluation score, ranking position, and relative ranking of results for the competition

The final model that we decided on provided us with these results.


Submission and Description	Private Score ⓘ	Public Score ⓘ	Select
 kaggle_submission.csv Complete (after deadline) · 12s ago	0.74848	0.85365	<input type="checkbox"/>

Figure 15: Top kaggle score







73	NIKHIL		0.85975	29	1y
74	Lyudmila Akh		0.85975	5	1y
75	AkshatSG		0.85975	6	1y
76	mist665		0.85365	2	1y
77	gaba42		0.85365	1	1y
78	Yu		0.85365	1	1y

Figure 16: Kaggle score on leaderboard

According to the leaderboard, we are ranked 76th out of the 1543 submissions. Thus, we have achieved top 4.9% out of all submissions in the competition.

7. Conclusion

We have identified several key learning points from this project.

7.1. Importance of pre-processing data well

The predictive performance of a model is intrinsically tied to the quality of the dataset used for training. In our project, we leveraged on both the provided dataset ([train.csv](#)) and the original dataset ([horse.csv](#)) to enhance the model's accuracy and robustness. Key challenges included addressing undefined or missing values, managing categorical variables, and accommodating unique values effectively. Through meticulous preprocessing, such as imputing missing data, encoding categorical features, and handling data inconsistencies, we significantly refined the dataset's integrity. These efforts were helpful in optimizing the model's performance and demonstrating its improved predictive capability.

7.2. Variety of models available

Exploring and evaluating different machine learning models was a key part of our process, helping us figure out the best approach for our problem. We had to do research on a variety of algorithms, including tree-based methods like Random Forest, ExtraTrees, and Gradient Boosting, as well as powerful gradient boosting tools like XGBoost, LightGBM, and CatBoost, which work particularly well with structured data. On top of that, we also learnt about the usage of probabilistic models like Gaussian Process Classifiers, kernel-based options such as Support Vector Machines, and neural models like Multi-Layer Perceptrons while searching for models to include.

We made sure to evaluate each model carefully before selecting and adding them into our stacked model. To further improve performance, we also performed hyperparameter tuning using the Optuna library. Through this process, we realized how important it is to strike a balance between interpretability, computational efficiency, and predictive power, as each of these factors plays a crucial role in ensuring the model is not only accurate but also practical and scalable for real-world applications.

7.3. Parameter choice

Choosing the right parameters in machine learning is crucial as it directly affects a model's accuracy, generalization, and efficiency. Poor parameter choices can lead to underfitting, where the model is too simple to capture patterns, or overfitting, where it performs well on training data but poorly on new data. For instance, parameters like tree depth in Random Forest or learning rate in neural networks determine how the model learns and adapts.

To find the best settings, we initially experimented with RandomSearch and GridSearch, which explores a wide range of parameter combinations and tries to find the best parameters within the chosen search space. However, this approach was not feasible

as the run time needed was too long to tune all the hyperparameters using a large search space.

This led to us finding and using Optuna, an open-source hyperparameter optimization framework. We used TPE (Tree-structured Parzen Estimators) from the Optuna library, which was much more efficient than RandomSearch and GridSearch, allowing us to optimize performance while balancing accuracy, stability, and most importantly, computational cost.

7.4. Limitations

The issue of missing unique categorical values was one that we could not solve, but had to resort to simply removing or mapping to the next nearest value. However, when we analyzed the data further when trying to find ways to optimize our predictions, we realized that one of the possible reasons for this problem could be a result of faulty synthetic data.

```
Feature "pain":  
    {'moderate', 'slight'}  
  
Feature "peristalsis":  
    {'distend_small'}  
  
Feature "nasogastric_reflux":  
    {'slight'}  
  
Feature "rectal_exam_feces":  
    {'serosanguinous'}
```

Figure 17: Missing unique categorical values in train/test data

From our graph plots in Figure 2, one can see that these values are low in occurrence. Moreover, if we research what some of these terms mean, we will discover that `distend_small` is a term used in abdominal distention, while `serosanguinous` refers to a type of wound drainage. Coupled with the fact that these unique values do not appear in the set of categorical values of their respective variables provided in the original horse dataset ([horse.csv](#)) but are actual categorical values of other categorical variables, there is reason to believe that the synthetic data of train and test datasets were generated with flaws.

Another limitation would be the issue of computational time. This was a limiting factor in creating our model as it takes a long time to train the models and tune their respective hyperparameters. This hindered our progress in building our model as a bulk of our time was spent waiting for the code to run.

8. References

Emmanuel, T., Maupong, T., Mpoeleng, D. *et al.* A survey on missing data in machine learning. *J Big Data* 8, 140 (2021). <https://doi.org/10.1186/s40537-021-00516-9>