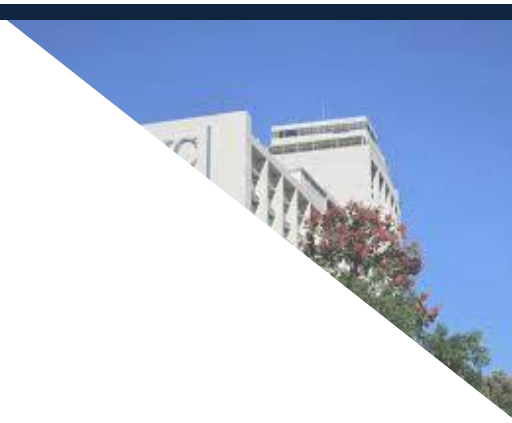


PUC
RIO

Relatório de Data Mining

Horse Colic Data Set

BI Master
Pós-Graduação abordando
a área de Ciência de Dados

Professora Manoela Kohler

Marcos Alonso Guimarães

INTRODUÇÃO

O presente trabalho apresenta as principais observações e resultados encontrados na análise da base de dados amostral, com características e sinais vitais de equinos. Trata-se de relatório avaliativo em relação aos conceitos e modelos apresentados na disciplina da Data Mining, do curso de pós-graduação em BI e Inteligência Artificial do Laboratório de Inteligência Computacional Aplicada - ICA/PUC-Rio.

Isso posto, o objetivo com a obtenção dos atributos disponibilizados na base de dados é treinar modelos de Machine Learning que possam, a partir de informações vitais do organismo do animal, avaliar se o cavalo irá sobreviver. Trata-se, portanto, de típico problema de classificação através do treino supervisionado de máquinas.

Assim, esse relatório apresenta-se da seguinte maneira:

- Capítulo 1 – Análise Exploratória dos Dados;
- Capítulo 2 – Pré-processamento dos Dados;
- Capítulo 3 – Modelos Supervisionados de Machine Learning; e
- Considerações Finais.

1. ANÁLISE EXPLORATÓRIA

1.1. Base de dados

Foram enviadas duas bases de dados, já separadas, contendo observações para treino e para teste. Totalizam-se 27 atributos, 1 rótulo de saída e 388 observações (299 da base de treino e 89 da base de teste).

No intuito de aplicar de forma sistemática os conhecimentos aprendidos utilizando a linguagem em Python, foi realizada a junção das bases apenas como forma de prática, sendo em seguida separada novamente usando a função de split do pacote skrlearn. Esse passo realizado em seguida foi realizado com o intuito de evitar o vazamento de informações entra a base de treino e a base a ser testada.

Assume-se que a base de treino contém dados conhecidos e que ocorreram, enquanto a base de teste entende-se como premissa tratar-se de informação não observada, sendo justamente o perfil de avaliação para a análise de modelos treinados.

Assim, o primeiro passo foi analisar as colunas existentes, o shape da base, bem como os tipos de dados a se analisar.

```
In [20]: train
```

Out[20]:

	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_
0	no	adult	530101	38.5	66.0	28.0	cool	reduced	NaN	more_
1	yes	adult	534817	39.2	88.0	20.0	NaN	NaN	pale_cyanotic	less_
2	no	adult	530334	38.3	40.0	24.0	normal	normal	pale_pink	less_
3	yes	young	5290409	39.1	164.0	84.0	cold	normal	dark_cyanotic	more_
4	no	adult	530255	37.3	104.0	35.0	NaN	NaN	dark_cyanotic	more_
...	...	...	...	...	...	...	...	...	...	
294	yes	adult	533886	NaN	120.0	70.0	cold	NaN	pale_cyanotic	more_
295	no	adult	527702	37.2	72.0	24.0	cool	increased	pale_cyanotic	more_
296	yes	adult	529386	37.5	72.0	30.0	cold	reduced	pale_cyanotic	less_
297	yes	adult	530612	36.5	100.0	24.0	cool	reduced	pale_pink	less_
298	yes	adult	534618	37.2	40.0	20.0	NaN	NaN	NaN	

299 rows × 28 columns

```
In [22]: train.head() # first five rows
```

Out[22]:

	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_ti
0	no	adult	530101	38.5	66.0	28.0	cool	reduced	NaN	more_3_
1	yes	adult	534817	39.2	88.0	20.0	NaN	NaN	pale_cyanotic	less_3_
2	no	adult	530334	38.3	40.0	24.0	normal	normal	pale_pink	less_3_
3	yes	young	5290409	39.1	164.0	84.0	cold	normal	dark_cyanotic	more_3_
4	no	adult	530255	37.3	104.0	35.0	NaN	NaN	dark_cyanotic	more_3_

5 rows × 28 columns

```
In [23]: train.shape
```

```
Out[23]: (299, 28)
```

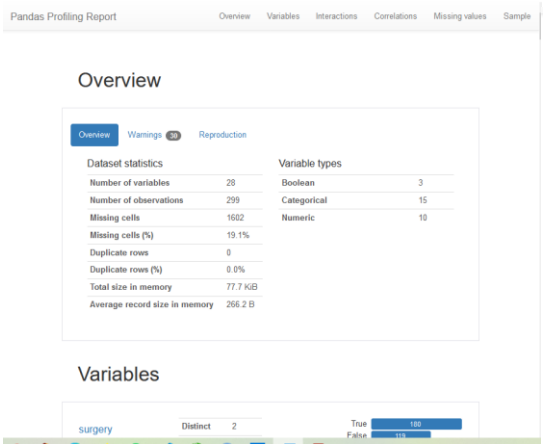
```
In [26]: train.columns # - number and name of columns

Out[26]: Index(['surgery', 'age', 'hospital_number', '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', 'outcome', 'surgical_lesion', 'lesion_1', 'lesion_2',
                'lesion_3', 'cp_data'],
               dtype='object')
```

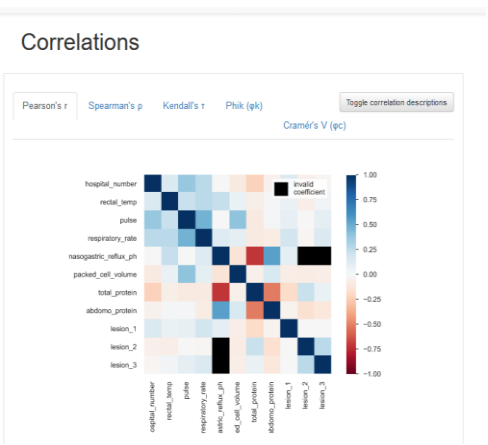
Os tipos de dados são importantes de serem avaliados pois é possível analisar, olhando já para o horizonte de análise, se os dados são inteiros, strings, booleanos. A utilização de dados de CPF ou CNPJ, por exemplo, exige especial atenção no seu uso, pois em geral é um dado com característica de identificador/chave primária, sendo importante a leitura desse dado como caractere e não número (caso de existir zeros à esquerda).

Assim, tem-se[1] que:

```
In [27]: train.dtypes
         # or data.info() as above

Out[27]: surgery                 object
         age                     object
         hospital_number          int64
         rectal_temp            float64
         pulse                  float64
         respiratory_rate       float64
         temp_of_extremities     object
         peripheral_pulse        object
         mucous_membrane         object
         capillary_refill_time   object
         pain                    object
         peristalsis             object
         abdominal_distention    object
         nasogastric_tube        object
         nasogastric_reflux      object
         nasogastric_reflux_ph  float64
         rectal_exam_feces       object
         abdomen                 object
         packed_cell_volume     float64
         total_protein          float64
         abdomo_appearance       object
         abdomo_protein         float64
         outcome                 object
         surgical_lesion         object
         lesion_1                 int64
         lesion_2                 int64
         lesion_3                 int64
         cp_data                 object
         dtype: object
```

Através do pacote pandas_profiling, é possível obter uma análise de perfil detalhada dos atributos e observações da base de dados. Por este motivo, foram realizadas essas análises também por meio da função ProfileReport ().





[1] Ressalta-se que as figuras apresentadas no presente relatório resumem as principais análises realizadas com as bases de dados. Assim, ressalta-se que a base de teste também foi analisada.

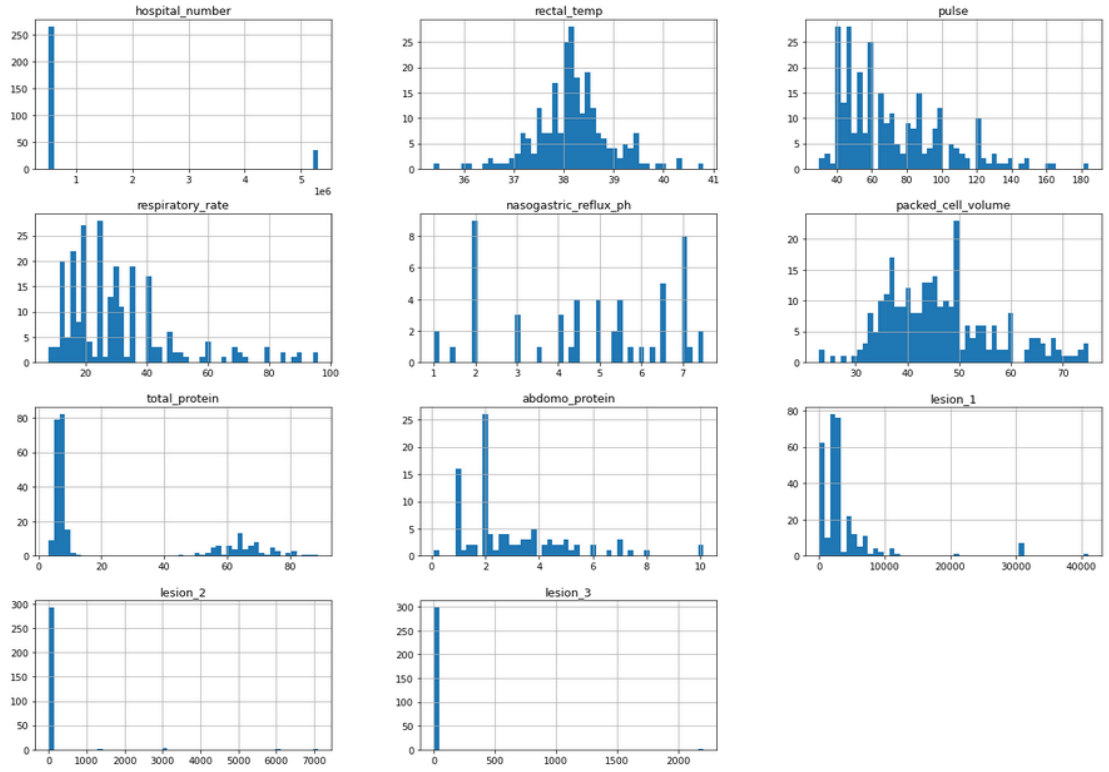
Sabendo que existem variáveis numéricas que são apenas identificadores, como o caso da numeração do hospital, por exemplo, foi usada a função describe() para obter as principais estatísticas descritivas das variáveis numéricas e que podem ser analisadas sob essas perspectivas.

Através dessa análise, foi possível reparar através dos quartis, que a maioria dos cavalos sofreram apenas uma lesão cirúrgica. Isso for confirmado observando o percentual de cada valor dos atributos lesion_2 e lesion_3:

```
In [30]: train.lesion_2.value_counts(dropna = False, normalize = True) # more than 97% of registers is zero
Out[30]: 0       0.976589
         3111    0.010033
         6112    0.003344
         7111    0.003344
         1400    0.003344
         3112    0.003344
         Name: lesion_2, dtype: float64

In [31]: train.lesion_3.value_counts(dropna = False, normalize = True) # more than 99% of registers is zero
Out[31]: 0       0.996656
         2209    0.003344
         Name: lesion_3, dtype: float64
```

1.2. Distribuição de probabilidade e correlação de variáveis

Também buscou-se verificar a distribuição de probabilidade das observações de cada variável numérica, através da visualização por gráficos do pacote matplotlib:

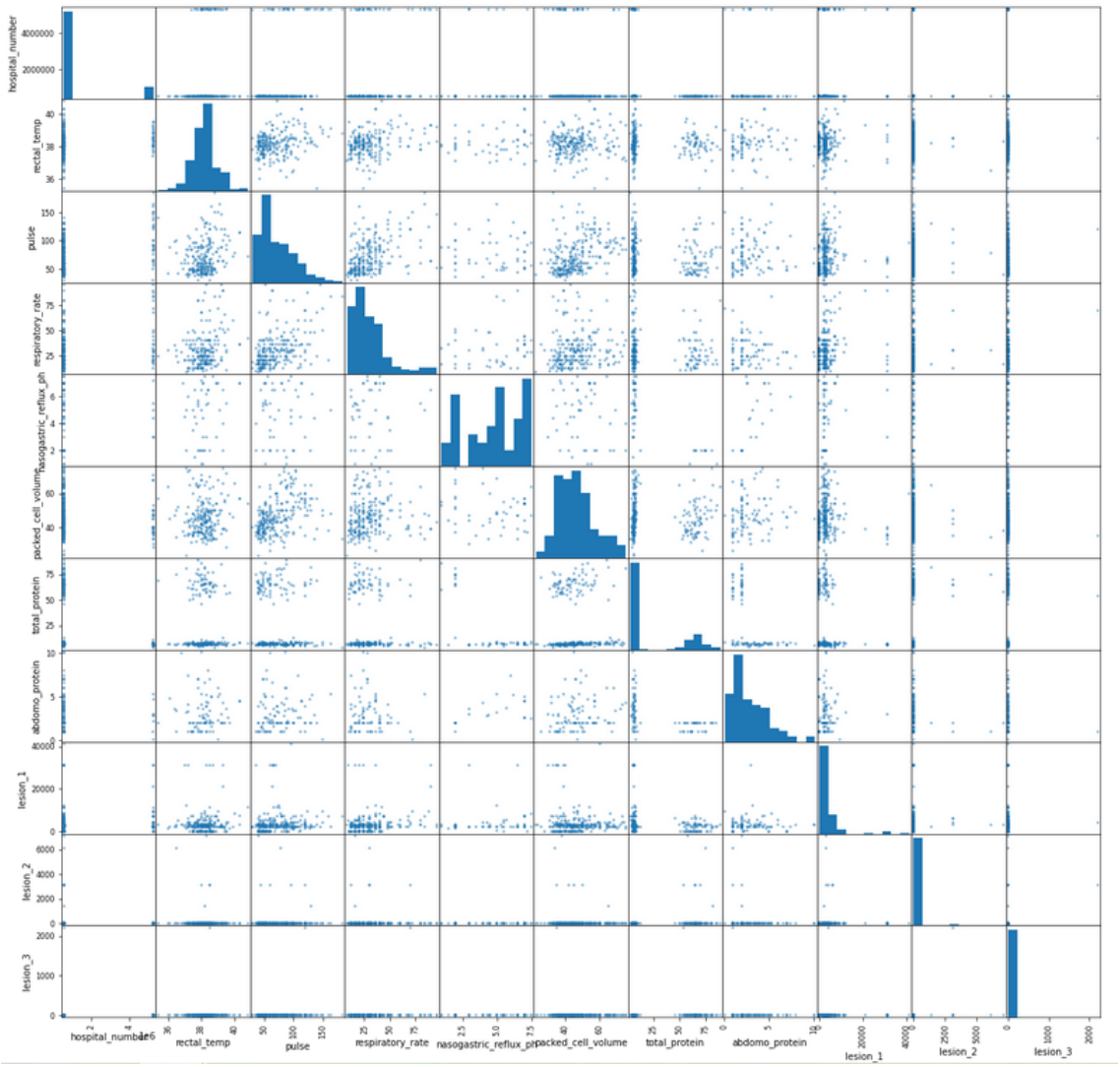


O pacote pandas também permite uma análise das variáveis através da distribuição e scatter plot, conforme figura a seguir.

Através do plote das observações através do histograma, foi possível verificar, por exemplo, que a variável rectal_temp segue uma distribuição gaussiana,

enquanto a que a variável pulse possui assimetria à direita ou positiva aproximadamente.

Em relação à correlação entre as variáveis, não foi possível obter uma relação explícita entre as variáveis numéricas. Talvez uma correlação positiva entre o ph do refluxo nasogástrico e a variável abonomo_protein.



1.3. Análise de dados faltantes - Missing Values

Foram usadas técnicas visuais e de contagem para obtenção do número de valores faltantes na base de dados. Essa análise é fundamental nesta etapa, de maneira a observar o grau de intercorrência que dados faltantes possui na análise.

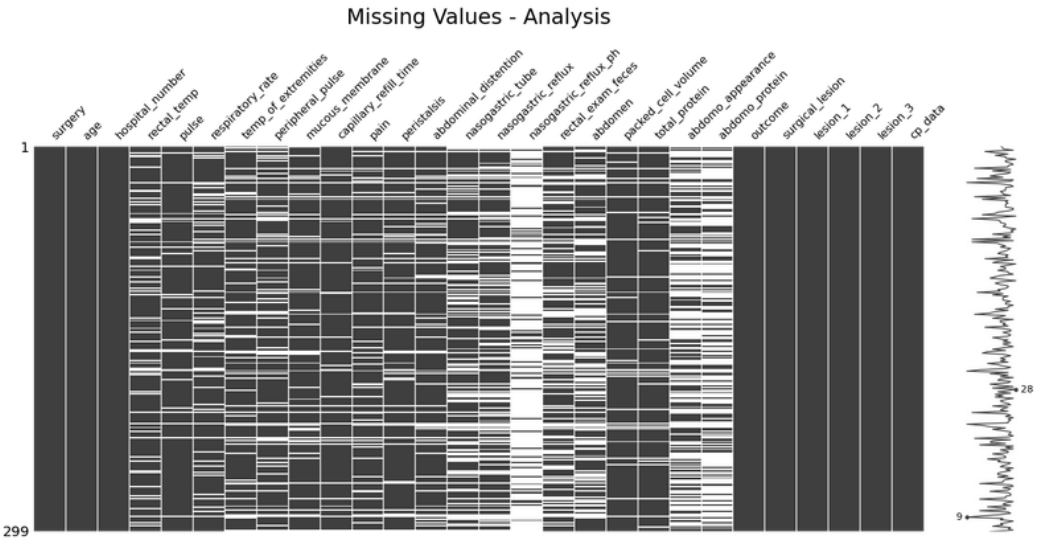
Para isso, usa-se o pacote missingno para uma análise gráfica desses valores por atributo. É possível também, por exemplo, agregar e obter a soma de valores faltantes por variável sem a análise visual.

Utilizou-se também técnica para apresentar a contagem de dados faltantes, bem como o percentual em relação ao tamanho da coluna da base. Essa abordagem

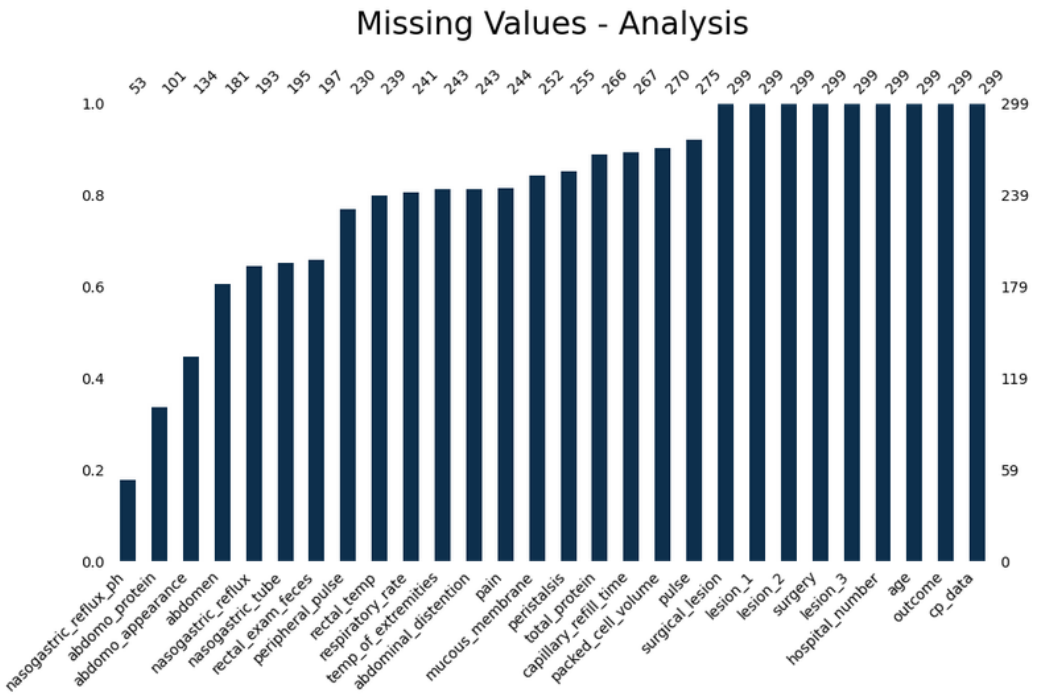
foi realizada já pensando em um possível corte a partir de um certo percentual elevado de valores faltantes.

3.2 Checking for missing values.

In [35]:
```
msno.matrix(train) #- matrix
plt.title('Missing Values - Analysis', size = 30, pad=30)
plt.show()
```



In [36]:
```
#msno.matrix(data) #- matrix
#msno.heatmap(data) #- heatmap
#msno.bar(data)# - bar plot
msno.bar(train, color="#0D2F4C", sort="ascending", figsize=(15,8), fontsize=14)
plt.title('Missing Values - Analysis', size = 30, pad=30)
plt.show() # - show plot/remove obs after running the plot
```



In [37]:
```
#X_train.isnull().any(axis=1) #rows
#X_train.isnull().any()#columns
null_columns = train.columns[train.isnull().any()] # Defining columns with missing values
null_columns
```

Out[37]:
```
Index(['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'],
      dtype='object')
```

In [38]:
```
train[null_columns].isnull().sum().sort_values() # Counting number of rows.
```

Out[38]:
```
pulse                      24
packed_cell_volume         29
capillary_refill_time      32
total_protein              33
peristalsis                44
mucous_membrane            47
pain                       55
abdominal_distention       56
temp_of_extremities        56
respiratory_rate           58
rectal_temp                60
peripheral_pulse           69
rectal_exam_feces         102
nasogastric_tube          104
nasogastric_reflux        106
abdomen                   118
abdomo_appearance         165
abdomo_protein            198
nasogastric_reflux_ph     246
dtype: int64
```

Let's check one single column...

In [39]:
```
train[pd.isnull(train['pulse'])] # Display only row with missing value in pulse column
```

Out[39]:

	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_ti
5	no	adult	528355	NaN	NaN	NaN	warm	normal	pale_pink	less_3_
28	yes	adult	5279442	NaN	NaN	NaN	NaN	NaN	NaN	N
52	no	adult	529483	NaN	NaN	NaN	normal	normal	pale_pink	less_3_
56	yes	adult	528872	NaN	NaN	NaN	NaN	NaN	NaN	N
58	yes	adult	528298	NaN	NaN	20.0	cold	reduced	pale_pink	N
74	yes	young	5292929	NaN	NaN	NaN	NaN	NaN	NaN	N
78	yes	adult	530693	NaN	NaN	NaN	cool	reduced	pale_pink	
83	yes	adult	5279822	38.0	NaN	24.0	cool	reduced	dark_cyanotic	more_3_
93	no	adult	530310	NaN	NaN	NaN	cool	reduced	normal_pink	less_3_
115	no	adult	533723	NaN	NaN	40.0	cool	normal	normal_pink	less_3_
117	no	adult	5290482	39.5	NaN	NaN	cool	reduced	pale_cyanotic	more_3_
126	yes	adult	530384	38.7	NaN	NaN	cool	normal	pale_pink	less_3_
135	yes	adult	530693	NaN	NaN	NaN	cool	reduced	pale_pink	more_3_
150	yes	adult	529399	39.3	NaN	NaN	cold	reduced	dark_cyanotic	more_3_
158	no	adult	528134	NaN	NaN	12.0	normal	normal	bright_pink	less_3_
159	yes	adult	527916	NaN	NaN	NaN	NaN	NaN	NaN	N
173	no	adult	518476	NaN	NaN	NaN	cool	absent	dark_cyanotic	N
174	yes	adult	527929	NaN	NaN	NaN	NaN	NaN	NaN	N

In [40]:
```
percent_missing = train.isnull().sum() * 100 / len(train)
missing_value_data = pd.DataFrame({'column_name': train.columns,
                                   'percent_missing': percent_missing}) # creating dictionary...
missing_value_data.sort_values('percent_missing', inplace=True)
missing_value_data
```

Out[40]:

	column_name	percent_missing
surgery	surgery	0.000000
lesion_2	lesion_2	0.000000
lesion_1	lesion_1	0.000000
surgical_lesion	surgical_lesion	0.000000
outcome	outcome	0.000000
lesion_3	lesion_3	0.000000
cp_data	cp_data	0.000000
age	age	0.000000
hospital_number	hospital_number	0.000000
pulse	pulse	8.026756
packed_cell_volume	packed_cell_volume	9.698997
capillary_refill_time	capillary_refill_time	10.702341
total_protein	total_protein	11.036789
peristalsis	peristalsis	14.715719
mucous_membrane	mucous_membrane	15.719064
pain	pain	18.394649
abdominal_distention	abdominal_distention	18.729097
temp_of_extremities	temp_of_extremities	18.729097
respiratory_rate	respiratory_rate	19.397993
rectal_temp	rectal_temp	20.066890
peripheral_pulse	peripheral_pulse	23.076923
rectal_exam_feces	rectal_exam_feces	34.113712
nasogastric_tube	nasogastric_tube	34.782609

2. PRÉ-PROCESSAMENTO DOS DADOS

2.1. Lidando com o rótulo de saída – Balanceamento dos dados

Após finalizada a parte de análise exploratória dos dados, leitura do dicionário com as informações complementares de cada variável, iniciou-se o processo de tratamento dos dados.

Um primeiro passo está justamente na observação quanto ao conteúdo presente nos rótulos de saída, que correspondem ao fato se o cavalo sobreviveu, morreu ou sofreu um processo de eutanásia. Assim, tem-se que a distribuição de resultados da base de treino é a seguinte:

```
In [41]: outcome_list = pd.DataFrame()
         #outcome_list['antes'] = data.outcome.value_counts(dropna = False)
         outcome_list['antes'] = train.outcome.value_counts(dropna = False,normalize=True)
         outcome_list
         #data.drop_duplicates(subset='outcome') #keep="last" mantain index - we could use this to verify
```

Out[41]:

	antes
lived	0.595318
died	0.257525
euthanized	0.147157

No entanto, é interessante pensar, principalmente quando estamos trabalhando com dados desbalanceados, de como os resultados se relacionam. É possível perceber, por exemplo, que o objetivo principal do modelo treinado está em prever se o animal irá sobreviver ou não. Assim, o resultado entre morrer e sofrer eutanásia acabam por ter um mesmo 'fim', de modo que unir os dois resultados como um só (no caso, rótulo 'died') é uma estratégia interessante. Permite maior observações concentradas em duas saídas a ser usada no modelo, bem como permite um maior balanceamento entre os dados, o que poderia certamente afetar o treinamento e a análise dos resultados.

```
#train.loc[train["outcome"] == "euthanized", ["outcome"]] = "died"
train.loc[train.outcome == "euthanized", "outcome"] = "died"
#y_train.loc[y_train == "euthanized"] = "died"
#data.loc[data.outcome == "euthanized", "outcome"] = "died" # it's possible to do a 'for' with 'if' condition too.
```

```
In [45]: outcome_list['depois'] = train.outcome.value_counts(dropna = False, normalize=True)
         outcome_list
```

Out[45]:

	antes	depois
lived	0.595318	0.595318
died	0.257525	0.404682
euthanized	0.147157	NaN

Agora, tem-se uma base bem mais distribuída e balanceada, que é um passo fundamental na etapa de pré-processamento dos dados.

2.2. Análise das colunas – Dicionário

Nessa etapa, cumpre esclarecer que podemos utilizar técnicas de seleção de atributos para especificar melhor o modelo a ser utilizado e evitar problemas de dimensionalidade também.

Poderiam ser aplicadas, por exemplo, técnicas de ganho de informação ou métodos de Wrapper. No entanto, devido ao apoio do relatório com informações das variáveis e menor tempo para a realização das análises, optou-se por abordar, ainda que de forma mais simplificada (mas não menos importante) a parte relativa à seleção de atributos, dedicando um tempo maior na implementação dos modelos e análise dos mesmos. Ademais, ao aplicar um dos modelos de classificação, foi extraída informação de importância de atributos, especificamente através da aplicação do modelo de XGBoost.

Assim, pelo relatório de variáveis que a variável cp_data, que confirma se há dados patológicos presentes nos casos, não teria significado na nossa análise. Assim, retirou-se a coluna da análise:

Ao mesmo tempo, verificou-se que existia um número elevado de hospitais, sem ganho de informação nessa variação elevada, bem como a taxa respiratória, que segundo o próprio dicionário, apresenta grandes flutuações no ser vivo e na base. Optou-se por retirá-las.

Como os atributos lesion_2 e lesion_3 teve praticamente todos os seus valores zerados, apresentando pouquíssima variância, também foram retirados.

Assim:

2.3. Lidando com os Dados Faltantes

Pata tratar o caso dos dados faltantes, alguns critérios e etapas consecutivas foram realizadas e detalhadas a seguir.

O primeiro critério utilizado foi o de retirar as colunas que representavam uma perda de informação considerável. Dessa forma, qualquer tipo de imputação de dados pode gerar alguma distorção não condizente com a realidade. Dessa forma e utilizando-se do dataframe criado com os percentuais de missing, foi tomada a decisão de retirar da base colunas que apresentavam mais de 30% de dados faltantes.

```
In [50]: missing_value_data.loc[percent_missing > 30]
```

Out[50]:

	column_name	percent_missing
rectal_exam_feces	rectal_exam_feces	34.113712
nasogastric_tube	nasogastric_tube	34.782609
nasogastric_reflux	nasogastric_reflux	35.451505
abdomen	abdomen	39.464883
abdomo_appearance	abdomo_appearance	55.183946
abdomo_protein	abdomo_protein	66.220736
nasogastric_reflux_ph	nasogastric_reflux_ph	82.274247

```
In [51]: filter_data = missing_value_data.loc[percent_missing > 30].index
         train = train.drop(columns = filter_data)
         train.head()
```

Out[51]:

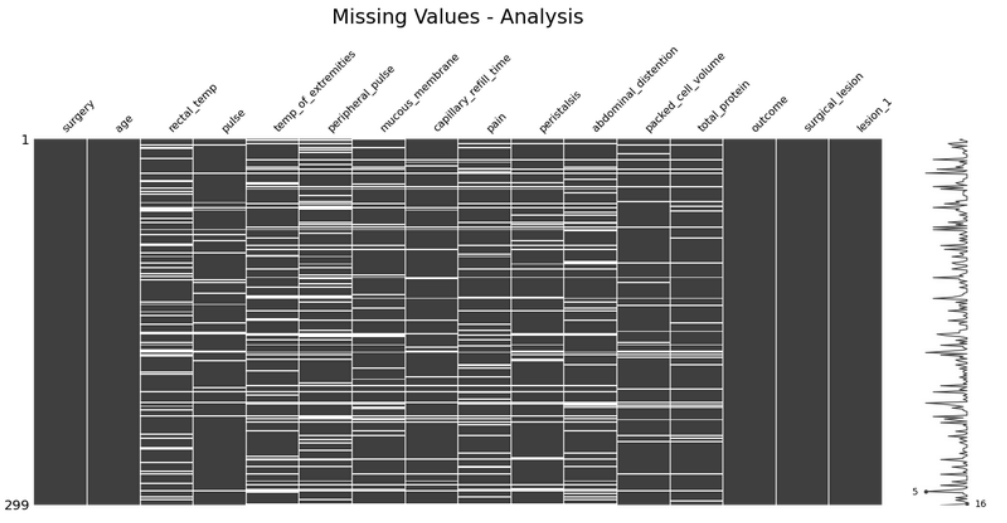
	surgery	age	rectal_temp	pulse	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	peristalsis	abdom
0	no	adult	38.5	66.0	cool	reduced	NaN	more_3_sec	extreme_pain	absent	
1	yes	adult	39.2	88.0	NaN	NaN	pale_cyanotic	less_3_sec	mild_pain	absent	
2	no	adult	38.3	40.0	normal	normal	pale_pink	less_3_sec	mild_pain	hypomotile	
3	yes	young	39.1	164.0	cold	normal	dark_cyanotic	more_3_sec	depressed	absent	
4	no	adult	37.3	104.0	NaN	NaN	dark_cyanotic	more_3_sec	NaN	NaN	

```
In [52]: train.shape
```

Out[52]: (299, 16)

Como resultado, o novo gráfico com dados faltantes apresentou-se da seguinte forma:

```
In [53]: msno.matrix(train) #- matrix
         plt.title('Missing Values - Analysis', size = 30, pad=30)
         plt.show()
```



Além das colunas, optou-se também pela retirada de linhas que não continham pelo menos 10 atributos completos.

```
In [54]: train = train.dropna(thresh=10) #eleven before...changed to 10
         train.shape
Out[54]: (274, 16)
```

```
In [55]: train
```

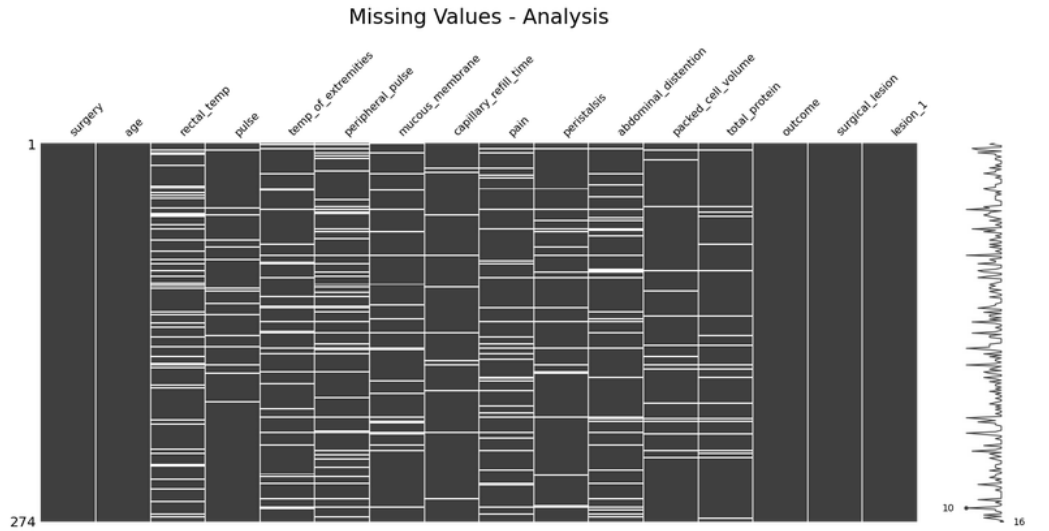
Out[55]:

	surgery	age	rectal_temp	pulse	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	peristalsis	abd
0	no	adult	38.5	66.0	cool	reduced	NaN	more_3_sec	extreme_pain	absent	
1	yes	adult	39.2	88.0	NaN	NaN	pale_cyanotic	less_3_sec	mild_pain	absent	
2	no	adult	38.3	40.0	normal	normal	pale_pink	less_3_sec	mild_pain	hypomotile	
3	yes	young	39.1	164.0	cold	normal	dark_cyanotic	more_3_sec	depressed	absent	
4	no	adult	37.3	104.0	NaN	NaN	dark_cyanotic	more_3_sec	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...
293	no	adult	38.5	40.0	normal	normal	normal_pink	less_3_sec	depressed	hypermotile	
294	yes	adult	NaN	120.0	cold	NaN	pale_cyanotic	more_3_sec	depressed	absent	
295	no	adult	37.2	72.0	cool	increased	pale_cyanotic	more_3_sec	severe_pain	hypomotile	
296	yes	adult	37.5	72.0	cold	reduced	pale_cyanotic	less_3_sec	severe_pain	absent	
297	yes	adult	36.5	100.0	cool	reduced	pale_pink	less_3_sec	mild_pain	hypomotile	

274 rows × 16 columns

A perda de informação, nesse caso, não foi significativa demais, reduzindo as observações de 299 para 274, uma perda de 25 casos observáveis.

```
In [56]: msno.matrix(train) #- matrix
         plt.title('Missing Values - Analysis', size = 30, pad=30)
         plt.show()
```



Também foi realizado cálculo do coeficiente de variação das colunas. A ideia é verificar quais colunas apresentavam pouca variação e com isso, utilizar a técnica de imputação da mediana, o que não afetaria muito os resultados encontrados.

```
In [57]: # Creating a coefficient of variation function
         def cv_value(X):
             cv = np.std(X)/np.mean(X)
             return cv
```

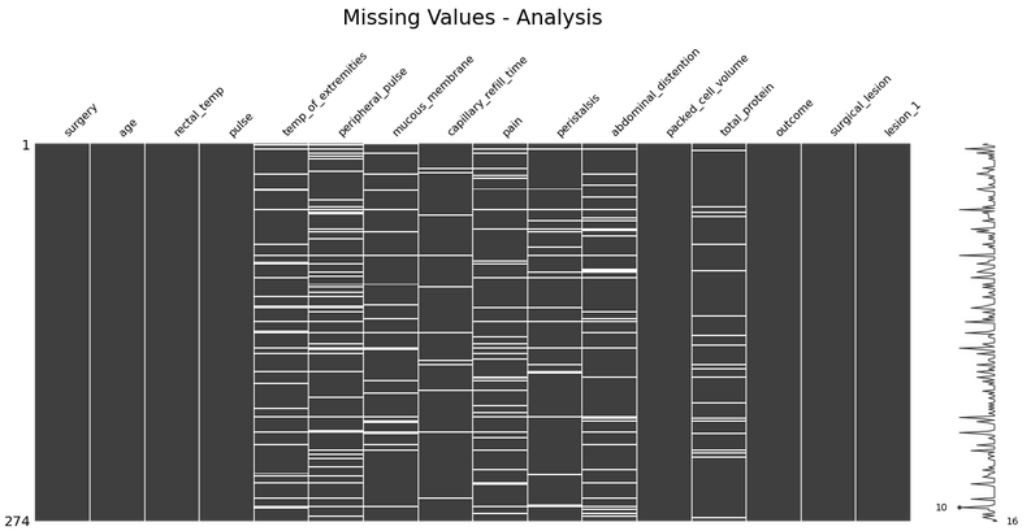
```
In [58]: columns_cv = pd.DataFrame(train,columns=train.columns)
         #columns_cv['c.variation'] = np.std(X_train.packed_cell_volume)/np.mean(X_train.packed_cell_volume)
         cv_value(columns_cv)
Out[58]: rectal_temp          0.019191
         pulse                0.389802
         packed_cell_volume   0.223815
         total_protein        1.123355
         lesion_1             1.518558
         dtype: float64
```

Assim, optou-se por imputar dados faltantes com a mediana as variáveis rectal_temp, pulse e packed_cell_volume.

```
In [59]: median_pulse = train["pulse"].median()
         median_packed = train["packed_cell_volume"].median()
         median_rectal = train["rectal_temp"].median()

In [60]: train["pulse"] = train["pulse"].replace(np.NaN, train["pulse"].median())
         train["packed_cell_volume"] = train["packed_cell_volume"].replace(np.NaN, train["packed_cell_volume"].median())
         train["rectal_temp"] = train["rectal_temp"].replace(np.NaN, train["rectal_temp"].median())
```

É importante assinalar que esse valor de mediana será o mesmo para imputar a base de teste depois nestes mesmos atributos, essa técnica permite resolver os problemas de pré-processamento da base de teste sem que haja vazamento de informações para o modelo.

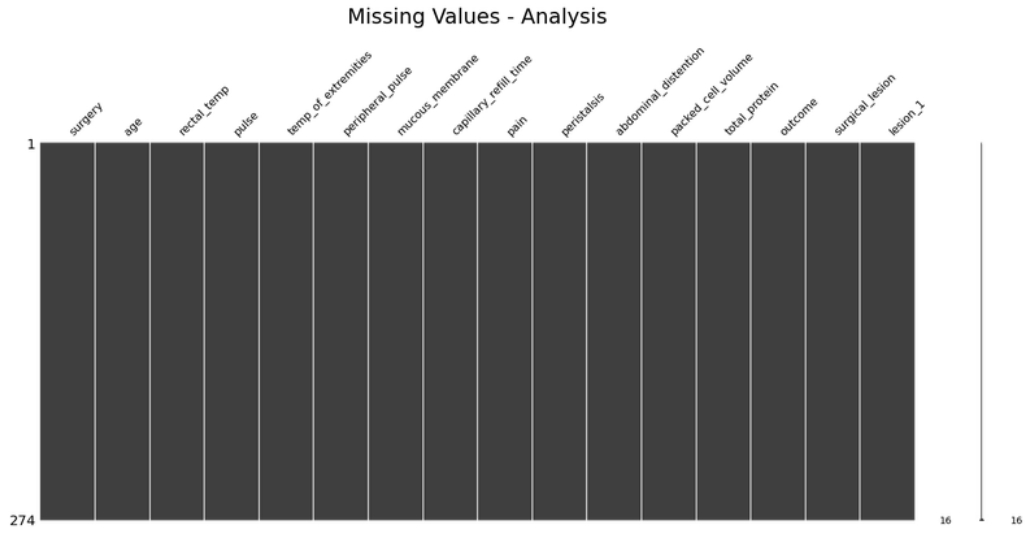
Temos então novamente o gráfico de valores faltantes:



Por fim, utilizou-se a moda nos demais atributos, ou seja, o valor mais frequente na base de dados em cada variável como forma de imputação de valores. É possível ainda, principalmente quando estamos lidando com muitos dados numéricos (na base atual, entendeu-se não ser tão adequado pelo elevado número de variáveis categóricas) a aplicação de modelos de regressão para preencher esses valores.

Com a aplicação da moda, tem-se que:

```
In [63]: msno.matrix(train) #- matrix
         plt.title('Missing Values - Analysis', size = 30, pad=30)
         plt.show()
```




```
In [64]: train.describe()
```

Out[64]:

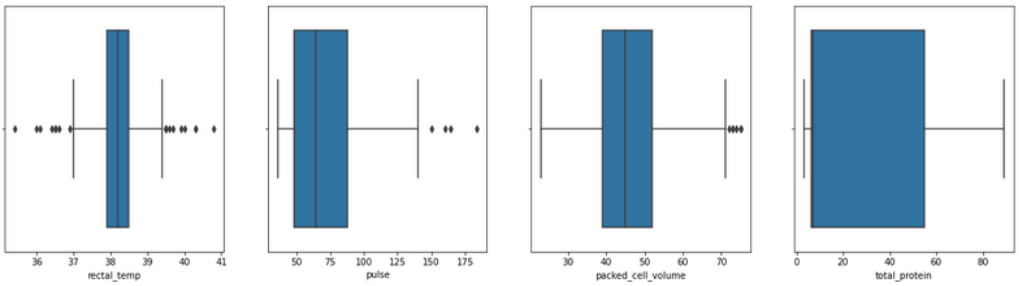
	rectal_temp	pulse	packed_cell_volume	total_protein	lesion_1
count	274.000000	274.000000	274.000000	274.000000	274.000000
mean	38.182117	71.835766	46.543796	23.174088	3659.594891
std	0.668173	27.511543	10.155111	26.943989	5567.474180
min	35.400000	36.000000	23.000000	3.300000	0.000000
25%	37.900000	48.250000	39.000000	6.500000	2111.000000
50%	38.200000	64.000000	45.000000	7.300000	2673.500000
75%	38.500000	88.000000	52.000000	54.750000	3208.500000
max	40.800000	184.000000	75.000000	89.000000	41110.000000

2.4. Removendo Outliers

Foi utilizado também, para as quatro variáveis numéricas que fariam sentido a análise de limites superiores e inferiores, a análise por box-plot e criação de função para a remoção de dados considerados outliers.

A ideia com isso não é evitar ‘treinar’ modelo sem casos de valores incomuns, mas sabendo justamente por se tratarem de dados incomuns, para efeito de treinamento e melhor ajuste do modelo, uma boa opção de pré-processamento em alguns casos.

```
In [65]: fig, axs = plt.subplots(ncols=4,figsize=(20, 5))  #sns.regplot(x='value', y='wage', data=df_melt, ax=axs[0])
sns.boxplot(x=train['rectal_temp'],ax=axs[0])
sns.boxplot(x=train['pulse'],ax=axs[1])
sns.boxplot(x=train['packed_cell_volume'],ax=axs[2])
sns.boxplot(x=train['total_protein'],ax=axs[3])
plt.show()
```



As we can see, using box-plot method, there are outliers in rectal_temp, pulse and packed_cell_volume attributes. If we remove all outliers, we are going to deal with a lot of loss information.
Then, he must to take careful with it. The rectal_temp variable doesn't have a lot of variance. The range seemns to be inside 36 - 41. At the other side, pulse and packed_cell_volume have a lot of variance, some of them identidy as outliers. So let's drop these values and see what happens...

```
In [66]: def remove_outlier(dtframe, col_name):   ### function to remove outliers
             q1 = dtframe[col_name].quantile(0.25)
             q3 = dtframe[col_name].quantile(0.75)
             iqr = q3-q1 #Interquartile range
             fence_low  = q1-1.5*iqr
             fence_high = q3+1.5*iqr
             dtframe_out = dtframe.loc[(dtframe[col_name] > fence_low) & (dtframe[col_name] < fence_high)]
             return dtframe_out
```

Before:

```
In [67]: train.shape
Out[67]: (274, 16)
```

Depois de retirados os outliers, é importante esclarecer a perda de mais 10 observações, ainda assim não representa necessariamente uma perda significativa. Assim, optou-se por manter essa técnica.

296	yes	adult	37.5	72.0	cold	reduced	pale_cyanotic	less_3_sec	severe_pain	absent
297	yes	adult	36.5	100.0	cool	reduced	pale_pink	less_3_sec	mild_pain	hypomotile

268 rows × 16 columns

```
In [70]: train = remove_outlier(train,'pulse')
In [71]: train = remove_outlier(train,'packed_cell_volume')
In [72]: train.shape
Out[72]: (264, 16)
```

We have lost 10 rows doing this. It's not a lot of loss and we gain further in model performance. It seems ok...

2.5. Padronização de Dados

Os dados foram padronizados nessa base, principalmente pela diferença entre as unidades de grandeza observadas entre as variáveis. É possível perceber, por exemplo, que a variável lesion_1, por se tratar de um conjunto de códigos, possui valores muito elevados se comparado com a variável rectal_temp.

Nesse ponto, pensou-se se de fato a padronização da variável de lesão cirúrgica poderia afetar a análise dos dados. No entanto, ainda que essa padronização seja aplicada, a relação existente entre os números codificados e o número

padronizada vão permanecer, de modo que será possível ainda extrair relação entre as variáveis.

4.5 Data Standardization

It's important to standardize these attributes. For models, there are gains in performance.

In [73]:
```
cols = "rectal_temp,pulse,packed_cell_volume,total_protein,lesion_1".split(",") # transform one large str into a l
cols
```

Out[73]:
```
['rectal_temp', 'pulse', 'packed_cell_volume', 'total_protein', 'lesion_1']
```

In [74]:
```
train[cols]
```

Out[74]:

	rectal_temp	pulse	packed_cell_volume	total_protein	lesion_1
0	38.5	66.0	45.0	8.4	11300
1	39.2	88.0	50.0	85.0	2208
2	38.3	40.0	33.0	6.7	0
5	38.2	64.0	45.0	6.5	0
6	37.9	48.0	37.0	7.0	3124
...	...	...	...	...	...
293	38.5	40.0	37.0	67.0	0
294	38.2	120.0	55.0	65.0	3205
295	37.2	72.0	44.0	6.5	2208
296	37.5	72.0	60.0	6.8	3205
297	36.5	100.0	50.0	6.0	2208

264 rows × 5 columns

In [75]:
```
from sklearn import preprocessing
sca = preprocessing.StandardScaler()
train[cols] = sca.fit_transform(train[cols])
```

In [76]:
```
train.shape
```

Out[76]:
```
(264, 16)
```

In [77]:
```
x_train = train.loc[:,train.columns != "outcome"] #input
y_train = train.outcome
```

2.6. One-hot encoding

Por fim, por se tratar de base com variáveis categóricas, realizou-se processo de aplicação da técnica de one-hot encoding, de modo a transformar as variáveis categóricas em variáveis numéricas.

Para isso, ainda seria possível apenas codificar com numeração simples entre essas variáveis (1,2,3...). No entanto, esse tipo de codificação precisa ser aplicado com cuidado, uma vez que pode passar para o modelo uma relação de ordenamento ou aproximação entre as características das variáveis. Assim, para o caso analisado, a técnica de one-hot encoding mostra-se mais adequada.

So, we are going to apply **One-Hot Encoding** technique.

```
78]: import pandas as pd
     import numpy as np# creating initial dataframe
     #bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')
     #bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])# generate binary values using get_dummies
     x_train = pd.get_dummies(x_train)# merge with main df bridge_df on key values
     #teste = bridge_df.join(dum_df)
     #bridge_df
     x_train
```

78]:

	rectal_temp	pulse	packed_cell_volume	total_protein	lesion_1	surgery_no	surgery_yes	age_adult	age_young	temp_of_extremities_cold	.
0	0.495327	-0.160519	-0.105454	-0.560569	1.357707	1	0	1	0	0	.
1	1.559705	0.711598	0.425841	2.277913	-0.252633	0	1	1	0	0	.
2	0.191219	-1.191202	-1.380563	-0.623564	-0.643706	1	0	1	0	0	.
5	0.039165	-0.239802	-0.105454	-0.630975	-0.643706	1	0	1	0	0	.
6	-0.416996	-0.874069	-0.955526	-0.612447	-0.090395	0	1	1	0	0	.
...	...	...	...	...	...	...	...	...	...	...	.
293	0.495327	-1.191202	-0.955526	1.610907	-0.643706	1	0	1	0	0	.
294	0.039165	1.980132	0.957136	1.536795	-0.076049	0	1	1	0	1	.
295	-1.481374	0.077331	-0.211713	-0.630975	-0.252633	1	0	1	0	0	.
296	-1.025212	0.077331	1.488432	-0.619858	-0.076049	0	1	1	0	1	.
297	-2.545752	1.187298	0.425841	-0.649503	-0.252633	0	1	1	0	0	.

264 rows × 41 columns

```
79]: x_train.info() #confirming 5 float objects
```

2.7. Aplicação das Técnicas na Base de Teste

Como forma de aplicar as técnicas de pré-processamento realizadas na base de treino, apresenta-se de forma sucinta as codificações realizadas para os tratamentos elencados nos tópicos anteriores.

Salienta-se que devido à base de dados como um todo ter uma quantidade de observações mais contida, entende-se que a aplicação de tratamento de outliers na base de teste não seria ideal, por conter menos observações ainda. Assim, foi a única técnica não aplicada na base de teste.

4.6 Treatment on Test data

Now we have to do some treatments on test dataset. We have to:

1. Treat outcome label "died" = "euthanazied"
2. Select the same columns of train dataset
3. We have to fill missing values with median and mode of train dataset (avoid leakage information).
4. Data standardization
5. One-hot encoding

4.6.2 Columns Analysis

```
In [85]: test = test.drop(columns = ['cp_data'])

In [86]: test.shape
Out[86]: (89, 27)

In [87]: test = test.drop(columns = ['hospital_number','respiratory_rate','lesion_2','lesion_3'])

In [88]: test.shape
Out[88]: (89, 23)
```

4.6.3 Missing Values

```
In [89]: #missing_value_data.loc[percent_missing > 30].index.values
         #missing_value_data.loc[percent_missing > 30].index.

In [90]: test = test.drop(columns = filter_data)

In [91]: test.shape
Out[91]: (89, 16)

In [92]: test["pulse"] = test["pulse"].replace(np.NaN, median_pulse)
```

In [92]:
```
test["pulse"] = test["pulse"].replace(np.NaN, median_pulse)
test["packed_cell_volume"] = test["packed_cell_volume"].replace(np.NaN, median_packed)
test["rectal_temp"] = test["rectal_temp"].replace(np.NaN, median_rectal)
```

In [93]:
```
test = test.fillna(train.mode().iloc[0]) ###we have to input the train mode to avoid data leakage
```

4.6.4 Data standardization

In [94]:
```
from sklearn import preprocessing
sca = preprocessing.StandardScaler()
test[cols] = sca.fit_transform(test[cols])
```

In [95]:
```
x_test = test.loc[:,test.columns != "outcome"] #input
y_test = test.outcome
```

4.6.5 One-Hot Encoding

In [96]:
```
import pandas as pd
import numpy as np# cresting initial dataframe
#bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')
#bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])# generate binary values using get_dummies
x_test = pd.get_dummies(x_test)# merge with main df bridge_df on key values
#teste = bridge_df.join(dum_df)
#bridge_df
x_test
```

Out[96]:

In [97]:
```
type(x_test)
```

Out[97]:
```
pandas.core.frame.DataFrame
```

In [98]:
```
np.array(x_test)
```

Out[98]:
```
array([[-1.20931581,  0.98914963,  3.23937024, ...,  0.        ,
         1.        ,  0.        ],
       [ 1.26491653, -0.07270418,  0.5044711 , ...,  1.        ,
         0.        ,  1.        ],
       [-1.34677316, -1.06819212, -0.06529956, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [ 0.99000183,  0.19275928,  0.96028762, ...,  0.        ,
         0.        ,  1.        ],
       [-1.62168786, -0.27180177, -1.20484087, ...,  0.        ,
         0.        ,  1.        ],
       [-2.30897462,  0.85641791,  0.5044711 , ...,  0.        ,
         0.        ,  1.        ]])
```

3. MODELOS SUPERVISIONADOS DE ML

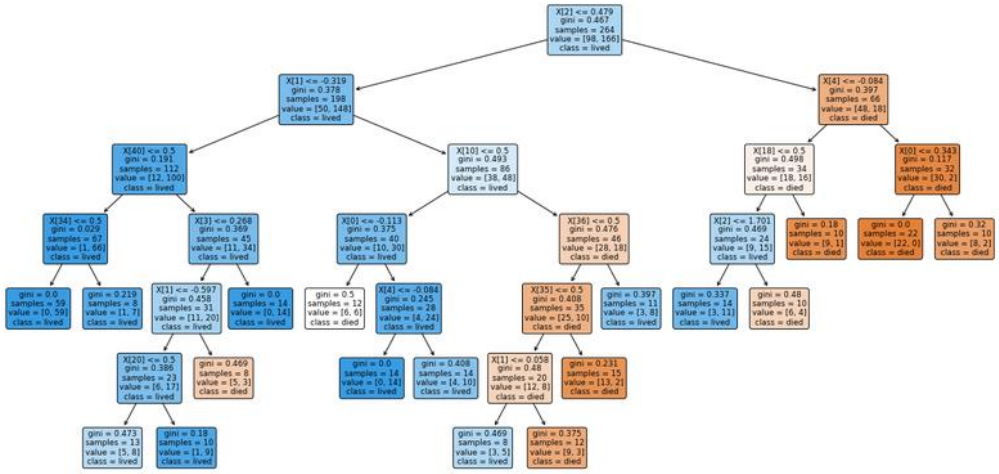
Após pré-processar os dados, diferentes modelos supervisionados com aplicação em problemas de classificação foram utilizados, conforme listado a seguir com respectivos resultados.

3.1. Decision Tree

```
In [100]: # training model
          from sklearn.tree import DecisionTreeClassifier

          def train(x_train, y_train, seed):
            model = DecisionTreeClassifier(min_samples_leaf=8, random_state=seed) # tried 8 min sample leaf...
            model.fit(x_train, y_train);
            return model

          model = train(x_train, y_train, seed)
```

```
In [101]: model
Out[101]: DecisionTreeClassifier(min_samples_leaf=8, random_state=1)
```

```
In [102]: # Tree Plot visualization
          from sklearn import tree
          fig, ax = plt.subplots(figsize=(20, 10))
          tree.plot_tree(model, class_names=['died', 'lived'], filled=True, rounded=True);
```



Função com medidas de análise dos modelos:

```
In [103]: def predict_and_evaluate(x_test, y_test):

            y_pred = model.predict(x_test) #inference

            # Accuracy
            from sklearn.metrics import accuracy_score
            accuracy = accuracy_score(y_test, y_pred)
            print('Accuracy: ', accuracy)

            # Kappa
            from sklearn.metrics import cohen_kappa_score
            kappa = cohen_kappa_score(y_test, y_pred)
            print('Kappa: ', kappa)

            # F1
            from sklearn.metrics import f1_score
            f1 = f1_score(y_test, y_pred, pos_label = "lived")
            print('F1: ', f1)

            # Confusion matrix
            from sklearn.metrics import confusion_matrix
            confMatrix = confusion_matrix(y_pred, y_test)

            ax = plt.subplot()
            sns.heatmap(confMatrix, annot=True, fmt=".0f")
            plt.xlabel('Real')
            plt.ylabel('Predicted')
            plt.title('Confusion Matrix')

            # set names
            ax.xaxis.set_ticklabels(['died', 'lived'])
            ax.yaxis.set_ticklabels(['died', 'lived'])
            plt.show()

          predict_and_evaluate(x_test, y_test)
```

Destaca-se que, por se tratar de variável categórica, era necessário codificar os resultados de saída para aplicar no modelo ou definir o label, principalmente para o cálculo da medida F1, conforme acima (pos_label = 'lived').

```
Acurácia:  0.7865168539325843
Kappa:  0.5627101111973106
F1:  0.8155339805825242
```

O resultado da matriz de confusão permite dizer que o modelo treinado apresentou valores de acurácia, kappa e F1 relativamente baixos.

Foi utilizado a função GridSearch para realizar testes dos hiperparâmetros e obter a melhor estimativa do modelo. Nesse caso, houve melhora, mas praticamente sem mudança.

In [268]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer

f1_scorer = make_scorer(f1_score, pos_label="lived")
```

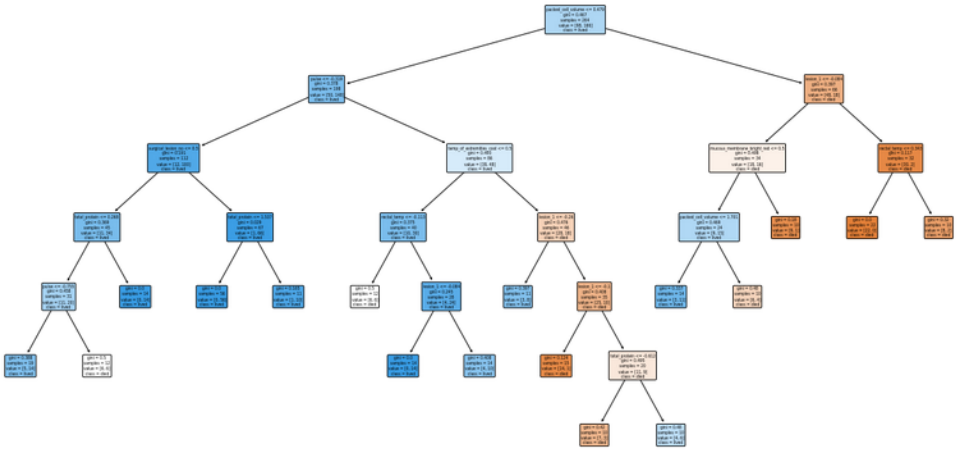
In [269]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
# Set the parameters by cross-validation
tuned_parameters = [{'criterion': ['gini', 'entropy'], 'max_depth': [2,4,6,7,8,9,10,12],
                     'min_samples_leaf': [1, 2, 3, 4, 5, 8, 10,12]}] # using gridsearch to try different tuned par

print("# Tuning hyper-parameters for F1 score")
print()

model = GridSearchCV(DecisionTreeClassifier(), tuned_parameters, scoring=f1_scorer) # instead of 'f1'
model.fit(x_train, y_train)

y_true, y_pred = y_test, model.predict(x_test)
print(classification_report(y_true, y_pred))
print()
```

```
# Tuning hyper-parameters for F1 score

              precision    recall  f1-score   support

        died       0.72      0.81      0.76        36
       lived       0.86      0.79      0.82        53

    accuracy                           0.80        89
   macro avg       0.79      0.80      0.79        89
weighted avg       0.80      0.80      0.80        89
```

In [270]:
```
predict_and_evaluate(x_test, y_test)
```

```
Accuracy:  0.797752808988764
Kappa:  0.5875386199794026
F1:  0.8235294117647058
```

```
In [108]: fig, ax = plt.subplots(figsize=(20, 10))
          tree.plot_tree(model.best_estimator_, class_names=['died', 'lived'],
                         filled=True, rounded=True, feature_names=grid_col.columns);
```



3.2. KNN

O modelo KNN, considerando o parâmetro dos três vizinhos mais próximos, apresentou um ganho sensível nas medidas de precisão.

```
Accuracy:  0.8202247191011236
Kappa:  0.6130434782608696
F1:  0.8152966686378869
```

A aplicação do GridSearch para testar diferentes parâmetros mostrou queda nos valores usando os 10 vizinhos mais próximos.

```
In [275]: evaluate(y_test, y_pred)

          Accuracy:  0.7640449438202247
          Kappa:  0.4944549634839058 3
          F1:  0.7586480140124369
```


```
In [276]: model.best_params_
Out[276]: {'n_neighbors': 10}
```

3.3. SVM

Aplicando o modelo de Support Vector Machine, houve melhora nos resultados encontrados, com aumento do percentual de acurácia, da medida de Kappa e do F1. O tipo de kernel usado para modelagem é um função de base radial.

```
Accuracy:  0.8539325842696629
Kappa:  0.6898954703832753
F1:  0.8828828828828829
```

Para a o uso do GridSearch, mantendo-se a função de base radial e variando alguns dos parâmetros, não houve melhora na modelagem.

```
In [286]: predict_and_evaluate(x_test, y_test)

Accuracy:  0.7752808988764045
Kappa:  0.5117937465715854
F1:  0.8275862068965516
```


```
In [287]: model.best_params_
Out[287]: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

Aplicou-se, assim, técnica de redução de dimensionalidade, pelo alto número de variáveis categóricas codificadas. A técnica de redução definida é a PCA:

```
In [124]: from sklearn.decomposition import PCA
          pca = PCA().fit(x_train)
          df = pd.DataFrame(pca.explained_variance_ratio_.cumsum())
          with pd.option_context('display.max_rows', None, 'display.max_columns', None):
              print(df)

                0
0    0.209343
1    0.326688
2    0.426987
3    0.514967
4    0.573035
5    0.626551
6    0.667394
7    0.706510
8    0.736516
9    0.764555
10   0.791350
11   0.816197
12   0.835263
13   0.853325
14   0.870077
15   0.885969
16   0.900822
17   0.914159
18   0.927148
19   0.939776
20   0.949681
21   0.958561
22   0.966013
23   0.972964
24   0.979220
25   0.984546
26   0.989597
27   0.993746
28   0.997407
29   0.999068
```

Definido o corte em 30 e treinando novamente o modelo, houve melhora em relação ao primeiro modelo SVM treinado e consideravelmente melhor em relação ao uso do GridSearch.

```
Accuracy:  0.8651685393258427
Kappa:  0.7124394184168013
F1:  0.8928571428571428
```

3.4. Random Forest

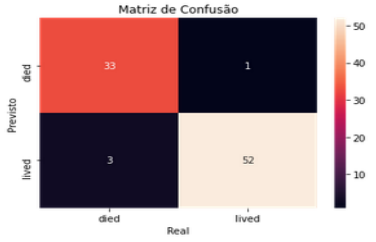
O uso do modelo de Random Forest, muito comum nos problemas de classificação envolvendo números menores de observações, apresentou uma melhora considerável em relação aos modelos anteriores. Ainda assim, é necessário considerar que a definição do número mínimo de folhas nesse modelo pode fazer diferença no treinamento do modelo, de forma a gerar overfitting.

Utilizando-se o mínimo de folhas equivalente a três, os resultados foram positivos no treinamento do modelo.

```
Accuracy:  0.8764044943820225
Kappa:  0.7302287131441167
F1:  0.905982905982906
```

O uso do GridSearch gerou melhora nos valores observados:

```
In [300]: predict_and_evaluate(x_test, y_test, model)
[Parallel(n_jobs=50)]: Using backend ThreadingBackend with 50 concurrent workers.
[Parallel(n_jobs=50)]: Done   2 out of  50 | elapsed:    0.0s remaining:    0.8s
[Parallel(n_jobs=50)]: Done  50 out of  50 | elapsed:    0.0s finished
Accuracy:  0.9550561797752809
Kappa:  0.9058699101004759
F1:  0.9629629629629629
```



3.5. XGBoost

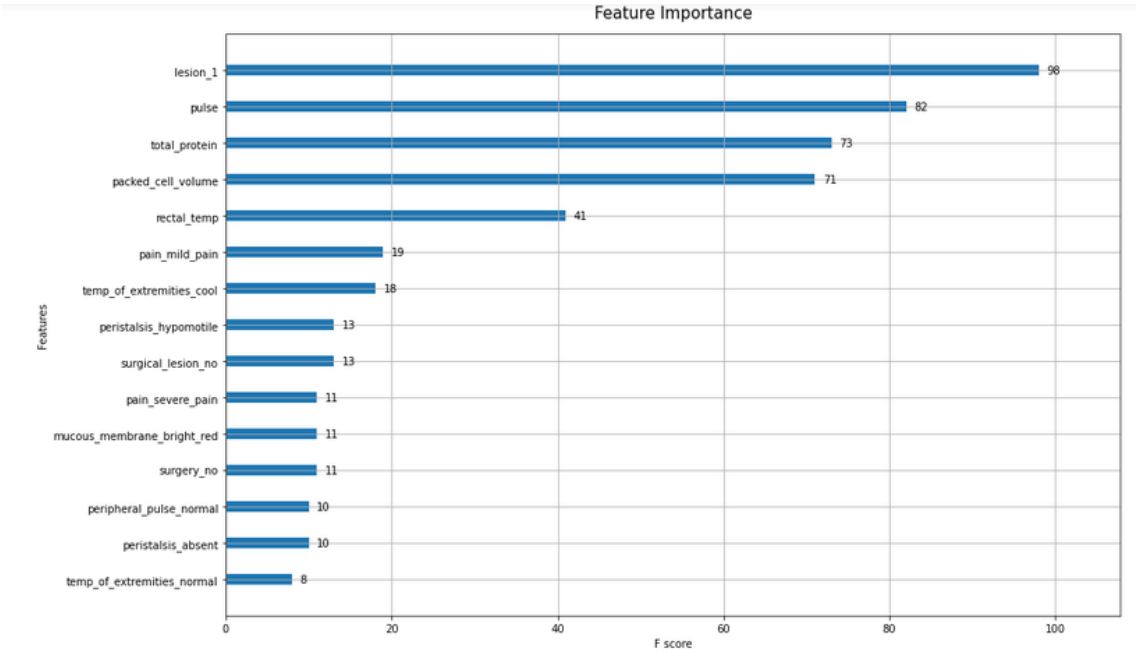
Por fim, aplicou-se o modelo de XGBoost, que tem a vantagem de aplicar técnicas de ensemble, tornando um método de aprendizagem mais fraca em um método mais forte.

```
In [135]: #pip install xgboost

In [136]: import xgboost as xgb
          from sklearn.metrics import mean_squared_error

In [137]: #data_dmatrix = xgb.DMatrix(data=x_train, label=y_train) ...think y_train must be coded (1,0)
          xgb_class = xgb.XGBClassifier (random_state=seed)

In [138]: xgb_class.fit(x_train, y_train, early_stopping_rounds = 10, eval_set=[(x_test,y_test)])

[16:56:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting i
n XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error
' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[0]     validation_0-logloss:0.58336
[1]     validation_0-logloss:0.53187
[2]     validation_0-logloss:0.47831
[3]     validation_0-logloss:0.46342

C:\Users\marco\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGB
Classifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) P
ass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as inte
gers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[4]     validation_0-logloss:0.43217
[5]     validation_0-logloss:0.40930
[6]     validation_0-logloss:0.39250
[7]     validation_0-logloss:0.39645
[8]     validation_0-logloss:0.39073
[9]     validation_0-logloss:0.39497
[10]    validation_0-logloss:0.39105
[11]    validation_0-logloss:0.37907
[12]    validation_0-logloss:0.37589

Out[138]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                        colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
                        importance_type='gain', interaction_constraints='',
                        learning_rate=0.300000012, max_delta_step=0, max_depth=6,
                        min_child_weight=1, missing=nan, monotone_constraints='()',
                        n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=1,
                        reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                        tree_method='exact', validate_parameters=1, verbosity=None)

In [139]: xgb_class.score(x_test,y_test)
Out[139]: 0.8426966292134831

In [140]: def train(x_train, y_train, seed):
            model = xgb.XGBClassifier (random_state=seed) # tente mudar parâmetro para evitar overfitting
            model.fit(x_train, y_train, early_stopping_rounds = 10, eval_set=[(x_test,y_test)]);
            return model

          model = train(x_train, y_train, seed)

[16:56:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting i
n XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error
' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Os resultados mostram um valor pior do que o modelo anterior, mas ainda melhor do que os primeiros modelos.

```
Accuracy:  0.8426966292134831
Kappa:  0.6645126548196015
F1:  0.875
```

O cálculo do erro de raiz quadrático médio (RMSE) foi de 0.237023. Com a biblioteca do XGBoost é possível obter uma relação de importância dos atributos, conforme mostrado a seguir.

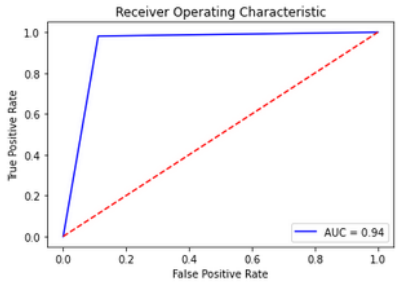


Como medida de análise da precisão, tem-se ainda a curva característica de operação do receptor (ROC curve). Quanto mais próximo do extremo esquerdo, mais adequada é a classificação. Quanto mais próxima da linha pontilhada em vermelho, mais próximo de uma classificação aleatória do modelo. Assim, os resultados mostram grau mais adequado do modelo.

ROC Curve

In [149]:
```
from sklearn import metrics
def buildROC(y_test, y_pred):
    fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.gcf().savefig('roc.png')
```

In [150]:
```
buildROC(y_test, y_pred)
```



4. CONSIDERAÇÕES FINAIS

O presente relatório teve como objetivo aplicar os principais conceitos desenvolvidos na disciplina de Data Mining de modo a obter e treinar modelos que possam classificar rótulo atinentes ao contexto de sobrevivência de animais equinos.

Para isso, buscou-se aplicar diversas técnicas mais desenvolvidas, de modo a obter um conjunto de técnicas mais completas. Em vista disso, limitações possam ter surgido, principalmente em termos de perda de informação, que levou a uma precisão ruim em alguns modelos, mas muito boas em outros modelos.

Ao mesmo tempo, a ideia é poder aplicar posteriormente esse conjunto de técnicas, principalmente em relação à questão de pré-processamento, para verificar o grau de melhoria e ajuste dessas técnicas em relação à base original.

Por fim, é perceptível que o tratamento dos dados e aplicação e testes de diversos modelos, aliados a uma análise detalhada e disponibilizada base de dados, é possível extrair informações valiosas para a tomada de decisões em negócios.