Data Preprocessing

In [ ]:
```
# Importing necessery packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn

df = pd.read_csv('Data.csv')
df
```

Out[ ]:

	surgery	age	hospital_number	rectal_temperature	pulse	respiratory_rate	t
0	1	1	534817	39.2	88	20	
1	2	1	530334	38.3	40	24	
2	1	9	5290409	39.1	164	84	
3	2	1	530255	37.3	104	35	
4	2	1	528355	?	?	?	
...	...	...	...	...	...	...	
288	1	1	529126	38	50	36	
289	2	1	535054	38.6	45	16	
290	1	1	528890	38.9	80	44	
291	1	1	530034	37	66	20	
292	1	1	534004	?	78	24	

293 rows × 25 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 293 entries, 0 to 292
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   surgery                        293 non-null    object
 1   age                            293 non-null    int64 
 2   hospital_number                293 non-null    int64 
 3   rectal_temperature             293 non-null    object
 4   pulse                          293 non-null    object
 5   respiratory_rate               293 non-null    object
 6   temperature_of_extremities     293 non-null    object
 7   peripheral_pulse               293 non-null    object
 8   capillary_refill_time          293 non-null    object
 9   pain                           293 non-null    object
 10  peristalsis                    293 non-null    object
 11  abdominal_distension           293 non-null    object
 12  nasogastric_tube               293 non-null    object
 13  nasogastric_reflux             293 non-null    object
 14  nasogastric_reflux_ph          293 non-null    object
 15  rectal_examination_feces       293 non-null    object
 16  packed_cell_volume             293 non-null    object
 17  total_protein                  293 non-null    object
 18  abdominocentesis_total_protein 293 non-null    object
 19  lesion_site                    293 non-null    int64 
 20  lesion_type                    293 non-null    int64 
 21  lesion_subtype                 293 non-null    int64 
 22  PossibleY                      293 non-null    int64 
 23  PossibleY2                     293 non-null    object
 24  PossibleY3                     293 non-null    int64 
dtypes: int64(7), object(18)
memory usage: 57.4+ KB
```

Since most of the columns are showing as objects and we can see that all the columns of the dataset is in numerical. So we came to know that the missing values are represented in some kind of special character. '?' So we have to replace those '?' with NaN values first and then process with the null values

In [ ]:
```
df = df.replace('?', np.NaN)
df
```

Out[ ]:

	surgery	age	hospital_number	rectal_temperature	pulse	respiratory_rate	t
0	1	1	534817	39.2	88	20	
1	2	1	530334	38.3	40	24	
2	1	9	5290409	39.1	164	84	
3	2	1	530255	37.3	104	35	
4	2	1	528355	NaN	NaN	NaN	
...	...	...	...	...	...	...	
288	1	1	529126	38	50	36	
289	2	1	535054	38.6	45	16	
290	1	1	528890	38.9	80	44	
291	1	1	530034	37	66	20	
292	1	1	534004	NaN	78	24	

293 rows × 25 columns

Now see the null values and do the appropriate task for fillling the data or drop it.

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
surgery                          1
age                              0
hospital_number                  0
rectal_temperature              59
pulse                           24
respiratory_rate                58
temperature_of_extremities      55
peripheral_pulse                67
capillary_refill_time           31
pain                            54
peristalsis                     43
abdominal_distension            54
nasogastric_tube               100
nasogastric_reflux             102
nasogastric_reflux_ph          240
rectal_examination_feces       101
packed_cell_volume              29
total_protein                   32
abdominocentesis_total_protein  194
lesion_site                      0
lesion_type                      0
lesion_subtype                   0
PossibleY                        0
PossibleY2                       1
PossibleY3                       0
dtype: int64
```

In [ ]:
```
df.shape
```

Out[ ]: (293, 25)

We came to know that some of the rows are having null values more than half of the whole dataset. So we can drop them. The rest we will replace it with the mean or median or most frequent(mode).

In [ ]:
```
for column in df.columns:
    if df[column].isnull().sum() > 70:
        df = df.drop(column, axis=1)
        print(column, "Dropped!")

df.isnull().sum()
```

```
nasogastric_tube Dropped!
nasogastric_reflux Dropped!
nasogastric_reflux_ph Dropped!
rectal_examination_feces Dropped!
abdominocentesis_total_protein Dropped!
```

Out[ ]:
```
surgery                          1
age                              0
hospital_number                  0
rectal_temperature              59
pulse                           24
respiratory_rate                58
temperature_of_extremities      55
peripheral_pulse                67
capillary_refill_time           31
pain                            54
peristalsis                     43
abdominal_distension            54
packed_cell_volume              29
total_protein                   32
lesion_site                      0
lesion_type                      0
lesion_subtype                   0
PossibleY                        0
PossibleY2                       1
PossibleY3                       0
dtype: int64
```

Now the columns which have more than 70 null values are removed. They do not contribut much to our model. Now replace the null values with the median in every column.

In [ ]:
```
for column in df.columns:
    x_median = df[column].median()
    df[column] = df[column].fillna(x_median)

df.isnull().sum()
```

Out[ ]:
```
surgery                        0
age                            0
hospital_number                0
rectal_temperature             0
pulse                          0
respiratory_rate               0
temperature_of_extremities     0
peripheral_pulse               0
capillary_refill_time          0
pain                           0
peristalsis                    0
abdominal_distension           0
packed_cell_volume             0
total_protein                  0
lesion_site                    0
lesion_type                    0
lesion_subtype                 0
PossibleY                      0
PossibleY2                     0
PossibleY3                     0
dtype: int64
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(293, 20)
```

Now all the null dataset are cleared. Now lets find the correlation of the dataset.

In [ ]:
```
df.corr()
df.shape
```

Out[ ]:
```
(293, 20)
```

In [ ]:
```
#Normalize

from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()
df_1 = df.drop('PossibleY', axis=1)
std_df = scalar.fit_transform(df)
df_std = pd.DataFrame(std_df, columns=df.columns)
```

In [ ]:
```
X = df_std.drop('PossibleY', axis=1)
y = df['PossibleY']
```

From the correlation matrix, we didn't find any revelent correlation amoung the features. Lets visualize and see once more.

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Decision Tree for 1st variable

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report

clf = DecisionTreeClassifier(criterion='entropy', max_depth=9)

clf.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=9)
```

In [ ]:
```
predicted_dt = clf.predict(X_train)
print("Accuracy for training set : ", (accuracy_score(y_train, predicted_
```

```
Accuracy for training set :  0.9316239316239316
```

In [ ]:
```
confusion_matrix(y_train, predicted_dt)
```

Out[ ]:
```
array([[ 80,   0],
       [ 16, 138]])
```

In [ ]:
```
print(classification_report(y_train, predicted_dt))
```

```
              precision    recall  f1-score   support

           1       0.83      1.00      0.91        80
           2       1.00      0.90      0.95       154

    accuracy                           0.93       234
   macro avg       0.92      0.95      0.93       234
weighted avg       0.94      0.93      0.93       234
```

In [ ]:
```
from sklearn.metrics import accuracy_score

y_pred = clf.predict(X_test)
print("Accuracy for testing set", accuracy_score(y_test, y_pred))
```

```
Accuracy for testing set 0.7627118644067796
 0.7627118644067796
```

In [ ]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

Out[ ]:
```
array([[14,  3],
       [11, 31]])
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.56      0.82      0.67        17
           2       0.91      0.74      0.82        42

    accuracy                           0.76        59
   macro avg       0.74      0.78      0.74        59
weighted avg       0.81      0.76      0.77        59
```

SVM for 1st variable

In [ ]:
```
from sklearn.svm import SVC

svc_clf = SVC(kernel='poly')

svc_clf.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
predicted = svc_clf.predict(X_train)
print("Accuracy for training set : ", (accuracy_score(y_train, predicted)
```

```
Accuracy for training set :  0.7991452991452992
```

In [ ]:
```
confusion_matrix(y_train, predicted)
```

Out[ ]:
```
array([[ 35,  45],
       [  2, 152]])
```

In [ ]:
```
print(classification_report(y_train, predicted))
```

```
              precision    recall  f1-score   support

           1       0.95      0.44      0.60        80
           2       0.77      0.99      0.87       154

    accuracy                           0.80       234
   macro avg       0.86      0.71      0.73       234
weighted avg       0.83      0.80      0.77       234
```

In [ ]:
```
y_pred = svc_clf.predict(X_test)
print("Accuracy for testing set : ", accuracy_score(y_test, y_pred))
```

```
Accuracy for testing set :  0.7457627118644068
```

In [ ]:
```
confusion_matrix(y_test, y_pred)
```

Out[ ]:
```
array([[ 3, 14],
       [ 1, 41]])
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.75      0.18      0.29        17
           2       0.75      0.98      0.85        42

    accuracy                           0.75        59
   macro avg       0.75      0.58      0.57        59
weighted avg       0.75      0.75      0.68        59
```

Taking PossibleY3 as the target

In [ ]:
```
# Normalizing
std_df = scalar.fit_transform(df)
df_std = pd.DataFrame(std_df, columns=df.columns)
X = df_std.drop('PossibleY3', axis=1)
y = df['PossibleY3']
```

Splitting dataset

In [ ]:
```
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X, y, tes
```

Decision Tree for 2nd variable

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier(max_depth=7, criterion='entropy')

dt_clf.fit(X_train_dt, y_train_dt)
```

Out[ ]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=7)
```

In [ ]:
```
predicted_dt2 = svc_clf.predict(X_train_dt)
print("Accuracy for training set : ", (accuracy_score(y_train_dt, predict
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[550], line 1
----> 1 predicted_dt2 = svc_clf.predict(X_train_dt)
      2 print("Accuracy for training set : ", (accuracy_score(y_train_dt, predicted_dt2)))

File /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/svm/_base.py:818, in BaseSVC.predict(self, X)
    816     y = np.argmax(self.decision_function(X), axis=1)
    817 else:
--> 818     y = super().predict(X)
    819 return self.classes_.take(np.asarray(y, dtype=np.intp))

File /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/svm/_base.py:431, in BaseLibSVM.predict(self, X)
    415 def predict(self, X):
    416     """Perform regression on samples in X.
    417
    418     For an one-class model, +1 (inlier) or -1 (outlier) is returned.
   (...)
    429         The predicted values.
    430     """
--> 431     X = self._validate_for_predict(X)
    432     predict = self._sparse_predict if self._sparse else self._dense_predict
    433     return predict(X)

File /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/svm/_base.py:611, in BaseLibSVM._validate_for_predict(self, X)
    608 check_is_fitted(self)
    610 if not callable(self.kernel):
--> 611     X = self._validate_data(
    612         X,
    613         accept_sparse="csr",
    614         dtype=np.float64,
    615         order="C",
    616         accept_large_sparse=False,
    617         reset=False,
    618     )
    620 if self._sparse and not sp.issparse(X):
    621     X = sp.csr_matrix(X)

File /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/base.py:580, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **check_params)
    509 def _validate_data(
    510     self,
    511     X="no_validation",
   (...)
    516     **check_params,
    517 ):
    518     """Validate input data and set or check the `n_features_in_` attribute.
    519
    520     Parameters
```

```
   (...)
    578         validated.
    579     """
--> 580     self._check_feature_names(X, reset=reset)
    582     if y is None and self._get_tags()["requires_y"]:
    583         raise ValueError(
    584             f"This {self.__class__.__name__} estimator "
    585             "requires y to be passed, but the target y is None."
    586         )

File /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/base.py:507, in BaseEstimator._check_feature_names(self, X, reset)
    502 if not missing_names and not unexpected_names:
    503     message += (
    504         "Feature names must be in the same order as they were in fit.\n"
    505     )
--> 507 raise ValueError(message)

ValueError: The feature names should match those that were passed during fit.
Feature names unseen at fit time:
- PossibleY
Feature names seen at fit time, yet now missing:
- PossibleY3
```

In [ ]:
```
confusion_matrix(y_train_dt, predicted_dt2)
```

Out[ ]:
```
array([[140,   7],
       [ 17,  70]])
```

In [ ]:
```
print(classification_report(y_train_dt, predicted_dt2))
```

```
              precision    recall  f1-score   support

           1       0.89      0.95      0.92       147
           2       0.91      0.80      0.85        87

    accuracy                           0.90       234
   macro avg       0.90      0.88      0.89       234
weighted avg       0.90      0.90      0.90       234
```

In [ ]:
```
y_pred_dt = dt_clf.predict(X_test)
print("Accuracy for testing set : ", accuracy_score(y_test_dt, y_pred_dt))
```

```
Accuracy for testing set :  0.9152542372881356
```

In [ ]:
```
print(confusion_matrix(y_test_dt, y_pred_dt))
```

```
[[38  3]
 [ 2 16]]
```

In [ ]:
```
print(classification_report(y_test_dt, y_pred_dt))
```

```
                precision    recall  f1-score   support

           1       0.95      0.93      0.94        41
           2       0.84      0.89      0.86        18

    accuracy                           0.92        59
   macro avg       0.90      0.91      0.90        59
weighted avg       0.92      0.92      0.92        59
```

SVM for 2nd variable

In [ ]:
```
svc_clf2 = SVC(kernel='linear')
svc_clf2.fit(X_train_dt, y_train_dt)
```

Out[ ]:
```
SVC(kernel='linear')
```

In [ ]:
```
predicted_svc = svc_clf.predict(X_train_dt)
print("Accuracy for training set : ", (accuracy_score(y_train_dt, predict
```

```
Accuracy for training set :  0.8974358974358975
```

In [ ]:
```
confusion_matrix(y_train_dt, predicted_svc)
```

Out[ ]:
```
array([[140,   7],
       [ 17,  70]])
```

In [ ]:
```
print(classification_report(y_train_dt, predicted_svc))
```

```
              precision    recall  f1-score   support

           1       0.89      0.95      0.92       147
           2       0.91      0.80      0.85        87

    accuracy                           0.90       234
   macro avg       0.90      0.88      0.89       234
weighted avg       0.90      0.90      0.90       234
```

In [ ]:
```
y_perd_svc2 = svc_clf.predict(X_test_dt)
print("Accuracy for testing set : ", accuracy_score(y_test, y_perd_svc2))
```

```
Accuracy for testing set :  0.8983050847457628
```

In [ ]:
```
confusion_matrix(y_test_dt, y_perd_svc2)
```

Out[ ]:
```
array([[40,  1],
       [ 5, 13]])
```

In [ ]:
```
print(classification_report(y_test_dt, y_perd_svc2))
```

	precision	recall	f1-score	support
1	0.89	0.98	0.93	41
2	0.93	0.72	0.81	18
accuracy			0.90	59
macro avg	0.91	0.85	0.87	59
weighted avg	0.90	0.90	0.89	59