

URINALYSIS TEST RESULTS ANALYSIS AND BINARY CLASSIFICATION USING MACHINE LEARNING MODELS

By: Tran Dinh An

I. About the dataset

- Description: This dataset was collected from a local clinic in Northern Mindanao, Philippines, and is from April of 2020 to January of 2023. UTIs are common infections that happen when bacteria, often from the skin or rectum, enter the urethra, and infect the urinary tract.
- Source: [Urinalysis Test Results \(kaggle.com\)](https://www.kaggle.com/datasets/urinalysis-test-results)

	Unnamed: 0	Age	Gender	Color	Transparency	Glucose	Protein	pH	Specific Gravity	WBC	RBC	Epithelial Cells	Mucous Threads	Amorphous Urates	Bacteria	Diagnosis
0	0	76.00	FEMALE	LIGHT YELLOW	CLEAR	NEGATIVE	NEGATIVE	5.0	1.010	1-3	0-2	OCCASIONAL	RARE	NONE SEEN	OCCASIONAL	NEGATIVE
1	1	9.00	MALE	DARK YELLOW	SLIGHTLY HAZY	NEGATIVE	1+	5.0	1.030	1-3	0-2	RARE	FEW	FEW	MODERATE	NEGATIVE
2	2	12.00	MALE	LIGHT YELLOW	SLIGHTLY HAZY	NEGATIVE	TRACE	5.0	1.030	0-3	0-2	RARE	FEW	MODERATE	RARE	NEGATIVE
3	3	77.00	MALE	BROWN	CLOUDY	NEGATIVE	1+	6.0	1.020	5-8	LOADED	RARE	RARE	NONE SEEN	FEW	NEGATIVE
4	4	29.00	FEMALE	YELLOW	HAZY	NEGATIVE	TRACE	6.0	1.025	1-4	0-2	RARE	RARE	NONE SEEN	FEW	NEGATIVE
...
1431	1431	0.06	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.0	1.015	1-2	0-1	RARE	FEW	NONE SEEN	RARE	NEGATIVE
1432	1432	42.00	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.5	1.010	0-2	0-2	RARE	NONE SEEN	NONE SEEN	RARE	NEGATIVE
1433	1433	47.00	FEMALE	DARK YELLOW	CLEAR	NEGATIVE	TRACE	6.0	1.030	2-4	0-2	MODERATE	MODERATE	NONE SEEN	RARE	NEGATIVE
1434	1434	57.00	FEMALE	DARK YELLOW	CLEAR	NEGATIVE	TRACE	5.0	1.030	0-2	0-2	PLENTY	PLENTY	NONE SEEN	FEW	NEGATIVE
1435	1435	3.00	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.0	1.020	0-1	0-2	RARE	RARE	NONE SEEN	RARE	NEGATIVE

Urinalysis Test Results Dataset (uncleaned)

II. Python libraries used:

- **pandas**: Data manipulation library for data analysis.
- **sqlite3**: Module for interacting with SQLite databases.
- **matplotlib.pyplot, seaborn, vaeplot**: Data visualization libraries.
- **LabelEncoder, RobustScaler, chi2_contingency, tabulate**: Data preprocessing and statistical analysis utilities.
- **numpy**: Fundamental package for scientific computing.
- **imblearn.over_sampling.SMOTE, imblearn.under_sampling.RandomUnderSampler, imblearn.pipeline.Pipeline, Counter**: Libraries for handling imbalanced datasets and constructing pipelines.
- **train_test_split**: Utility for splitting datasets into training and testing sets.
- **LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, SVC, KNeighborsClassifier, GaussianNB**

LGBMClassifier, XGBClassifier, VotingClassifier, CatBoostClassifier: Machine learning and classification algorithms.

- **optuna:** Library for automated hyperparameter optimization.
- **accuracy_score, classification_report, confusion_matrix, roc_auc_score:** Evaluation metrics for classification models.
- **torch, nn, optim, DataLoader, TensorDataset:** Deep learning libraries and utilities.
- **tensorflow:** Framework for training neural networks

III. Data Inspection

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unnamed: 0             1436 non-null   int64
1   Age                    1436 non-null   float64
2   Gender                 1436 non-null   object
3   Color                  1435 non-null   object
4   Transparency           1436 non-null   object
5   Glucose                1436 non-null   object
6   Protein                1436 non-null   object
7   pH                    1436 non-null   float64
8   Specific Gravity       1436 non-null   float64
9   WBC                   1436 non-null   object
10  RBC                   1436 non-null   object
11  Epithelial Cells      1436 non-null   object
12  Mucous Threads        1436 non-null   object
13  Amorphous Urates      1436 non-null   object
14  Bacteria              1436 non-null   object
15  Diagnosis              1436 non-null   object
dtypes: float64(3), int64(1), object(12)
memory usage: 179.6+ KB
```

Dataset's information (uncleaned)

The columns of the dataset are:

- **Age** (The age of the patient) Note: Some patients are months old, so the age of these patients are preprocessed by dividing it by a hundred) e.g, 8 MONTHS OLD, $8/100 = 0.08$.
- **Gender** (The gender of the patient) Note: Either male or female.
- **Color** (urine color).
- **Transparency** (urine transparency).
- **Glucose** (glucose is a type of sugar, and its presence in the urine can be an important indicator of certain health conditions).
- **Protein** (the presence of protein in the urine is one of the parameters examined to assess kidney function and detect potential).
- **pH** (the pH level measures the acidity or alkalinity of urine).
- **Specific Gravity** (urine specific gravity is a measure of the concentration of particles in urine compared to water).
- **WBC** (White Blood Cells) Note: Also known as leukocytes, white blood cells are a crucial part of the immune system.

- **RBC** (Red Blood Cells) Note: RBC are responsible for carrying oxygen throughout the body.
- **Epithelial Cells** (epithelial cells are cells that line the surfaces and cavities of the body, including the urinary tract).
- **Mucous Threads** (mucous threads are strands of mucus that can be present in urine).
- **Amorphous Urates** (amorphous urates are non-crystalline formations in the urine that consist of uric acid).
- **Bacteria** (presence of bacteria in the urine).
- **Diagnosis** (UTI Diagnosis) Note: Either NEGATIVE or POSITIVE, **THIS IS THE TARGET COLUMN.**

Some remarks:

- The data has 1 missing values at the column **Color**. There are 1000+ records in this dataset so that missing value may be easily eliminated without altering the nature of the dataset. No duplicated values were found in this dataset.
- The column 'Unnamed: 0' acted as an index column in this dataset. However, because the dataset is very small, an index column is not required to speed up the query process. Some columns also include whitespace characters.
- According to the dataset description, the age of some patients are divided by a hundred if they are newborns (under 1 year old). This rule, however, in my opinion is not optimal since the scale differences between patients ≥ 1 year old and < 1 year old are not in sync (e.g, 1 year has 12 months, if the patient is 5 months old, the age value should be like 5/12, in the dataset the value for 5 months old is 0.05 which is 5/100 of a year??? This makes absolutely no sense at all).
- The 'WBC' and 'RBC' columns include data points with ranging values (1-3, 2-4,...), with over 50+ unique entries each. On top of that, these columns are classified as categorical columns, resulting in extremely large dimensions.

IV. Data Preprocessing

```
df.dropna(inplace=True)

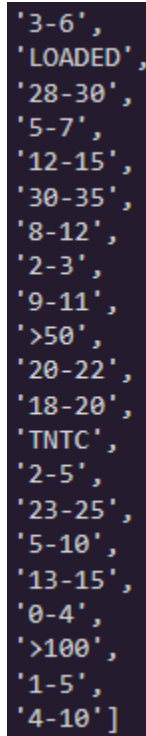
df.rename(columns=lambda x: x.replace(" ", "_"), inplace=True)

df = df.drop(columns=['Unnamed: 0'])
```

dropna(), rename() & drop()

The first step is to eliminate the rows that have the missing value in column Color, replace any whitespace characters with underscores if any column name has it, and remove the

'Unnamed: 0' column. The next stage is to transform the 'WBC' and 'RBC' columns. Let us look at some of the values in these two columns.

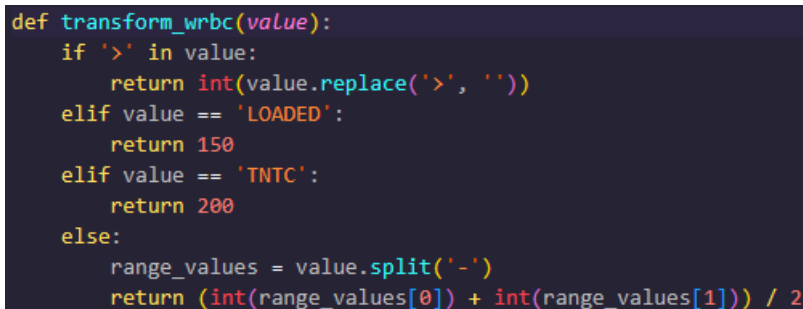


```
'3-6',  
'LOADED',  
'28-30',  
'5-7',  
'12-15',  
'30-35',  
'8-12',  
'2-3',  
'9-11',  
'>50',  
'20-22',  
'18-20',  
'TNTC',  
'2-5',  
'23-25',  
'5-10',  
'13-15',  
'0-4',  
'>100',  
'1-5',  
'4-10']
```

A snippet of matching values from 'WBC' & 'RBC' columns

The values of these two columns share the same meaning - ranging values represent the number of blood cells of patient. We will transform these values by calculating the middle point of each ranging value. There are some exceptional values:

- '>50' & '>100': more than 50 or 100 cells. For these values we will remove '>' character.
- 'LOADED' & 'TNTC': the two highest values in those two columns. To maintain the meaning of these values and the ordinality, we will replace 'LOADED' with 150 and 'TNTC' with 200.



```
def transform_wrbc(value):  
    if '>' in value:  
        return int(value.replace('>', ''))  
    elif value == 'LOADED':  
        return 150  
    elif value == 'TNTC':  
        return 200  
    else:  
        range_values = value.split('-')  
        return (int(range_values[0]) + int(range_values[1])) / 2
```

Function to transform 'WBC' & 'RBC'

The final thing to do is transform the 'Age' column. For this column we are just going to change any value lower than 1 by multiply it with 100 then divide by 12.

```
def transform_age(age):
    if age < 1:
        return age * 100 / 12
    else:
        return age
```

Function to transform 'Age'

That's should be it for the pre-processing part, below is a sneak peak of the new cleaned dataset.

Age	Gender	Color	Transparency	Glucose	Protein	pH	Specific_Gravity	WBC	RBC	Epithelial_Cells	Mucous_Threads	Amorphous_Urates	Bacteria	Diagnosis
76.0	FEMALE	LIGHT YELLOW	CLEAR	NEGATIVE	NEGATIVE	5.0	1.010	2.0	1.0	OCCASIONAL	RARE	NONE SEEN	OCCASIONAL	NEGATIVE
9.0	MALE	DARK YELLOW	SLIGHTLY HAZY	NEGATIVE	1+	5.0	1.030	2.0	1.0	RARE	FEW	FEW	MODERATE	NEGATIVE
12.0	MALE	LIGHT YELLOW	SLIGHTLY HAZY	NEGATIVE	TRACE	5.0	1.030	1.5	1.0	RARE	FEW	MODERATE	RARE	NEGATIVE
77.0	MALE	BROWN	CLOUDY	NEGATIVE	1+	6.0	1.020	6.5	150.0	RARE	RARE	NONE SEEN	FEW	NEGATIVE
29.0	FEMALE	YELLOW	HAZY	NEGATIVE	TRACE	6.0	1.025	2.5	1.0	RARE	RARE	NONE SEEN	FEW	NEGATIVE
...
0.5	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.0	1.015	1.5	0.5	RARE	FEW	NONE SEEN	RARE	NEGATIVE
42.0	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.5	1.010	1.0	1.0	RARE	NONE SEEN	NONE SEEN	RARE	NEGATIVE
47.0	FEMALE	DARK YELLOW	CLEAR	NEGATIVE	TRACE	6.0	1.030	3.0	1.0	MODERATE	MODERATE	NONE SEEN	RARE	NEGATIVE
57.0	FEMALE	DARK YELLOW	CLEAR	NEGATIVE	TRACE	5.0	1.030	1.0	1.0	PLENTY	PLENTY	NONE SEEN	FEW	NEGATIVE
3.0	MALE	YELLOW	CLEAR	NEGATIVE	NEGATIVE	6.0	1.020	0.5	1.0	RARE	RARE	NONE SEEN	RARE	NEGATIVE

Urinalysis Test Results Dataset (cleaned)

V. EDA

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1435 entries, 0 to 1434
Data columns (total 15 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Age                 1435 non-null   float64
1   Gender              1435 non-null   object
2   Color               1435 non-null   object
3   Transparency        1435 non-null   object
4   Glucose             1435 non-null   object
5   Protein             1435 non-null   object
6   pH                  1435 non-null   float64
7   Specific_Gravity    1435 non-null   float64
8   WBC                 1435 non-null   float64
9   RBC                 1435 non-null   float64
10  Epithelial_Cells    1435 non-null   object
11  Mucous_Threads      1435 non-null   object
12  Amorphous_Urates    1435 non-null   object
13  Bacteria            1435 non-null   object
14  Diagnosis           1435 non-null   object
dtypes: float64(5), object(10)
memory usage: 168.3+ KB
```

```
Number of unique values in Age: 99
Number of unique values in Gender: 2
Number of unique values in Color: 10
Number of unique values in Transparency: 5
Number of unique values in Glucose: 6
Number of unique values in Protein: 5
Number of unique values in pH: 6
Number of unique values in Specific_Gravity: 6
Number of unique values in WBC: 54
Number of unique values in RBC: 44
Number of unique values in Epithelial_Cells: 7
Number of unique values in Mucous_Threads: 6
Number of unique values in Amorphous_Urates: 6
Number of unique values in Bacteria: 6
Number of unique values in Diagnosis: 2
```

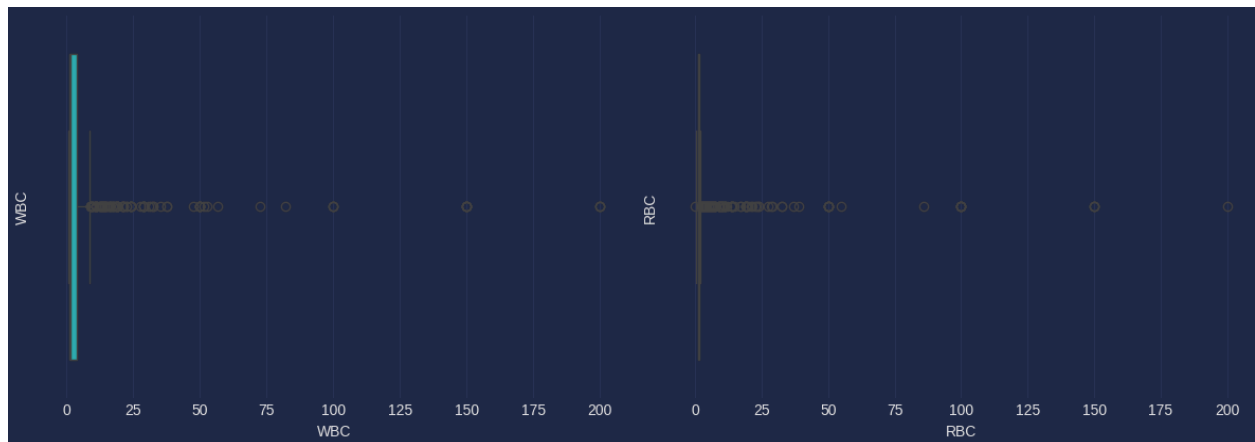
Dataset's information (cleaned)

This dataset comprises 1435 total rows and 15 columns; the first 14 columns are features, while the final column is the target column. Only 3 of the feature columns are numerical ('Age', 'WBC', and 'RBC'), with the rest being ordinal. Although Python classifies 'pH' & 'Specific_Gravity' as float64, each has just 6 different values. This implies they can also be termed ordinal columns.

	Age	pH	Specific_Gravity	WBC	RBC
count	1435.000000	1435.000000	1435.000000	1435.000000	1435.000000
mean	27.229268	6.052962	1.015847	7.411150	3.688502
std	23.449380	0.598682	0.007287	22.666183	13.850209
min	0.083333	5.000000	1.005000	0.500000	0.000000
25%	6.000000	6.000000	1.010000	1.000000	1.000000
50%	23.000000	6.000000	1.015000	1.500000	1.000000
75%	45.000000	6.500000	1.020000	4.000000	1.500000
max	92.000000	8.000000	1.030000	200.000000	200.000000

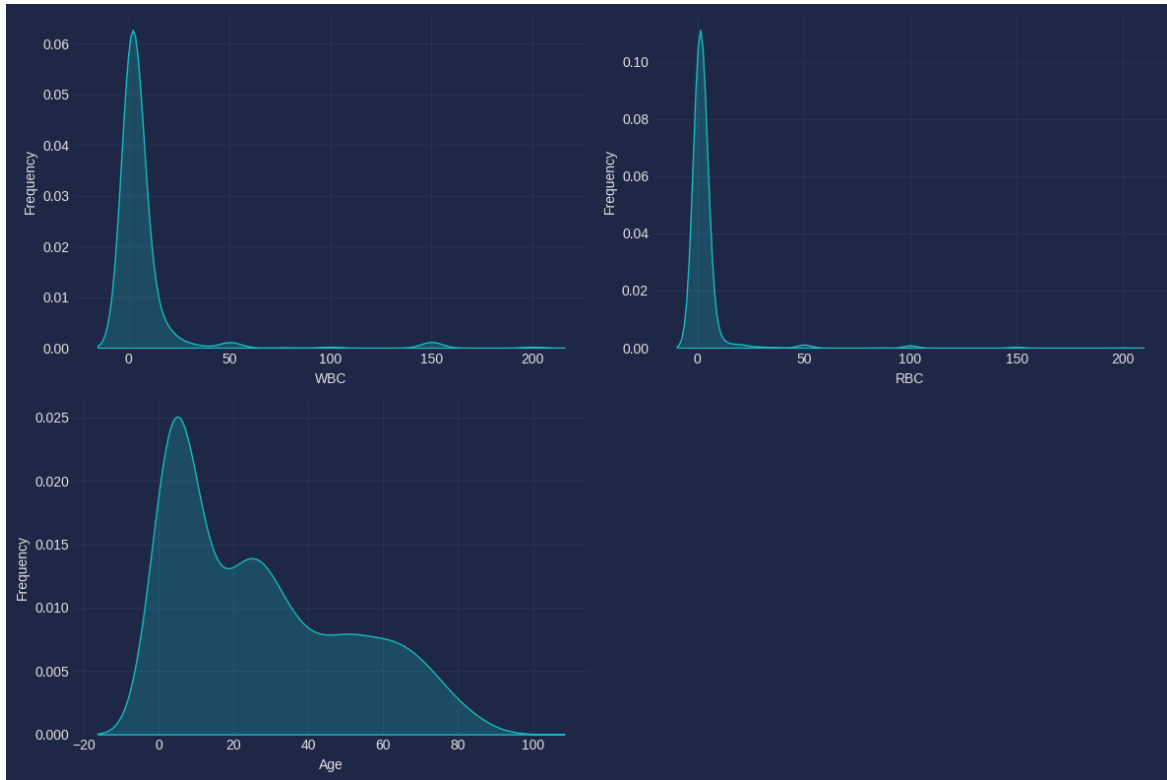
.describe()

We will omit 'pH' and 'Specific_Gravity' in this section because they are considered ordinal columns. The age range of the patients in the dataset varies widely, from 0.08 (which is 1/12 aka 1 month old) to 92 years old. But since the mean is 27.2, we can guess that the distribution of age is not symmetrical at all (we will see it more clearly with the kde plot in later part). The same thing may be stated of 'WBC' and 'RBC'. We can see the boxplot of these two columns below, there is definitely a handful of outliers in these columns. But we can't really decide if we are going to remove this or not. Because the impact of these values on the target column are not known yet, we are going to keep these columns intact.



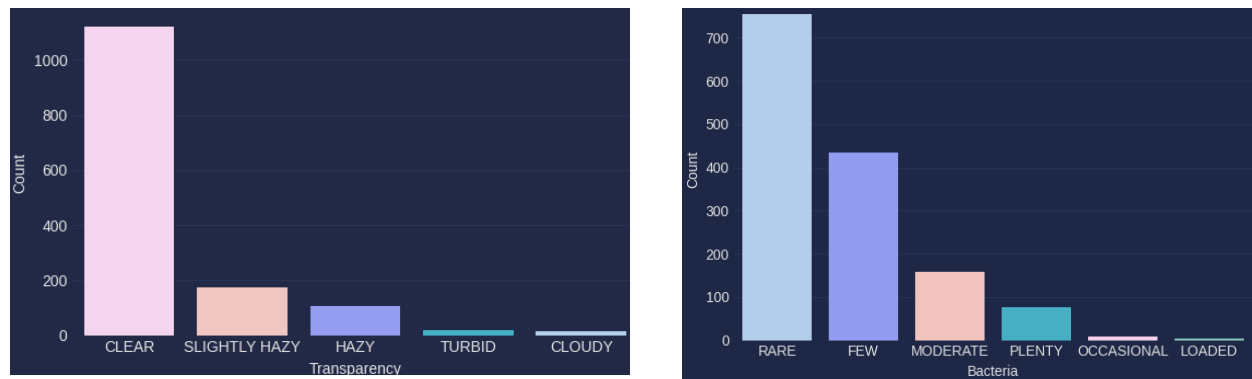
Boxplots of 'WBC' & 'RBC'

As expected in the preceding section, the distributions of the numerical columns are all not normal (in this case, all of them are right-skewed). The majority of patients have low WBC and RBC counts and are young in age.



KDE plots for 'WBC', 'RBC' & 'Age'

Several bar charts were visualized using seaborn and matplotlib. For the purpose of preventing this report from becoming too long, only some of the plots will be chosen. The chosen plots represent the most significant columns for modeling (as discovered later on when models were trained and features important were visualized).



Bar charts for 'Transparency' & 'Bacteria'

More than 70% of patients have clear urine transparency, and more than half contain rare presence of bacteria in the urine. When compared to other variables, these two columns have decent feature importance ratings, suggesting that patients with a positive diagnosis may have differing urine transparency and bacteria levels.

Noted: Please refer to the notebook file '3 - EDA', to see the visualizations for other columns of your desire.

For the purpose of showcasing my SQL skill, a SQLite database was created using the cleaned data and several queries have been made. Here are some of the most important one:

	Diagnosis	COUNT(*)
0	NEGATIVE	1354
1	POSITIVE	81

Diagnosis counts

	Transparency	COUNT(*)
0	CLEAR	29
1	SLIGHTLY HAZY	23
2	HAZY	17
3	TURBID	7
4	CLOUDY	5

Transparency counts (POSITIVE only)

	Bacteria	COUNT(*)
0	MODERATE	41
1	PLENTY	18
2	RARE	14
3	FEW	8

Bateria counts (POSITIVE only)

Only 81 individuals in this dataset (<6%) had a positive diagnosis. This shows that the dataset is heavily imbalanced. Based on the previous bar chart of Transparency distribution, there are more than 1000 patients with clear transparency, which is significantly greater than the other transparency classes. However, if only Positive cases are considered, the ratio of the 'Clear' class to the other groups becomes less extreme. This indicates that patients with positive diagnosis usually do not have clear urine transparency. In the Bacteria column, the majority of patients are classified as 'Rare'; however, if only positive patients are considered, the majority of patients are classified as 'Moderate'.

```
query = """SELECT AVG(WBC)
FROM UTI
WHERE Diagnosis = 'POSITIVE';"""
pd.read_sql(query, conn)
```

	AVG(WBC)
0	38.944444

Average WBC (Positive)

```
query = """SELECT AVG(WBC)
FROM UTI
WHERE Diagnosis = 'NEGATIVE';"""
pd.read_sql(query, conn)
```

	AVG(WBC)
0	5.524742

Average WBC (Negative)

Later on, WBC was discovered to be the most important column in different models. It can be seen that Positive patients have much higher WBC compare to Negative patients (38.9 vs 5.5) (*Note:* To see more queries performed, please refer to the notebook file '2 - SQL Queries').

Variable	Chi-square stat	P-value	Degrees of freedom
Color	18.6699	0.0281506	9
Transparency	113.476	1.31996e-23	4
Glucose	9.22428	0.100446	5
Protein	19.3505	0.000670615	4
pH	0.955408	0.966082	5
Specific_Gravity	6.89316	0.228708	5
Epithelial_Cells	26.1285	0.000210695	6
Mucous_Threads	3.38439	0.640949	5
Amorphous_Urates	15.1143	0.00988508	5
Bacteria	200.57	2.14538e-41	5
Gender	16.3232	5.34055e-05	1

χ^2 test between categorical variables vs the target variable.

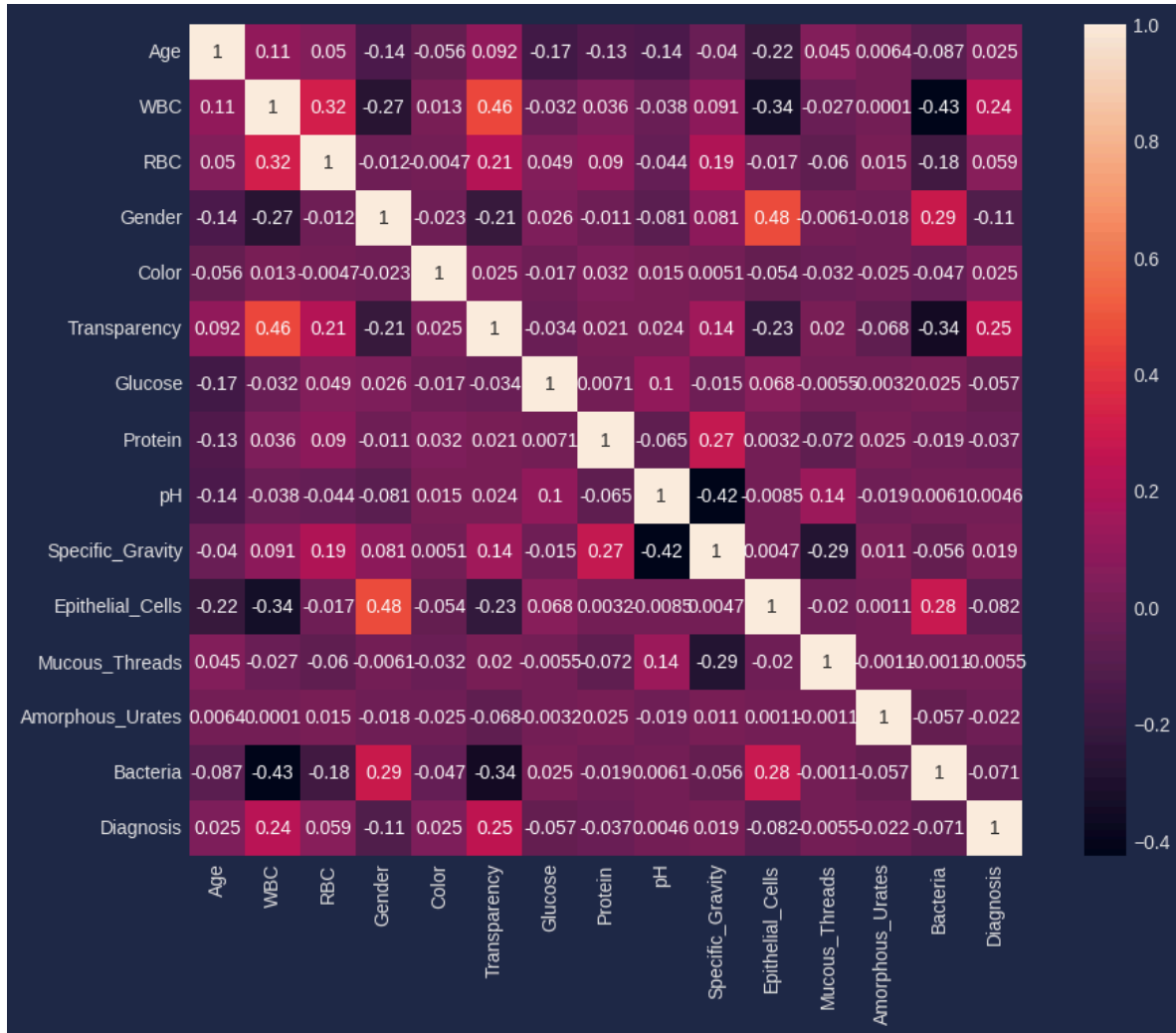
A chi-square tests of independence between each independent variable and the dependent variable was conducted. Based on the results, variables such as 'Transparency' and 'Bacteria' have extremely low p-values and high chi-square statistics, indicating a significant association with the dependent variable 'Diagnosis', as mentioned before. Conversely, "pH" and "Mucous_Threads" have high p-values but low chi-square statistics, indicating a lack of significant association.

Before continuing, the dataset is going to be transformed by encoding the categorical columns and scaling the numerical columns. Since some categorical columns are ordinal, the order for each columns will be specified first, then LabelEncoder() will be used to maintain the ordinality of the columns. For the numerical columns, RobustScaler() will be used to maintain the distribution of the data.

	Age	WBC	RBC	Gender	Color	Transparency	Glucose	Protein	pH	Specific_Gravity	Epithelial_Cells	Mucous_Threads	Amorphous_Urates	Bacteria	Diagnosis
0	1.358974	0.166667	0.0	0	4	0	4	4	0	1	4	6	3	4	0
1	-0.358974	0.166667	0.0	1	2	3	4	0	0	5	6	0	0	2	0
2	-0.282051	0.000000	0.0	1	4	3	4	5	0	5	6	0	2	6	0
3	1.384615	1.666667	298.0	1	1	1	4	0	1	3	6	6	3	0	0
4	0.153846	0.333333	0.0	0	9	2	4	5	1	4	6	6	3	0	0
...
1430	-0.576923	0.000000	-1.0	1	9	0	4	4	1	2	6	0	3	6	0
1431	0.487179	-0.166667	0.0	1	9	0	4	4	2	1	6	3	3	6	0
1432	0.615385	0.500000	0.0	0	2	0	4	5	1	5	2	2	3	6	0
1433	0.871795	-0.166667	0.0	0	2	0	4	5	0	5	5	5	3	0	0
1434	-0.512821	-0.333333	0.0	1	9	0	4	4	1	3	6	6	3	6	0

Transformed data

The next step is to compute the correlation matrix, which measures the strength and direction of the relationship between any two variables. Spearman method will be used since it assesses monotonic relationships between variables, and the dataset have both ordinal or continuous variables. For the target column 'Diagnosis', there are only 2 columns that have somewhat significant relationship with it: 'Transparency' (0.25) & 'WBC' (0.24).



Correlation matrix

VI. MODELING

Before model training, the data must be separated into train and test sets. However, because the target variable is severely unbalanced, several additional steps must be taken before building any ML models. One way of dealing with imbalanced dataset is resampling method. For this dataset, a combination of Synthetic Minority Over-sampling Technique (SMOTE) and Random Under-sampling were utilized to fix this problem. First, SMOTE will generate synthetic samples for the minority class until it reaches 10% of the majority class. Then, using random under-sampling, the number of samples in the majority class will be reduced until it is double of

that in the minority class. Of course the techniques will only be performed on train set to avoid leaking information from the test set into the training set.

```
Counter({0: 1089, 1: 59})
```

y_train before

```
Counter({0: 216, 1: 108})
```

y_train after

Below are the classification reports and confusion matrices for both the training and testing sets of Scikit-learn's logistic regression model using default parameters. Despite the fact that the model's performance on the test set is more accurate than on the train set, the precision, recall, and F1-score values of the minority class (class 1) in test set is low. Although resampling methods have been utilized, the model still struggles to correctly identify instances of the minority class.

```
Classification Report for Train Set:
      precision    recall  f1-score   support

     0       0.84      0.93      0.88       216
     1       0.82      0.65      0.73       108

 accuracy          0.84       324
 macro avg       0.83      0.79      0.80       324
 weighted avg    0.84      0.84      0.83       324

Confusion Matrix for Train Set:
[[201  15]
 [ 38  70]]

Classification Report for Test Set:
      precision    recall  f1-score   support

     0       0.96      0.92      0.94       265
     1       0.38      0.55      0.44        22

 accuracy          0.90       287
 macro avg       0.67      0.73      0.69       287
 weighted avg    0.92      0.90      0.90       287

Confusion Matrix for Test Set:
[[245  20]
 [ 10  12]]
```

LogisticRegression() using default parameter

To improve the performance of the model, Optuna - a hyperparameter optimization framework has been used to find optimal hyperparameter values. It will pick the best trial which is the one with the best accuracy for test set. For logistic regression, 3 parameters will be tuned: penalty, C and tol.

```

Classification Report for Train Set:
      precision    recall  f1-score   support

     0       0.82      0.95      0.88       216
     1       0.85      0.58      0.69       108

 accuracy          0.83       324
 macro avg          0.84      0.77      0.79       324
 weighted avg       0.83      0.83      0.82       324

Confusion Matrix for Train Set:
[[205  11]
 [ 45  63]]

Classification Report for Test Set:
      precision    recall  f1-score   support

     0       0.96      0.97      0.96       265
     1       0.58      0.50      0.54        22

 accuracy          0.93       287
 macro avg          0.77      0.73      0.75       287
 weighted avg       0.93      0.93      0.93       287

Confusion Matrix for Test Set:
[[257   8]
 [ 11  11]]

roc_auc_score(y_test, y_pred_test)

0.7349056603773585

```

LogisticRegression() using Optuna's best trial parameter

The model trained with Optuna's best parameters shows improvement in overall accuracy on both the training and testing sets compared to the model with default parameters. It also exhibits higher precision and recall values for class 1 (the minority class). However, the ROC AUC score is not as high as the accuracy, implying that this model may not be the best at differentiating between positive and negative classifications.

(Note: There are a total of 12 models deployed for this part. Since they all follow the same procedure with the above logistic regression model - train model with default parameter, use optuna to find best parameters values and train model with the best parameters found by optuna, I decide not to include every single one of them here to avoid making this report become too long. Only the best one will be discussed in the next part. To see all the models as well as their best parameters find by Optuna, please refer to the notebooks '4 - Modeling (Part 1)' & '5 - Modeling (Part 2)').

VII. MODELS EVALUATION

```

most_frequent_class = y.unique()[0]

baseline_predictions = np.full_like(y, most_frequent_class)

baseline_accuracy = accuracy_score(y, baseline_predictions)
print("Baseline Accuracy:", baseline_accuracy)

Baseline Accuracy: 0.9435540069686411

```

Baseline accuracy

The baseline accuracy is often computed by simply predicting the most frequent class for all instances in the dataset. In this case, it is just the percentage of patients who have been diagnosed as negative (the majority class). If a model performs worse than this baseline, it suggests that model is not learning anything meaningful from the data.

The following is a collection of performance metrics for several deployed model using this dataset after using Optuna to find the best parameters value:

	model	precision0	recall0	f1-score0	precision1	recall1	f1-score1	acc	auc-roc
0	Logistic Regression	0.96	0.97	0.96	0.58	0.50	0.54	0.93	0.73
1	Naïve Bayes	0.95	0.98	0.96	0.62	0.36	0.46	0.93	0.67
2	Decision Trees	0.97	0.95	0.96	0.52	0.68	0.59	0.93	0.81
3	Random Forests	0.97	0.97	0.97	0.65	0.68	0.67	0.95	0.83
4	Support Vector Machines	0.97	0.95	0.96	0.48	0.59	0.53	0.92	0.77
5	K-Nearest Neighbors	0.96	0.98	0.97	0.65	0.50	0.56	0.94	0.74
6	Gradient Boosting Machine	0.97	0.98	0.97	0.78	0.64	0.70	0.96	0.81
7	XGBoost	0.97	0.98	0.97	0.70	0.64	0.67	0.95	0.81
8	LGBM	0.97	0.97	0.97	0.67	0.64	0.65	0.95	0.80
9	CatBoost	0.97	0.98	0.98	0.74	0.64	0.68	0.95	0.81
10	MLP (Keras)	0.96	0.97	0.96	0.55	0.50	0.52	0.93	0.73
11	MLP (PyTorch)	0.96	0.95	0.96	0.48	0.55	0.51	0.92	0.75

Performance metrics of different models

Based on the collected metrics, it can be seen that gradient boosting-based models (e.g., Gradient Boosting, XGBoost, LGBM, CatBoost) generally outperform other algorithms in terms of overall accuracy and AUC-ROC. The worst models are Logistic Regression and Naïve Bayes with the lowest AUC-ROC score, indicates that these models does not distinguish well between classes. MLP models (both Keras and PyTorch implementations) show competitive performance but seem to have lower precision and recall compared to some of the tree-based and boosting algorithms. This might indicate that for this dataset, complex neural network architectures do not necessarily outperform simpler models.

Four best models have been selected for further analysis. Those models are:

- Random Forest from Scikit-learn (with accuracy of 0.95 and AUC-ROC score of 0.83).
- eXtreme Gradient Boosting a.k.a. XGBoost (with accuracy of 0.95 and AUC-ROC score of 0.81).
- Gradient Boosting from Scikit-learn (with accuracy of 0.96 and AUC-ROC score of 0.81).
- CatBoost from Yandex (with accuracy of 0.95 and AUC-ROC score of 0.81).

VIII. MODELS INTERPRETATION

```
RFmodel = RandomForestClassifier(  
    class_weight={0: 1, 1: 3},  
    random_state=42,  
    n_estimators=80,  
    max_depth=20,  
    min_samples_split=5,  
    min_samples_leaf=1,  
    max_features='sqrt')
```

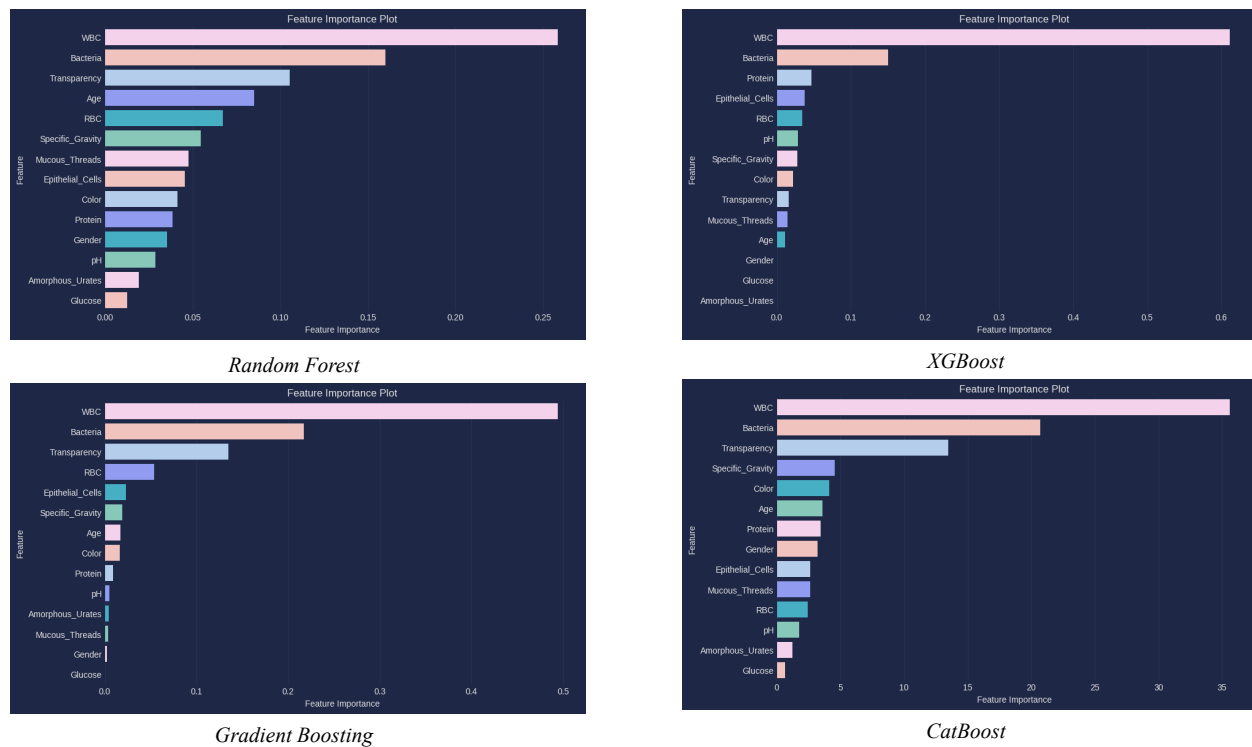
```
GBmodel = GradientBoostingClassifier(  
    n_estimators=59,  
    learning_rate=0.00942960365477529,  
    max_depth=4,  
    min_samples_split=13,  
    min_samples_leaf=4,  
    subsample=0.9662391695759919,  
    max_features=0.6489569238642884)
```

```
XGmodel = XGBClassifier(  
    objective='binary:logistic',  
    random_state=42,  
    reg_alpha=0.3069059937405213,  
    n_estimators=700,  
    max_depth=6,  
    learning_rate=0.0009902885160053942,  
    gamma=0.08145843854983292,  
    reg_lambda=1.6551035395467457,  
    alpha=1.499811272697868)
```

```
CBmodel = CatBoostClassifier(  
    iterations=42,  
    depth=6,  
    learning_rate=0.015592556206349632,  
    l2_leaf_reg=0.025419689599191234,  
    border_count=160,  
    random_state=42)
```

The parameters of the best models

The above shown the best parameter values from Optuna's best trial of the top 4 best models. Some models have one or two parameters defined manually to suit the job of binary classification. Random Forest has its 'class_weight' parameter set to '{0: 1, 1: 3}' to weight toward the minority class more. XGBoost have 'binary:logistic' for 'objective' since our goal is to build a model to do a binary classification task.



Feature Importance plots of the 4 top models

As mentioned in the previous part, the columns with the greatest influence on the model's predictions are 'WBC' & 'Bacteria', 'Transparency' was also somewhat important for all models except for XGBoost model. Random Forest and CatBoost take account of every features to determine the predictions, feature important scores between variables are not significantly different. Gradient Boosting and XGBoost, on the other hand, rely solely on some of the most important features, since the feature important score for 'WBC' is substantially higher than the others, some features do not even have any feature important score at all.

IX. VOTING ENSEMBLE

```

Classification Report for Train Set:
      precision    recall  f1-score   support

     0       0.87       0.99       0.93       216
     1       0.96       0.71       0.82       108

 accuracy          0.90       324
 macro avg       0.92       0.85       0.87       324
weighted avg       0.90       0.90       0.89       324

Confusion Matrix for Train Set:
[[213  3]
 [ 31 77]]

Classification Report for Test Set:
      precision    recall  f1-score   support

     0       0.97       0.98       0.98       265
     1       0.74       0.64       0.68        22

 accuracy          0.95       287
 macro avg       0.85       0.81       0.83       287
weighted avg       0.95       0.95       0.95       287

Confusion Matrix for Test Set:
[[260  5]
 [  8 14]]

roc_auc_score(y_test, y_pred_test)

0.8087478559176672

```

Hard Voting

```

Classification Report for Train Set:
      precision    recall  f1-score   support

     0       0.91       0.99       0.95       216
     1       0.97       0.81       0.88       108

 accuracy          0.93       324
 macro avg       0.94       0.90       0.92       324
weighted avg       0.93       0.93       0.93       324

Confusion Matrix for Train Set:
[[213  3]
 [ 20 88]]

Classification Report for Test Set:
      precision    recall  f1-score   support

     0       0.97       0.98       0.98       265
     1       0.74       0.64       0.68        22

 accuracy          0.95       287
 macro avg       0.85       0.81       0.83       287
weighted avg       0.95       0.95       0.95       287

Confusion Matrix for Test Set:
[[260  5]
 [  8 14]]

roc_auc_score(y_test, y_pred_test)

0.8087478559176672

```

Soft Voting

One way of improving models performance even further is by combining multiple machine learning models together to make predictions, which is voting ensemble technique. The above are the classification reports, confusion matrices and AUC-ROC score for the voting ensemble model combining the top 4 models. The differences between hard voting and soft voting are not significant, in this case. They have the same AUC-ROC score and performance on the test set. The only difference is the performance on train set, with soft voting performs a bit better than hard voting. But when compare both voting ensembles cases to Random Forest by itself, Random Forest performs a bit better (AUC-ROC score of Random Forest is 0.83 while both voting ensemble types are 0.81).

X. CONCLUSION:

- The dataset contains various attributes related to urine analysis. The target column for analysis is "Diagnosis," which indicates whether a urinary tract infection (UTI) is present or not.
- Some data preprocessing steps are necessary, including handling missing values in the "Color" column, and addressing the peculiar scaling of patient age, particularly for infants under 1 year old.

- Adjustments are needed for the categorical representation of the "WBC" and "RBC" columns, considering the wide range of values and the potential high dimensionality.
- The dataset contains 1435 rows and 15 columns, with 14 features and 1 target column. Most features are ordinal, and only 'Age', 'WBC', and 'RBC' are numerical.
- The age distribution ranges widely, from infants to elderly individuals, with a mean age of 27.2 years. 'WBC' and 'RBC' exhibit outliers, but their impact on the diagnosis is unclear until later on.
- The dataset is heavily imbalanced, with less than 6% positive diagnoses. Features like urine transparency and bacteria presence show significant associations with the diagnosis, while others like pH lack significance.
- Data transformation involves label encoding categorical columns and scaling numerical ones. Correlation analysis suggests 'Transparency' and 'WBC' are somewhat correlated with the diagnosis.
- Due to severe imbalance in the target variable, additional steps are needed before model training. SMOTE and Random Under-sampling techniques are employed on the training set to address the imbalance.
- Optuna's hyperparameter optimization is utilized to train the model with the best parameters. The optimized model demonstrates improved overall accuracy on both training and testing sets compared to default parameters.
- In all models, class 0 performs better than class 1 due to data imbalance.
- The baseline accuracy, computed by predicting the majority class (negative diagnosis) for all instances, is **94.36%**. Any model performing below this baseline suggests that it's not effectively learning from the data.
- Gradient boosting-based models (Gradient Boosting, XGBoost, LGBM, CatBoost) generally outperform other algorithms in terms of overall accuracy and AUC-ROC. These models are efficient in distinguishing between positive and negative classes.
- MLP models show competitive performance but have lower precision and recall compared to some tree-based and boosting algorithms, suggesting that for this dataset, complex neural network architectures might not necessarily outperform simpler models.
- Four models with high accuracy & AUC-ROC score are chosen for further analysis and combined for a new ensemble models. They are Random Forest, XGBoost, Gradient Boosting and CatBoost.
- The top models have something in common, they treated 'WBC' and 'Bacteria' as the most important features to make predictions, 'Transparency' was also somewhat important for all models except for XGBoost model.

- Voting Ensemble model of the top 4 performs with no significant improvement when compared with each model separately, moreover Random Forest even performed better than Voting Ensemble.