Introduction

Kidney Stone Prediction Classifier

This binary classification model predicts the likelihood of kidney stones in patients using numerical features such as osmolality and medical history. Trained on 414 entries and tested on 276, it's implemented in Python with Pandas and Scikit-learn.

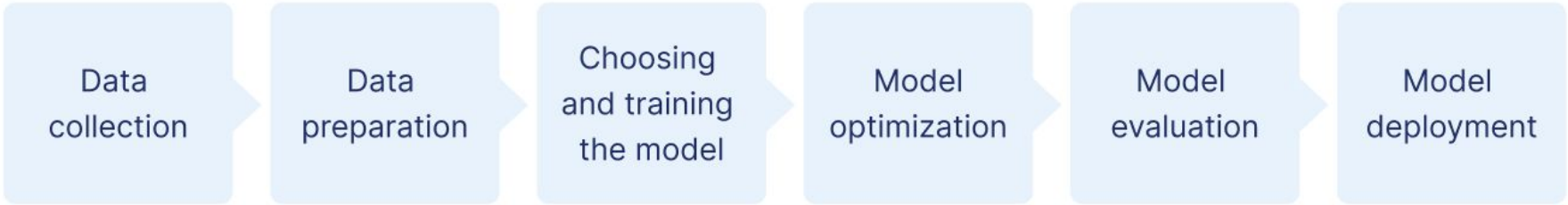
What is Binary Classification ?

Binary classification is the process of categorizing data into two distinct groups or classes.

Why It Is Needed ?

- ***Early Diagnosis:*** Identify kidney stone risk early.
- ***Personalized Treatment:*** Tailor treatment based on risk.
- ***Preventive Measures:*** Implement lifestyle changes for high risk.
- ***Resource Allocation:*** Optimize healthcare resources.
- ***Insights and Research:*** Drive nephrology research

Workflow Diagram



Raw Data Collection

Train Data

	id	gravity	ph	osmo	cond	urea	calc	target
0	0	1.013	6.19	443	14.8	124	1.45	0
1	1	1.025	5.40	703	23.6	394	4.18	0
2	2	1.009	6.13	371	24.5	159	9.04	0
3	3	1.021	4.91	442	20.8	398	6.63	1
4	4	1.021	5.53	874	17.8	385	2.21	1

```
train data
----------------------------------------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       414 non-null    int64
 1   gravity  414 non-null    float64
 2   ph       414 non-null    float64
 3   osmo     414 non-null    int64
 4   cond     414 non-null    float64
 5   urea     414 non-null    int64
 6   calc     414 non-null    float64
 7   target   414 non-null    int64
dtypes: float64(4), int64(4)
memory usage: 26.0 KB
```

- ***id:*** Unique identifier for each entry.
- ***gravity:*** Urine specific gravity.
- ***ph:*** Urine pH level.
- ***osmo:*** Urine osmolality.
- ***cond:*** Urine conductivity.
- ***urea:*** Urea level in urine.
- ***calc:*** Calcium concentration in urine.
- ***target:*** Binary indicator for ***kidney stone presence (1)*** or ***absence (0)***.

Raw Data Collection

Test Data

Sample data

	id	gravity	ph	osmo	cond	urea	calc
0	414	1.017	5.24	345	11.5	152	1.16
1	415	1.020	5.68	874	29.0	385	3.46
2	416	1.024	5.36	698	19.5	354	13.00
3	417	1.020	5.33	668	25.3	252	3.46
4	418	1.011	5.87	567	29.0	457	2.36

```
test data
---------------------------------------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 276 entries, 0 to 275
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       276 non-null    int64
 1   gravity  276 non-null    float64
 2   ph       276 non-null    float64
 3   osmo     276 non-null    int64
 4   cond     276 non-null    float64
 5   urea     276 non-null    int64
 6   calc     276 non-null    float64
dtypes: float64(4), int64(3)
memory usage: 15.2 KB
```

- ***"target"*** *column is not available , we have to predict for this data*

Raw Data Collection

Provided Predicted Sample Data for Test Data

```
sample.head()
```

	id	target
0	414	0.5
1	415	0.5
2	416	0.5
3	417	0.5
4	418	0.5

➢ ***We will use this for evaluating the model accuracy on Test Data***

Data Analysing

Statistical Analysis of Training Data

mean	206.500000	1.017894	5.955459	651.545894	21.437923	278.657005	4.114638	0.444444
std	119.655756	0.006675	0.642260	234.676567	7.514750	136.442249	3.217641	0.497505
min	0.000000	1.005000	4.760000	187.000000	5.100000	10.000000	0.170000	0.000000
25%	103.250000	1.012000	5.530000	455.250000	15.500000	170.000000	1.450000	0.000000
50%	206.500000	1.018000	5.740000	679.500000	22.200000	277.000000	3.130000	0.000000

Calculating the number of NaN values

```
train.isna().sum()
```

```
id          0
gravity     0
ph          0
osmo        0
cond        0
urea        0
calc        0
target      0
```

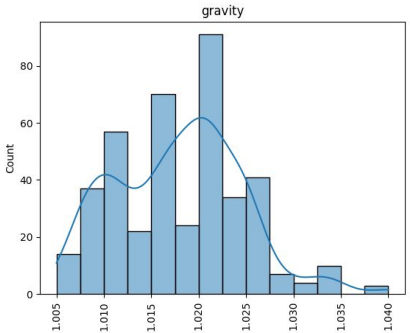
Calculating the number of Duplicated values

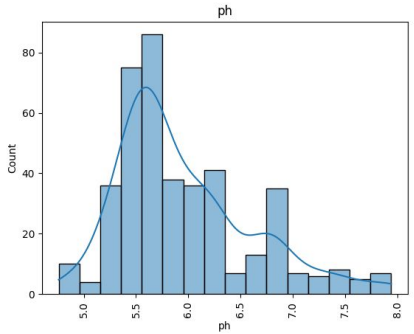
```
train.duplicated().sum()
```

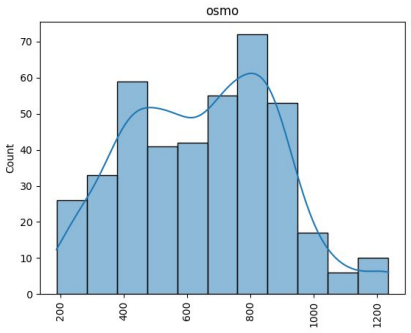
```
0
```

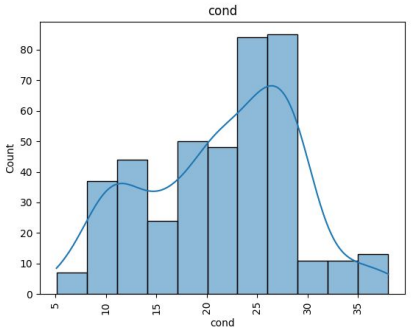
Data Analysing

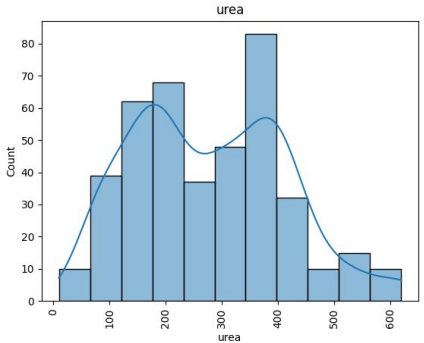
Analysis the Training data of columns using Charts

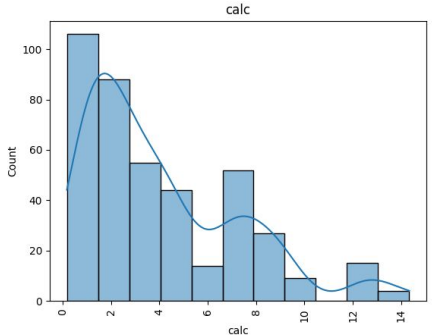


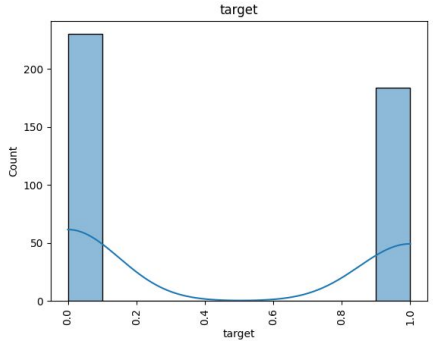






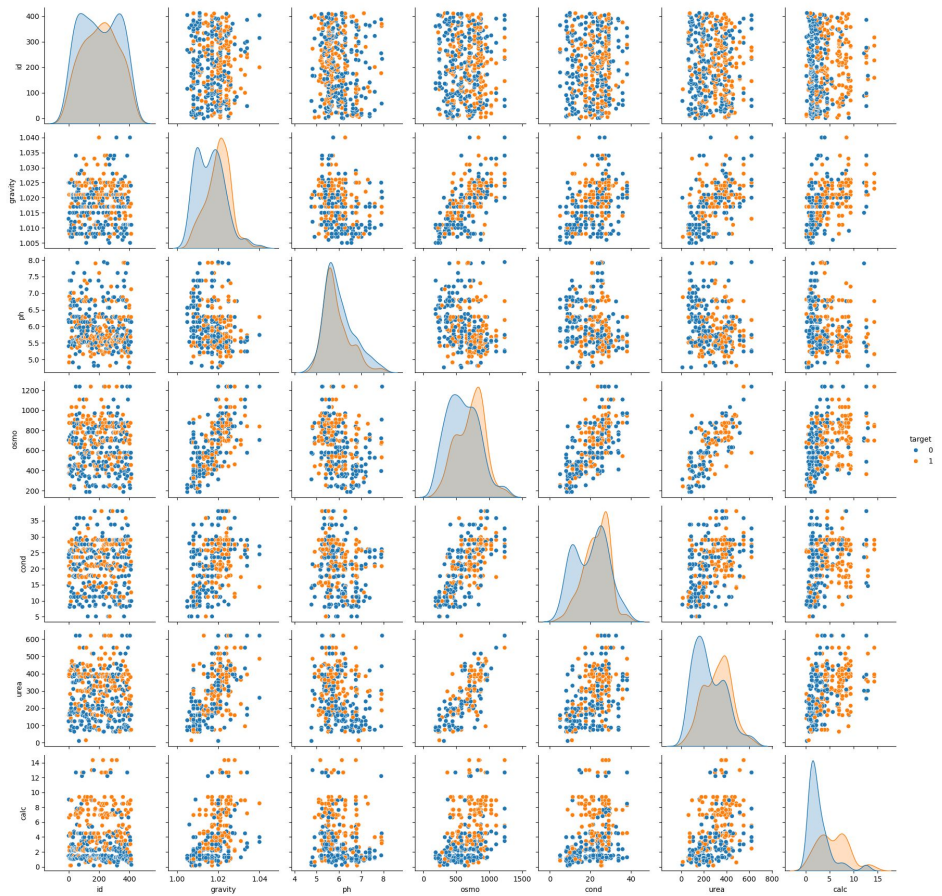






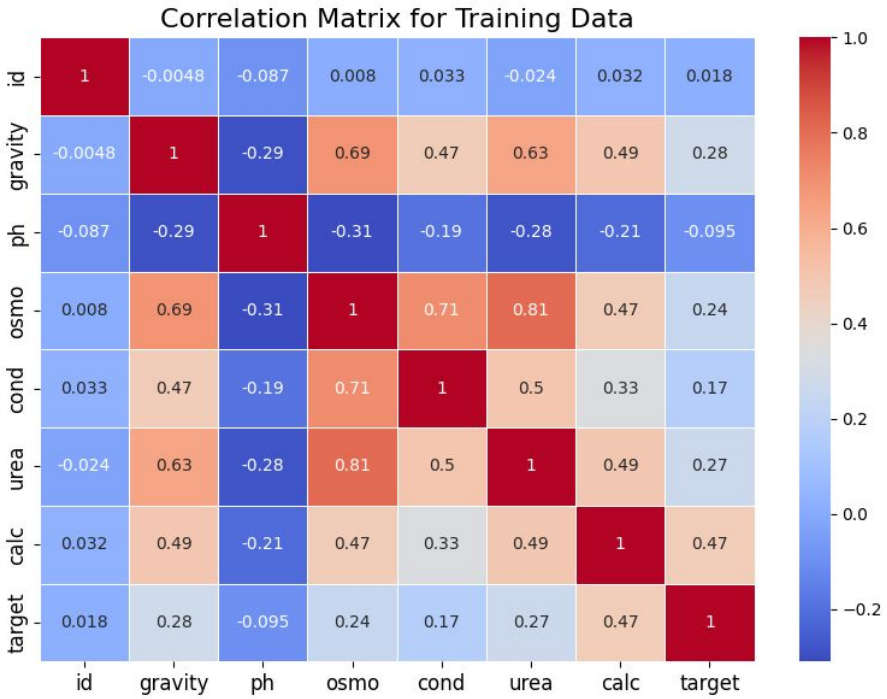
Data Analysing

Pair Plots for Various Parameters



Data Analysing

Correlation Heatmap for Selected Features

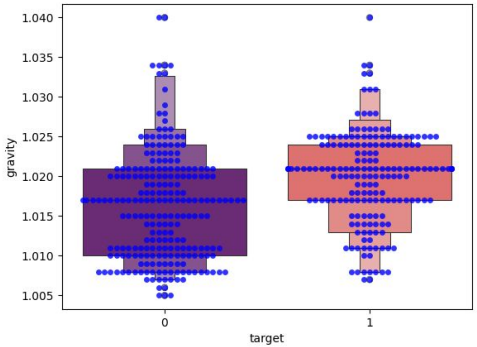


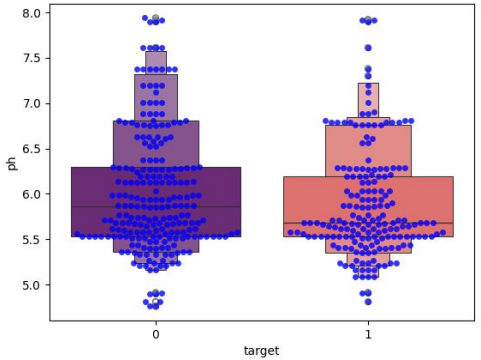

```
1 corr_matrix['target']
```

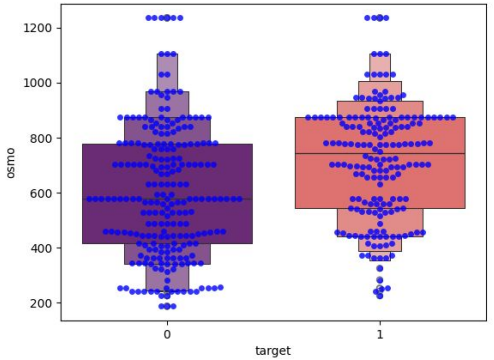
```
id          0.018222
gravity     0.282577
ph         -0.094983
osmo        0.244770
cond        0.172224
urea        0.265211
calc        0.467439
target      1.000000
Name: target, dtype: float64
```

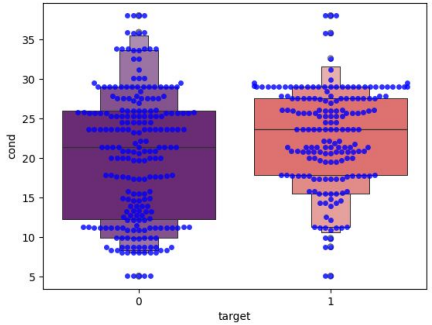
Data Analysing

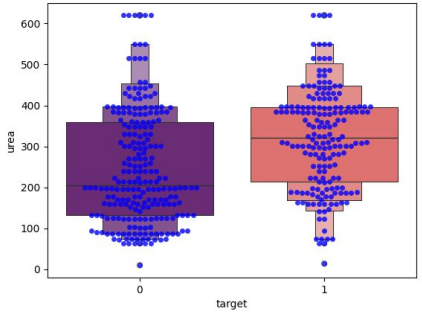
Box Plots for Various Parameters with respect to “target” Variable

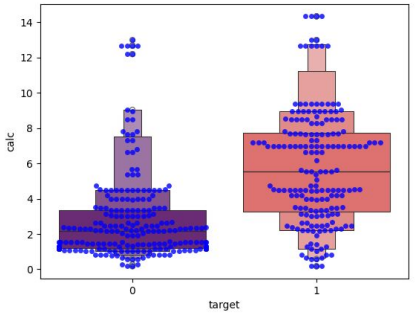












Data Preparation and Preprocessing

Split data into input (X) and output (y) variables

```
X = train.drop('target', axis=1)
y = train['target']
```

Data Preprocessing

```
numerical_features =['gravity', 'ph', 'osmo', 'cond', 'urea', 'calc']

# Create a transformer for the numerical features
numerical_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler())  # Scale the data
])

# Define the preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features)
    ])
```

Split data into training and testing sets

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

Define models and Fit & Evaluate on data to choose best model

- ***Logistic Regression:*** Linear model estimating the probability of a binary outcome.
- ***Gradient Boosting:*** Sequential ensemble method minimizing a loss function by adding weak learners.
- ***AdaBoost:*** Boosting ensemble technique that adjusts the weights of misclassified observations iteratively.
- ***Random Forest:*** Ensemble learning method constructing multiple decision trees and aggregating their outputs.
- ***XGBoost:*** Gradient boosting library employing a regularized objective function and tree pruning.
- ***SVM:*** Supervised learning algorithm seeking the optimal hyperplane to separate classes in feature space.
- ***K-Nearest Neighbors:*** Instance-based learning method classifying data points based on their nearest neighbors.
- ***Decision Tree:*** Hierarchical structure of decisions based on feature values to classify instances.
- ***Extra Trees:*** Ensemble learning technique similar to Random Forests, with more randomized splitting.
- ***Gaussian Naive Bayes:*** Probabilistic classifier assuming independence among features given the class label.
- ***CatBoost:*** Gradient boosting algorithm optimized for categorical features with ordered boosting.
- ***SGD Classifier:*** Linear classifier trained using stochastic gradient descent.
- ***Bagging Classifier:*** Bootstrap aggregating method training base classifiers on bootstrapped subsets of data.

Define models and Fit & Evaluate on data to choose best model

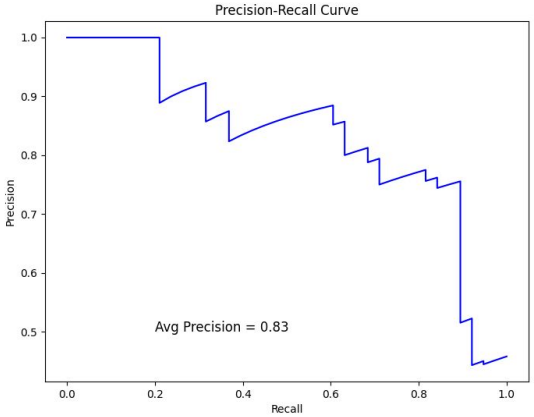
Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	ROC AUC Score (%)
Logistic Regression	78.31	81.25	68.42	74.29	77.54
Gradient Boosting	78.31	77.78	73.68	75.68	77.95
Gaussian Naive Bayes	78.31	76.32	76.32	76.32	78.16
XGBoost	74.7	74.29	68.42	71.23	74.21
CatBoost	74.7	71.79	73.68	72.73	74.62
AdaBoost	73.49	72.22	68.42	70.27	73.1
Random Forest	72.29	74.19	60.53	66.67	71.37
Extra Trees	72.29	72.73	63.16	67.61	71.58
Bagging Classifier	71.08	71.88	60.53	65.71	70.26
Decision Tree	67.47	67.74	55.26	60.87	66.52
SVM	62.65	60.0	55.26	57.53	62.08
K-Nearest Neighbors	56.63	54.17	34.21	41.94	54.88
SGD Classifier	49.4	47.5	100.0	64.41	53.33

Best Model

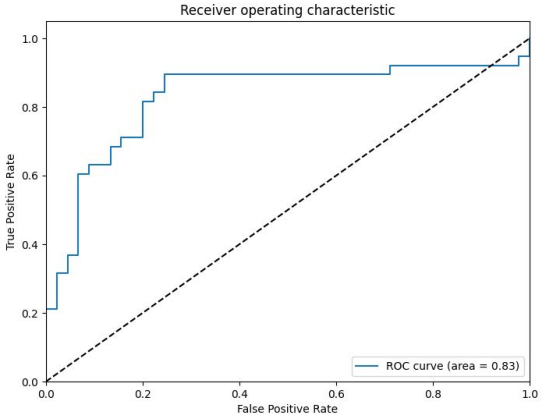
Logistic Regression

Confusion Matrix

Precision-Recall Curve



ROC Curve



Calculate accuracy on Test Data

```
y_test_true = sample['target']
accuracy = accuracy_score(y_test_true, y_test_pred)
print("Accuracy on test data:", accuracy*100,"%")
```

```
Accuracy on test data: 100.0 %
```

THANK YOU …