Neural Networks Final Project

Project 1: Predictions for Tabular Dataset Using Neural Network Model

1. Problem Definition

Task Identification

The objective of this project is to develop a predictive model for kidney disease based on patient data. Since the target variable indicates whether a patient has kidney disease or not, this is a **classification task**.

Input Data and Desired Output

- **Input Data:** The model will utilize patient features such as age, blood pressure, blood test results, and other relevant medical attributes. These features include both numeric and categorical data.
- **Desired Output:** The output is a binary label indicating the presence (1) or absence (0) of kidney disease for each patient.

Justification for Using a Neural Network

Neural networks are suitable for this problem due to their ability to:

- Learn complex, non-linear relationships between features and the target variable.
- Handle a mixture of numeric and categorical inputs after appropriate preprocessing.
- Generalize well to unseen data when properly trained, making them effective for medical diagnosis tasks.

2. Data Acquisition

Dataset Loading

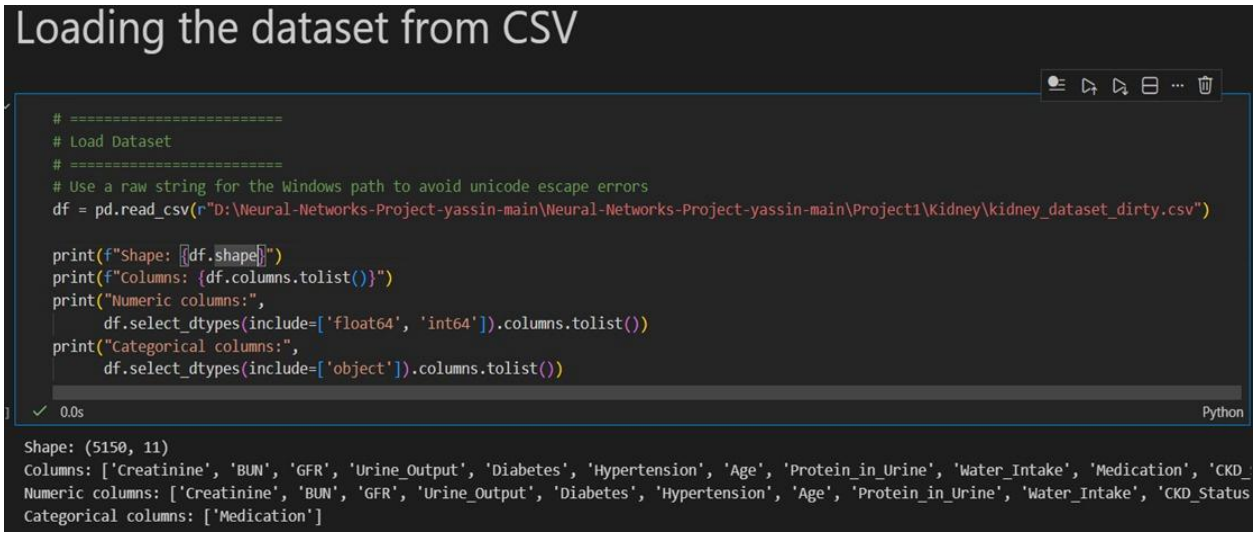
The kidney dataset was imported from a CSV file. A raw string was used to correctly handle the Windows file path and avoid potential escape-character issues.

Dataset Source

The dataset was obtained from Kaggle: Kidney Function Health Dataset

Preliminary Data Inspection

- **Dataset Dimensions:** The dataset's shape (number of rows and columns) was examined to understand its scale and complexity.
- **Feature Overview:** All column names were listed to identify the available attributes.
- **Data Types:** Features were categorized into numeric features (containing integer or floating-point values) and categorical features (containing non-numeric values requiring encoding).



3. Exploratory Data Analysis (EDA)

Dataset Dimensions:

- Rows: 5150
- Columns: 11

Basic Statistics

Statistical measures (mean, standard deviation, minimum, maximum) were calculated for each feature to understand their distributions and ranges.

Discribe Data Before Cleaning

```
df.describe()
```

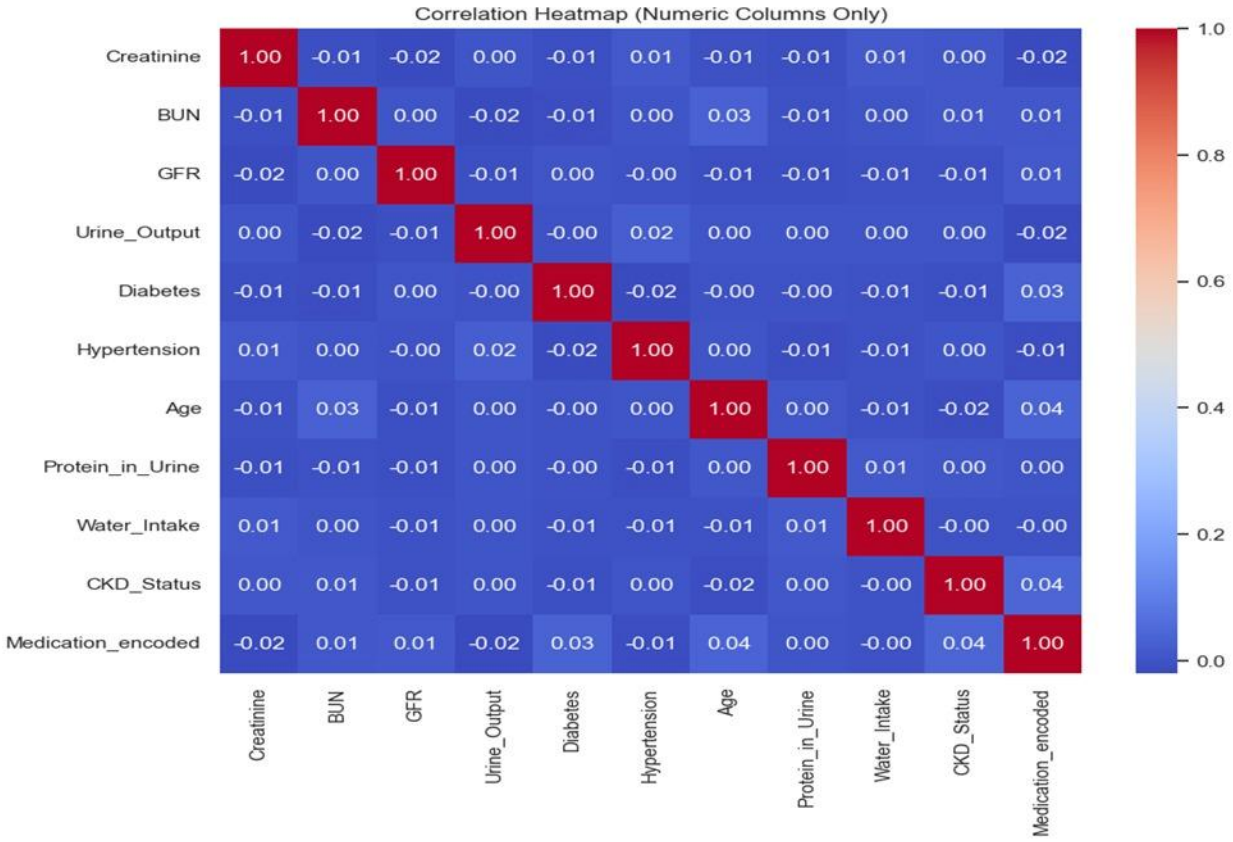
✓ 0.0s Python

	Creatinine	BUN	GFR	Urine_Output	Diabetes	Hypertension	Age	Protein_in_Urine	Water_Intake
count	4904.000000	4902.000000	4905.000000	4904.000000	4912.000000	4907.000000	4906.000000	4904.000000	4905.000000
mean	-52200.338271	-52602.724386	-51714.458200	-51032.853267	-51302.605863	-52169.958834	-52337.378291	-51484.385790	-52800.840329
std	222457.626006	223326.225930	221629.026614	223648.883098	220637.218540	222392.983515	222835.484031	222167.969450	223663.645911
min	-999999.000000	-999999.000000	-999999.000000	-999999.000000	-999999.000000	-999999.000000	-999999.000000	-999999.000000	-999999.000000
25%	0.768240	10.611929	40.034309	998.987827	0.000000	0.000000	38.107222	78.581190	1.614044
50%	0.990965	15.293598	88.947948	1780.127924	0.000000	0.000000	48.819922	116.061608	2.425191
75%	1.871044	34.395003	93.724429	2129.348915	1.000000	1.000000	59.252177	471.947577	3.220039
max	7.996428	119.931652	105.451432	2499.939696	1.000000	1.000000	90.000000	2997.724192	3.998043

Visualizations

Several visualizations were created to understand the data better:

- Feature distribution plots
- Correlation heatmap showing relationships between variables



Missing Values and Anomalies Detection

Missing values were counted for each column, and outliers were identified using statistical methods.

```
Final dataset shape: (5150, 12)

Diabetes distribution:
Diabetes
 0.0          3312
 1.0          1348
-999999.0      252
Name: count, dtype: int64

Hypertension distribution:
Hypertension
 0.0          2895
 1.0          1756
-999999.0      256
Name: count, dtype: int64

Target distribution:
CKD_Status
 0.0          3451
 1.0          1200
-999999.0      256
Name: count, dtype: int64
```

```
Outliers replaced with NaN:
Creatinine: 1416
BUN: 1224
GFR: 499
Urine_Output: 504
Age: 501
Protein_in_Urine: 1439
Water_Intake: 504
```

4. Data Cleaning

The data cleaning process involved several key steps:

- **Missing Values:** Continuous features were imputed with the median, while categorical features were imputed with the mode.
- **Outliers:** Detected using boxplots and the Interquartile Range (IQR) method, then extreme values were capped or replaced.

```
Final dataset shape: (4651, 11)

Diabetes distribution:
Diabetes
0.0    3432
1.0    1219
Name: count, dtype: int64

Hypertension distribution:
Hypertension
0.0    3045
1.0    1606
Name: count, dtype: int64

Target distribution:
CKD_Status
0.0    3451
1.0    1200
Name: count, dtype: int64
```

- **Duplicates:** Duplicate records were identified and removed.
- **Numeric Consistency:** All numeric features were converted to float type, and categorical features were converted to integer type.

5. Data Preprocessing

Encoding Categorical Variables

Label encoding was applied to transform categorical variables into numerical format:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Medication_encoded'] = le.fit_transform(df['Medication'])
```

Normalization

MinMax normalization (0-1 scaling) was applied to standardize the range of continuous features:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[continuous] = scaler.fit_transform(df[continuous])
```

	Creatinine	BUN	GFR	Urine_Output	Diabetes	Hypertension	Age	Protein_in_Urine	Water_Intake	CKD_Status
count	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000	4651.000000
mean	0.264794	0.263291	0.682610	0.606201	0.262094	0.345302	0.443381	0.200138	0.500560	0.258009
std	0.273766	0.272651	0.314628	0.281162	0.439821	0.475518	0.190694	0.257752	0.274750	0.437586
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.077650	0.075619	0.496665	0.339702	0.000000	0.000000	0.323121	0.036929	0.273014	0.000000
50%	0.134883	0.133380	0.847571	0.695250	0.000000	0.000000	0.443952	0.063806	0.503292	0.000000
75%	0.514239	0.510521	0.881529	0.819032	1.000000	1.000000	0.563441	0.405241	0.728628	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Train/Validation/Test Split

The dataset was split into three subsets with shuffling enabled:

```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['CKD_Status'])
y = df['CKD_Status']

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

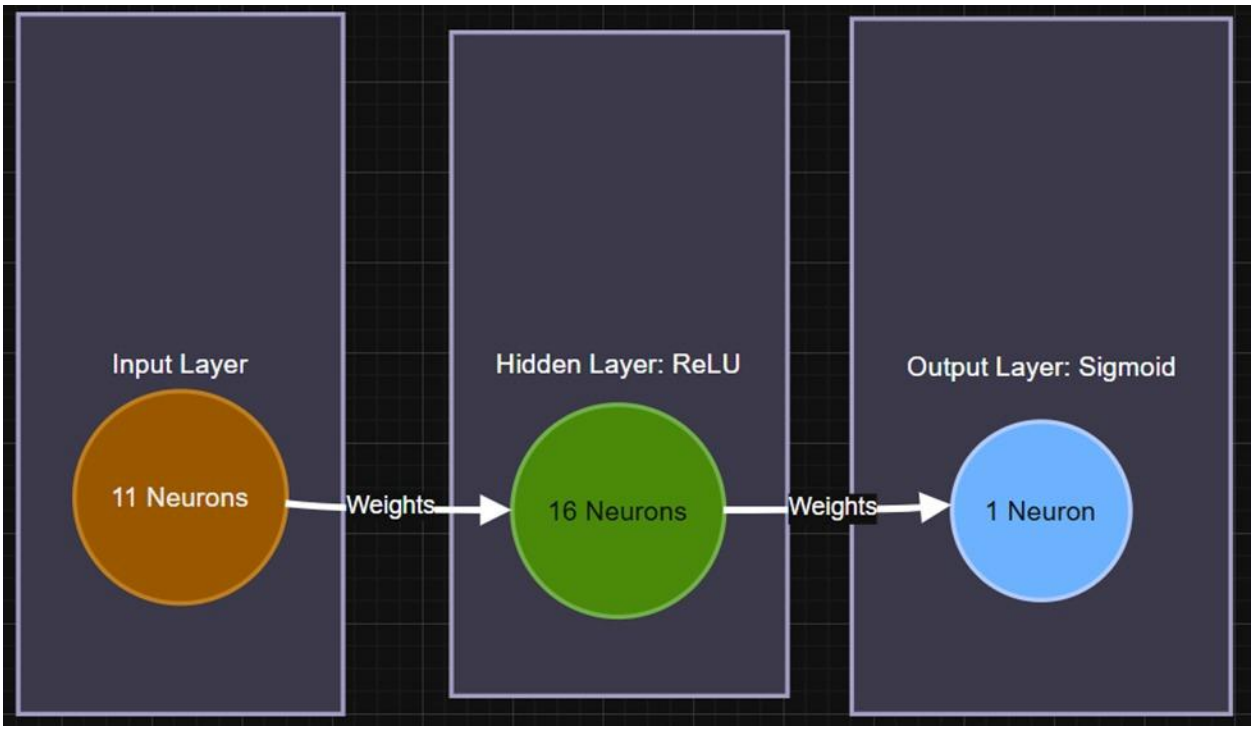
This results in a 60% training, 20% validation, and 20% test split.

6. Neural Network Architecture Design

Architecture Summary:

- **Input Layer:** 11 neurons (matching the number of features)
- **Hidden Layer:** 1 layer with 16 neurons using ReLU activation
- **Output Layer:** 1 neuron using Sigmoid activation

Architecture Flow: Input (11) → Hidden Layer (16, ReLU) → Output (1, Sigmoid)



7. Parameter Initialization

Weights and biases were initialized using random values from a normal distribution:

```
np.random.seed(42)
W1 = np.random.randn(hidden_size, input_size) * 0.01
b1 = np.zeros((hidden_size, 1))
W2 = np.random.randn(output_size, hidden_size) * 0.01
b2 = np.zeros((output_size, 1))
```

Note: Possible improvements include Xavier or He initialization methods.

8. Forward Propagation

The forward propagation process computes the network's predictions through the following steps:

```
Z1 = np.dot(W1, X) + b1
A1 = relu(Z1)
Z2 = np.dot(W2, A1) + b2
A2 = sigmoid(Z2)
cache = (X, Z1, A1, Z2, A2, W1, W2)
```

- **Linear Transformation:** $Z = W \cdot X + b$
- **Activation:** $A = f(Z)$
- Intermediate values are stored for use in backpropagation

9. Loss Function Computation

Binary cross-entropy loss with L2 regularization was used:

```
eps = 1e-8
bce = -(1/m) * np.sum(y*np.log(A2+eps) + (1-y)*np.log(1-A2+eps))
reg = (lambda_reg/(2*m))*(np.sum(W1**2)+np.sum(W2**2))
loss = bce + reg
```

- Binary cross-entropy was selected for the classification task
- L2 regularization was added to prevent overfitting

10. Backpropagation

Gradients were computed using the chain rule:

```
dZ2 = A2 - y
dW2 = (1/m)*np.dot(dZ2, A1.T) + (lambda_reg/m)*W2
db2 = (1/m)*np.sum(dZ2, axis=1, keepdims=True)

dA1 = np.dot(W2.T, dZ2)
dZ1 = dA1 * relu_derivative(Z1)
dW1 = (1/m)*np.dot(dZ1, X.T) + (lambda_reg/m)*W1
db1 = (1/m)*np.sum(dZ1, axis=1, keepdims=True)
```

Gradients were computed for both output and hidden layers using activation function derivatives.

11. Parameter Update

Gradient descent was applied to update weights and biases:

```
W1 -= learning_rate * dW1
b1 -= learning_rate * db1
W2 -= learning_rate * dW2
b2 -= learning_rate * db2
```

12. Training Loop

The training process consisted of iterating through epochs with the following steps:

1. Forward propagation
2. Loss computation
3. Backpropagation
4. Weight updates

Training Configuration:

- Epochs: 250-1000
- Dropout: keep_prob = 0.9
- Early Stopping: patience = 20 epochs based on validation loss

Training and validation losses were tracked throughout the process and visualized:

```
plt.plot(losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

13. Model Evaluation

The model was evaluated using multiple classification metrics:

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

y_pred = (A2_test > 0.5).astype(int).flatten()
accuracy_score(y_test, y_pred)
precision_score(y_test, y_pred)
recall_score(y_test, y_pred)
f1_score(y_test, y_pred)
confusion_matrix(y_test, y_pred)
```

Performance Results:

- Before regularization: Accuracy ≈ 0.992
- After regularization (L2 + Dropout): Accuracy ≈ 0.732

14. Model Testing

The trained model was applied to the test set to evaluate its performance on unseen data. Predicted labels were compared against true labels, and a slight accuracy drop was observed after regularization, indicating a successful reduction in overfitting.

15. Experiments and Hyperparameter Tuning

Various hyperparameters were systematically tested:

- **Hidden Neurons:** 8, 16, 32
- **Learning Rates:** 1e-3, 1e-4, 1e-5
- **Activation Functions:** ReLU, Tanh

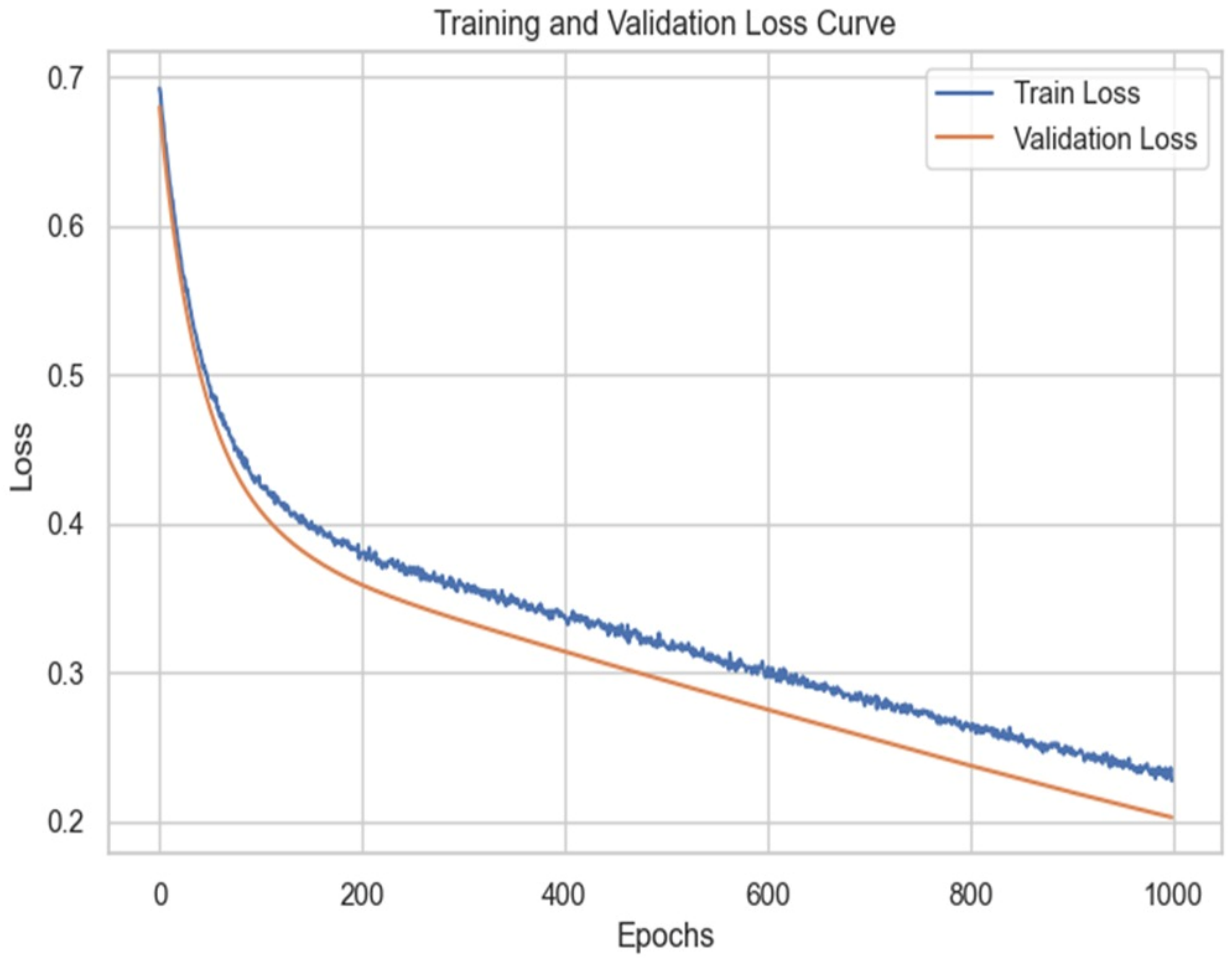
```
for h in hidden_sizes:
    for lr in learning_rates:
        for act in activations:
                val_acc, losses, val_losses = train_evaluate(...,
hidden_size=h, learning_rate=lr, activation=act)
```

```
Hidden=8, LR=0.001, Activation=relu, Val Acc=0.9925
Hidden=8, LR=0.001, Activation=tanh, Val Acc=0.9925
Hidden=8, LR=0.0001, Activation=relu, Val Acc=0.9914
Hidden=8, LR=0.0001, Activation=tanh, Val Acc=0.9914
Hidden=8, LR=1e-05, Activation=relu, Val Acc=0.7505
Hidden=8, LR=1e-05, Activation=tanh, Val Acc=0.7505
Hidden=16, LR=0.001, Activation=relu, Val Acc=0.9925
Hidden=16, LR=0.001, Activation=tanh, Val Acc=0.9925
Hidden=16, LR=0.0001, Activation=relu, Val Acc=0.9925
Hidden=16, LR=0.0001, Activation=tanh, Val Acc=0.9925
Hidden=16, LR=1e-05, Activation=relu, Val Acc=0.7538
Hidden=16, LR=1e-05, Activation=tanh, Val Acc=0.7538
Hidden=32, LR=0.001, Activation=relu, Val Acc=0.9925
Hidden=32, LR=0.001, Activation=tanh, Val Acc=0.9925
Hidden=32, LR=0.0001, Activation=relu, Val Acc=0.9925
Hidden=32, LR=0.0001, Activation=tanh, Val Acc=0.9925
Hidden=32, LR=1e-05, Activation=relu, Val Acc=0.7505
Hidden=32, LR=1e-05, Activation=tanh, Val Acc=0.7505
```

Results were plotted comparing training and validation loss curves as well as accuracy curves across different configurations.

16. Visualization and Reporting

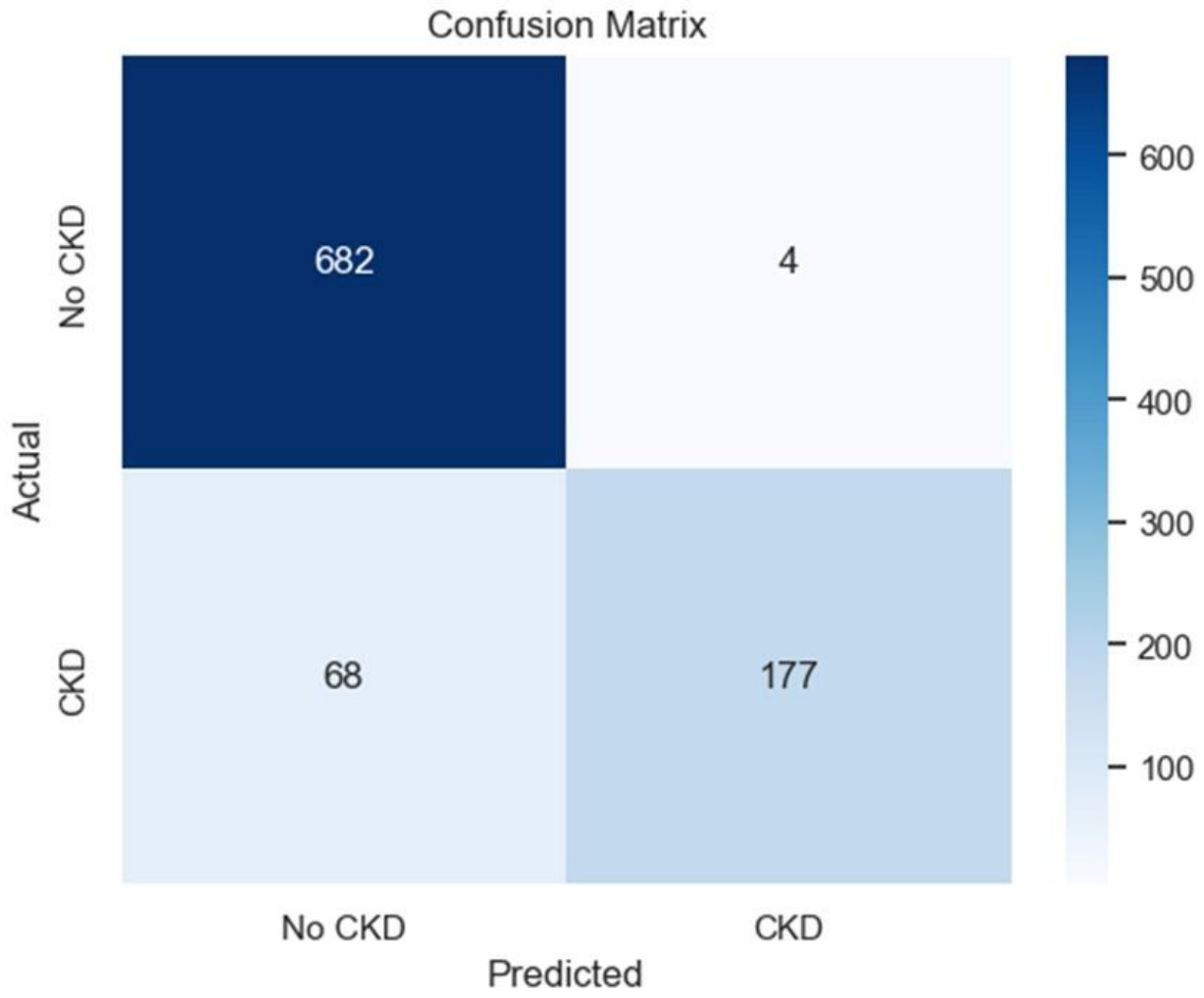
Training and Validation Loss Curves



Accuracy Curves

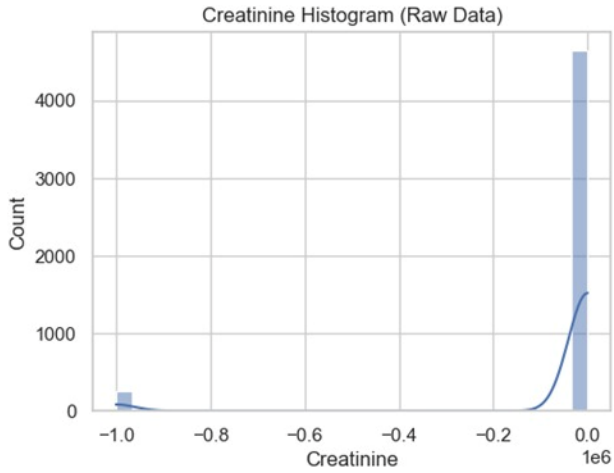
Similar plotting was performed for training and validation accuracy over epochs.

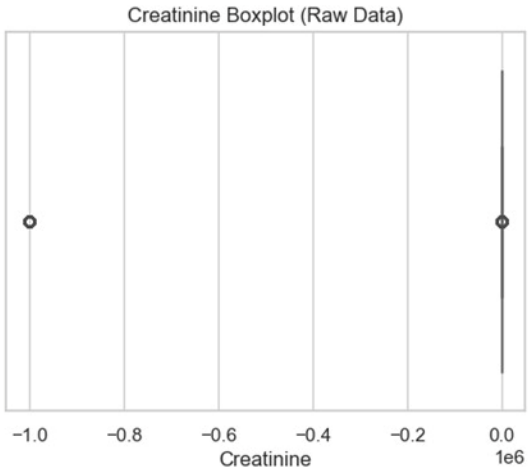
Confusion Matrix



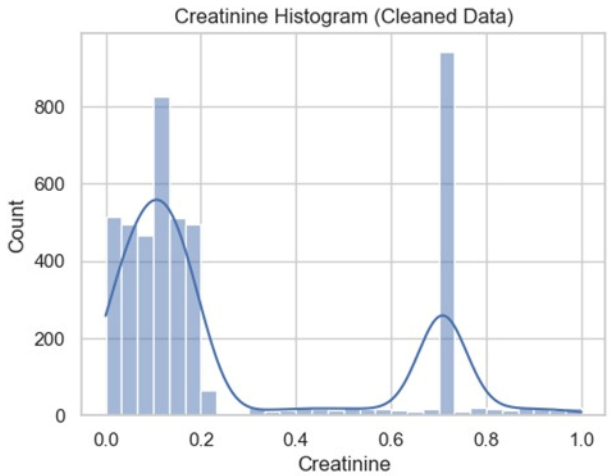
Feature Distributions

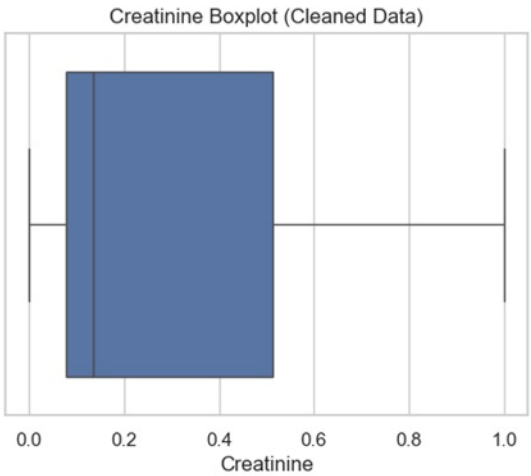
Before Cleaning





After Cleaning





Correlation Heatmap

Only numeric features and label-encoded categorical features were included in the correlation analysis.