Identifying Patients at a High-Risk of Chronic Kidney Disease

Group 5

Ramon Gonzalez and Alanis Perez

Master of Science in Applied Data Science, University of San Diego

ADS-502B: Applied Data Mining

August 11, 2025

Abstract

Chronic kidney disease (CKD) is a progressive medical condition characterised by kidney damage that may lead to more severe complications if not treated early. The purpose of this project is to classify patients as high- or low/moderate-risk of CKD using a synthetic dataset obtained from Kaggle created for machine learning research and practice. This clean dataset did not contain missing values or inconsistency, allowing for smooth data analysis and modeling. The two primary classification models applied are logistic regression, to estimate the probability of CKD, and random forest, to identify key predictors. These models were evaluated by looking at evaluation metrics (accuracy, precision, recall), ROC curves, and confusion matrices. Analyses of feature importance and coefficients were run to interpret results. An initial accuracy of 100% for random forest provoked a sanity check. Correlation analysis revealed strong correlations with GFR which was removed to make for a more realistic and challenging model. The results demonstrate the robustness of these models in identifying high-risk for CKD to pave the way for preventative healthcare interventions.

Keywords: Chronic kidney disease, high-risk classification, risk prediction, logistic regression, random forest, feature importance, predictive modeling, machine learning

Introduction

Chronic Kidney Disease (CKD) as defined by the National Kidney Foundation is a progressive disease of the kidneys that develop after damage which inhibits its ability to perform essential functions such as waste removal, maintaining blood pressure, and producing healthy red blood cells (2023). Early detection of kidney disease is imperative to avoid morbidity or mortality, as kidney malfunction can lead to severe complications such as heart disease and high blood pressure. The objective of this project is to develop a classification model that can identify patients at a high risk of CKD based on demographic and clinical data. Identifying a patient that may be high risk can allow for early intervention to reduce the need for painful and stressful procedures like dialysis or transplants. A classification model can help to enable healthcare professionals to extend the quality of care needed for a sick patient before the disease progresses.

The dataset that will be used in this project mimics real-world patient records based on relevant features that are commonly associated with kidney disease and function. To satisfy the objective, logistic regression and random forest will be implemented as primary models as they are well suited for interpretable and strong predictive capabilities. A binary logistic regression model estimates the probability of CKD by modeling the log of the odds for the outcome, reflecting the magnitude and direction (positive or negative) of each feature's influence (IBM, 2025). A random forest classification model will give insight into the predictive capabilities of each feature by determining feature importance, ranking features by their contribution to accuracy (IBM). Depending on classification results and accuracy measures, XGBoost analysis will be conducted to evaluate more complex patterns and relationships that the primary models may not capture.

Method

Data Source

The dataset that was used for this project was obtained from Kaggle and consisted of synthetic “but medically realistic” data created for the purpose of helping data scientists and healthcare professionals approach CKD through machine learning models (Miah, 2025). It consists of 2,304 rows of data and a total of 9 features with two target variables. The features include demographic data and clinical measurements of kidney function. The demographic data consists of age (numeric, measured in years), hypertension (categorical, labeled 1 for patients with high blood pressure, 0 otherwise), and diabetes (categorical, labeled 1 for patients with diabetes, 0 otherwise). The clinical data consists of glomerular filtration rate (numeric, measured in ml/min/1.73m^2), creatinine blood level (numeric, measured in mg/dL), blood urea nitrogen level (numeric, measured in mg/dL), and urine output (numeric, measured in ml of urine produced in a day). The two target variables included in this dataset are CKD status (1 for patients with CKD, 0 otherwise) and dialysis needed (1 for patients that will require dialysis based on CKD progression, 0 otherwise). The column for ‘dialysis needed’ was excluded for this project because the focus was to determine if a patient is considered high risk for CKD and dialysis is aimed at determining post-diagnosis data. Including this variable could have improperly and prematurely influenced the risk level of CKD.

Preprocessing

Preliminary exploratory data analysis was performed to give an overview of variable types and data structure using ‘.info()’ and ‘.describe()’ methods. Due to the cleanliness of the raw dataset, all variables were properly typed, no encoding was necessary, there were no duplicate entries, and there were no missing values, making preprocessing smooth and quick. The dataset was split into a traditional 80/20 split with a stratified split to preserve class balance given the small sample size of the data. During initial model training, however, the random forest classifier resulted in 100% accuracy on the test set, which raised some red flags. A sanity check was performed to check for suspected data leakage. A feature correlation analysis revealed that glomerular filtration rate (GFR) had a significantly higher correlation with CKD status than other variables at 0.60. Running a feature importance analysis confirmed that GFR dominated over the other predictors, making for a trivial classification task. Notably, GFR is one of the primary measurements taken in clinical settings to examine CKD and overall kidney function, so keeping it as a predictor variable leads to an unrealistic exploratory analysis project. For this reason, GFR was dropped from the feature set for modeling.

Model Training

Logistic regression (LR) was used as a baseline model to provide insights on linear relationships between the coefficients of features that measure the effect on the target variable. Random forest (RF) was used to investigate more complex and/or non-linear relationships and provided feature importance scores to identify which variables contributed the most to prediction accuracy. After removing GFR to reduce data leakage and create more realistic performances, both models demonstrated to be robust predictors as evaluated by strong performance metrics and high AUC scores. As an attempt to increase accuracy, an XGBoost analysis was performed. However, it was unsuccessful in outperforming the RF model, even with tuning and the inclusion of cross-validation, indicating that a higher-complexity model was not necessary. XGBoost may have been a more suitable model for a messier or more complex dataset, but given the cleanliness and synthetic nature of this data, it was not helpful. For this reason, further interpretations and the creation of predictions on a positive class and risk labels were used with LR and RF as primary modes.

Results

Model Evaluation

As shown in table 1, LR resulted in a weighted accuracy of predicting the presence of CKD at 0.62, precision of 0.63, and recall of 0.62. The ROC curve for LR is shown in figure 3, indicating a stronger performance than random chance but only at moderate strength of discriminating between class separation with an AUC score of 0.68. RF yielded higher accuracy and stronger performance also shown in table 1, with a weight accuracy of 0.71, precision of 0.77, and recall of 0.57. The ROC curve for RF is shown in figure 4, indicating a stronger performance than LR at a slightly higher strength of discriminating between classes with an AUC score of 0.75. The XGBoost model resulted in an accuracy of 0.70, just 0.1% shy of outperforming RF.

Confusion Matrices

Confusion matrices were created to assess predictive capability and determine the reliability of appropriate classifications. Figure 1 shows a confusion matrix for LR. This model showed strength in correctly predicting positive (146) and negative (142) classes, but missed the mark with a few false positives (84) and false negatives (89). Figure 2 shows a confusion matrix for RF. Unfortunately, some precision was lost as a higher number of false negatives (100) were miscalculated and true positives (135) were weaker. It did, however, excel as calculating true positives (186) and only accounted for a few false positives (40).

Feature Importance

Looking at the RF model, a feature importance barplot was created to visualize which features held the most weight in predicting CKD, shown in figure 6. Blood urea nitrogen level (BUN) was the most important with an importance score of ~0.34, followed by creatinine level at ~0.26, then urine output at ~0.20, and age at ~0.17. However, both hypertension (~0.02) and diabetes (~0.02) suddenly tapered off from the graph indicating little to no importance on determining CKD risk.

Coefficients and Log-Odds

Using the LR model, a coefficient plot gave an overview of the direction of influence by the feature set on the target variable, shown in figure 5. Blood urea nitrogen level (BUN) was unsurprisingly the most influential with a positive log-odd effect score of approximately 0.7. This was followed by creatinine level with a positive log-odd effect score of approximately 0.5. Urine output was the third most influential with a negative effect but at a much lower strength than expected at -0.1. Hypertension was also negatively associated with CKD at a small strength smaller than -0.1. Diabetes (< 0.1) and age (< 0.1) were both miniscule in comparison, indicating little to no influence in the direction of predicting CKD risk. BUN and creatinine level were the strongest influential predictors on determining the risk of CKD.

Classifications and Risk Labels

Using predictions and probabilities on the positive class (CKD = 1) for RF with a threshold of 0.7 allowed for the insertion of a CKD probability column with a probability score of disease on the testing set. This probability score was used to create a risk label for patients which labeled ‘high risk’ for those with a probability score above the threshold and ‘low/moderate risk’ for those below the threshold. The test set consisted of 461 patients (taken from the 20 split of the original set). This threshold determined that 112 out of the 461 patients were at high risk of CKD and 349 were at a low/moderate risk.

Discussion

The results of the preliminary models demonstrate that both were robust in predicting patients at a high risk of CKD, each giving insights into feature influences. Logistic regression offered interpretable feature coefficients, describing direction and strength of influence. Random forest provided an overview of non-linear relationships, feature interactions, and high predictive performance. There were some hiccups with initial testing due to the GFR variable which resulted in unrealistic model performance that does not appropriately reflect real-world clinical measurements and complexities. As a dominant predictor because of its direct correlation in measuring primary kidney function, it was removed to properly mirror a more realistic and challenging model.

Limitations

There are a few limitations of this study. To reiterate, this is a synthetic dataset that does not fully capture medical data, variability, or complexity of clinical measurements and demographic data. The exclusion of a very strong predictor (GFR) reduced model accuracy significantly. This dataset is also relatively small, and although measures were taken to balance sample size and classes, a larger dataset can provide greater understanding.

Next Steps

Replication studies can explore more features, including GFR, and test more complex models. Additionally, shifting the perspective to look at time-series models to evaluate CKD progression over time can advance intervention strategies in healthcare. In the future, these models could be applied to real datasets. Reintroducing a variable such as 'dialysis_needed' as found in the original dataset can guide research to develop models that can predict advanced stages of disease that can or will require more intense procedures.

References

IBM. (2025, May 14). *What Is Logistic Regression?* IBM. ibm.com/think/topics/logistic-regression

IBM. *What Is Random Forest?* IBM. ibm.com/think/topics/random-forest

Miah, A. (2025). *Kidney Disease Risk Dataset* (Version 1). [Data set]. Kaggle. kaggle.com/datasets/miadul/kidney-disease-risk-dataset

National Kidney Foundation Patient Education Team. (2023, September 11). *Chronic Kidney Disease (CKD)*. National Kidney Foundation. kidney.org/kidney-topics/chronic-kidney-disease-ckd

Table 1

Model Performance Summary

Model	**Weighted Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.62	0.63	0.62	0.63
Random Forest	0.71	0.77	0.57	0.66
XGBoost	0.70	-	-	-

Figure 1

Confusion Matrix for Logistic Regression

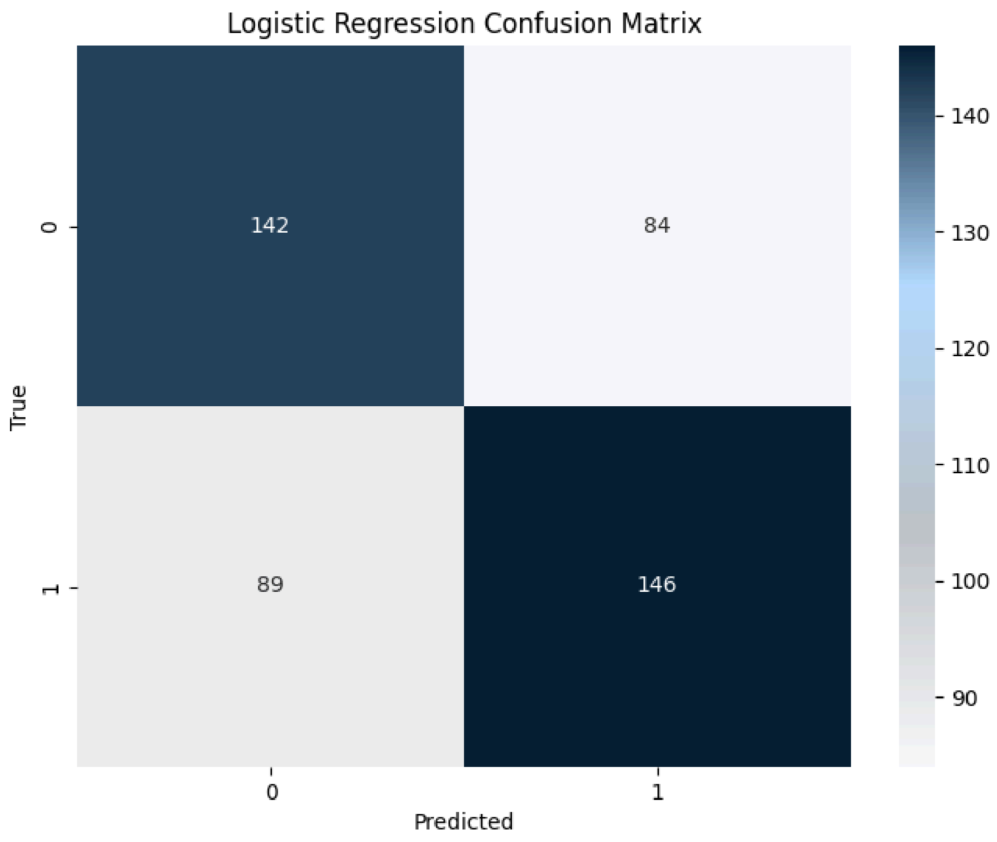


Figure 2

Confusion Matrix for Random Forest

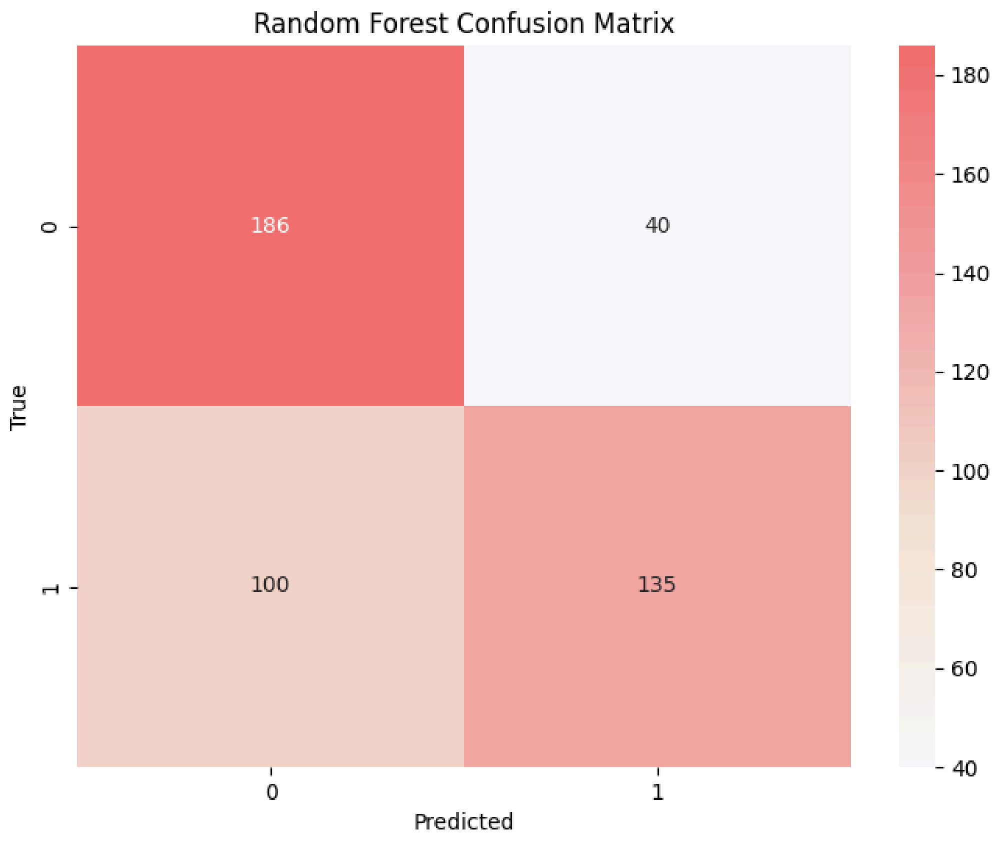


Figure 3

ROC Curve for Logistic Regression

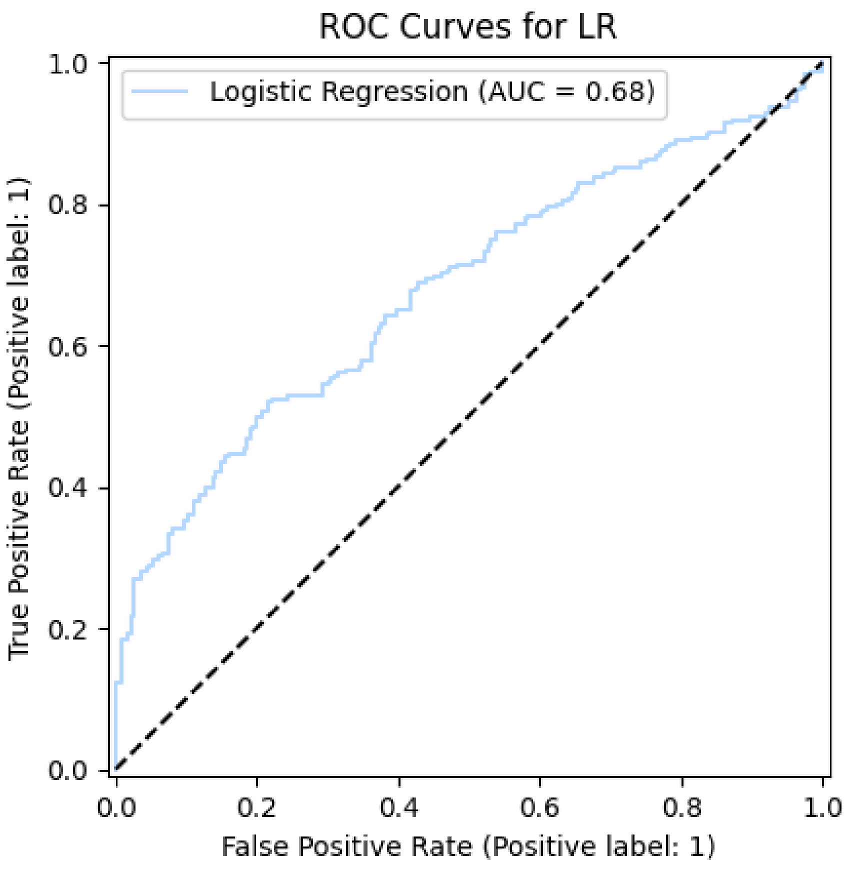


Figure 4

ROC Curve for Random Forest

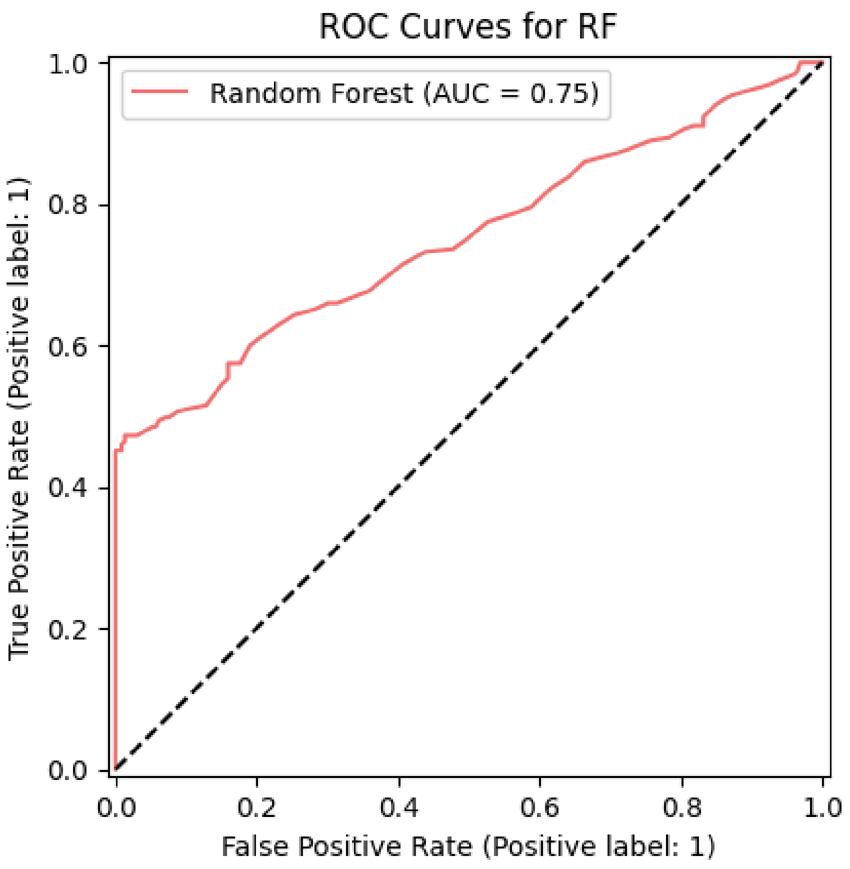


Figure 5

Coefficients of Logistic Regression

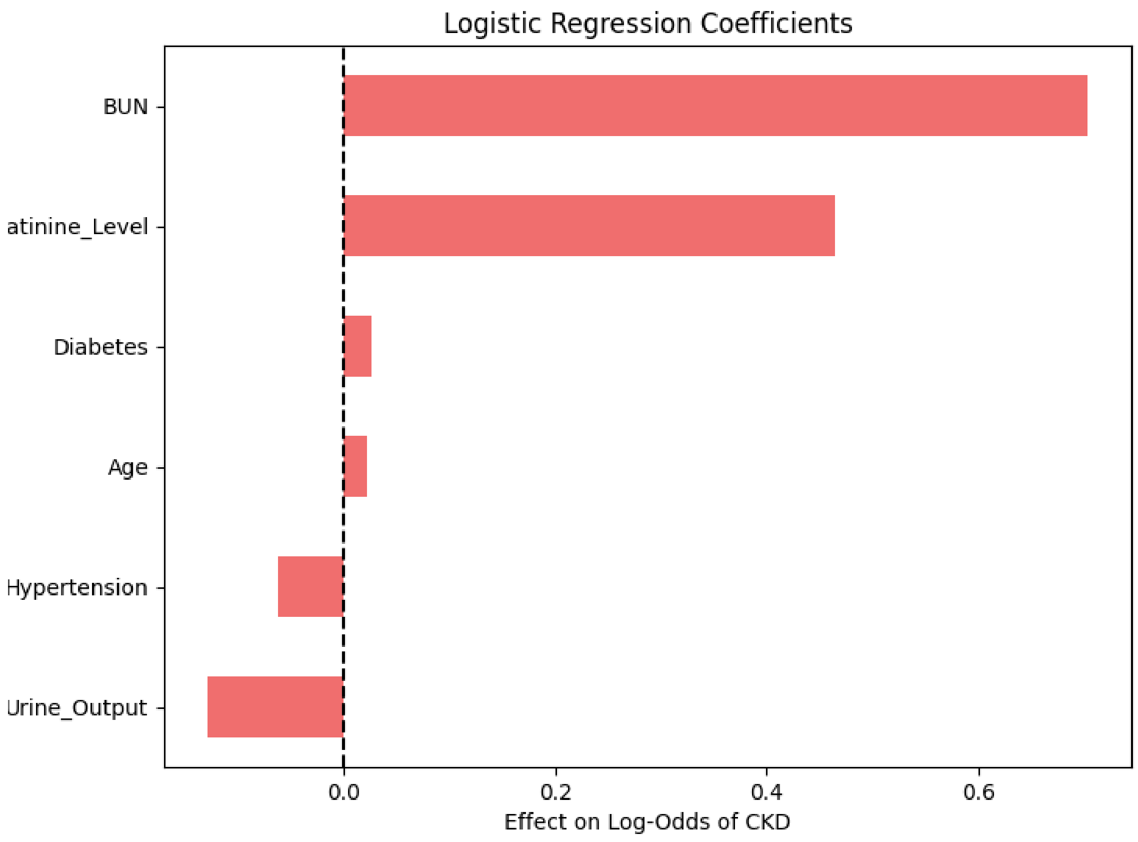
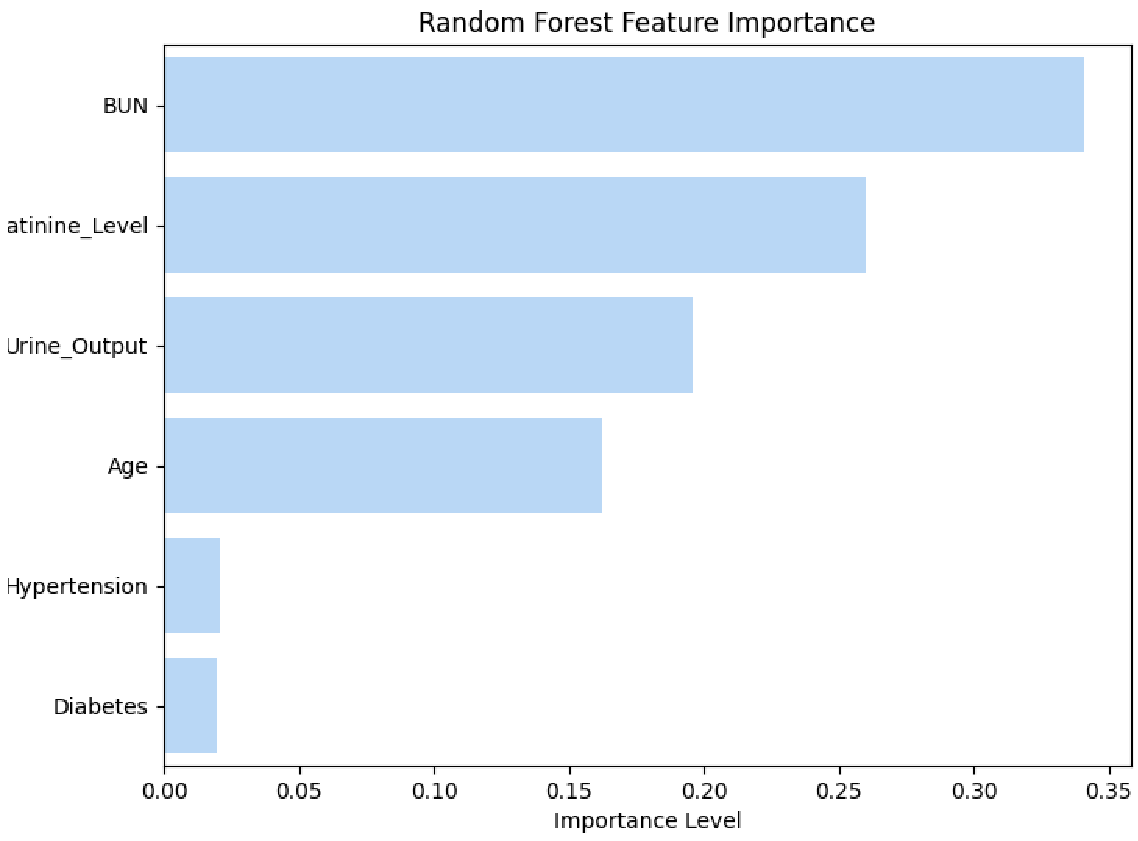


Figure 6

Feature Importance of Random Forest



500b_Final_notebook_RGonzalez,_APerez

August 11, 2025

```
[ ]: # Imports
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split, StratifiedKFold,␣
      ↪cross_val_score
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, confusion_matrix,␣
      ↪roc_auc_score, RocCurveDisplay, accuracy_score
     import matplotlib.pyplot as plt
     from matplotlib.colors import LinearSegmentedColormap
     import seaborn as sns
     from xgboost import XGBClassifier, plot_importance
```

Preprocessing * No null values * No encoding needed

```
[ ]: # Load data
     kidney_df = pd.read_csv("kidney_disease_dataset.csv")
     # Drop 'Dialysis_Needed' variable -- not used for this classification analysis
     kidney_df = kidney_df.drop('Dialysis_Needed', axis=1)
```

```
[ ]: kidney_df.head()
```

```
[ ]:    Age  Creatinine_Level   BUN  Diabetes  Hypertension   GFR  Urine_Output  \
     0   71              0.30  40.9         0             1  46.8        1622.0
     1   34              1.79  17.1         0             0  43.8        1428.0
     2   80              2.67  15.0         0             1  78.2        1015.0
     3   40              0.97  31.1         0             1  92.8        1276.0
     4   43              2.05  22.8         1             1  62.2        1154.0

        CKD_Status
     0           1
     1           1
     2           1
     3           1
     4           0
```

```
[ ]: # Start w/ basic EDA to give an overview of the data
     print(kidney_df.info())
     print(kidney_df.describe())
     print(kidney_df['CKD_Status'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2304 entries, 0 to 2303
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Age               2304 non-null   int64
 1   Creatinine_Level  2304 non-null   float64
 2   BUN               2304 non-null   float64
 3   Diabetes          2304 non-null   int64
 4   Hypertension      2304 non-null   int64
 5   GFR               2304 non-null   float64
 6   Urine_Output      2304 non-null   float64
 7   CKD_Status        2304 non-null   int64
dtypes: float64(4), int64(4)
memory usage: 144.1 KB
None
               Age  Creatinine_Level          BUN     Diabetes  Hypertension  \
count  2304.000000       2304.000000  2304.000000  2304.000000   2304.000000
mean     54.159288          1.305638    18.813672     0.406684      0.498264
std      20.513729          0.789594    10.508358     0.491322      0.500106
min      20.000000          0.300000     5.000000     0.000000      0.000000
25%      36.000000          0.620000     9.975000     0.000000      0.000000
50%      54.000000          1.240000    18.200000     0.000000      0.000000
75%      72.000000          1.842500    26.000000     1.000000      1.000000
max      90.000000          4.130000    61.900000     1.000000      1.000000

               GFR  Urine_Output   CKD_Status
count  2304.000000   2304.000000  2304.000000
mean     68.953863   1309.189670     0.508681
std      24.660191    491.951914     0.500033
min       5.000000    100.000000     0.000000
25%      51.300000    967.750000     0.000000
50%      69.150000   1295.500000     1.000000
75%      86.300000   1633.500000     1.000000
max     120.000000   2899.000000     1.000000
CKD_Status
1    1172
0    1132
Name: count, dtype: int64
```

Train-test split * Traditional 80/20 split * Stratified K-fold to balance small sample size

[ ]:
```
# Define features
X = kidney_df.drop('CKD_Status', axis=1)
y = kidney_df['CKD_Status']
```

[ ]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

[ ]:
```
# Scaling (for LR only)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Models * Logistic Regression (scaling) * Random Forests (not scaled)

[ ]:
```
# Logisitc Regresion model (LR)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_scaled, y_train)
```

[ ]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
# Random Forest model (RF)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(random_state=42)
```

Model Evaluation

[ ]:
```
# Logistic Regression
y_pred_lr = lr.predict(X_test_scaled)
print("Logistic Regression:")
print(classification_report(y_test, y_pred_lr))
print("ROC AUC:", roc_auc_score(y_test, lr.predict_proba(X_test_scaled)[:, 1]))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       226
           1       0.85      0.83      0.84       235

    accuracy                           0.84       461
   macro avg       0.84      0.84      0.84       461
weighted avg       0.84      0.84      0.84       461

ROC AUC: 0.9091696479005837
```

```
[ ]: # Random Forest
     y_pred_rf = rf.predict(X_test)
     print("Random Forest:")
     print(classification_report(y_test, y_pred_rf))
     print("ROC AUC:", roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1]))
```

```
Random Forest:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       226
           1       1.00      1.00      1.00       235

    accuracy                           1.00       461
   macro avg       1.00      1.00      1.00       461
weighted avg       1.00      1.00      1.00       461

ROC AUC: 0.9999999999999999
```

SANITY CHECK - why does RF result in 100% accuracy?

```
[ ]: # Create function to test accuracy of both LR and RF
     def test_models(X_subset, y): # restate test/train split
         X_train, X_test, y_train, y_test = train_test_split(
             X_subset, y, test_size=0.2, random_state=42, stratify=y
         )

         lr = LogisticRegression(max_iter=1000)
         lr.fit(X_train, y_train)
         lr_acc = lr.score(X_test, y_test) # no scaling (see if separation is linear)

         rf = RandomForestClassifier(random_state=42)
         rf.fit(X_train, y_train)
         rf_acc = rf.score(X_test, y_test)

         return rf_acc, lr_acc
```

```
[ ]: # Run correlation for each variable with CKD_Status
     correlations_ckd = kidney_df.corr(numeric_only=True)['CKD_Status'].abs().
      ↪sort_values(ascending=False)
     print(f"Feature correlations with CKD_Status \n {correlations_ckd}")
```

```
Feature correlations with CKD_Status
 CKD_Status          1.000000
GFR                 0.601672
BUN                 0.305803
Creatinine_Level    0.185017
Urine_Output        0.041600
Hypertension        0.027722
Diabetes            0.018322
```

```
Age                   0.012416
Name: CKD_Status, dtype: float64
```

[ ]:
```
# Determine how dropping features changes accuracy
features_sorted = correlations_ckd.index[1:]  # skip CKD_Status of course
X_current = X.copy() # make copy -- ensure cleanliness!

for variable in range(len(features_sorted)):
    rf_acc, lr_acc = test_models(X_current, y)
    print(f"Dropping top {variable} features: RF = {rf_acc:.4f}, LR = {lr_acc:.
 ↪4f}")

    # Drop the next most correlated feature
    if variable < len(features_sorted):
        X_current = X_current.drop(columns=[features_sorted[variable]])
```

```
Dropping top 0 features: RF = 1.0000, LR = 0.8330
Dropping top 1 features: RF = 0.6963, LR = 0.6247
Dropping top 2 features: RF = 0.5597, LR = 0.5271
Dropping top 3 features: RF = 0.4967, LR = 0.4902
Dropping top 4 features: RF = 0.5228, LR = 0.5033
Dropping top 5 features: RF = 0.5336, LR = 0.5054
Dropping top 6 features: RF = 0.5076, LR = 0.5119
```

RF accuracy drops to 0.6963 from 1.0 by dropping GFR. Retrain both models without GFR.

[ ]:
```
# Define NEW features
kidney_df_new = kidney_df.drop('GFR', axis=1)

X_new = kidney_df_new.drop('CKD_Status', axis=1)
y_new = kidney_df_new['CKD_Status']
```

[ ]:
```
# NEW Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_new, y_new, test_size=0.2, random_state=42, stratify=y
)
```

[ ]:
```
# NEW Scaling (for LR only)
scaler = StandardScaler()
X_train_scaled_new = scaler.fit_transform(X_train)
X_test_scaled_new = scaler.transform(X_test)
```

[ ]:
```
# NEW Logisitc Regresion model (LR)
lr_new = LogisticRegression(max_iter=1000)
lr_new.fit(X_train_scaled_new, y_train)
```

[ ]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
# NEW Random Forest model (RF)
rf_new = RandomForestClassifier(n_estimators=100, random_state=42)
rf_new.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(random_state=42)
```

[ ]:
```
# NEW Logistic Regression predictions
y_pred_lr_new = lr_new.predict(X_test_scaled_new)
print("Logistic Regression:")
print(classification_report(y_test, y_pred_lr_new))
print("ROC AUC:", roc_auc_score(y_test, lr_new.
 ↪predict_proba(X_test_scaled_new)[:, 1]))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.61      0.63      0.62       226
           1       0.63      0.62      0.63       235

    accuracy                           0.62       461
   macro avg       0.62      0.62      0.62       461
weighted avg       0.62      0.62      0.62       461

ROC AUC: 0.6785539446431933
```

[ ]:
```
# NEW Random Forest predictions
y_pred_rf_new = rf_new.predict(X_test)
print("Random Forest:")
print(classification_report(y_test, y_pred_rf_new))
print("ROC AUC:", roc_auc_score(y_test, rf_new.predict_proba(X_test)[:, 1]))
```

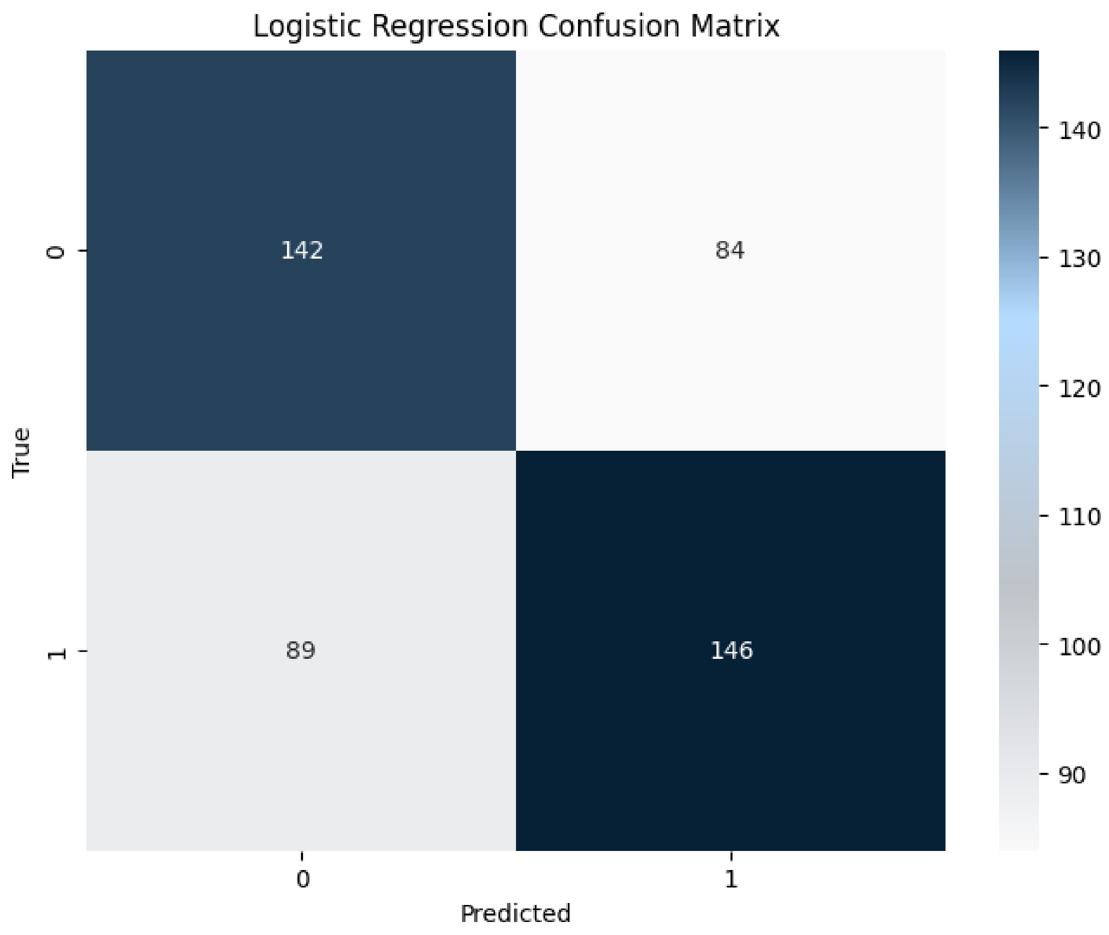
```
Random Forest:
              precision    recall  f1-score   support

           0       0.65      0.82      0.73       226
           1       0.77      0.57      0.66       235

    accuracy                           0.70       461
   macro avg       0.71      0.70      0.69       461
weighted avg       0.71      0.70      0.69       461

ROC AUC: 0.7506495951798156
```

Visualizations: * Confusion Matrix for each model * ROC Curve * Feature Importance (RF) * Coefficients (LR)

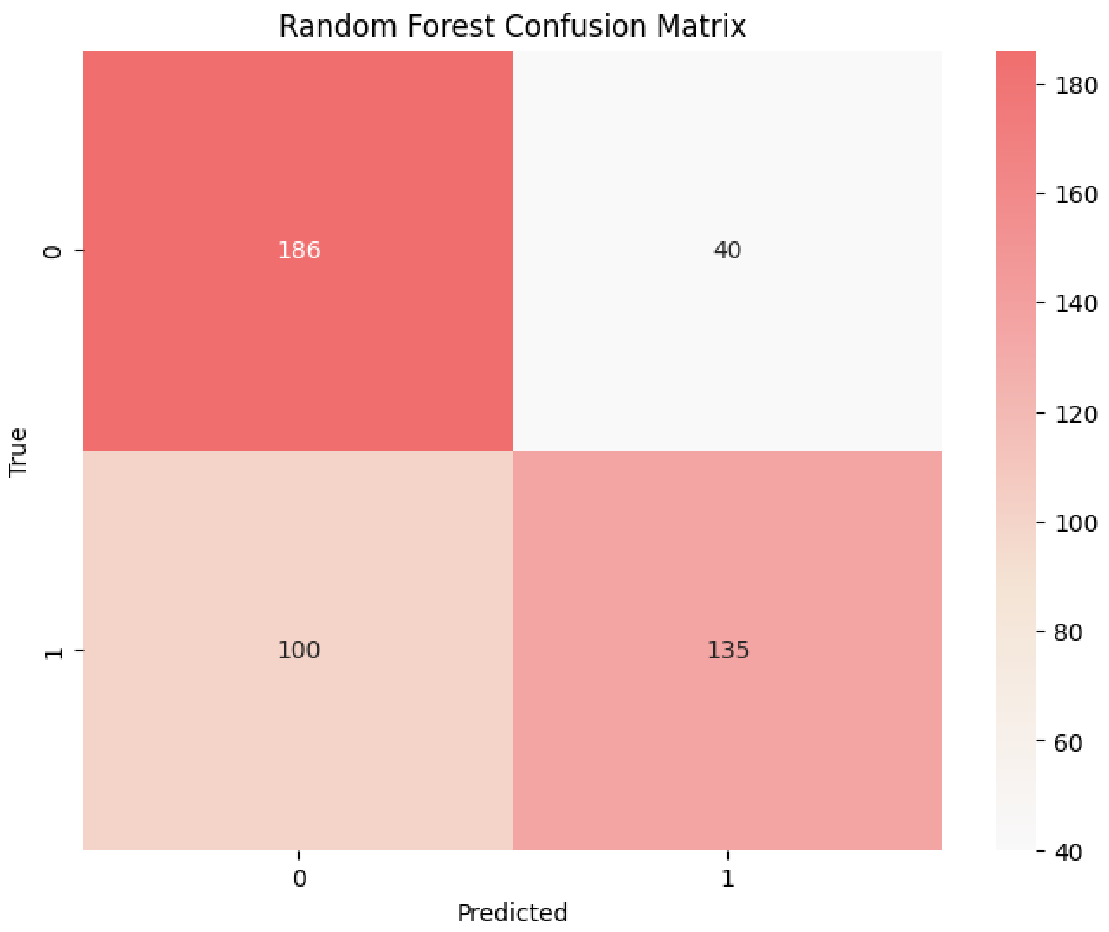
[ ]:
```
# Custom colors to match slides for presentation!
colors_red = ['#f9f9fa', '#f4e3d4', '#f3a2a2', '#f06e6e']
```

```
custom_cmap_red = LinearSegmentedColormap.from_list("custom_cmap_red",␣
    ↪colors_red)
colors_blue = ['#f9f9fa', '#bec3c9', '#b4dbff', '#062036']
custom_cmap_blue = LinearSegmentedColormap.from_list("custom_cmap_blue",␣
    ↪colors_blue)
```

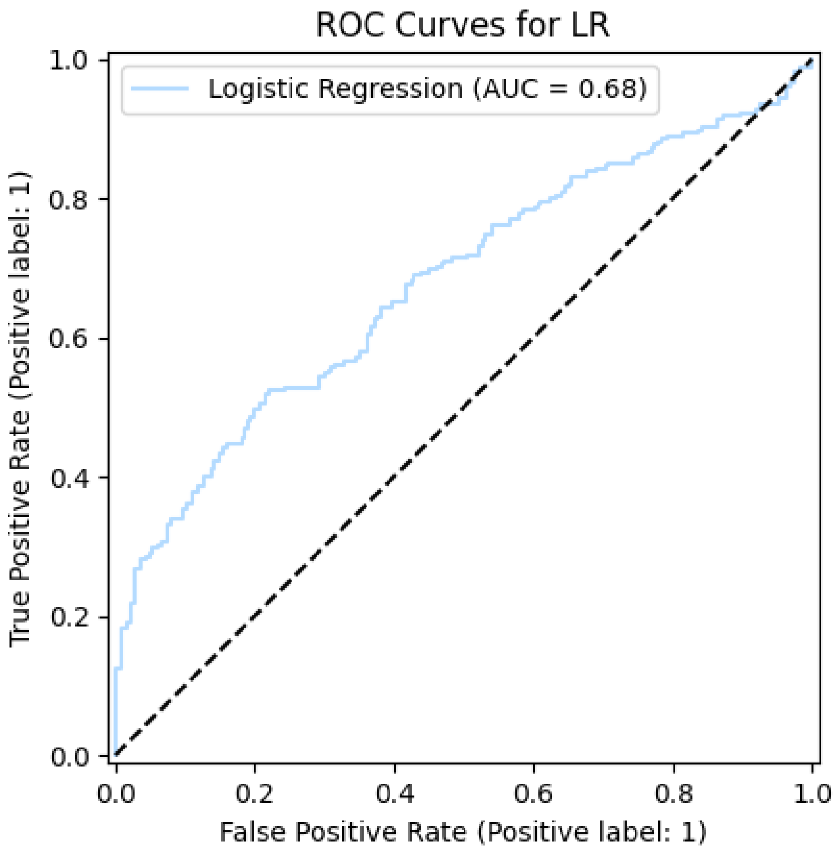
[ ]:
```
# Confusion Matrix for Logistic Regression
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_lr_new), annot=True, fmt='d',␣
    ↪cmap=custom_cmap_blue)
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.savefig("viz/confusion_lr.png")
plt.show()
```



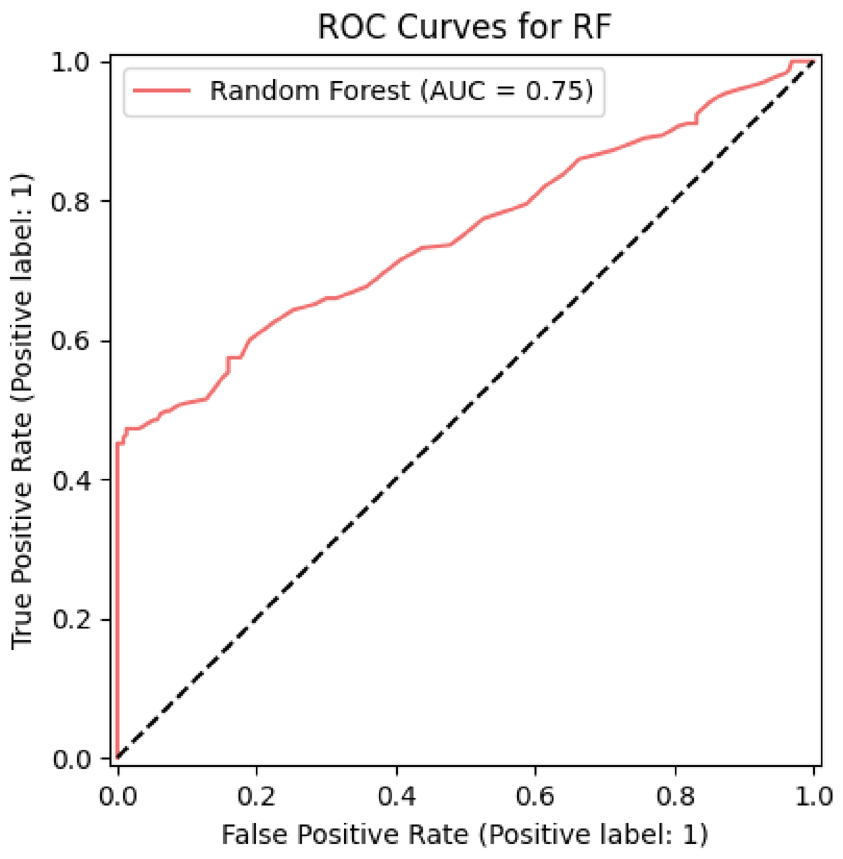
```
[ ]: # Confusion Matrix for Random Forest
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_rf_new), annot=True, fmt='d',␣
 ↪cmap=custom_cmap_red)
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.savefig("viz/confusion_rf.png")
plt.show()
```



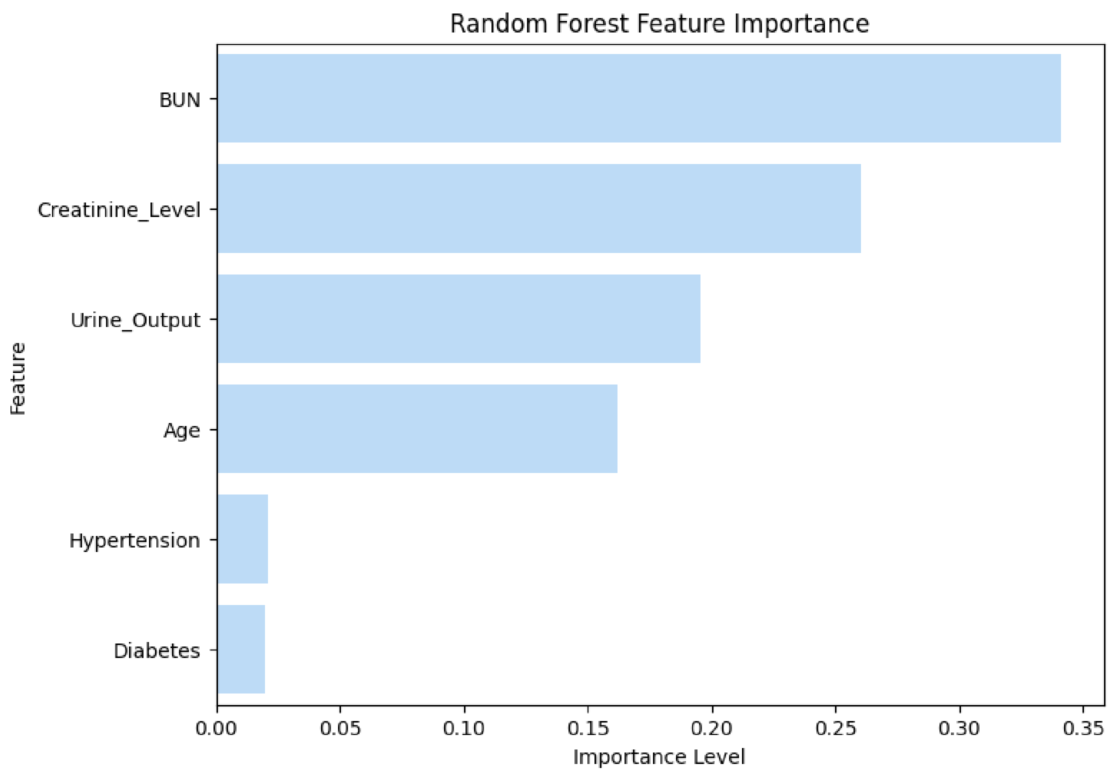
```
[ ]: # ROC Curve for LR
RocCurveDisplay.from_estimator(lr_new, X_test_scaled_new, y_test,␣
 ↪name='Logistic Regression', color='#b4dbff')
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curves for LR")
plt.legend()
plt.savefig("viz/ROC_lr.png")
plt.show()
```



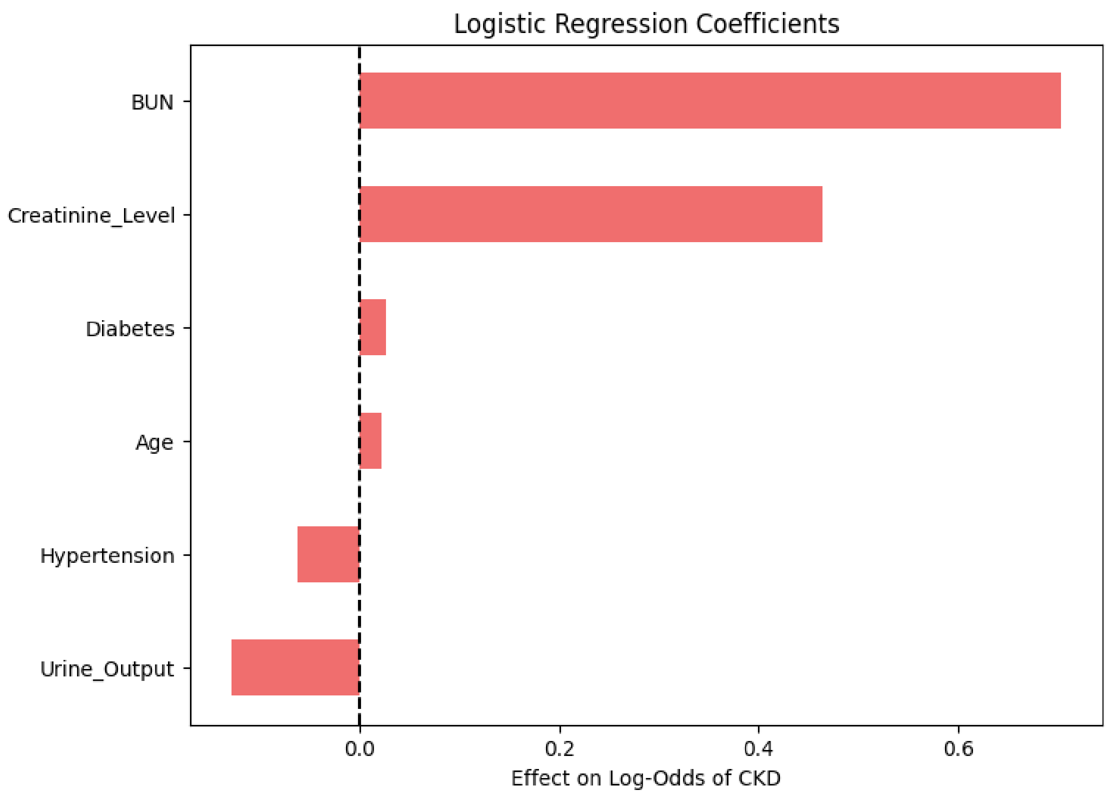
[ ]:
```
# ROC Curves for RF
RocCurveDisplay.from_estimator(rf_new, X_test, y_test, name='Random Forest',␣
 ↪color='#f06e6e')
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curves for RF")
plt.legend()
plt.savefig("viz/ROC_rf.png")
plt.show()
```



```
[ ]: # Feature Importance of the Random Forest model - which features are most␣
      ↪influential?
     importances = rf_new.feature_importances_
     features = X_new.columns
     feature_importance = pd.Series(importances, index=features).
      ↪sort_values(ascending=False)

     plt.figure(figsize=(8, 6))
     sns.barplot(x=feature_importance, y=feature_importance.index, color='#b4dbff')
     plt.title("Random Forest Feature Importance")
     plt.xlabel("Importance Level")
     plt.ylabel("Feature")
     plt.savefig("viz/feat_imp_rf.png")
     plt.show()
```



[ ]:
```
# Coefficient Plot of the Logistic Regression model - in which direction do the␣
↪features influence?
coefs = pd.Series(lr_new.coef_[0], index=X_new.columns).sort_values()

plt.figure(figsize=(8, 6))
coefs.plot(kind="barh", color='#f06e6e')
plt.title("Logistic Regression Coefficients")
plt.xlabel("Effect on Log-Odds of CKD")
plt.axvline(0, color='black', linestyle='--')
plt.savefig("viz/coeff_lr.png")
plt.show()
```



Testing model classifications

```
[ ]: # Probabilities of positive class (CKD = 1)
     prob_rf = rf_new.predict_proba(X_test)[:, 1]
     prob_lr = lr_new.predict_proba(X_test_scaled_new)[:, 1]
```

```
[ ]: # Probability threshold (to define "high risk")
     threshold = 0.7 # experiment with different thresholds
     high_risk = prob_rf >= threshold # mess w/ lr to see if probability changes
```

```
[ ]: # Add risk labels to patients ("high" vs "low/moderate")
     risk_labels = np.where(high_risk, "High Risk", "Low/Moderate Risk")
     results_df = X_test.copy()
     results_df["CKD_Prob"] = prob_rf
     results_df["Risk_Label"] = risk_labels
```

```
[ ]: results_df.head()
```

```
[ ]:      Age  Creatinine_Level   BUN  Diabetes  Hypertension  Urine_Output  \
     749   70              1.23  17.3         0             0        1103.0
     768   83              1.30  22.6         0             1        1256.0
     95    56              0.30  33.9         1             0        1699.0
```

```
1618   84              0.30  25.2         1             0         737.0
1968   24              1.83   7.3         1             0        1394.0

      CKD_Prob         Risk_Label
749       0.49  Low/Moderate Risk
768       0.32  Low/Moderate Risk
95        1.00          High Risk
1618      0.43  Low/Moderate Risk
1968      0.16  Low/Moderate Risk
```

[ ]:
```
risk_counts = results_df['Risk_Label'].value_counts()
print(risk_counts)
print(f"Total number of patients in testing set: {len(results_df)}")
```

```
Risk_Label
Low/Moderate Risk    349
High Risk            112
Name: count, dtype: int64
Total number of patients in testing set: 461
```

Try XGBoost to improve model accuracy

[ ]:
```
# XGBoost model
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
xgb_acc = accuracy_score(y_test, y_pred_xgb)
print(f"XGBoost Accuracy: {xgb_acc:.4f}")
```

```
XGBoost Accuracy: 0.6811
```

[ ]:
```
# Tune XGBoost with different parameters to improve accuracy
xgb_tune = XGBClassifier(
    n_estimators=200, # increases tree size (default is 100)
    learning_rate=0.05, # decrease step size
    max_depth=4, # limit tree depth (avoid overfitting)
    subsample=0.8, # only uses 80% of data per tree (avoid overfitting)
    colsample_bytree=0.8, # only uses 80% of features per tree (avoid␣
 ↪overfitting)
    random_state=42
    )
```

[ ]:
```
# Include cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(xgb_tune, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')
print(f"XGBoost Cross Validation Accuracy: {np.mean(cv_scores):.4f}")
```

```
XGBoost Cross Validation Accuracy: 0.7021
```

[ ]:
```
# Train on training set
xgb_tune.fit(X_train, y_train)
```

[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.05, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=4,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=200,
              n_jobs=None, num_parallel_tree=None, …)
```

[ ]:
```
# Predictions on XGBoost
y_pred_xgb = xgb_tune.predict(X_test)
xgb_acc = accuracy_score(y_test, y_pred_xgb)
print(f"XGBoost Accuracy (test set): {xgb_acc:.4f}")
```

```
XGBoost Accuracy (test set): 0.7093
```

XGBoost does not give further insight or improve model accuracy.