STATS/CSE 780 - Final Project Report

Chronic Kidney Disease Detection

John Tweedie

Student number: 400550023

2023-12-12

Abstract

Chronic Kidney Disease (CKD) is a disease characterized by a progressive loss of kidney function, and left untreated, will result in renal failure. The prevalence of the disease is significantly increasing, and as such, early diagnosis and treatment are imperative to assure effective treatment and to not over-run hospital/dialysis capacity. Machine learning (ML) techniques offer a contemporary and accurate method for early detection of CKD from measured patient biomarkers and disease histories. This study applies two ML classifiers to develop diagnostic models. Extra consideration was given to clinical applicability of the models, as similar models have not seen widespread adoption. Further, K-Means clustering was performed on the data to investigate the potential of the method as tool for inference and subsequent treatment planning.

Introduction

Chronic Kidney Disease (CKD), a condition affecting millions worldwide, is characterized by a general progressive loss of kidney structure and function (Levey and Coresh, 2012). A significant increase in the prevalence of the disease since the 1980s has been noted, likely due to an aging population, increases in diagnostic capabilities that have reduced the occurrence of patients going undiagnosed, and improvements in treatments that can significantly extend a patients lifespan (Lewis, 2012; Locatelli et al., 2002). Early detection and intervention is imperative to slow disease progression, but remains challenging given the current resources available in typical clinical settings. If treatment is not sufficient in the earlier stages, the disease can progress to a severe end-stage wherein a patient must complete frequent dialysis treatment, which can constitute a large expense for both the patient and healthcare system. Recognizing these shortcoming in the mid-2000s, the United Kingdom's National Health Service stated a need for improving methods for identification of at-risk individuals, a screening program, and measures to reduce progression of the disease starting at the early stages (Lewis, 2012). Further, recent reform efforts have strongly recommended for community-based and primary care (i.e. outside of hospitals), such that the financial burden of intensive treatments is reduced, thus highlighting a need for accessible and robust clinical tools for diagnostics, prognostics, and therapeutics (Lewis, 2012).

CKD is defined largely in general terms of kindney function impairment due to the heterogeneous nature of the condition; the disease can manifest in a variety of expressions and pathologies, and can progress at different rates from patient to patient (Levey and Coresh, 2012). In a more clinically-formal sense, CKD is defined as either when the kidneys excretory capacity is reduced, or simply by the presence of proteinuria (excess protein present in the urine). The

kidney organ acts to filter the bloodstream and remove waste (producing urine), and regulate various chemicals and bloodstream components such as hormones, electrolytes, and vitamins. As such, they represent a vital and robust aspect of bodily function, and patients of CKD are thereby often significantly negatively impacted by their waning function. CKD imposes various cardiovascular implications to patients, including an increased risk of cardiovascular events, chronic inflammation leading to accelerated aging at a cellular level, cardiomyopathy, and vascular calcification (Jankowski and Noels, 2021; Lewis, 2012). Further implications of the disease can include disruption of circadian rhythm, bone demineralization, gut dysbiosis and associated digestive issues, hyperparathyroidism, and hyperphosphatemia (Jankowski and Noels, 2021). At its most severe stage, CKD can cause complete kidney failure, resulting in the patient either receiving a transplant, or completing dialysis treatment indefinitely, as there exists no cure for CKD.

The causes of CKD remain variable among populations. In developed nations, the disease is more commonly associated with aging (with the loss of kidney function being benign), diabetes, hypertension, obesity, and cardiovascular disease. While these causes are still prevalent in developing nations, CKD in these populations is relatively more-likely to arise as a secondary effect from infections, drug exposure, and toxicity (Levey and Coresh, 2012).

Considering the function of the kidney, CKD is often identifiable by distinct compositional abnormalities in the bloodstream or urine. Some hallmark testable biomarkers of the disease manifest in the bloodstream as abnormal levels of vitamin D, calcium, and phosphorous, or in the urine as excess sediment or protein (Levey and Coresh, 2012; Lewis, 2012). Common diagnostic criteria regarding the excretory capacity is defined by when measurements of the glomerular filtration rate of the kidney are less than 60/ml on two occasions over a 90-day period; a measure typically defaulted to by clinicians. (Lewis, 2012). Regarding the proteinuria, diagnostic criteria are defined as when a patient exhibits a ratio of urine albumin- (a type of protein present in the bloodstream) to-creatine greater than 30 mg/mmol, or a urine protein-to-creatine ratio greater than 50 mg/mmol (Lewis, 2012). Criteria exist for diagnosing CKD based on various pathological abnormalities and observable markers of kidney damage, such as polycystic kidney diseases, renal tubular acidosis, renal masses, and enlarged kidneys (Levey and Coresh, 2012). Some more-recent machine learning-based diagnostic techniques have shown promising results in non-clinical, but often require wide arrays of variables to be present as mandatory features in the predictive diagnostic models (Sanmarchi et al., 2023).

Similar to diagnostic methods, prognosis of the disease has undergone significant reforms in the mid-2000s CKD used to be described by a 5-stage scale developed by the US National

Kidney Foundation Kidney Disease Outcomes Quality Initiative (NKF-KDOQI), with each stage pertaining to a degree of glomerular filtration rate impairment (Lewis, 2012). At the least-sever end of the spectrum; Stage 1 is describes the presence of proteinuria and/or abnormal kidney anatomy. Conversely, Stage 5 describes end-stage renal disease (i.e. renal failure) (Lewis, 2012). Two clinically-significant shortcomings to this classification scale prompted inclusion of additional considerations in the diagnostic criteria. Most notably, there was no consideration as to the cause of the patient's CKD (i.e the patient disease history). Further, the previous scale was based upon a binary indicator of proteinuria, as opposed to a more-continuous indicators. The cause of CKD and a 3-level scale of proteinuria severity were consequently introduced into the accepted clinical diagnostic standards, thereby representing a more-robust set of biomarkers for which to base diagnoses on (Lewis, 2012). Recent advances in ML/AI techniques (e.g. decision trees) have been applied in research to improve prognosis methods and therapeutic treatment regimes, but similar to the diagnostic methods, have not seen widespread adoption in clinical settings (Sanmarchi et al., 2023).

With the recent advents of machine learning (ML) and artificial intelligence (AI), much research has been focused on applying these techniques in diagnostic, prognostic, and treatment contexts for various medical conditions, including CKD. In the context of CKD, prognosis-related research typically involves regression-based methods to predict disease severity and progression, whereas therapeutic research may employ stochastic Markov decision process models to optimize treatment regimes (Sanmarchi et al., 2023). The majority of the research is focused on classification-based approaches to diagnosis; the most popular methods being artificial neural networks, tree algorithms, logistic regression, and support vector machines (Sanmarchi et al., 2023). Utilization of clustering-based analysis methods are relatively sparse throughout the existing literature. Commonly used predictor variables in the CKD diagnostic studies include blood pressure, blood hemoglobin, pus cell presence, blood glucose, age, serum creatine, and red blood cell count, patient diabetes, specific gravity of urine, and patient cardiovascular disease, among others (Hossain et al., 2022; Sanmarchi et al., 2023). In the context of assessing risk of CKD development, the important and widely-adopted variables throughout the literature consist of blood pressure, age, sex, cardiovascular disease, diabetes, and various blood and urine abnormalities (Sanmarchi et al., 2023). However, feature selection can be challenging in some instances, as certain feature selection methods may perform adequately (Hossain et al., 2022).

Despite the influx of novel research on the topic, a highly limited amount of models will actually see clinical use; in their review, Sanmarchi et al., 2023 noted only one model (a lasso regression) that was applied in a clinical setting, indicating either a lack of consideration for

applicability in model-development, or simply that the clinical settings have not yet had ample time to properly test and adopt these recent advancements into practice. Major limitations of current ML/AI diagnostic methods are that the majority of studies do not explicitly consider daily use and clinical integration of these models, particularly regarding reliance on high-dimensional (and therefore not always available) data and a lack interpretability in complex models (Sanmarchi et al., 2023). As such, there is an apparent need for predictive diagnostic models that can perform sufficiently given the following considerations:

- High-accuracy of detection
- Low amount of required variables (low amount of patient testing required)
- Applicable and easy to use in clinical settings
- Ability to draw inference
- Robustness of diagnostic capability (able to diagnose early, or even identify high-risk patients)

To address these considerations, this study attempts to develop a simple, yet accurate classification model that has low dimensions and can also serve as an inferential tool. Here, a dataset (Soundarapandian et al., 2015) comprising of 250 early-stage CKD patients and 150 healthy control patients is adopted to develop the predictive diagnostic models. The diagnostic models are assessed in terms of accuracy, interpretability, dimensionality, and versatility. Further, a clustering-based analysis is completed to investigate the potential for the method as a tool for inference and potential identification of patient sub-groups. Clustering analysis and patient grouping may be useful in the context of disease prognosis and for developing more-targeted therapeutic treatment regimes. The results of this study are compared to its contemporaries.

1 Methods

The general methodological framework utilized in this study contains four steps; data exploration, clustering analysis, logistic regression classification, and support vector machine (SVM) classification. Appendix B contains a comprehensive description of data processing methods, as well as the final classification and clustering methods.

The dataset utilized for this study is the "Early stage of Indians Chronic Kidney Disease (CKD)" dataset (Soundarapandian et al., 2015), available on the UCI Machine Learning Repository. Data processing was minimal, consisting of outlier screening for removal, binary encoding

of the categorical variables, standardization of the numeric variables, and imputation of missing-values. Imputation was completed using the *Iterative Imputer* function from the python package *scikit-learn* (Pedregosa et al., 2011); a function that predicts missing or null variables using a Bayesian ridge regression on the remaining, non-null variables. A principal components analysis was performed using scikit-learn (Pedregosa et al., 2011) to provide a reduced-dimension feature set for model fittings.

K-means clustering was performed to identify potential patient subgroups. This represents a methodology that has seen limited use and testing in the context of CKD diagnosis and prognosis (Sanmarchi et al., 2023). Further, the variable clinical expression and characterization of CKD (Levey and Coresh, 2012; Lewis, 2012) suggests there may be distinct subgroups comprised of patients that exhibit similar diagnostic indicators in certain subsets of biomarkers (e.g. a group for abnormal blood composition as a diagnostic indicator). The *scikit-learn* (Pedregosa et al., 2011) package was used too perform the K-means clustering. The K-means clustering method attempts to partition the observations into characteristically-distinct subgroups, with minimal combined in-group variance; letting $C_k \in C$ denote cluster $k \in K$, the objective of the algorithm is to minimize the following:

$$\underset{C}{\text{minimize}} \left\{ \sum_{k=0}^{K} (Var(C_k)) \right\} \tag{1}$$

where the solution used in scikit-learn is achieved using Elkan's algorithm (James et al., 2023; Pedregosa et al., 2011). The number of clusters K is pre-defined hyper-parameter. Tuning for K was achieved through iterative clustering and assessing silhouette score. Additional exploratory and charting analyses was completed on the data with regard to identifying distinct group characteristics.

The first classification method performed was logistic regression. The logistic regression model is fitted using maximum likelihood approach, and maps predictions to binary outcomes using the logistic function (James et al., 2023):

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \tag{2}$$

where X is a predictor variable, and β_0 and β_1 are the fitted regression parameter coefficients. LASSO feature selection was performed for the logistic regression to reduce dimensions. The regularization penalty hyper-parameter *alpha* was tuned via cross-validation. Additional logistic regression models were fit using the PCA components, features only from the blood-test, and features only from the urine test.

The second classification method used was support vector machines (SVM). SVMs attempt

to fit a kernel through the feature space that maximizes the separation of features (James et al., 2023). A linear, polynomial, and radial kernel were fitted, and the respective cross-validation results were compared to identify the optimal kernel. Similar reduced-dimension models were also fit using SVM.

Classification models were evaluated by cross validation; specifically sensitivity and specificity. Sensitivity reflects the estimated probability that a CKD patient would be correctly identified by the model. In this context, extra emphasis was given to sensitivity, due to the implications of not identifying early-stage CKD cases and giving a misleading impression to a patient that they are disease-free (Maxim et al., 2014). Specificity reflects the estimated probability of correctly identifying a healthy patient. In this instance, the patient would be misled into believe they have the disease, when in fact they do not, potentially leading to unnecessary psychological.

2 Results

2.0.1 Data Exploration

A summary of the variables contained within the dataset is provided in Table 1. A total of 24 variables were present; 9 from the blood test data (B), 8 from the urine tests (U), and the remain from the patient history assessment (P).

Kernel density plots of each numeric variable are provided in Figure 1, with outliers truncated for visual purposes. A portion of the numeric variables (e.g. *su*, *bu*) exhibit some skewness, as several observations contain abnormally high values, likely indicative of kidney malfunction.

Bar plots are provided for the categorical variables in Figure 2, each of which being a binary outcome. In general, most patients appear much more likely to exhibit the 'normal' or 'healthy' outcome for a given categorical variable. As such, it is likely that those with CKD may exhibit a limited amount of abnormal health indicators (i.e. 1 or 2), providing some additional intuitive evidence of distinct sub-groups for clustering.

A correlation matrix is provided in Figure 3. Due to similarity in the measured, and thus redundant parameters in the data, correlation between several features were noted. High-correlation variables were removed during model fitting using the variance inflation factor with a cutoff of 5.

The PCA-reduced feature set was derived using three components. While the associate scree plot (Figure 4) suggests that a significant portion of the data remains unexplained, the selection of three components was intentional to drastically reduce dimensions while still retaining

Table 1: Variable Summary

Variable	Desc.	Test	Variable	Desc.	Test
age	age (years)	n/a	rbcc	red blood cell count (mill/cmm)	B
bp	blood pressure (mm/Hg)	n/a	rbc	red blood cells, {normal, abnormal}	U
sg	specific gravity (ratio)	U	pc	pus cell {normal, abnormal}	U
al	albmin, ordinal numeric	U	pcc	pus cell clumps {present, not}	U
su	sugar, ordinal numeric	U	ba	bacteria {present, not}	U
bgr	blood glucose random (mgs/dl)	B	htn	hypertension {yes, no}	P
bu	blood urea (mgs/dl)	B	dm	diabetes mellitus {yes, no}	P
sc	serum creatinine (mgs/dl)	B	cad	coronary artery disease {yes, no}	P
sod	sodium (mEq/L)	B	appet	appetite {good, poor}	P
pot	potassium (mEq/L)	B	pe	pedal edema {yes, no}	P
hemo	hemoglobin (g)	B	ane	anemia {yes, no}	P
pcv	packed cell volume	B	class	CKD {ckd, not ckd}	n/a
wbcc	white blood cell count (cells/cumm)	B			

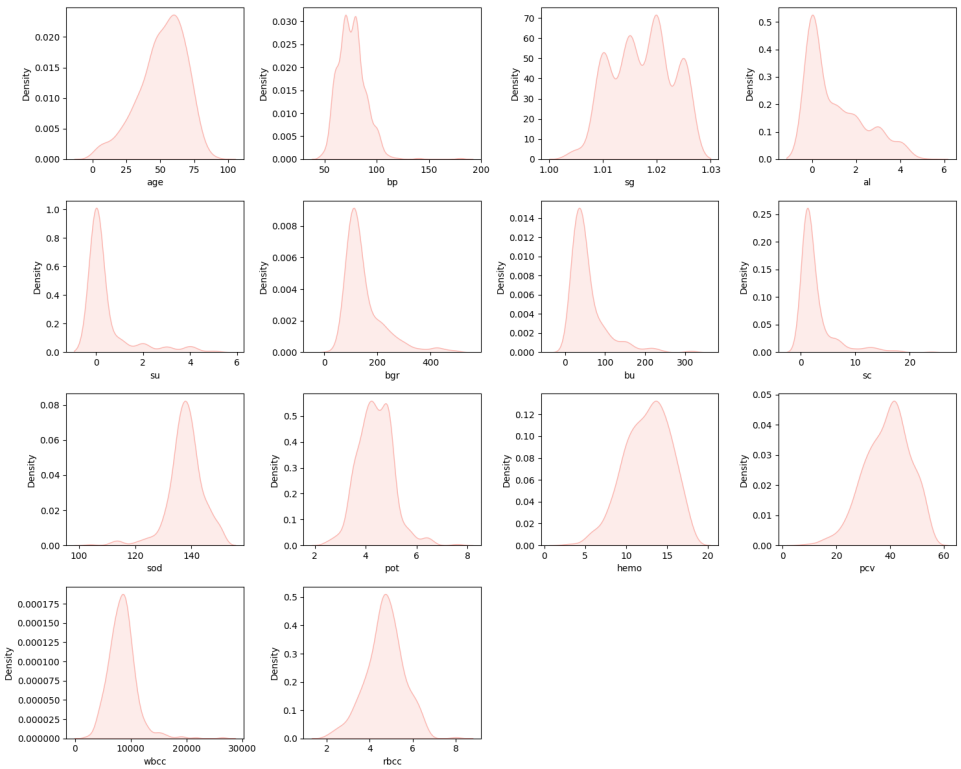


Figure 1: Kernel density plots of numeric CKD variables

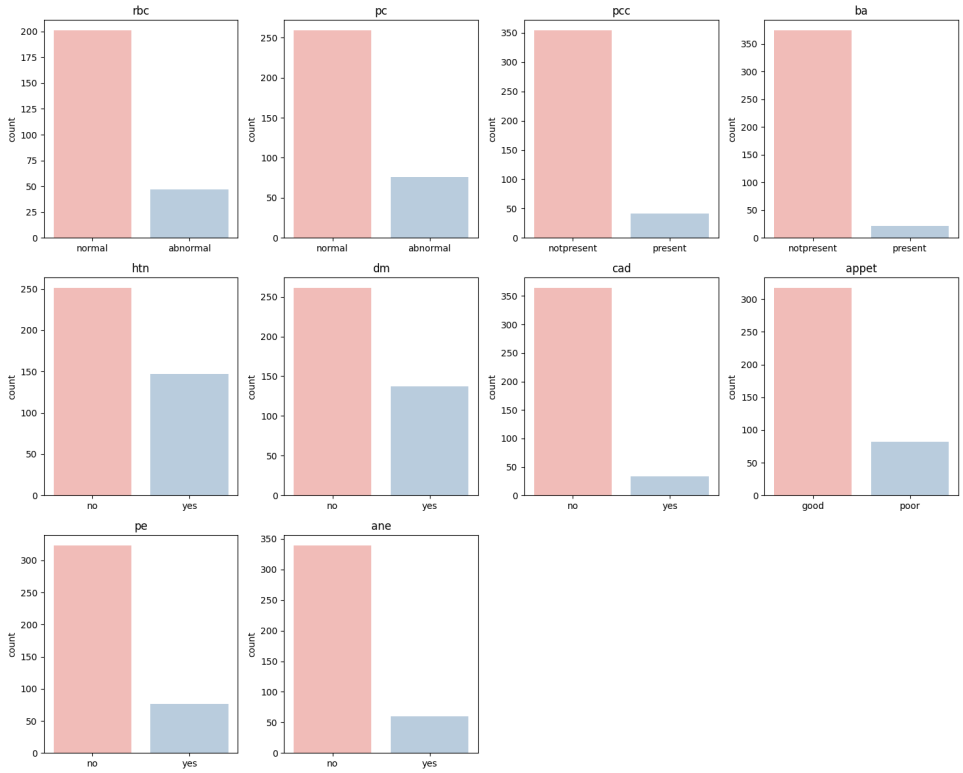


Figure 2: Bar plots of categorical CKD variables

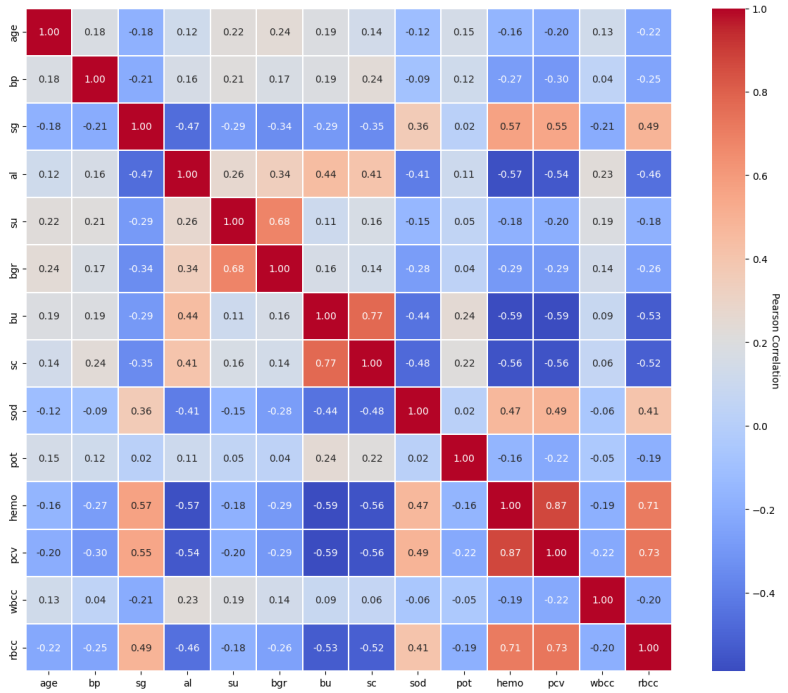


Figure 3: Correlation matrix for numeric CKD variables

adequate validation performance.

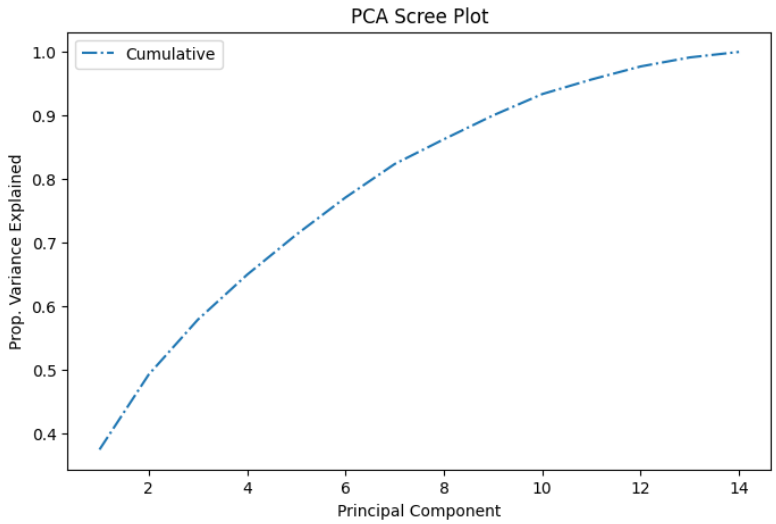


Figure 4: Scree plot of principal component analysis of CKD data

2.0.2 Clustering

Clustering was ultimately completed on the non-transformed data; this was done to preserve the original feature variance and inter-relationships, and with the intention of finding groups that exhibit extreme or outlying observations in similar variables. Clustering was also performed on standardized and PCA-transformed data, but the resulting groups were not as objectively distinct in regard to CKD classification. Conversely, using the non-transformed data would always produce a cluster containing all of the healthy patients and some CKD patients, with the remaining clusters containing only CKD patients.

The number of clusters (K) was tuned on the basis of silhouette score, and the results are provided in Table 2. While 2 clusters contained the highest score, $K = 4$ clusters was ultimately selected, as 2 clusters provides only a basic classification-type segmentation of the data. Adopting 4 clusters was completed in attempts to describe additional sub-groups among CKD patients, and resulted in success isolation of healthy patients from 3 clusters of CKD patients (Figure 5).

Table 2: Tuning results based of silhouette score for K-Means clustering

Num. clusters (K)	Silhouette score
2	0.44
3	0.39
4	0.41
5	0.38
6	0.36

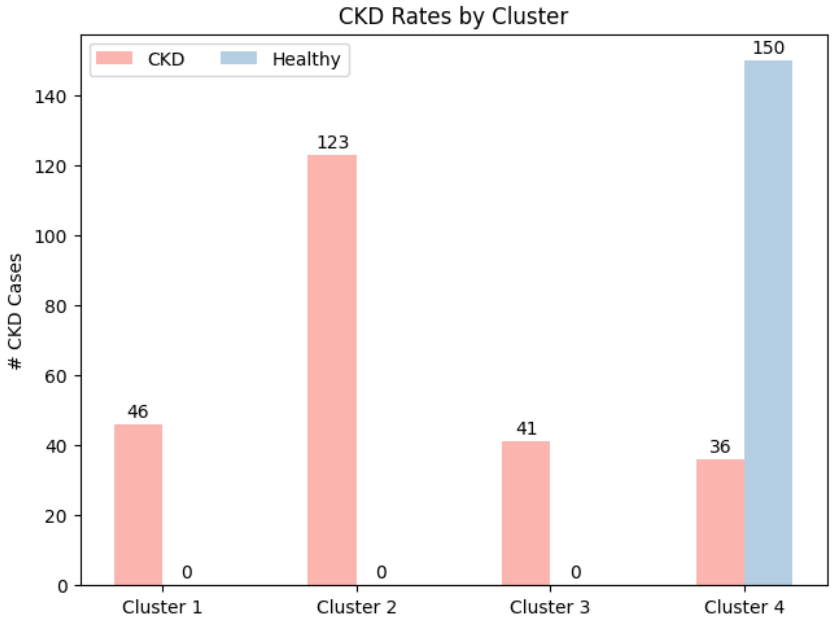


Figure 5: CKD rates by cluster

The resulting clusters, plotted against their respective principal components with transformed cluster centroids, are provided in Figure 6. Clusters 4 (containing all the healthy patients) appears densely concentrated and highly similar. The same is true for cluster 2, but to a lesser extent. Clusters 1 and 3 appear to have captured two groups with higher in-group variability, but with distinct contributions to the overall variance.

Cluster statistics are presented in Table 3 for the categorical variables, and in Table 4 for the numeric variables. From this, it is noted that cluster 3 patients exhibit fewer disease indicators for some urine tests variables (i.e. *rbc* and *pc*), exhibit a higher relative proportion of patient history indicators (specifically *htn*, *ane*, and *pe*), and show distinct abnormalities in *al, bu, hemo, sc,* and *pcv*. Categorical outcomes between clusters 1 and 2 are relatively similar, but show some unique characteristics within their respective numeric variables. Cluster 1 patients tend to exhibit very high levels of *su, bgr,* and *wbcc.*, and are the oldest on average. Cluster 2 patients exhibit lower relative blood pressure (*bp*) and are the youngest, on average, among CKD patients. Additional plotting for each variable by cluster is available in Appendix A, for reference.

Table 3: Proportion of '1' outcomes $\in \{1, 0\}$ of each categorical variable by cluster

Cluster #	class	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane
1	1	0.74	0.67	0.24	0.09	0.7	**0.83**	0.26	0.33	0.22	0.17
2	1	0.85	0.70	0.20	0.09	0.58	0.48	**0.10**	0.33	0.29	0.18
3	1	**0.46**	**0.39**	0.17	0.15	**0.83**	0.66	0.22	0.44	**0.54**	**0.66**
4	0.19	0.98	0.99	0	0.01	0.04	0.06	0	0.04	0.04	0.01

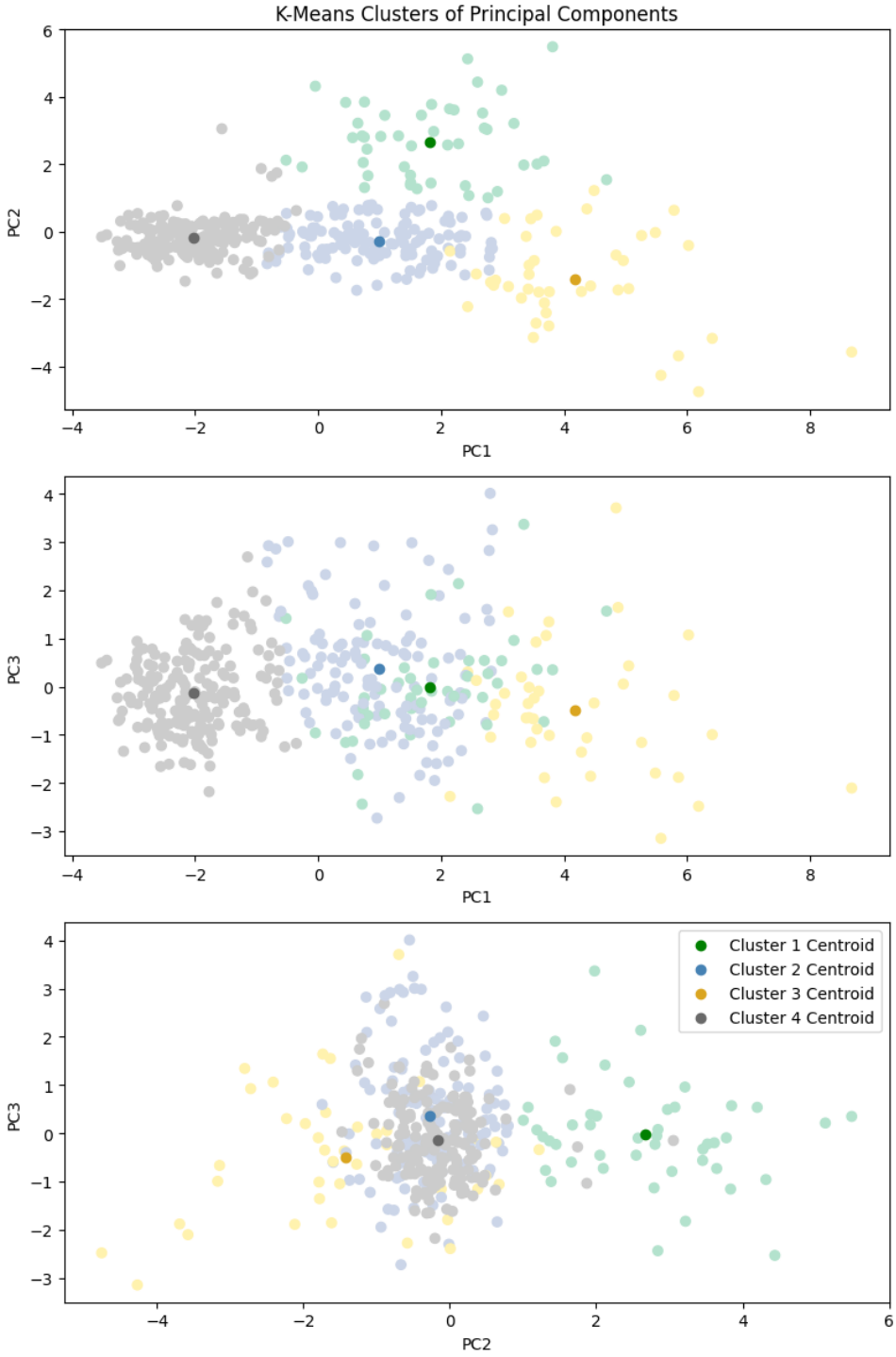


Figure 6: K-Means clusters and PCA components of CKD data

Table 4: Average values of each numeric variable by cluster (notable values bolded)

Cluster	age	bp	sg	al	su	bgr	bu
1	61.17	83.99	1.01	1.91	**2.48**	**305.37**	50.72
2	**54.39**	**76.4**	1.01	1.59	0.17	140.38	63.33
3	55.71	85.18	1.01	**2.38**	0.73	161.25	**148.09**
4	45.79	72.38	1.02	0.17	0.06	112.46	32.22

Cluster	sc	sod	pot	hemo	pcv	wbcc	rbcc
1	2.33	134.86	4.22	11.72	36.07	**9968.6**	4.39
2	2.71	136.32	4.4	10.92	34.28	9094.31	4.31
3	**10.38**	130.48	4.9	**8.21**	**25.73**	8471.54	3.44
4	1.04	141.13	4.29	14.82	45.51	7676.05	5.31

2.0.3 Logistic Regression

Logistic regression classification performance was strong across all model variations. Cross validation results are presented in 5. For the 'baseline' full model, with manual feature selection, a sensitivity rate of 99% was achieved. This was improved upon with the LASSO-selected features, PCA components, and blood test features, wherein a sensitivity rate of 100% was achieved; however, this came with the cost of decrease specificity for each. A specificity rate of 71.8, seen in the blood test, is likely not acceptable for clinical settings. The urine test feature model achieved a sensitivity rate of 98% and a specificity of 90%.

Table 5: Logistic regression classification model cross validation results

Model	# Params.	Sensitivity (%)	Specificity (%)	Missclass Rate (%)
Baseline	12	99.0	98.2	1.3
LASSO	11	100	93.3	2.5
PCA	3	100	77.8	10.1
Blood test	9	100	71.8	13.8
Urine test	8	98.0	90.0	5.0

Through the hyperparameter tuning, the LASSO feature selection was able to achieve dimensions of 11 without a reduction in sensitivity; the results of which are provided in Figure 7. Dimensionality could be further reduced, but at the cost of additional contextutally-problematic false negative results.

2.0.4 SVM Classification

The support vector machine classification achieved similarly-accurate results (Table 6). The 'baseline' full model, using all available features achieved both high sensitivity and specificity rates, at 99% and 100%, respectively. These results were not improved upon by adopting either a radial or polynomial kernel, with the polynomial kernel slightly reducing performance. The highest sensitivity rate was observed in the model utilizing the PCA components, suggesting that the transformation preserved the variance structure in such a way that linear separability

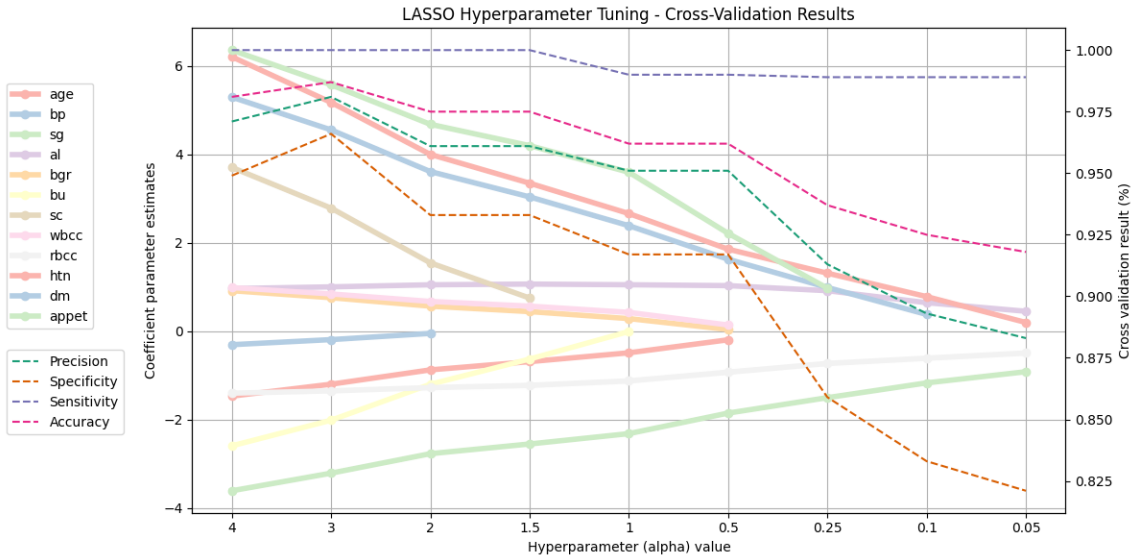


Figure 7: Cross-validation results of LASSO hyperparameter tuning trials

was successfully retained, and even somewhat improved. Compared to the logistic regression models, the blood test feature model performed worse regarding sensitivity (87.1%), but much better regarding specificity (95.3%). The urine test feature models achieved higher sensitivity and specificity performance relative to the corresponding logistic regression classification model.

Table 6: Support vector machine classification model cross validation results

Model	# Params.	Sensitivity (%)	Specificity (%)	Missclass Rate (%)
Baseline	24	99.0	100	0.6
LASSO	11	99.0	98.2	1.3
PCA	3	100	98.2	0.6
Blood test	9	87.1	95.3	10.7
Urine test	8	98.1	96.4	2.5

3 Discussion

The clustering analysis performed in this study represents a relatively underutilized method in the context of CKD prognosis and diagnosis. Through the K-Means clustering, 4 distinct patient sub-groups were observed; 3 of which contained only CKD patients. This suggests that K-Means may also perform well as a classification method. From the resulting clusters, cluster 1 is characterized by an older cohort of patients with relatively high blood-sugar (reflected by *bu* and *bgr*) and who are more-likely diabetic (*dm*). Cluster 2 is characterized by a younger cohort of patients that are unlikely to have coronary artery disease, high blood pressure, or high blood sugar (i.e. the biomarkers more-so resemble that of a healthy patient), but who still have CKD. This represented the largest cluster of the three CKD-patient clusters. Cluster 3

is characterized by patients with significant abnormalities in urine composition (e.g. *al*) and who were more likely to have either anemia or pedal edema. The inter-relation of variables that yield abnormal values within clusters warrants further analysis, as there is potential to concretely identify patient groups with specific causes and specific manifestations of CKD. As such, treatment regimes could be tailored to best address a patient needs; e.g. to address the diabetic related issues common amongst cluster 1 patients.

The classification exercises in this study illustrates that theses methods can provide robust and accurate diagnostic models for early-stage CKD. For all but one classifier model and feature selection method, the sensitivity rate was above 98.1%. This suggests that if one of these models were to be adopted, CKD would go undetected in no more than approximately 1.8% of patients with early-stages of the disease. The outlying model in regards to sensitivity was the blood test SVM classifier, which achieved only 87.1%. The prediction accuracy achieved in all models this study is highly comparable to other diagnostic screening tests (Maxim et al., 2014), and thus, can likely be considered at clinically-acceptable levels. Further, results were relatively in line with contemporary ML/AI diagnostic models for CKD (Akben, 2018; Hossain et al., 2022); for instance Akben, 2018, using K-Nearest Neighbors classification, achieved miss-classification rates of 2.2% and 6.7% while using urine test features only and blood test features only, respectively.

The models that performed best overall were the baseline models (both logistic and SVM), the LASSO models, and the SVM PCA model. Regarding applicability in a clinical setting, the SVM model using the LASSO-selected feature set is likely the most relevant due to the lower dimensions relative to the prediction accuracy. As such, the SVM model, which achieves sensitivity and specificity rates in line with other clinically-accepted diagnostic tests (Maxim et al., 2014) could therefore be implemented as a screening tool for patients that may receive routine blood and urine tests. Further, it was demonstrated that one could run the screening using only one of the tests (for instance, if only one was available), while still retaining high diagnostic accuracy.

While the SVM models achieved favourable cross validation results, the models themselves lack an avenue for further inference. In the case of the logistic model, inference can be achieved by review of the resulting coefficients and p-vales. For instance, in the LASSO logistic model, the *htn* coefficient of 3.35 suggests that patients with *hypertension* are approximately 28.5 times more likely (based on the log-odds) to have early-stage CKD than those without. Conversely, the SVM model offers no such avenue for similar inference. Further, one can assess the importance of biomarkers in terms of predictive value. In this instance, specific gravity, age,

diabetes, hypertension, red blood cell count, and urine albumin were among the most-relevant diagnostic predictors, which is largely consistent with the relevant literature (Hossain et al., 2022; Sanmarchi et al., 2023).

The methods performed in this study demonstrate the effectiveness and value of machine learning techniques in medical fields. Unique inference can be gained through clustering, as well as through review of logistic regression model fittings. The classification exercise illustrates how simple ML techniques can be used to develop highly accurate, and therefore clinically-relevant, diagnostic models. As such, it is likely that these tools will see an increase in clinical adoption in the near-future.

References

Akben, S. (2018). Early Stage Chronic Kidney Disease Diagnosis by Applying Data Mining Methods to Urinalysis, Blood Analysis and Disease History. *IRBM*, *39*(5), 353–358. https://doi.org/10.1016/j.irbm.2018.09.004

Hossain, M. M., Swarna, R. A., Mostafiz, R., Shaha, P., Pinky, L. Y., Rahman, M. M., Rahman, W., Hossain, M. S., Hossain, M. E., & Iqbal, M. S. (2022). Analysis of the performance of feature optimization techniques for the diagnosis of machine learning-based chronic kidney disease. *Machine Learning with Applications*, *9*, 100330. https://doi.org/10.1016/j.mlwa.2022.100330

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An introduction to statistical learning with applications in python*. Springer. https://doi.org/10.1007/978-3-031-38747-0

Jankowski, J., & Noels, H. (2021, March). *Comorbidities in Chronic Kidney Disease (CKD)*. MDPI. https://doi.org/10.3390/books978-3-03936-669-9

Levey, A. S., & Coresh, J. (2012). Chronic kidney disease. *The Lancet*, *379*(9811), 165–180. https://doi.org/10.1016/S0140-6736(11)60178-5

Lewis, R. (2012). *Understanding Chronic Kidney Disease: A Guide for the Non-Specialist*. M & K Update Limited. https://books.google.ca/books?id=ssrnUtJqrV0C

Locatelli, F., Vecchio, L. D., & Pozzoni, P. (2002). The importance of early detection of chronic kidney disease. *Nephrology Dialysis Transplantation*, *17*(suppl 11), 2–7. https://doi.org/10.1093/ndt/17.suppl_11.2

Maxim, L. D., Niebo, R., & Utell, M. J. (2014). Screening tests: A review with examples. *Inhalation Toxicology*, *26*(13), 811–828. https://doi.org/10.3109/08958378.2014.955932

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Sanmarchi, F., Fanconi, C., Golinelli, D., Gori, D., Hernandez-Boussard, T., & Capodici, A. (2023). Predict, diagnose, and treat chronic kidney disease with machine learning: A systematic literature review. *Journal of Nephrology*, *36*(4), 1101–1117. https://doi.org/10.1007/s40620-023-01573-4

Soundarapandian, P., Jerlin Rubini, L., & Eswaran, P. (2015). Early stage of Indians Chronic Kidney Disease (CKD). https://archive.ics.uci.edu/dataset/336/chronic+kidney+disease

A Appendix A: Additional Plotting for K-Means Clusters

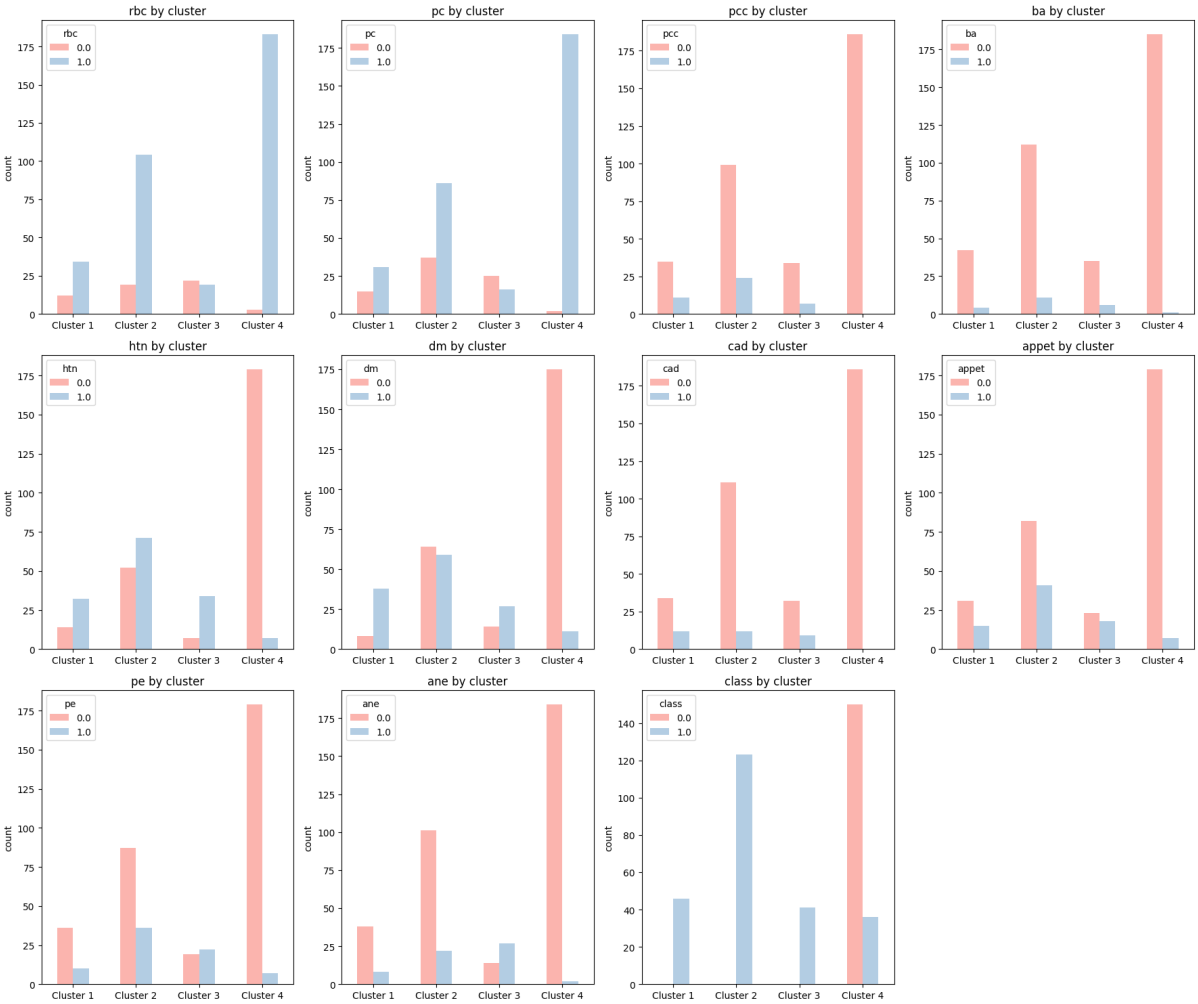


Figure 8: K-Means clusters and PCA components of CKD data

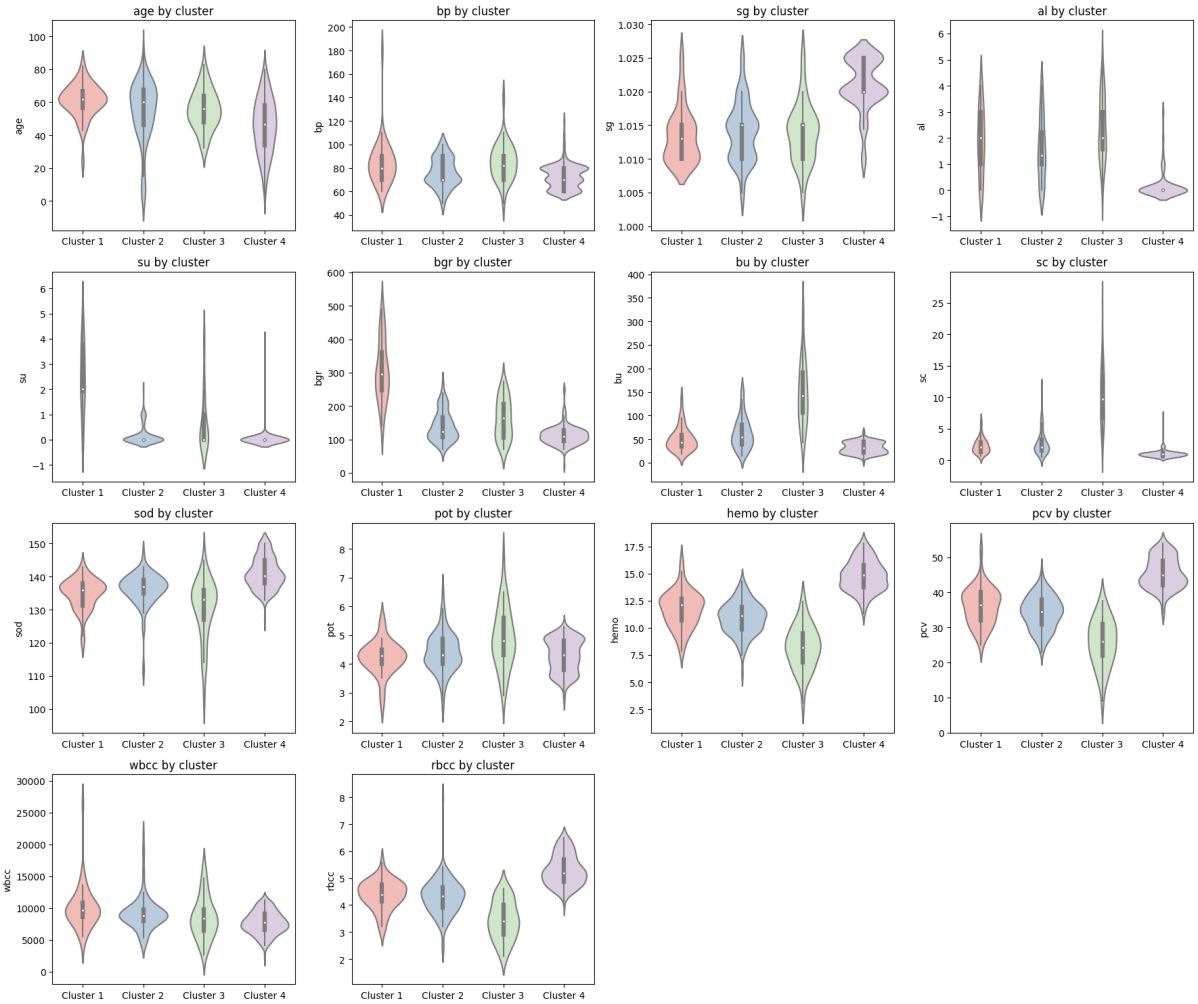


Figure 9: K-Means clusters and PCA components of CKD data

B Appendix B: Jupyter Notebook - Data processing, figures, classification, and clustering

jtweedie_CKD_780final

December 12, 2023

John Tweedie, SN:400550023, 2023-12-12, STATS 780 Final Report

[293]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import KNNImputer
from sklearn.metrics import accuracy_score, mean_squared_error, rand_score,␣
 ↪silhouette_samples, silhouette_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.decomposition import PCA, TruncatedSVD, FactorAnalysis
import statsmodels.api as sm
from statsmodels.tools import add_constant
from statsmodels.stats.outliers_influence import variance_inflation_factor
import warnings
warnings.filterwarnings('ignore')
```

Chronic_Kidney_Disease - Data info:

https://www.researchgate.net/publication/328213160_Early_Stage_Chronic_Kidney_Disease_Diagnosis_by_A

```
age     -   age {numeric}
bp      -   blood pressure {numeric}
sg      -   specific gravity {1.005,1.010,1.015,1.020,1.025}
al      -   albumin {0,1,2,3,4,5} ***
su      -   sugar {0,1,2,3,4,5} ***
rbc     -   red blood cells {normal,abnormal}
pc      -   pus cell {normal,abnormal}
pcc     -   pus cell clumps {normal,abnormal}
ba      -   bacteria {normal,abnormal}
bgr     -   blood glucose random {numeric}
bu      -   blood urea {numeric}
sc      -   serum creatinine {numeric}
sod     -   sodium {numeric}
pot     -   potassium {numeric}
hemo    -   hemoglobin {numeric}
```

```
pcv     -   packed cell volume
wbcc    -   white blood cell count {numeric}
rbcc    -   red blood cell count {numeric}
htn     -   hypertension {yes,no}
dm      -   diabetes mellitus {yes,no}
cad     -   coronary artery disease {yes,no}
appet   -   appetite {good,poor}
pe      -   pedal edema {yes,no}
ane     -   anemia {yes,no}
class   -   class {ckd,notckd}
```

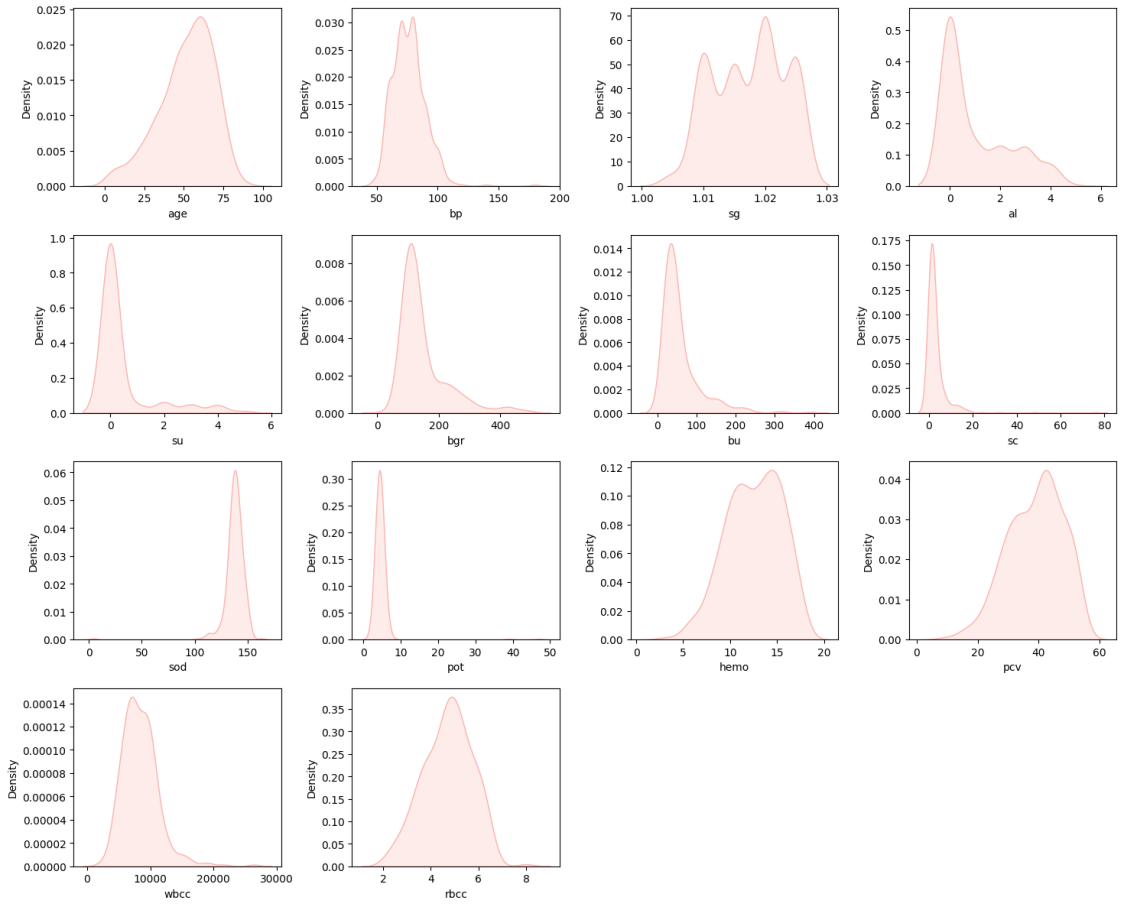
1 Data Processing

[294]:
```
columns =␣
 ↪['age','bp','sg','al','su','rbc','pc','pcc','ba','bgr','bu','sc','sod',
          ␣
 ↪'pot','hemo','pcv','wbcc','rbcc','htn','dm','cad','appet','pe','ane','class']
numeric_cols = ['age','bp','sg','al','su','bgr','bu','sc','sod',
           'pot','hemo','pcv','wbcc','rbcc']
blood_cols = ['bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'rbcc', 'wbcc']
urine_cols = ['sg', 'al', 'su', 'al', 'rbc', 'pc', 'pcc', 'ba']
cat_cols = [cat_col for cat_col in columns if cat_col not in numeric_cols]
```

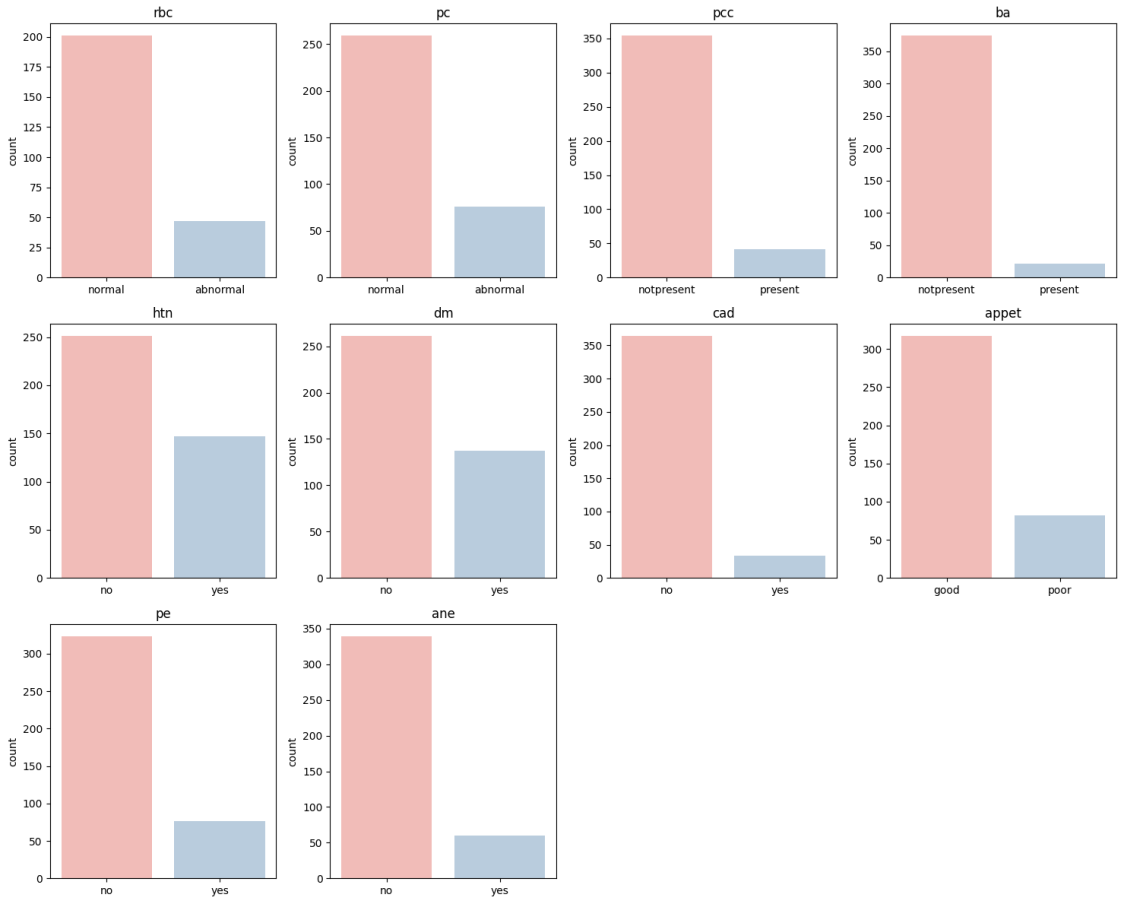
[295]:
```
df = pd.read_csv(r'chronic_kidney_disease.csv', header=None, names=columns)
df[numeric_cols] = df[numeric_cols].apply(pd.to_numeric, errors='coerce')
```

[296]:
```
# data clean-up
df = df.replace('?', np.nan)
df = df.replace('\tno', 'no')
df = df.replace('\tyes', 'yes')
df = df.replace(' yes', 'yes')
```

[297]:
```
# KDE plots to view distributions of data
# we will check for outliers here
sns.set_palette('Pastel1')
fig, axs = plt.subplots(4, 4, figsize=(15, 12))
axs = axs.flatten()
for i, col in enumerate(df[numeric_cols].columns):
    sns.kdeplot(df[col], ax=axs[i], fill=True)
plt.tight_layout()
fig.delaxes(axs[-2])
fig.delaxes(axs[-1])
```

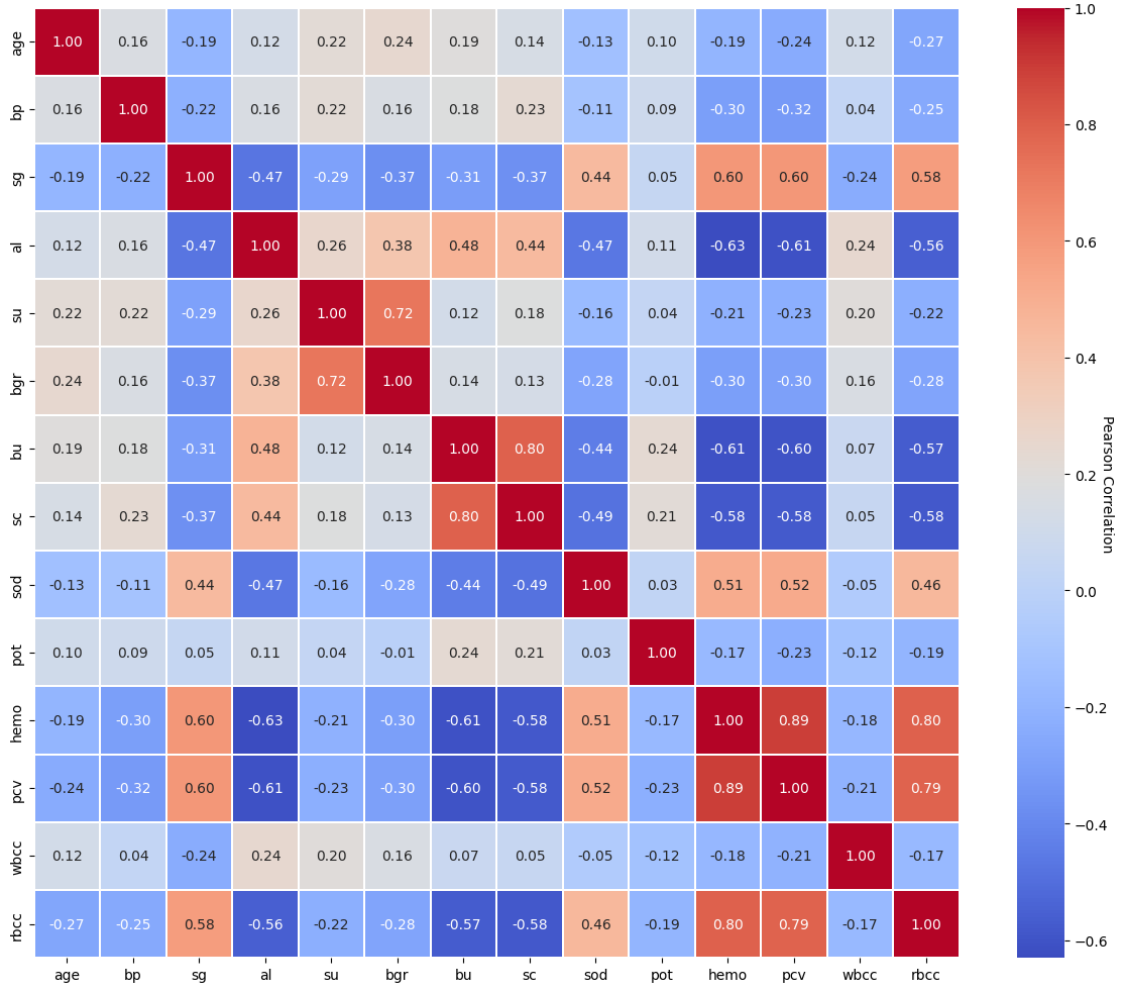


[298]:
```
# Categorical data - bar plots
sns.set_palette('Pastel1')
fig, axs = plt.subplots(3, 4, figsize=(15, 12))
axs = axs.flatten()
for i, col in enumerate(df[cat_cols].columns):
    df_plot = pd.DataFrame(data=df[col].value_counts(), columns=[col])
    sns.barplot(df_plot, x=df_plot.index, y=col, ax=axs[i], errorbar=None)
    axs[i].set_ylabel('count')
    axs[i].set_xlabel('')
    axs[i].set_title(col)
plt.tight_layout()
fig.delaxes(axs[-2])
fig.delaxes(axs[-1])
```



[299]:
```
# Outlier removal from kde and scatter plots
df = df.drop(df['pot'].nlargest(2).index) # remove visual pot outliers
df = df.drop(df['sc'].nlargest(2).index) # remove visual sc outliers
```

[300]:
```
# correlation matrix of all variables
plt.figure(figsize=(12, 10))
sns.heatmap(df[numeric_cols].corr(), annot=True,
            cmap="coolwarm",
            fmt=".2f",
            linewidths=.01)
ax = plt.gca()
grad = ax.collections[0].colorbar
grad.set_label("Pearson Correlation", rotation=270)
plt.tight_layout()
```



1.1 Data Processing - Categorical Data Encoding

[301]:
```
# explicitly specifying the categorical enconding, rather than automatically␣
↪assigning 1/0
# should help with making interpretation easier later on

# creating dictionary to specify what is a '1' or '0'
cat_dict = {'normal':1, 'abnormal':0, 'present':1, 'notpresent':0,
            'yes':1, 'no':0, 'poor':1, 'good':0, 'ckd':1, 'notckd':0}

# replace categorical values using the dictionary
df[cat_cols] = df[cat_cols].replace(cat_dict)

# convert to numeric values
df[cat_cols] = df[cat_cols].apply(pd.to_numeric, errors='coerce')
```

1.2 Data Processing - Imputation of Missing Values

1.2.1 Imputation by K-Nearest Neighbors for Numeric Variables

- Imputing missing numeric variables
- https://scikit-learn.org/stable/modules/impute.html

[302]:
```
imputer = KNNImputer(n_neighbors=9, weights="uniform")
df_imp_num = pd.DataFrame(imputer.fit_transform(df[numeric_cols]),␣
 ↪columns=numeric_cols)
```

1.2.2 Iterative Imputation for categorical variables

- Using SKlearn's iterative imputer
  - performs a multiple regression for each columns with missing data to predict missing values, essentially logit

[303]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

[304]:
```
imputer = IterativeImputer(max_iter=10, random_state=13)
df_imp_cat = pd.DataFrame(np.round(imputer.fit_transform(df)), columns=df.
 ↪columns)
df_imp_cat = df_imp_cat[cat_cols]
```

[305]:
```
df = pd.concat([df_imp_num, df_imp_cat], axis=1)
```

Data is now appropriately imputed and fully populated - ready for modelling

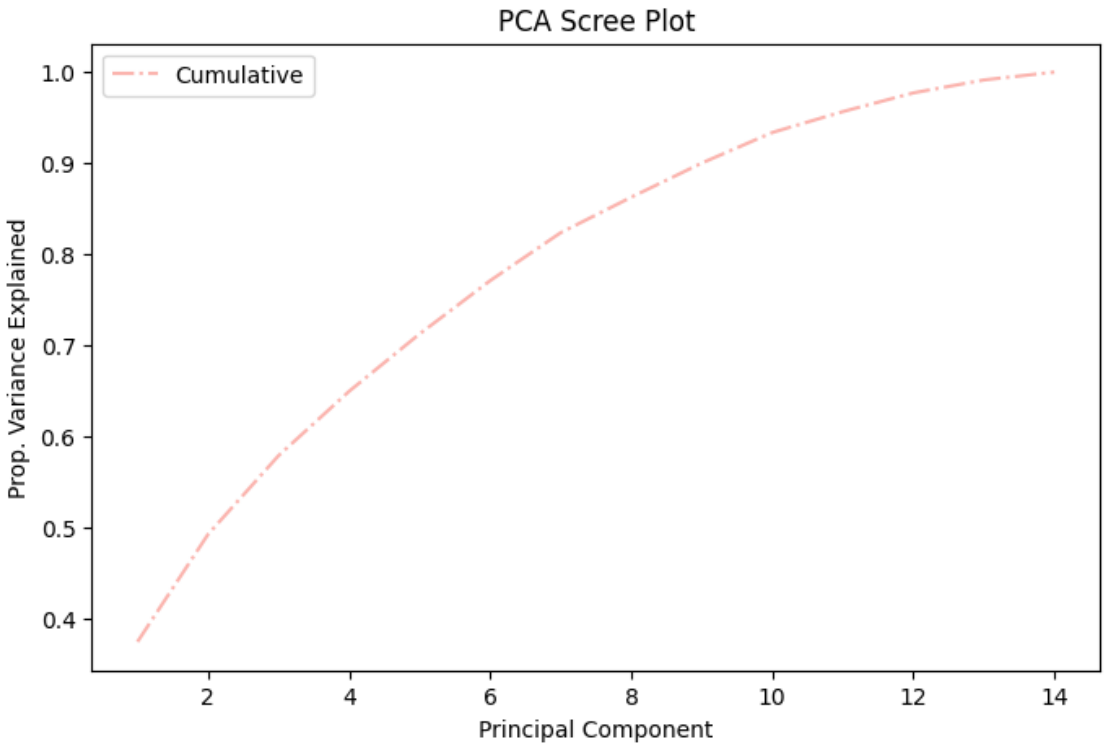
1.3 Principal Components Analysis

[306]:
```
numeric_vars = ['age','bp','sg','al','su','bgr','bu',
                'sc','sod','pot','hemo','pcv','wbcc','rbcc']

scaler = StandardScaler()
# scale the numeric data and load into dataframes
df_scaled = scaler.fit_transform(df[numeric_vars]) # scale numeric only

n_pca_comps = 14
pca = PCA(n_components=n_pca_comps)
pca_fit = pca.fit_transform(df_scaled)
```

[307]:
```
pca_cols = ['PC' + str(i) for i in range(1, 15)]
df_pca = pd.DataFrame(pca_fit, index=df.index, columns=pca_cols)
```

[308]:
```
# Generating a cumulative scree plot
fig , ax = plt.subplots(figsize=(8,5))
prop_var_exp = pca.explained_variance_ratio_
```

```
plt.plot([i for i in range(1, 15)], np.cumsum(prop_var_exp), '-.',␣
 ↪label='Cumulative')
plt.xlabel('Principal Component')
plt.ylabel('Prop. Variance Explained')
plt.title('PCA Scree Plot')
plt.legend()
plt.show()
```



[309]:
```
n_pca_comps = 3 # selecting 3 components
pca = PCA(n_components=n_pca_comps)
pca_fit = pca.fit_transform(df_scaled)
pca_cols = ['PC' + str(i) for i in range(1, n_pca_comps+1)]
df_pca = pd.DataFrame(pca_fit, index=df.index, columns=pca_cols)
```

1.4 Clustering

[310]:
```
# get data - standardized + encoded
df_cluster = pd.concat([(pd.DataFrame(df_scaled, index=df.index,␣
 ↪columns=numeric_vars)),
           df[cat_cols + ['class']]], axis=1)
```

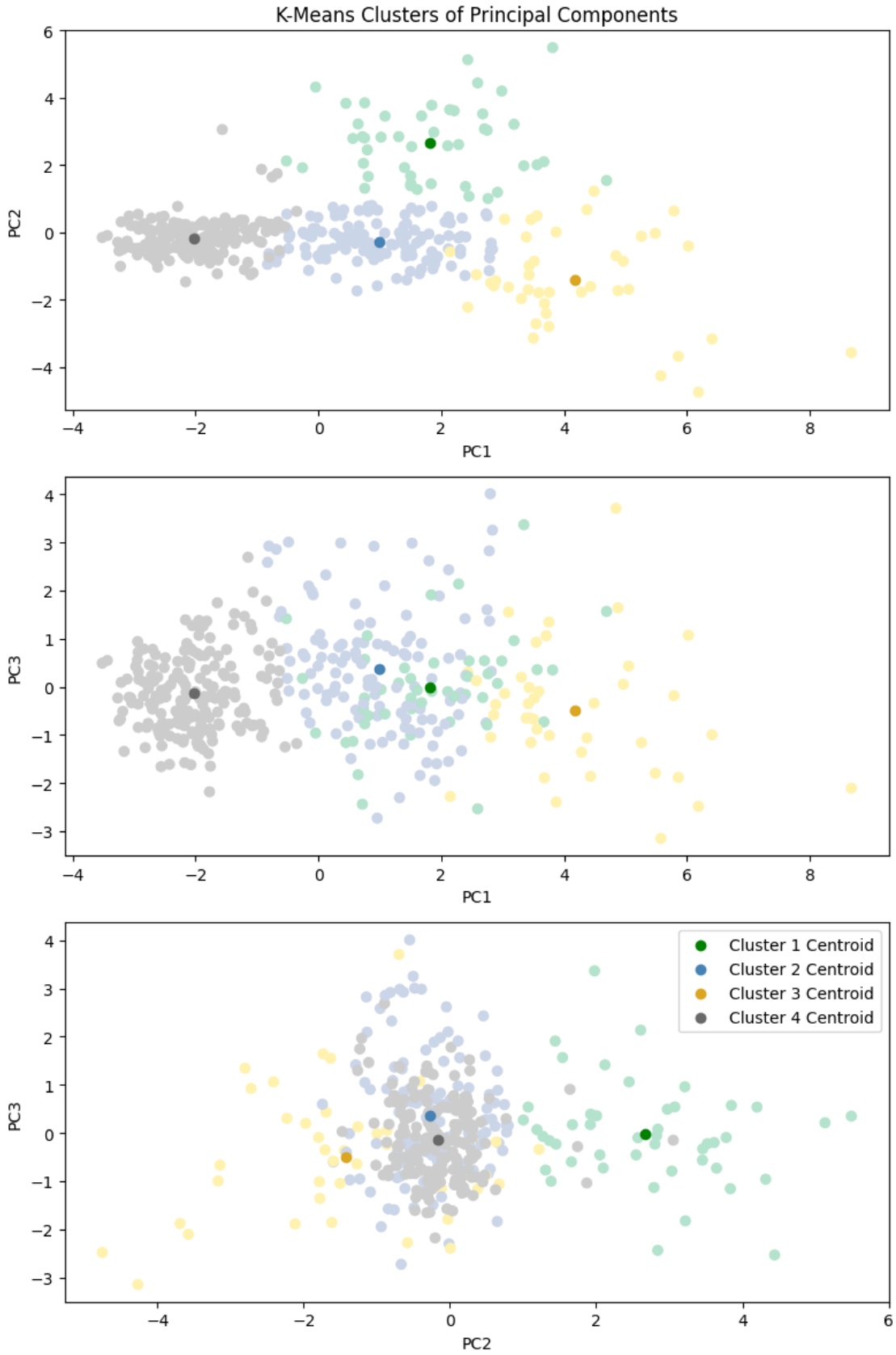
[311]:
```
for n_km_clusters in range(2,7):
    km_cluster = KMeans(n_clusters=n_km_clusters, random_state=13, n_init=10)
#     km_cluster.fit(df_cluster)
#     km_cluster.fit(df)
    km_cluster.fit(df_pca)
    km_labels = km_cluster.labels_ # training set labels
    print('# clusters: ', n_km_clusters,
          ' Sil. Score: ', silhouette_score(df_pca, km_labels).round(2))
```

```
# clusters:  2  Sil. Score:  0.44
# clusters:  3  Sil. Score:  0.39
# clusters:  4  Sil. Score:  0.41
# clusters:  5  Sil. Score:  0.38
# clusters:  6  Sil. Score:  0.36
```

[312]:
```
# 2 clusters has the highest silhouette score
km_cluster = KMeans(n_clusters=4, random_state=13, n_init=10)

# km_cluster.fit(df_cluster)
# km_cluster.fit(df)
km_cluster.fit(df_pca)

km_labels = km_cluster.labels_
km_centers = km_cluster.cluster_centers_

# find centroids of PCA components by cluster for plotting purposes (not true␣
 ↪centroid of cluster)
df_pca['cluster']=km_labels
c1_center = np.mean(df_pca.loc[df_pca['cluster']==0], axis=0)
c2_center = np.mean(df_pca.loc[df_pca['cluster']==1], axis=0)
c3_center = np.mean(df_pca.loc[df_pca['cluster']==2], axis=0)
c4_center = np.mean(df_pca.loc[df_pca['cluster']==3], axis=0)
```

[313]:
```
# plotting clusters by PCA
fig, axs = plt.subplots(3, 1, figsize=(8, 12))
axs = axs.flatten()

axs[0].scatter(df_pca['PC1'], df_pca['PC2'],
        c=km_labels, cmap='Pastel2')
# axs[0].scatter(km_centers[:,0], km_centers[:,1], c='red', label='centroids')
axs[0].scatter(c1_center['PC1'], c1_center['PC2'], c='green', label='Cluster 1␣
 ↪Centroid')
axs[0].scatter(c2_center['PC1'], c2_center['PC2'], c='steelblue',␣
 ↪label='Cluster 2 Centroid')
axs[0].scatter(c3_center['PC1'], c3_center['PC2'], c='goldenrod',␣
 ↪label='Cluster 3 Centroid')
```

```
axs[0].scatter(c4_center['PC1'], c4_center['PC2'], c='dimgrey', label='Cluster␣
 ↪4 Centroid')
axs[0].set_title('K-Means Clusters of Principal Components')
axs[0].set_xlabel('PC1')
axs[0].set_ylabel('PC2')

axs[1].scatter(df_pca['PC1'], df_pca['PC3'],
        c=km_labels, cmap='Pastel2')
# axs[1].scatter(km_centers[:,0], km_centers[:,2], c='red', label='centroids')
axs[1].scatter(c1_center['PC1'], c1_center['PC3'], c='green', label='Cluster 1␣
 ↪Centroid')
axs[1].scatter(c2_center['PC1'], c2_center['PC3'], c='steelblue',␣
 ↪label='Cluster 2 Centroid')
axs[1].scatter(c3_center['PC1'], c3_center['PC3'], c='goldenrod',␣
 ↪label='Cluster 3 Centroid')
axs[1].scatter(c4_center['PC1'], c4_center['PC3'], c='dimgrey', label='Cluster␣
 ↪4 Centroid')
axs[1].set_xlabel('PC1')
axs[1].set_ylabel('PC3')

axs[2].scatter(df_pca['PC2'], df_pca['PC3'],
        c=km_labels, cmap='Pastel2')
# axs[2].scatter(km_centers[:,1], km_centers[:,2], c='red', label='centroids')
axs[2].scatter(c1_center['PC2'], c1_center['PC3'], c='green', label='Cluster 1␣
 ↪Centroid')
axs[2].scatter(c2_center['PC2'], c2_center['PC3'], c='steelblue',␣
 ↪label='Cluster 2 Centroid')
axs[2].scatter(c3_center['PC2'], c3_center['PC3'], c='goldenrod',␣
 ↪label='Cluster 3 Centroid')
axs[2].scatter(c4_center['PC2'], c4_center['PC3'], c='dimgrey', label='Cluster␣
 ↪4 Centroid')
axs[2].set_xlabel('PC2')
axs[2].set_ylabel('PC3')

plt.legend()

plt.tight_layout()
```


K-Means Clusters of Principal Components
PC2
PC1
PC3
PC1
PC3
PC2
Cluster 1 Centroid
Cluster 2 Centroid
Cluster 3 Centroid
Cluster 4 Centroid

1.4.1 Additional Plotting by Cluster

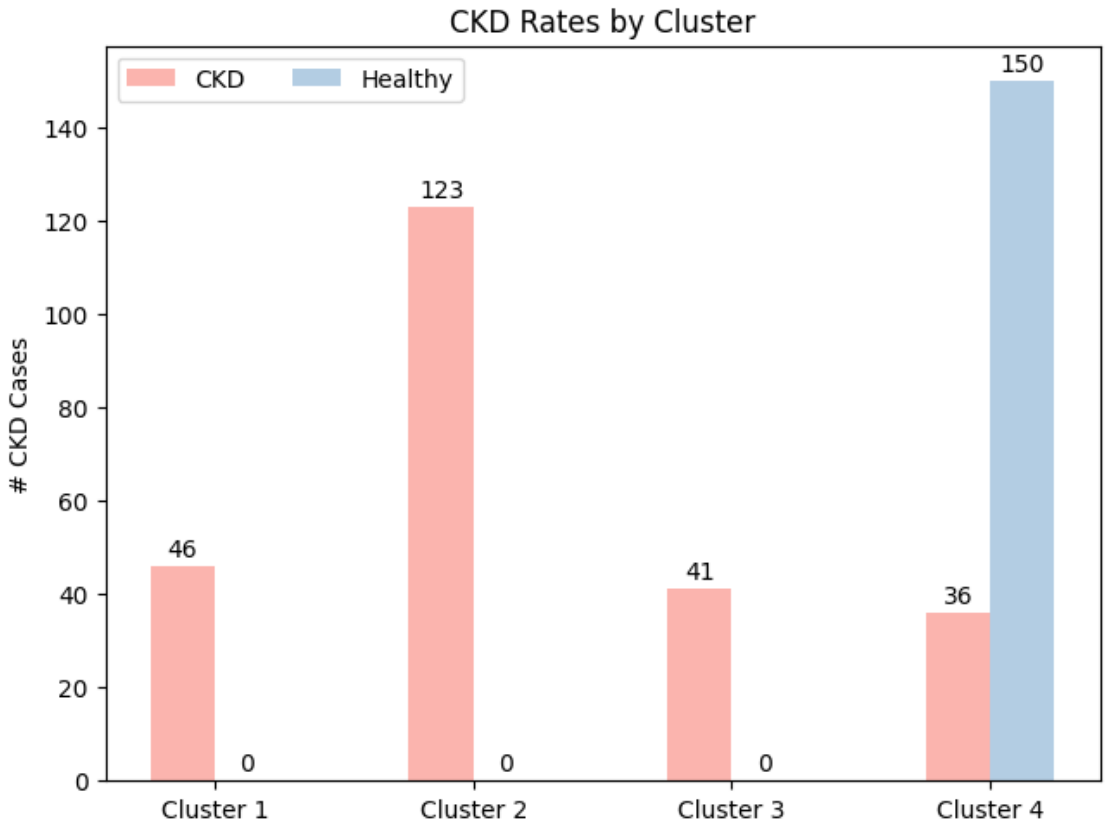
- useful for additional exploratory and cluster analysis/inference

[314]:
```
df_clusters = df.copy() # creating a copy for plotting later
df_clusters['cluster_label'] = km_labels
```

[315]:
```
# find CKD rates within each cluster, populate a dictionary for plotting

c1_ckd = df.loc[(df_clusters['cluster_label'] == 0) &␣
 ↪(df_clusters['class']==1)]['class'].count()
c2_ckd = df.loc[(df_clusters['cluster_label'] == 1) &␣
 ↪(df_clusters['class']==1)]['class'].count()
c3_ckd = df.loc[(df_clusters['cluster_label'] == 2) &␣
 ↪(df_clusters['class']==1)]['class'].count()
c4_ckd = df.loc[(df_clusters['cluster_label'] == 3) &␣
 ↪(df_clusters['class']==1)]['class'].count()

c1_hty = df.loc[(df_clusters['cluster_label'] == 0) &␣
 ↪(df_clusters['class']==0)]['class'].count()
c2_hty = df.loc[(df_clusters['cluster_label'] == 1) &␣
 ↪(df_clusters['class']==0)]['class'].count()
c3_hty = df.loc[(df_clusters['cluster_label'] == 2) &␣
 ↪(df_clusters['class']==0)]['class'].count()
c4_hty = df.loc[(df_clusters['cluster_label'] == 3) &␣
 ↪(df_clusters['class']==0)]['class'].count()

clusters = ['Cluster 1','Cluster 2','Cluster 3','Cluster 4']
plot_dict = {'CKD':(c1_ckd, c2_ckd, c3_ckd, c4_ckd),
           'Healthy':(c1_hty, c2_hty, c3_hty, c4_hty)}

# clusters = ['Cluster 1','Cluster 2']
# plot_dict = {'CKD':(c1_ckd, c2_ckd),
#            'Healthy':(c1_hty, c2_hty)}
```

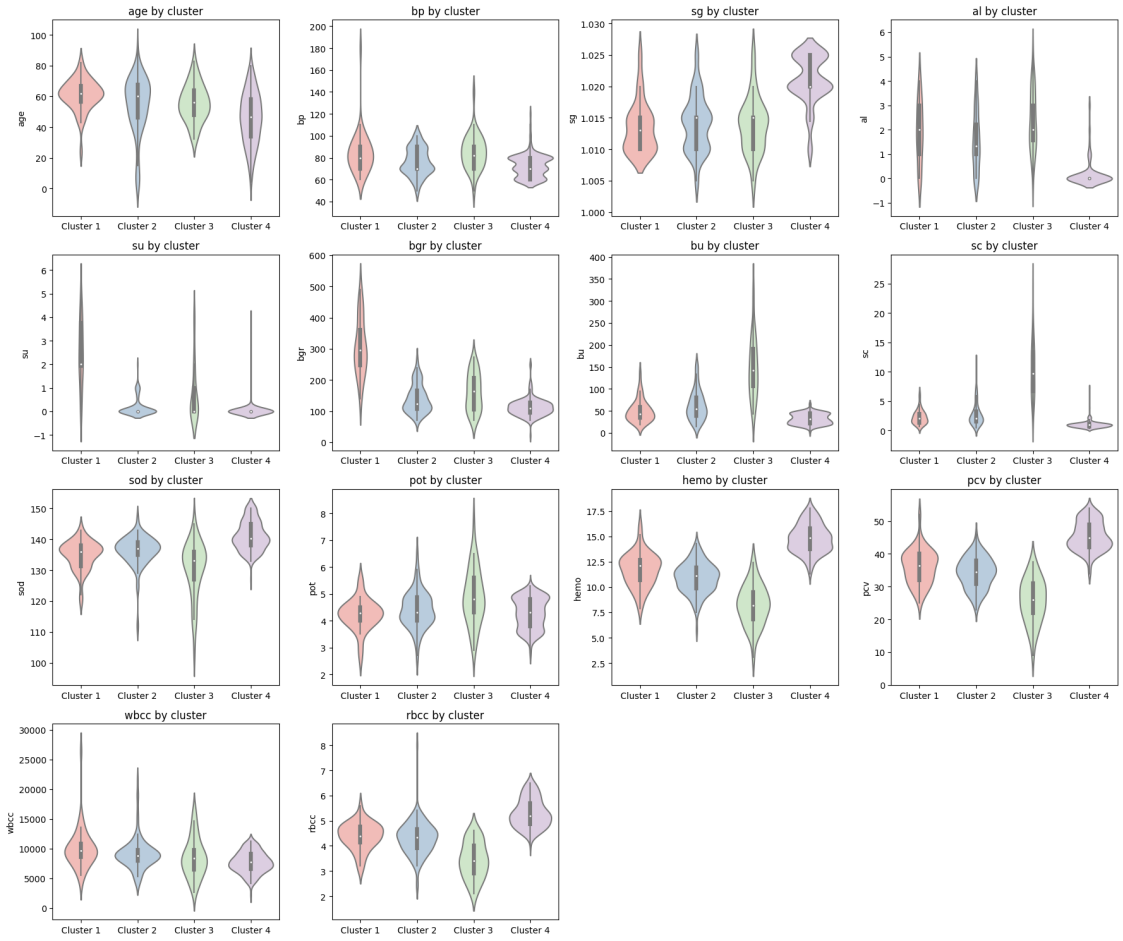
[316]:
```
# bar plot to investigate difference in CKD cases
sns.set_palette('Pastel1')

x = np.arange(len(clusters))
width = 0.25
multiplier = 0.5
fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in plot_dict.items():
```

```
    offset = width * multiplier
    rects = ax.bar(x + offset, measurement, width, label=attribute)
    ax.bar_label(rects, padding=2)
    multiplier += 1

ax.set_ylabel('# CKD Cases')
ax.set_title('CKD Rates by Cluster')
ax.set_xticks(x + width, clusters)
ax.legend(loc='upper left', ncols=2)
plt.show()
```



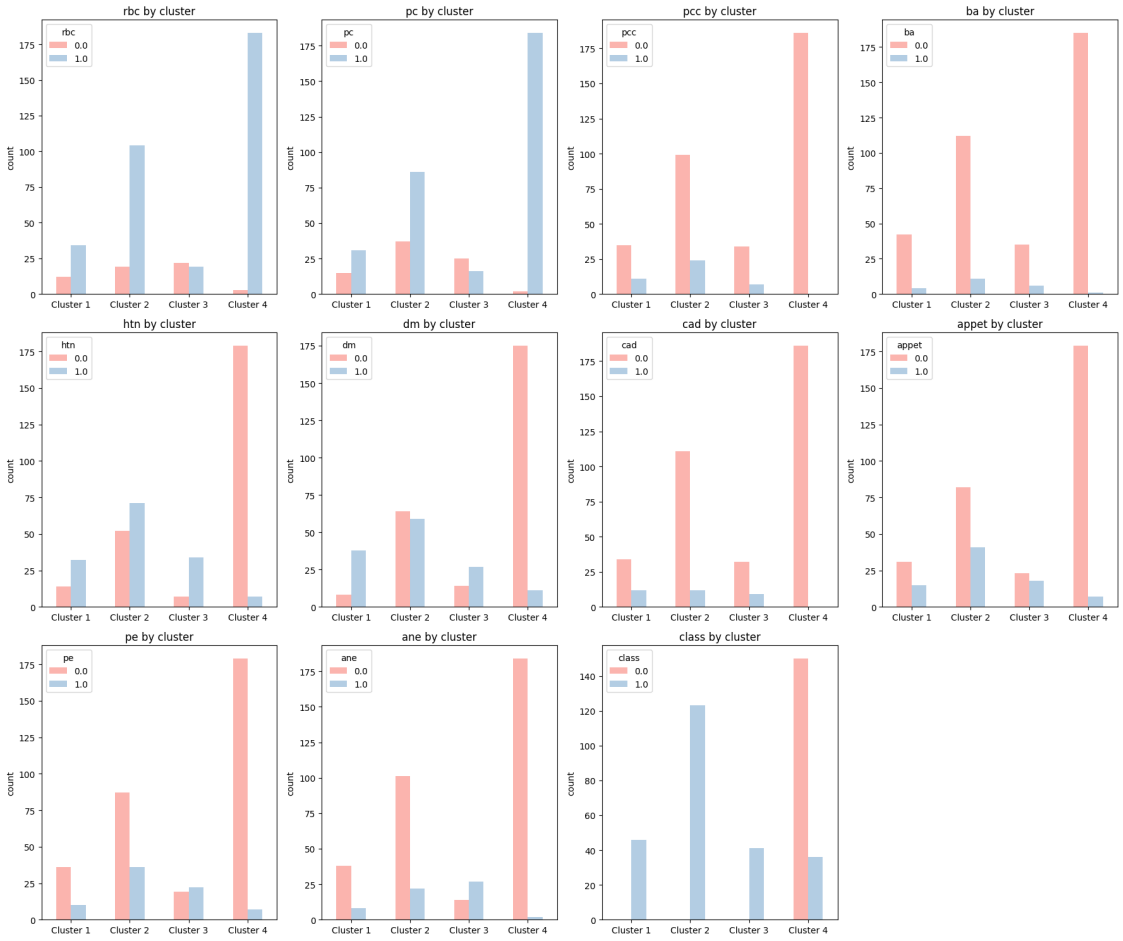
[317]:
```
# create violin plots for each numeric variable, grouped by cluster
sns.set_palette('Pastel1')
fig, axs = plt.subplots(4, 4, figsize=(18, 15))
axs = axs.flatten()
for i, var in enumerate(numeric_cols):
    sns.violinplot(x='cluster_label', y=var, data=df_clusters, ax=axs[i])
    axs[i].set_title('{} by cluster'.format(var))
    axs[i].set_ylabel(var)
    axs[i].set_xticklabels(clusters)
```

```
    axs[i].set_xlabel('')
plt.tight_layout()
fig.delaxes(axs[-2])
fig.delaxes(axs[-1])
plt.show()
```



[318]:
```
# create bar plots for each categorical variable, grouped by cluster

sns.set_palette('Pastel1')
fig, axs = plt.subplots(3, 4, figsize=(18, 15))
axs = axs.flatten()
for i, var in enumerate(cat_cols):
    var_count = df_clusters[cat_cols + ['cluster_label']].
 ↪groupby(['cluster_label', var]).size().unstack()
    var_count.plot(kind='bar', ax=axs[i])
    axs[i].set_title('{} by cluster'.format(var))
    axs[i].set_ylabel('count')
```

```
    axs[i].set_xticklabels(clusters, rotation=0)
    axs[i].set_xlabel('')
plt.tight_layout()
fig.delaxes(axs[-1])
plt.show()
```



1.4.2 Descriptive Statistics of Clusters

[319]:
```
cluster_stats = df_clusters.groupby('cluster_label').agg(['mean', 'median',␣
 ↪'std'])
cluster_stats = np.round(cluster_stats, 2)
# cluster_stats = cluster_stats[cat_cols]
cluster_stats.to_csv('cluster_stats.csv')
```

1.5 Test/Train Segmentation and Data Standardization

[320]:
```
# re-listing out the predictor variable names, and those that are numeric vs.␣
↪categorical

predictors = ['age','bp','sg','al','su','rbc','pc',
              'pcc','ba','bgr','bu','sc','sod','pot',
              'hemo','pcv','wbcc','rbcc','htn','dm',
              'cad','appet','pe','ane']

numeric_vars = ['age','bp','sg','al','su','bgr','bu',
                'sc','sod','pot','hemo','pcv','wbcc','rbcc']

cat_vars = ['rbc', 'pc', 'pcc', 'ba', 'htn',
            'dm', 'cad', 'appet', 'pe', 'ane']
```

[321]:
```
# Two functions to test/train split and standardize
# can pass through different subsets of variables for future feature selection,␣
↪if necessary



def run_test_train_split(df, predictors):
    # function to return test train split from dataframe and predictor subset

    X = df[predictors]
    y = df[['class']].astype('int')

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
↪random_state=100)

    return X_train, X_test, y_train, y_test



def run_standardize(X_train, X_test, numeric_vars, cat_vars):
    # function to scale numeric dataframe columns in test and train sets

    scaler = StandardScaler()

    # scale the numeric data and load into dataframes
    X_train_num_scaled = scaler.fit_transform(X_train[numeric_vars]) # scale␣
↪numeric only
    df_X_train_num_scaled = pd.DataFrame(X_train_num_scaled,␣
↪columns=numeric_vars, index=X_train.index)
    X_train = pd.concat([df_X_train_num_scaled, X_train[cat_vars]], axis=1)
```

```
    X_test_num_scaled = scaler.transform(X_test[numeric_vars]) # scale numeric␣
 ↪only, using fit from train
    df_X_test_num_scaled = pd.DataFrame(X_test_num_scaled,␣
 ↪columns=numeric_vars, index=X_test.index)
    X_test = pd.concat([df_X_test_num_scaled, X_test[cat_vars]], axis=1).
 ↪dropna()

    X_train = add_constant(X_train)
    X_test = add_constant(X_test)

    return X_train, X_test



X_train, X_test, y_train, y_test = run_test_train_split(df, predictors)
X_train, X_test = run_standardize(X_train, X_test, numeric_vars, cat_vars)
```

2 Logistic Regression Classification

[322]:
```
# function to print out model logistic performance (missclassification, AUC,␣
 ↪cross-val...)

def model_performance(model, features):

    if 'PC1' in features:
        X_test_fs = X_test_pca[features]
    else:
        X_test_fs = X_test[features] # feature selected X_test set
    y_prob = model.predict(X_test_fs)

    try:
        y_pred = round(y_prob)
        con_mtx = confusion_matrix(y_test, y_pred)
        clf_report = classification_report(y_test, y_pred)

    except:
        con_mtx = confusion_matrix(y_test, y_prob)
        clf_report = classification_report(y_test, y_prob)

    print('\nCross Validation Results for Chronic Kidney Disease (+) vs.␣
 ↪Healthy (-) Precitions:')
     ␣
 ↪print('=================================================================================
    print('\nTrue positives:', con_mtx[1][1])
    print('False positives:', con_mtx[1][0])
    print('True negatives:', con_mtx[0][0])
```

```
    print('False negatives:', con_mtx[0][1])
    print('Precision: ', round((con_mtx[1][1]/
 ↪(con_mtx[1][1]+con_mtx[1][0])),3)) # ratio of true+ to total predicted
    print('Sensitivity: ', round((con_mtx[1][1]/
 ↪(con_mtx[1][1]+con_mtx[0][1])),3)) # TP / (TP + FN)
    print('Specificity: ', round((con_mtx[0][0]/
 ↪(con_mtx[0][0]+con_mtx[1][0])),3)) # TN / (TN + FP)
    print('Accuracy: ', round((con_mtx[1][1]+con_mtx[0][0])/len(X_test),3))
    print('Missclass. Rate: ', round((con_mtx[0][1]+con_mtx[1][0])/
 ↪len(X_test),3))

    print('\nclf report: \n')
    print(clf_report)
```

Feature Selection - VIF

- Checks for multi-colinearity and avoids singular matrix

[323]:
```
df_vif = pd.DataFrame()
df_vif['Predictor'] = X_train.columns
df_vif['VIF'] = [variance_inflation_factor(X_train, i) for i in␣
 ↪range(len(X_train.columns))]
```

[324]:
```
vif_drops = ['hemo', 'pcv', 'const'] # high VIF
X_train = X_train.drop(columns=vif_drops)
X_test = X_test.drop(columns=vif_drops)
```

[325]:
```
# Fitting the full initial logisitic classsifier model
clf_lg = sm.Logit(y_train, X_train).fit(method='bfgs')
# Evaluate performance
# model_performance(clf_lg, X_train.columns)
```

```
        Current function value: 0.002622
        Iterations: 35
        Function evaluations: 36
        Gradient evaluations: 36
```

2.0.1 Feature Selection - Stepwise

[326]:
```
# remove features based on the model fit above
fs1_drops = ['cad', 'pcc', 'pc', 'su',
             'ba', 'ane', 'rbc', 'pe', 'sod', 'pot'] # feature selection 1 drop␣
 ↪columns
X_train = X_train.drop(columns=fs1_drops)
X_test = X_test.drop(columns=fs1_drops)
```

[327]:
```
# Fitting the full initial logisitic classsifier model
clf_lg = sm.Logit(y_train, X_train).fit(method='bfgs')
# print(clf_lg.summary())
```

```
Current function value: 0.040994
Iterations: 35
Function evaluations: 36
Gradient evaluations: 36
```

2.0.2 Outlier Test - High Leverage Observations

[328]:
```
# find high leverage points
lvg = clf_lg.get_influence().hat_matrix_diag
```

[329]:
```
# define cutoff: 'greatly exceeding' (p+1)/n (ISLP, 2023)
cutoff = (len(X_train.columns)+1) / len(lvg)
indexes = np.where(lvg > 13 * cutoff) # we will use 13 times the above, which␣
 ↪detects 4 outliers
```

[330]:
```
# drop high leverage points from training set
X_train = X_train.drop(X_train.index[indexes])
y_train = y_train.drop(y_train.index[indexes])
```

2.0.3 Final 'Baseline' Logistic Regression Model

[331]:
```
# Fitting the full initial logisitic classsifier model
clf_lg = sm.Logit(y_train, X_train).fit(method='bfgs')
print(clf_lg.summary())

# Evaluate performance
model_performance(clf_lg, X_train.columns)
```

```
Current function value: 0.028616
Iterations: 35
Function evaluations: 36
Gradient evaluations: 36
                           Logit Regression Results
==============================================================================
Dep. Variable:                  class   No. Observations:                  233
Model:                          Logit   Df Residuals:                      221
Method:                           MLE   Df Model:                           11
Date:                Tue, 12 Dec 2023   Pseudo R-squ.:                  0.9576
Time:                        20:22:41   Log-Likelihood:                -6.6676
converged:                      False   LL-Null:                       -157.13
Covariance Type:            nonrobust   LLR p-value:                5.591e-58
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
age           -4.4795      2.848     -1.573      0.116     -10.062       1.103
bp            -0.8134      1.363     -0.597      0.551      -3.484       1.857
sg            -6.5028      3.004     -2.164      0.030     -12.392      -0.614
al             3.4466      2.964      1.163      0.245      -2.364       9.257
bgr            1.6918      1.782      0.949      0.342      -1.801       5.184
bu            -5.1358      3.636     -1.413      0.158     -12.262       1.990
sc             5.7321      4.074      1.407      0.159      -2.253      13.717
wbcc           1.4997      2.310      0.649      0.516      -3.028       6.027
rbcc          -4.4551      3.336     -1.335      0.182     -10.994       2.083
htn           18.6020    192.869      0.096      0.923    -359.414     396.618
dm            19.0840    119.265      0.160      0.873    -214.671     252.839
appet         18.1213    102.232      0.177      0.859    -182.249     218.492
================================================================================

Possibly complete quasi-separation: A fraction 0.76 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.

Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 102
False positives: 1
True negatives: 55
False negatives: 1
Precision:  0.99
Sensitivity:  0.99
Specificity:  0.982
Accuracy:  0.987
Missclass. Rate:  0.013

clf report:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98        56
           1       0.99      0.99      0.99       103

    accuracy                           0.99       159
   macro avg       0.99      0.99      0.99       159
weighted avg       0.99      0.99      0.99       159
```

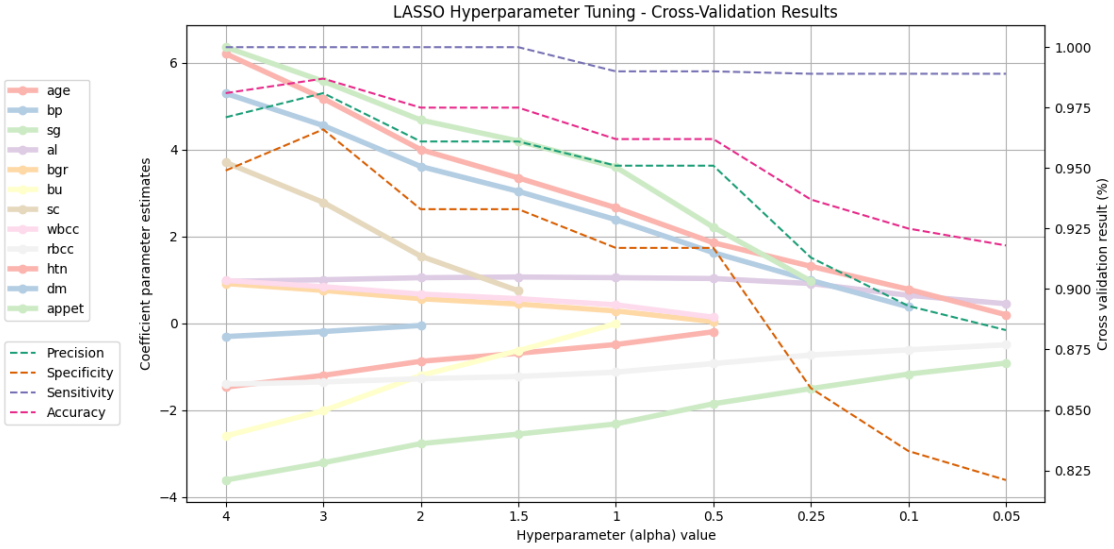
2.1 LASSO Feature Selection for Logit

[332]:
```
# fit lasso_model (same as above)
lasso_model = sm.Logit(y_train, X_train)
# print(clf_lg.summary())
```

[333]:
```
# We will perform LASSO regularization to shrink the regression coefficients
# Several tuning parameters (penalty coefficients) will be evaluated to select␣
 ↪the best-performing model

# list of all tuning parameters to test, full coefficient shrinkage begins at␣
 ↪~0.005
Cs = [4, 3, 2, 1.5, 1, 0.5, 0.25, 0.1, 0.05]

# initialize dataframes
lasso_coefs = pd.DataFrame({'Feature': X_train.columns})
lasso_cv = pd.DataFrame()
lasso_model = sm.Logit(y_train, X_train)
X_test = X_test[X_train.columns]

# itterate through tuning paramters, store results in initialized dataframes
for c in Cs:

    clf_lasso = lasso_model.fit_regularized(method='l1', L1_wt=1, alpha=(1/c),␣
 ↪disp=False) # alpha is the (inverse) penalty term
    y_prob = clf_lasso.predict(X_test)
    y_pred = round(y_prob)
    con_mtx = confusion_matrix(y_test, y_pred)

    lasso_coefs['coef_C={}'.format(c)] = clf_lasso.params.values

    # calculate cross validation metrics
    Precision = round((con_mtx[1][1]/(con_mtx[1][1]+con_mtx[1][0])),3)
    Specificity = round((con_mtx[0][0]/(con_mtx[0][0]+con_mtx[1][0])),3)
    Sensitivity = round((con_mtx[1][1]/(con_mtx[1][1]+con_mtx[0][1])),3) # TP /␣
 ↪(TP + FN)
    Accuracy = round((con_mtx[1][1]+con_mtx[0][0])/len(X_test),3)
    Missclass = round((con_mtx[0][1]+con_mtx[1][0])/len(X_test),3)

    results_list = [c, con_mtx[1][1], con_mtx[1][0], con_mtx[0][0],
                    con_mtx[0][1], Precision, Specificity, Sensitivity,␣
 ↪Accuracy, Missclass]

    # store cross validation metrics in dataframe
    lasso_cv = pd.concat([lasso_cv, pd.DataFrame([results_list])],␣
 ↪ignore_index=True)
```

```
columns=['Tuning Parameter', 'True Positives', 'False Positives', 'True␣
 ↪Negatives',
        'False Negatives', 'Precision', 'Specificity', 'Sensitivity',␣
 ↪'Accuracy', 'Missclass Rate']
lasso_cv = lasso_cv.rename(columns=dict(zip(lasso_cv.columns, columns)))
lasso_coefs = lasso_coefs.set_index('Feature')
```

[334]:
```
sns.set_palette('Pastel1')
plt.figure(figsize=(12, 6))
for col, row in lasso_coefs.replace(0, np.nan).iterrows():
    plt.plot(row, label=col, linewidth=4, marker='o')
plt.legend(loc='upper left', bbox_to_anchor=(-0.22, 0.9))
ax1 = plt.gca()
plt.grid(True)

sns.set_palette('Dark2')
ax2 = plt.gca().twinx()
for col, row in lasso_cv[['Precision', 'Specificity', 'Sensitivity',␣
 ↪'Accuracy']].T.iterrows():
    ax2.plot(row, linestyle='dashed', label=col)
plt.legend(loc='upper left', bbox_to_anchor=(-0.22, 0.35))
plt.subplots_adjust(left=0.1, right=0.85, top=0.9, bottom=0.1, wspace=0.3,␣
 ↪hspace=0.3)

plt.title('LASSO Hyperparameter Tuning - Cross-Validation Results')
ax2.set_ylabel('Cross validation result (%)')
ax1.set_xticklabels(Cs)
ax1.set_ylabel('Coefficient parameter estimates')
ax1.set_xlabel('Hyperparameter (alpha) value')
plt.tight_layout()
plt.show()
```



[335]:
```
# lasso_cv
```

Based on the LASSO results, we will employ a coefficient of 1.5 to achieve the best trade-off between dimensionality and accuracy, and minimizing false negative rate. However, there is no complete shrinkage of a coefficient without incurring additional false negatives

[336]:
```
lasso_coefs = pd.DataFrame({'Feature': X_train.columns})
lasso_cv = pd.DataFrame()
clf_lasso = lasso_model.fit_regularized(method='l1', L1_wt=1, alpha=(1/1.5),␣
 ↪disp=False) # alpha is the (inverse) penalty term
y_prob = clf_lasso.predict(X_test)
y_pred = round(y_prob)
con_mtx = confusion_matrix(y_test, y_pred)

lasso_coefs['coefficients'] = clf_lasso.params.values

# calculate cross validation metrics
Precision = round((con_mtx[1][1]/(con_mtx[1][1]+con_mtx[1][0])),3)
Specificity = round((con_mtx[0][0]/(con_mtx[0][0]+con_mtx[1][0])),3)
Sensitivity = round((con_mtx[1][1]/(con_mtx[1][1]+con_mtx[0][1])),3) # TP / (TP␣
 ↪+ FN)
Accuracy = round((con_mtx[1][1]+con_mtx[0][0])/len(X_test),3)
Missclass = round((con_mtx[0][1]+con_mtx[1][0])/len(X_test),3)

results_list = [c, con_mtx[1][1], con_mtx[1][0], con_mtx[0][0],
                con_mtx[0][1], Precision, Specificity, Sensitivity, Accuracy,␣
 ↪Missclass]

# store cross validation metrics in dataframe
lasso_cv = pd.concat([lasso_cv, pd.DataFrame([results_list])],␣
 ↪ignore_index=True)

columns=['Tuning Parameter', 'True Positives', 'False Positives', 'True␣
 ↪Negatives',
    'False Negatives', 'Precision', 'Specificity', 'Sensitivity', 'Accuracy',␣
 ↪'Missclass Rate']
lasso_cv = lasso_cv.rename(columns=dict(zip(lasso_cv.columns, columns)))
```

[337]:
```
model_performance(clf_lasso, X_test.columns)
final_model = pd.DataFrame({'Predictor': X_train.columns, 'Coefficient':␣
 ↪clf_lasso.params.values,
                            'p-value': clf_lasso.pvalues,
                             'VIF': [variance_inflation_factor(X_test, i) for i␣
 ↪in range(len(X_test.columns))]})
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 99
False positives: 4
True negatives: 56
False negatives: 0
Precision:  0.961
Sensitivity:  1.0
Specificity:  0.933
Accuracy:  0.975
Missclass. Rate:  0.025

clf report:

              precision    recall  f1-score   support

           0       0.93      1.00      0.97        56
           1       1.00      0.96      0.98       103

    accuracy                           0.97       159
   macro avg       0.97      0.98      0.97       159
weighted avg       0.98      0.97      0.98       159
```

[338]:
```
final_model # investigate coefficients
```

[338]:

	Predictor	Coefficient	p-value	VIF
age	age	-0.686739	0.120526	1.169383
bp	bp	0.000000	NaN	1.243678
sg	sg	-2.549323	0.000029	1.681317
al	al	1.067900	0.080782	1.635674
bgr	bgr	0.444929	0.542992	1.337049
bu	bu	-0.628937	0.610653	2.918202
sc	sc	0.751631	0.643957	2.477935
wbcc	wbcc	0.568557	0.252254	1.199250
rbcc	rbcc	-1.223918	0.089333	2.351167
htn	htn	3.349687	0.037534	2.948019
dm	dm	3.039868	0.049210	2.737267
appet	appet	4.199153	0.008284	1.605580

2.2 Logistic Regression using PCA components

[339]:
```
# prepare test/train sets with PCA data

X_pca_cat = pd.concat([df_pca.drop('cluster', axis=1), df[cat_vars]], axis=1)
y = df[['class']].astype('int')
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca_cat, y,␣
 ↪test_size=0.4, random_state=100)
```

[340]:
```
# Fitting the logisitic classsifier model using PCA components and binarys
clf_lg = sm.Logit(y_train, X_train_pca).fit(method='bfgs')
#print(clf_lg.summary())


# Evaluate performance
# model_performance(clf_lg, clf_lg.params.index)
```

```
Current function value: 0.029428
Iterations: 35
Function evaluations: 36
Gradient evaluations: 36
```

[341]:
```
fs1_drops = ['cad', 'dm', 'htn', 'ba', 'pe', 'ane'] # feature selection 1 drop␣
 ↪columns
X_train_pca = X_train_pca.drop(columns=fs1_drops)
X_test_pca = X_test_pca.drop(columns=fs1_drops)
```

[342]:
```
# Fit 2
clf_lg = sm.Logit(y_train, X_train_pca).fit(method='bfgs')
#print(clf_lg.summary())

# Evaluate performance
# model_performance(clf_lg, X_train_pca.columns)
```

```
Current function value: 0.082415
Iterations: 35
Function evaluations: 36
Gradient evaluations: 36
```

[343]:
```
fs1_drops = ['pc', 'pcc', 'appet', 'rbc'] # feature selection 1 drop columns
X_train_pca = X_train_pca.drop(columns=fs1_drops)
X_test_pca = X_test_pca.drop(columns=fs1_drops)
```

[344]:
```
# Final PCA fit
clf_lg = sm.Logit(y_train, X_train_pca).fit(method='bfgs')
print(clf_lg.summary())

# Evaluate performance
model_performance(clf_lg, X_train_pca.columns)
```

```
Optimization terminated successfully.
         Current function value: 0.277946
         Iterations: 16
         Function evaluations: 17
         Gradient evaluations: 17
                           Logit Regression Results
==============================================================================
Dep. Variable:                   class   No. Observations:                  237
Model:                           Logit   Df Residuals:                      234
Method:                            MLE   Df Model:                            2
Date:                Tue, 12 Dec 2023   Pseudo R-squ.:                  0.5862
Time:                        20:22:42   Log-Likelihood:                -65.873
converged:                       True   LL-Null:                       -159.17
Covariance Type:            nonrobust   LLR p-value:                 3.021e-41
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
PC1            1.4412      0.183      7.878      0.000       1.083       1.800
PC2            0.4245      0.214      1.986      0.047       0.006       0.843
PC3            0.9025      0.246      3.664      0.000       0.420       1.385
==============================================================================

Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 87
False positives: 16
True negatives: 56
False negatives: 0
Precision:  0.845
Sensitivity:  1.0
Specificity:  0.778
Accuracy:  0.899
Missclass. Rate:  0.101

clf report:

              precision    recall  f1-score   support

           0       0.78      1.00      0.88        56
           1       1.00      0.84      0.92       103

    accuracy                           0.90       159
   macro avg       0.89      0.92      0.90       159
weighted avg       0.92      0.90      0.90       159
```

3 Support Vector Machine Classification

[345]:
```
from sklearn import svm
```

[346]:
```
# reset test/train dataframe
X_train, X_test, y_train, y_test = run_test_train_split(df, predictors)
X_train, X_test = run_standardize(X_train, X_test, numeric_vars, cat_vars)

# fit SVM classifier
clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(X_train, y_train['class'])
clf_svm.predict(X_test)

# cross-validation
model_performance(clf_svm, X_test.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 103
False positives: 0
True negatives: 55
False negatives: 1
Precision:  1.0
Sensitivity:  0.99
Specificity:  1.0
Accuracy:  0.994
Missclass. Rate:  0.006

clf report:

              precision    recall  f1-score   support

           0       1.00      0.98      0.99        56
           1       0.99      1.00      1.00       103

    accuracy                           0.99       159
   macro avg       1.00      0.99      0.99       159
weighted avg       0.99      0.99      0.99       159
```

The SVM acheives near-perfect classification on the test set. We will test the accuracy of the SVM, but with reduced dimensions

3.0.1 SVM using LASSO-selected features from L.R. model

- we will use the features that remained in the final LASSO logistic regression model to compare performance

[347]:
```
coefs = final_model.dropna()
coefs = coefs['Coefficient']
fs_cols = coefs.index
```

[348]:
```
X_train_fs1 = X_train[fs_cols]
X_test_fs1 = X_test[fs_cols]
```

[349]:
```
clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(X_train_fs1, y_train['class'])
clf_svm.predict(X_test_fs1)
model_performance(clf_svm, X_test_fs1.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 102
False positives: 1
True negatives: 55
False negatives: 1
Precision:  0.99
Sensitivity:  0.99
Specificity:  0.982
Accuracy:  0.987
Missclass. Rate:  0.013

clf report:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98        56
           1       0.99      0.99      0.99       103

    accuracy                           0.99       159
   macro avg       0.99      0.99      0.99       159
weighted avg       0.99      0.99      0.99       159
```

We have achieved similar, but slightly worse accuracy as the logistic regression with LASSO feature selection, while using the same set of predictor variables given from the LASSO application

3.0.2 SVM using PCA

[350]:
```
clf_svm.fit(X_train_pca, y_train['class'])
clf_svm.predict(X_test_pca)
model_performance(clf_svm, X_train_pca.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 102
False positives: 1
True negatives: 56
False negatives: 0
Precision:  0.99
Sensitivity:  1.0
Specificity:  0.982
Accuracy:  0.994
Missclass. Rate:  0.006

clf report:

              precision    recall  f1-score   support

           0       0.98      1.00      0.99        56
           1       1.00      0.99      1.00       103

    accuracy                           0.99       159
   macro avg       0.99      1.00      0.99       159
weighted avg       0.99      0.99      0.99       159
```

[351]:
```
# radial and polynomial models - identical/minimially different predictions

clf_svm = svm.SVC(kernel='rbf')
clf_svm.fit(X_train, y_train['class'])
clf_svm.predict(X_test)
model_performance(clf_svm, X_test.columns)

clf_svm = svm.SVC(kernel='poly')
clf_svm.fit(X_train, y_train['class'])
clf_svm.predict(X_test)
model_performance(clf_svm, X_test.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
```

```
================================================================================
====

True positives: 103
False positives: 0
True negatives: 55
False negatives: 1
Precision:  1.0
Sensitivity:  0.99
Specificity:  1.0
Accuracy:  0.994
Missclass. Rate:  0.006

clf report:

              precision    recall  f1-score   support

           0       1.00      0.98      0.99        56
           1       0.99      1.00      1.00       103

    accuracy                           0.99       159
   macro avg       1.00      0.99      0.99       159
weighted avg       0.99      0.99      0.99       159


Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 103
False positives: 0
True negatives: 54
False negatives: 2
Precision:  1.0
Sensitivity:  0.981
Specificity:  1.0
Accuracy:  0.987
Missclass. Rate:  0.013

clf report:

              precision    recall  f1-score   support

           0       1.00      0.96      0.98        56
           1       0.98      1.00      0.99       103

    accuracy                           0.99       159
```

```
   macro avg       0.99      0.98      0.99       159
weighted avg       0.99      0.99      0.99       159
```

4 Blood and Urinalysis-only models

4.0.1 L.R. for blood test only

[352]:
```
X_train_blood = X_train[blood_cols]
X_test_blood = X_test[blood_cols]

clf_lg = sm.Logit(y_train, X_train_blood).fit(method='bfgs')
# print(clf_lg.summary())

# Evaluate performance
model_performance(clf_lg, X_train_blood.columns)
```

```
         Current function value: 0.287539
         Iterations: 35
         Function evaluations: 36
         Gradient evaluations: 36

Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 81
False positives: 22
True negatives: 56
False negatives: 0
Precision:  0.786
Sensitivity:  1.0
Specificity:  0.718
Accuracy:  0.862
Missclass. Rate:  0.138

clf report:

              precision    recall  f1-score   support

           0       0.72      1.00      0.84        56
           1       1.00      0.79      0.88       103

    accuracy                           0.86       159
   macro avg       0.86      0.89      0.86       159
weighted avg       0.90      0.86      0.86       159
```

4.0.2 L.R. for urinalysis only

[353]:
```
X_train_urine = X_train[urine_cols]
X_test_urine = X_test[urine_cols]

clf_lg = sm.Logit(y_train, X_train_urine).fit(method='bfgs')
# print(clf_lg.summary())

# Evaluate performance
model_performance(clf_lg, X_train_urine.columns)
```

```
         Current function value: 0.192927
         Iterations: 35
         Function evaluations: 36
         Gradient evaluations: 36

Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 97
False positives: 6
True negatives: 54
False negatives: 2
Precision:  0.942
Sensitivity:  0.98
Specificity:  0.9
Accuracy:  0.95
Missclass. Rate:  0.05

clf report:

              precision    recall  f1-score   support

           0       0.90      0.96      0.93        56
           1       0.98      0.94      0.96       103

    accuracy                           0.95       159
   macro avg       0.94      0.95      0.95       159
weighted avg       0.95      0.95      0.95       159
```

4.0.3 SVM for blood test only

[354]:
```
clf_svm.fit(X_train_blood, y_train['class'])
clf_svm.predict(X_test_blood)
model_performance(clf_svm, X_test_blood.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 101
False positives: 2
True negatives: 41
False negatives: 15
Precision:  0.981
Sensitivity:  0.871
Specificity:  0.953
Accuracy:  0.893
Missclass. Rate:  0.107

clf report:

              precision    recall  f1-score   support

           0       0.95      0.73      0.83        56
           1       0.87      0.98      0.92       103

    accuracy                           0.89       159
   macro avg       0.91      0.86      0.88       159
weighted avg       0.90      0.89      0.89       159
```

4.0.4 SVM for urinalysis only

[355]:
```
X_train_urine = X_train[urine_cols]
clf_svm.fit(X_train_urine, y_train['class'])
X_test_urine = X_test[urine_cols]
clf_svm.predict(X_test_urine)
model_performance(clf_svm, X_test_urine.columns)
```

```
Cross Validation Results for Chronic Kidney Disease (+) vs. Healthy (-)
Precitions:
================================================================================
====

True positives: 101
```

```
False positives: 2
True negatives: 54
False negatives: 2
Precision:  0.981
Sensitivity:  0.981
Specificity:  0.964
Accuracy:  0.975
Missclass. Rate:  0.025

clf report:

              precision    recall  f1-score   support

           0       0.96      0.96      0.96        56
           1       0.98      0.98      0.98       103

    accuracy                           0.97       159
   macro avg       0.97      0.97      0.97       159
weighted avg       0.97      0.97      0.97       159
```

[356]:
```
# complete
```