Final Project

Jenna Orvitz, Noah Ripstein (400311716), Viransh Shah (400394334)

2024-04-18

```
!pip3 install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in c:\users\viran\anaconda3\envs\proj02\lib\site-packa
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
import seaborn as sns
from patsy import dmatrices, dmatrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
import statsmodels.api as sm
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
from ucimlrepo import fetch_ucirepo

chronic_kidney_disease = fetch_ucirepo(id=336)

df = pd.concat([chronic_kidney_disease.data.features, chronic_kidney_disease.data.targets], axi

df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbc
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38.0	6000.0	Na
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	Na
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6

1. Classification Problem Identification: Define and describe a classification problem based on the dataset.

Using different health features we want to classify indivuals into one of two groups, has Chronic Kidney Disease or does not have Chronic Kidney Disease.

2. Variable Transformation: Implement any transformations chosen or justify the absence of such modifications.

```
df.describe()
```

	age	bp	sg	al	su	bgr	bu	sc
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000

	age	bp	sg	al	su	bgr	bu	sc
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000

```
df.dtypes
```

```
age      float64
bp       float64
sg       float64
al       float64
su       float64
rbc       object
pc        object
pcc       object
ba        object
bgr      float64
bu       float64
sc       float64
sod      float64
pot      float64
hemo     float64
pcv      float64
wbcc     float64
rbcc     float64
htn       object
dm        object
cad       object
appet     object
pe        object
ane       object
```

```
class     object
dtype: object
```

```
float64_columns = df.select_dtypes(
    include=['float64']
    ).columns
float64_columns
scaler = StandardScaler()
df[float64_columns] = scaler.fit_transform(df[float64_columns])
```

```
cat_columns = df.select_dtypes(
    include=['object']
    ).columns

for col in cat_columns:
    print(df[col].value_counts(normalize=True))
```

```
rbc
normal      0.810484
abnormal    0.189516
Name: proportion, dtype: float64
pc
normal      0.773134
abnormal    0.226866
Name: proportion, dtype: float64
pcc
notpresent    0.893939
present       0.106061
Name: proportion, dtype: float64
ba
notpresent    0.944444
present       0.055556
```

```
Name: proportion, dtype: float64
htn
no     0.630653
yes    0.369347
Name: proportion, dtype: float64
dm
no      0.653266
yes     0.344221
\tno    0.002513
Name: proportion, dtype: float64
cad
no     0.914573
yes    0.085427
Name: proportion, dtype: float64
appet
good    0.794486
poor    0.205514
Name: proportion, dtype: float64
pe
no     0.809524
yes    0.190476
Name: proportion, dtype: float64
ane
no     0.849624
yes    0.150376
Name: proportion, dtype: float64
class
ckd       0.620
notckd    0.375
ckd\t     0.005
Name: proportion, dtype: float64
```

```
for col in cat_columns:
    df[col] = df[col].astype('category').cat.codes
df.head(5)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv
0	-0.203139	0.258373	0.454071	-0.012548	-0.410106	-1	1	0	0	-0.341498	...	0.569881
1	-2.594124	-1.936857	0.454071	2.208413	-0.410106	-1	1	0	0	NaN	...	-0.098536
2	0.613295	0.258373	-1.297699	0.727772	2.323069	1	1	0	0	3.473064	...	-0.878356
3	-0.203139	-0.473370	-2.173584	2.208413	-0.410106	1	0	1	0	-0.392022	...	-0.766953
4	-0.028189	0.258373	-1.297699	0.727772	-0.410106	1	1	0	0	-0.530963	...	-0.432744

3. Dataset Overview: Provide a detailed description of the dataset, covering variables, summaries, observation counts, data types, and distributions (at least three statements).

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     391 non-null    float64
 1   bp      388 non-null    float64
 2   sg      353 non-null    float64
 3   al      354 non-null    float64
 4   su      351 non-null    float64
 5   rbc     400 non-null    int8
 6   pc      400 non-null    int8
 7   pcc     400 non-null    int8
 8   ba      400 non-null    int8
 9   bgr     356 non-null    float64
 10  bu      381 non-null    float64
 11  sc      383 non-null    float64
 12  sod     313 non-null    float64
 13  pot     312 non-null    float64
 14  hemo    348 non-null    float64
 15  pcv     329 non-null    float64
 16  wbcc    294 non-null    float64
 17  rbcc    269 non-null    float64
 18  htn     400 non-null    int8
 19  dm      400 non-null    int8
 20  cad     400 non-null    int8
 21  appet   400 non-null    int8
 22  pe      400 non-null    int8
 23  ane     400 non-null    int8
 24  class   400 non-null    int8
dtypes: float64(14), int8(11)
memory usage: 48.2 KB
```

```
df.describe()
```

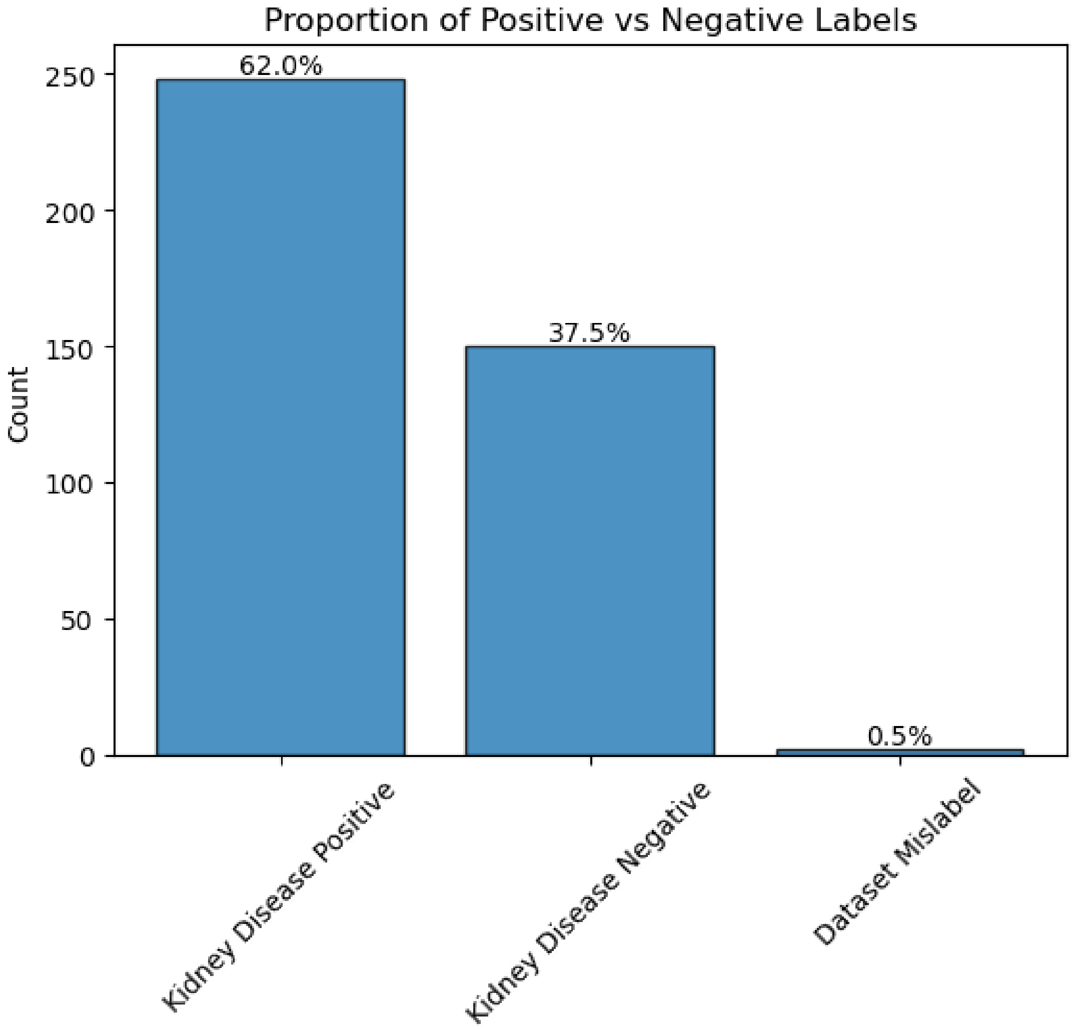
	age	bp	sg	al	su	rbc	pc	p
count	3.910000e+02	3.880000e+02	3.530000e+02	354.000000	351.000000	400.00000	400.000000	40
mean	9.994847e-17	-2.380684e-16	2.415443e-15	0.000000	0.000000	0.12250	0.485000	0.
std	1.001281e+00	1.001291e+00	1.001419e+00	1.001415	1.001428	0.93256	0.759089	0.
min	-2.885708e+00	-1.936857e+00	-2.173584e+00	-0.752868	-0.410106	-1.00000	-1.000000	-1
25%	-5.530393e-01	-4.733701e-01	-1.297699e+00	-0.752868	-0.410106	-1.00000	0.000000	0.
50%	2.050779e-01	2.583733e-01	4.540705e-01	-0.752868	-0.410106	1.00000	1.000000	0.
75%	7.590867e-01	2.583733e-01	4.540705e-01	0.727772	-0.410106	1.00000	1.000000	0.
max	2.246163e+00	7.575807e+00	1.329955e+00	2.948733	4.145186	1.00000	1.000000	1.

```
df["class"].value_counts()
```

```
class
```

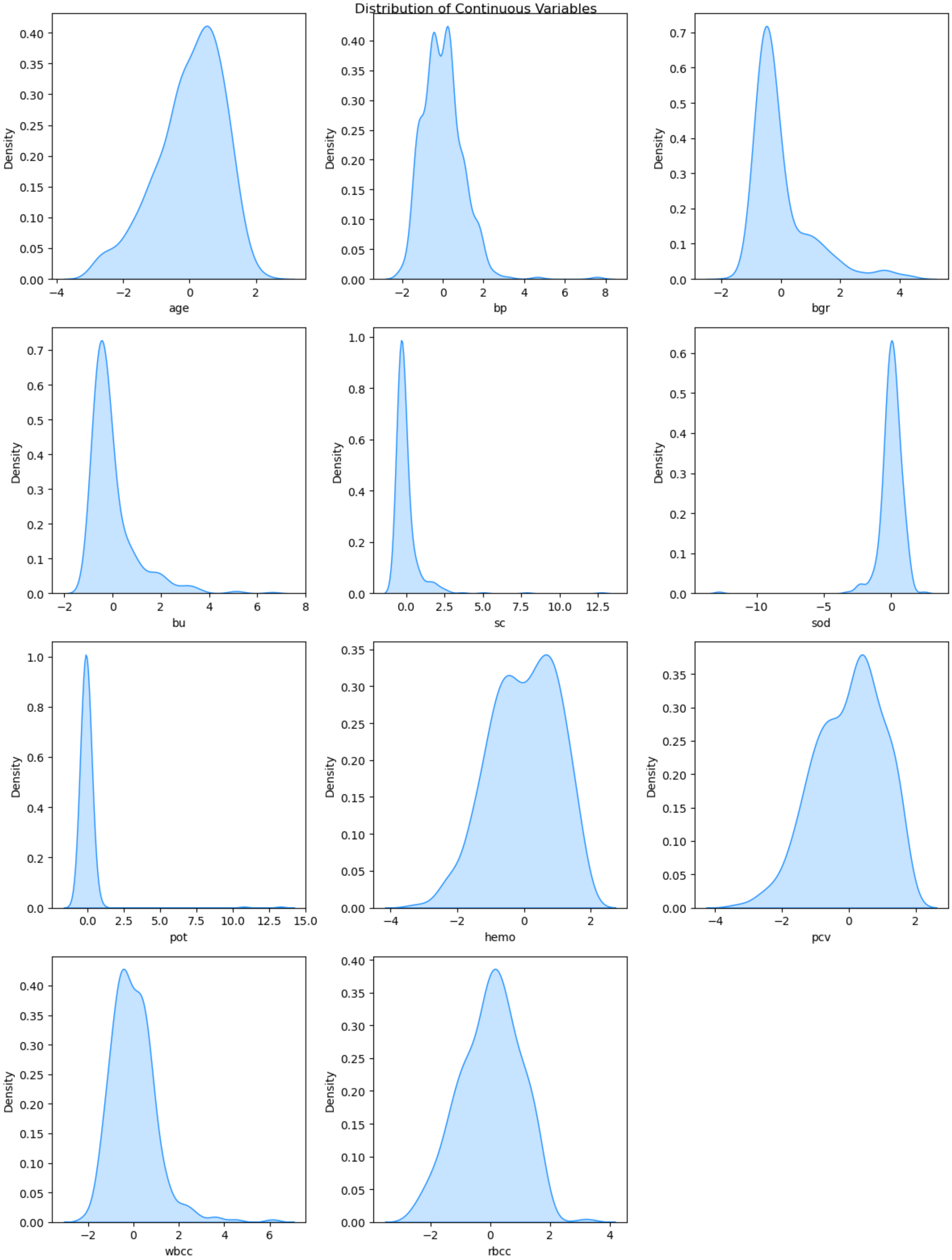
```
0    248
2    150
1      2
Name: count, dtype: int64
```

```python
fig, ax = plt.subplots(1, 1)
bar_data = df["class"].value_counts()
ax.bar(range(len(bar_data)), bar_data, edgecolor="black", alpha=0.8)

ax.set_xticks([0, 1, 2])
ax.set_xticklabels(["Kidney Disease Positive", "Kidney Disease Negative", "Dataset Mislabel"],

for i, count in enumerate(bar_data):
    percentage = count / bar_data.sum() * 100
    ax.text(i, count, f"{percentage:.1f}%", ha="center", va="bottom")
ax.set_ylabel("Count")
ax.set_title("Proportion of Positive vs Negative Labels")
plt.show()
```

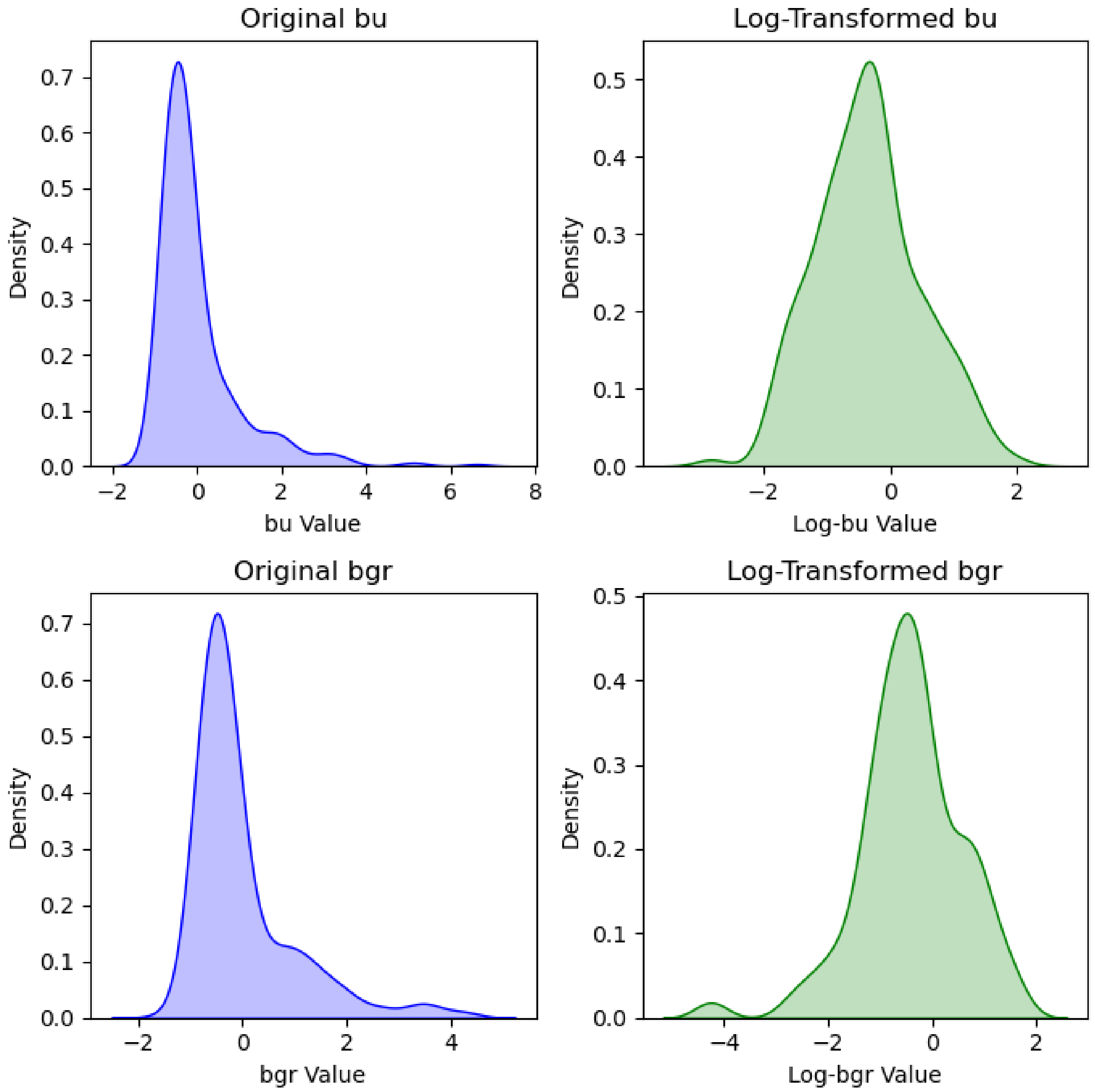


Visualizing distribution of continuous variables with Kernel Density Estimation

```
num_vars = ['age', 'bp', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']

num_features = len(num_vars)
num_rows = 4  # Number of rows in the subplot grid
num_cols = 3  # Number of columns in the subplot grid

fig, axes = plt.subplots(num_rows, num_cols, figsize=(4 * num_cols, 4 * num_rows))

for i, cont_feature in enumerate(df[num_vars]):
    row = i // num_cols  # Calculate the row index for the subplot
    col = i % num_cols  # Calculate the column index for the subplot

    ax_kde = axes[row, col]

    # Plot KDE for the feature
    sns.kdeplot(df[cont_feature], ax=ax_kde, fill=True, color="dodgerblue")

# Remove empty subplots
for i in range(num_features, num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.suptitle("Distribution of Continuous Variables")
plt.tight_layout()
plt.show()
```


Distribution of Continuous Variables
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
−4 −2 0 2
age
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
−2 0 2 4 6 8
bp
Density
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
−2 0 2 4
bgr
Density
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
−2 0 2 4 6 8
bu
Density
0.0 0.2 0.4 0.6 0.8 1.0
0.0 2.5 5.0 7.5 10.0 12.5
sc
Density
0.0 0.1 0.2 0.3 0.4 0.5 0.6
−10 −5 0
sod
Density
0.0 0.2 0.4 0.6 0.8 1.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0
pot
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
−4 −2 0 2
hemo
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
−4 −2 0 2
pcv
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
−2 0 2 4 6
wbcc
Density
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
−2 0 2 4
rbcc

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


# Create a new DataFrame with selected variables and their transformations
data_log_vis = pd.DataFrame({
    'bu': df['bu'],
    'log_bu': np.log(df['bu'] + 1),  # Log transform with handling zero values
    'bgr': df['bgr'],
    'log_bgr': np.log(df['bgr'] + 1)
})

# Variables to plot
variables = ['bu', 'bgr']

# Create a figure with 2 rows and 2 columns
fig, axes = plt.subplots(2, 2, figsize=(7, 7))
axes = axes.flatten()  # Flatten to simplify indexing

for i, var in enumerate(variables):
    # Original Data Plot
    sns.kdeplot(data_log_vis[var], ax=axes[2*i], fill=True, color="blue")
    axes[2*i].set_title(f"Original {var}")
    axes[2*i].set_xlabel(f"{var} Value")
    axes[2*i].set_ylabel("Density")

    # Log-Transformed Data Plot
    sns.kdeplot(data_log_vis[f'log_{var}'], ax=axes[2*i+1], fill=True, color="green")
    axes[2*i+1].set_title(f"Log-Transformed {var}")
    axes[2*i+1].set_xlabel(f"Log-{var} Value")
    axes[2*i+1].set_ylabel("Density")

plt.tight_layout()
plt.show()
```



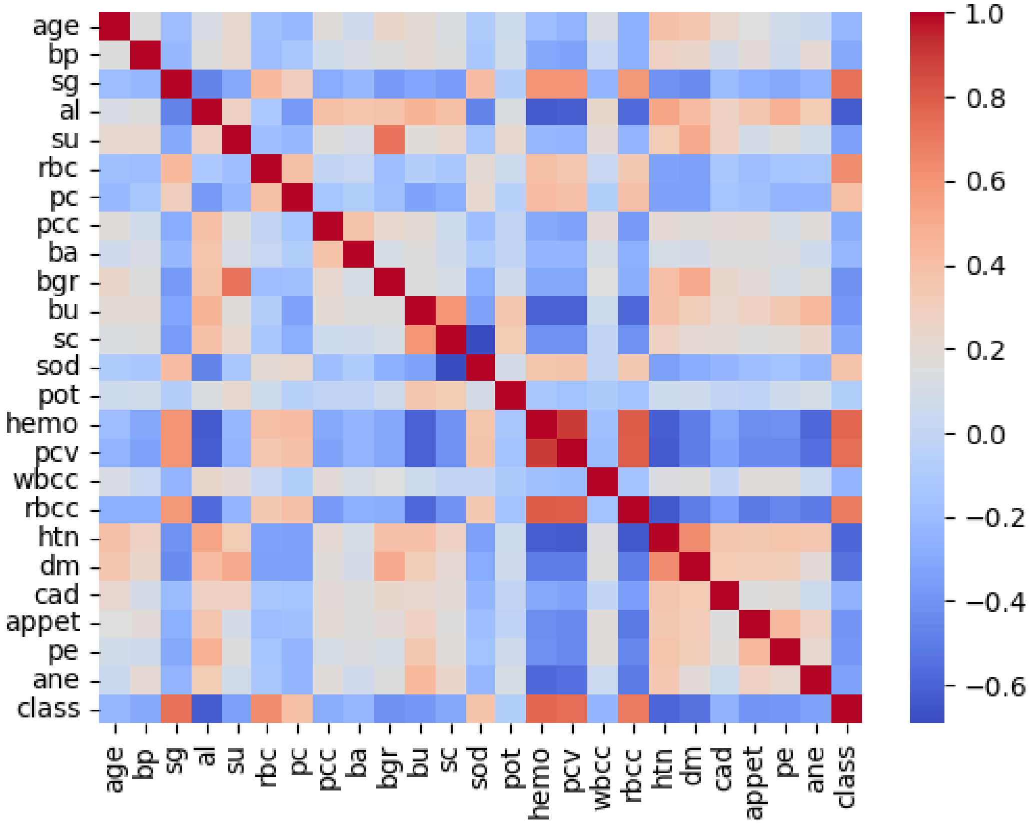
Observations: 1. The dataset has an imbalance in the number of kidney disease positive vs negative examples. Our visual exploratory data analysis also revealed that there are two mislabeled variables in the dataset's target column. The column in the dataset should include only "positive" or “negative” Kidney disease status, but there were a few examples with a third label. We discuss this more in the outliers section. 2. Many of the variables look roughly noramlly distributed, except that the blood glucode random and blood urea features are long-tailed. This has implications for feature engineering: we expect that log-transforming these features will make them closer to a normal distribution; this is likely to improve performance on classifiers such as logistic regression. We visualized these variables log-transformed to confirm that they look closer to a normal distribution after the transformation 3. Most variables are continuous, although the specific gravity, albumin and sugar levels are categorical.

4. Association Between Variables: Analyze variable relationships and their implications for feature selection or extraction (at least three statements)

```
correlation = df.corr()
sns.heatmap(correlation, cmap='coolwarm')


correlation
```

	age	bp	sg	al	su	rbc	pc	pcc	ba
age	1.000000	0.159480	-0.191096	0.122091	0.220866	-0.181683	-0.209743	0.169865	0.065425
bp	0.159480	1.000000	-0.218836	0.160689	0.222576	-0.194643	-0.129873	0.074018	0.126518
sg	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	0.421101	0.299093	-0.290210	-0.220317
al	0.122091	0.160689	-0.469760	1.000000	0.269305	-0.110803	-0.375461	0.403257	0.366845
su	0.220866	0.222576	-0.296234	0.269305	1.000000	-0.187230	-0.221037	0.156997	0.115534
rbc	-0.181683	-0.194643	0.421101	-0.110803	-0.187230	1.000000	0.393821	0.002845	0.019199
pc	-0.209743	-0.129873	0.299093	-0.375461	-0.221037	0.393821	1.000000	-0.136040	-0.088435
pcc	0.169865	0.074018	-0.290210	0.403257	0.156997	0.002845	-0.136040	1.000000	0.376102
ba	0.065425	0.126518	-0.220317	0.366845	0.115534	0.019199	-0.088435	0.376102	1.000000
bgr	0.244992	0.160193	-0.374710	0.379464	0.717827	-0.193079	-0.175899	0.215386	0.109492
bu	0.196985	0.188517	-0.314295	0.453528	0.168583	-0.071404	-0.323372	0.192276	0.167696
sc	0.132531	0.146222	-0.361473	0.399198	0.223244	-0.122191	-0.279445	0.060680	0.063784
sod	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	0.197653	0.218343	-0.183387	-0.100474
pot	0.058377	0.075151	-0.072787	0.129038	0.219450	0.061364	-0.058745	-0.003962	0.001224

	age	bp	sg	al	su	rbc	pc	pcc	ba
hemo	-0.192928	-0.306540	0.602582	-0.634632	-0.224775	0.402049	0.418814	-0.295985	-0.233115
pcv	-0.242119	-0.326319	0.603560	-0.611891	-0.239189	0.350038	0.391230	-0.326328	-0.230173
wbcc	0.118339	0.029753	-0.236215	0.231989	0.184893	0.029804	-0.079035	0.184171	0.115111
rbcc	-0.268896	-0.261936	0.579476	-0.566437	-0.237448	0.339400	0.390282	-0.371968	-0.266713
htn	0.389724	0.277324	-0.410243	0.525234	0.321166	-0.321229	-0.344689	0.206843	0.111083
dm	0.354065	0.235513	-0.436692	0.406456	0.500133	-0.345661	-0.345482	0.173907	0.099610
cad	0.221807	0.098398	-0.195717	0.272713	0.276542	-0.129224	-0.154193	0.184861	0.157115
appet	0.148648	0.184732	-0.268856	0.359009	0.089770	-0.190258	-0.172015	0.193949	0.155157
pe	0.085726	0.062676	-0.298504	0.477127	0.144712	-0.143371	-0.244199	0.113742	0.141271
ane	0.041271	0.204279	-0.243082	0.322958	0.077908	-0.135308	-0.233601	0.178299	0.064608
class	-0.222361	-0.297019	0.729117	-0.625585	-0.345589	0.630148	0.397401	-0.283455	-0.222438



Hemp and wbcc, hemo and rbcc, pcv and rbcc have the three highest positive correlations.

Sc and sod, hemo and htn, pcv and htn, hemo and ane, pcv and ane have the highest negative correlations.

Highly correlated features can lead to overfitting or redundant information. We can get rid of redundant features which leads to simpler models.

5. Missing Value Analysis and Handling: Implement your strategy for identifying and addressing missing values in the dataset, or provide reasons for not addressing them.

```
# Missing Value Analysis
missing_values = df.isnull().sum()


print(missing_values)
```

```
age        9
bp        12
sg        47
al        46
su        49
rbc        0
pc         0
pcc        0
ba         0
bgr       44
bu        19
sc        17
sod       87
pot       88
hemo      52
pcv       71
wbcc     106
rbcc     131
htn        0
```

```
dm         0
cad        0
appet      0
pe         0
ane        0
class      0
dtype: int64
```

```
# Mean imputer for numerical values and most frequent imputer for categorical values
num_vars = ['age', 'bp', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wbcc', 'rbcc']
cat_vars = ['sg', 'al', 'su']

imputer_num = SimpleImputer(strategy='mean')
imputer_cat = SimpleImputer(strategy='most_frequent')

df[num_vars] = imputer_num.fit_transform(df[num_vars])
df[cat_vars] = imputer_cat.fit_transform(df[cat_vars])
```

For numerical features (age, bp, bgr, bu, sc, sod, pot, hemo, pcv, wbcc, rbcc), we'll use mean imputation. For categorical features (sg, al, su), we'll use mode imputation. Binary features (rbc, pc, pcc, ba, htn, dm, cad, appet, pe, ane) already have no missing values.

6. Outlier Analysis: Implement your approach for identifying and managing outliers, or provide reasons for not addressing them.

```
# I noticed dm has 1s and 2s, so I converted them to 0s and 1s
# Class has 0s and 2s, so I converted them to 0s and 1s
df['dm'] = df['dm'].replace({'2':1, '1':0})
df['class'] = df['class'].replace({2:1})
```

7. Sub-group Analysis: Explore potential sub-groups within the data, employing appropriate data science methods to find the sub-groups of patients and visualize the sub-groups. The sub-group analysis must not include the labels (for CKD patients and healthy controls).

8. Data Splitting: Segregate 30% of the data for testing, using a random seed of 1. Use the remaining 70% for training and model selection.

```
# Split data into features and target variable
X = df.drop('class', axis=1)
y = df['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

9. Classifier Choices: Identify the two classifiers you have chosen and justify your selections.

```
# Classifier Choices
rf = RandomForestClassifier()
lr = LogisticRegression()

# Model Training
rf.fit(X_train, y_train)
lr.fit(X_train, y_train)

# Model Evaluation
rf_pred = rf.predict(X_test)
rf_y_prob = rf.predict_proba(X_test)
lr_pred = lr.predict(X_test)
lr_y_prob = lr.predict_proba(X_test)
probT_rf = pd.DataFrame(
    data = {'prob0': rf_y_prob[:,1], 'y_test': y_test}
    )
probT_lr = pd.DataFrame(
    data = {'prob0': lr_y_prob[:,1], 'y_test': y_test}
    )
probT_rf['y_test_pred'] = probT_rf.prob0.map(lambda x: 1 if x>0.5 else 0)
probT_lr['y_test_pred'] = probT_lr.prob0.map(lambda x: 1 if x>0.5 else 0)
```

We chose random forest because it is known for not overfitting and being able to handle multi dimensional data. It also works well when there is both numerical and catagorical data which we have in this case.

We chose logistic regression because it is simple and easy to interpret.

10. Performance Metrics: Outline the two metrics for comparing the performance of the classifiers.

```
def evaluate(y_test, y_test_pred):
    cm = confusion_matrix(y_test,y_test_pred)
    print('Confusion Matrix : \n', cm)
    total = sum(sum(cm))
    accuracy = (cm[0,0]+cm[1,1])/total
    print ('Accuracy : ', accuracy)
    sensitivity = cm[0,0]/(cm[0,0]+cm[0,1])
    print('Sensitivity : ', sensitivity )
    specificity = cm[1,1]/(cm[1,0]+cm[1,1])
    print('Specificity : ', specificity)
    print(classification_report(y_test, y_test_pred, zero_division=0.0))

print('Random Forest Classifier:\n')
evaluate(probT_rf.y_test, probT_rf.y_test_pred)

print('Logistic Regression Classifier:\n')
evaluate(probT_lr.y_test, probT_lr.y_test_pred)
```

```
Random Forest Classifier:


Confusion Matrix :
 [[70  0]
 [ 0 50]]
Accuracy :  1.0
Sensitivity :  1.0
```

```
Specificity :  1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        70
           1       1.00      1.00      1.00        50

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120

Logistic Regression Classifier:

Confusion Matrix :
 [[70  0]
 [ 0 50]]
Accuracy :  1.0
Sensitivity :  1.0
Specificity :  1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        70
           1       1.00      1.00      1.00        50

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

11. Feature Selection/Extraction: Implement methods to enhance the performance of at least one classifier in (9). The answer for this question can be included in (12).

12. Classifier Comparison: Utilize the selected metrics to compare the classifiers based on the test set. Discuss your findings (at least two statements).

13. Interpretable Classifier Insight: After re-training the interpretable classifier with all available

data, analyze and interpret the significance of predictor variables in the context of the data and the challenge (at least two statements).

14. Sub-group Improvement Strategy: If sub-groups were identified, propose and implement a method to improve one classifier performance further. Compare the performance of the new classifer with the results in (12).

Contributions

Jenna: Created/set up repository and jupyter notebook, started working on questions 1-4, started working on 11, made general edits

Viransh: References added, started working on questions 5-10

Noah: Finished question 3, added visualizations and discussion of normality/log-transformation

Github Link

Github link (https://github.com/JennaOrvitz/Stats3DA3FinalProject/tree/main)

References

Rubini, Soundarapandian, L., and P. Eswaran. 2015. "Chronic Kidney Disease." UCI Machine Learning Repository.

Sanmarchi, Francesco, Claudio Fanconi, Davide Golinelli, Davide Gori, Tina Hernandez-Boussard, and Angelo Capodici. 2023. "Predict, Diagnose, and Treat Chronic Kidney Disease with Machine Learning: A Systematic Literature Review - Journal of Nephrology." *SpringerLink*. Springer International Publishing. `https://link.springer.com/article/10.1007/s40620-023-01573-4`.