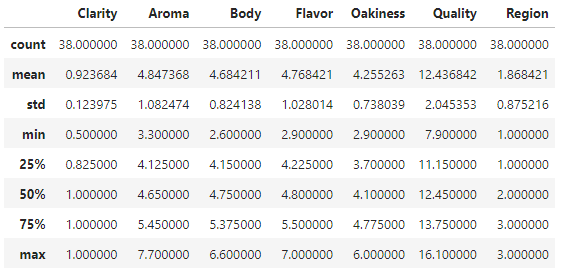
**STAT 4360 (Introduction to Statistical Learning, Spring 2023)**

**Mini Project 1  
Name: Ann Biju**

1. (a) I see that while Aroma, Body, Flavor, and Oakiness have similar mean values, Clarity does not. So I plotted Clarity against Quality (scatterplot), to see that they do not have a linear relationship.



*Using describe() to get a statistical overview of the wine dataset*

A graph with blue dots

Description automatically generated

*Scatterplot of Quality v. Clarity*

(b) Aroma, Body and Flavor of wine are most statistically significant predictors with regards to Quality. They have the highest R-squared values of 0.500, 0.301, and 0.002 respectively.

A line graph with purple dots

Description automatically generatedA line with purple dots

Description automatically generated

A line graph with purple dots

Description automatically generatedA line graph with purple dots

Description automatically generated

A diagram of wine and wine

Description automatically generated with medium confidenceA graph of a line with purple dots

Description automatically generated with medium confidence

*Simple Linear Regression models of Quality v. All other predictor variables*

(c) This model has an Adjusted R^2 value of 0.667. We reject the null hypothesis for predictors Flavor and Oakiness since their P >|t| values are 0.001 and 0.021, respectively – both of which are less than 0.05. This means that we will conclude that Flavor and Oakiness play a significant role in determining the quality of the wine.

(d) Since the Pr(>F) of model is high (0.909) we can conclude that the interaction between Region and Quality is not significant.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

(e) Final model in equation form

A math equations on a white sheet

Description automatically generated

Quality of the wine is 12.3766

(f) 

1. (a) Average BMI of 32.19 indicates that the majority of participants were not of healthy weight. Unsure if blood pressure indicates systolic or diastolic measures. A table with numbers and symbols

   Description automatically generated

(b) 0.784 accuracy is satisfactory. Specificity is 0.567.

A computer screen shot of a number

Description automatically generated

A graph of a positive rate

Description automatically generated

LDA and QDA curves overlap for the most part.

(c) Sensitivity of 0.8776 is excellent. Specificity of 0.5409 is good.

A computer screen with numbers and symbols

Description automatically generated

(d) While the accuracies of the LDA and QDA were very similar, the LDA is stronger (0.78), so it is recommended. Best posterior probability cutoff would be at 0.25.

1. (a) The empirical risk of the James Stein estimator (6.21) is significantly lower than the empirical risk of the MLE (9.75).

(b) The risk of both the JS estimator and MLE become similar to each other as the value of a increases and reaches 10. JS estimator’s risk starts off low (at 6) but sees considerable increase as the value of a increases.

A graph with blue and orange lines

Description automatically generated

(c) Here, the risks of both the JS estimator and the MLE increase as the value of sigma increases. The MLE risk sees a faster rate of increase from sigma = 5 than does the JS estimator.

A graph with a line and a point

Description automatically generated

**Python Code (or R Code)**

import pandas as pd

import numpy as np

wine = pd.read\_csv('wine.txt', sep = '\t')

wine.head()

# Question 1a

wine.describe()

# plotting Quality against Clarity

wine.plot.scatter(x = 'Clarity', y = 'Quality')

# Question 1b

# fitting a simple linear regression model for Clarity v. Quality

import statsmodels.api as sm

from statsmodels.formula.api import ols

import matplotlib.pyplot as plt

y = wine['Quality']

x = wine['Clarity']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Clarity'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Clarity'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Clarity'], a\*wine['Clarity']+b)

#add axis labels

plt.xlabel('Clarity of Wine')

plt.ylabel('Quality of Wine')

# fitting a simple linear regression model for Aroma v. Quality

y = wine['Quality']

x = wine['Aroma']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Aroma'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Aroma'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Aroma'], a\*wine['Aroma']+b)

#add axis labels

plt.xlabel('Aroma of Wine')

plt.ylabel('Quality of Wine')

# fitting a simple linear regression model for Body v. Quality

y = wine['Quality']

x = wine['Body']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Body'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Body'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Body'], a\*wine['Body']+b)

#add axis labels

plt.xlabel('Body of Wine')

plt.ylabel('Quality of Wine')

# fitting a simple linear regression model for Flavor v. Quality

y = wine['Quality']

x = wine['Flavor']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Flavor'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Flavor'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Flavor'], a\*wine['Flavor']+b)

#add axis labels

plt.xlabel('Flavor of Wine')

plt.ylabel('Quality of Wine')

# fitting a simple linear regression model for Oakiness v. Quality

y = wine['Quality']

x = wine['Oakiness']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Oakiness'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Oakiness'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Oakiness'], a\*wine['Oakiness']+b)

#add axis labels

plt.xlabel('Oakiness of Wine')

plt.ylabel('Quality of Wine')

# fitting a simple linear regression model for Region v. Quality

y = wine['Quality']

x = wine['Region']

x = sm.add\_constant(x)

model1 = sm.OLS(y,x).fit()

model1

# view model summary

print(model1.summary())

# create plot

a, b = np.polyfit(wine['Region'], wine['Quality'], 1)

#add points to plot

plt.scatter(wine['Region'], wine['Quality'], color='purple')

#add line of best fit to plot

plt.plot(wine['Region'], a\*wine['Region']+b)

#add axis labels

plt.xlabel('Region of Wine')

plt.ylabel('Quality of Wine')

# Question 1c

import statsmodels.api as sm

# Convert the "Region" column to categorical

wine['Region'] = pd.Categorical(wine['Region'])

# Define the response variable and predictor variables

response = wine['Quality']

predictors = wine[['Clarity', 'Aroma', 'Body', 'Flavor', 'Oakiness', 'Region']]

# Add a constant term to the predictor variables

predictors = sm.add\_constant(predictors)

# Fit the multiple linear regression model

model\_mlr = sm.OLS(response, predictors).fit()

# Print the summary of the model

print(model.summary())

# Question 1d

# check if Region is significant

# Define the response variable and predictor variables

response = wine['Quality']

predictors = wine[['Clarity', 'Aroma', 'Body', 'Flavor', 'Oakiness']]

# Add a constant term to the predictor variables

predictors = sm.add\_constant(predictors)

# Fit the multiple linear regression model

model\_no\_region = sm.OLS(response, predictors).fit()

# Print the summary of the model

print(model.summary())

sm.stats.anova\_lm(model\_no\_region, model\_mlr)

# since the Pr(>F) of model is high (0.909) we can conclude that the interaction between Region and Quality is not significant

# Question 1f

import scipy.stats as st

import statsmodels.api as sm

# we know final model is model\_no\_region

# Calculate the sample means of the predictor variables

means = wine[['Clarity', 'Aroma', 'Body', 'Flavor', 'Oakiness']].mean()

means.head()

# calculating the quality of wine using the sample mean of the predictors

y\_hat = 3.98 + 2.34\*0.923684 + 0.49\*4.847368 + 0.27\*4.684211 + 1.16\*4.768421 - 0.69\*4.255263

y\_hat

# we know final model is model\_no\_region

# 95% confidence interval

conf\_interval\_low = -0.645

conf\_inter\_high = 8.613

print("95% confidence interval:", conf\_interval\_low, ", ", conf\_inter\_high)

# prediction interval

prediction\_low = y\_hat - 1.755\*43.24

prediction\_high = y\_hat + 1.755\*43.24

print("95% prediction interval:", prediction\_low, ", ", prediction\_high)

# Question 2a

d = pd.read\_csv("diabetes.csv")

d.head()

#plot

from matplotlib import pyplot

fig, axes = pyplot.subplots(ncols=1, figsize=(18, 10), sharey=False)

d.boxplot()

# Question 2b

#LDA

length = len(d)

traini = np.random.randint(length,size=int(length \* 0.4))

alli = range(0,length)

testi = [i for i in alli if i not in traini]

# training data

train = d.iloc[traini, :]

trainX = train.iloc[:,0:8]

trainY = train['Outcome']

# test data

test = d.iloc[testi,:]

testX = test.iloc[:,0:8]

testY = test['Outcome']

testY.shape

#find the LDA

lda = LDA()

lda.fit(trainX, trainY)

lda.coef\_

ldapred = lda.predict(testX)

pd.crosstab(ldapred, testY, rownames = ['predict'], colnames = ['test'])

print(lda.predict\_proba(testX))

# confusion matrix

# we need a logistic regression model

logistic1 = sm.logit(formula='Outcome~Pregnancies+BloodPressure+Glucose+Insulin+SkinThickness+Age+BMI+DiabetesPedigreeFunction', data=d)

fitted1 = logistic1.fit()

fitted1.summary()

#predicting values

predicted\_values1=fitted1.predict(d[['Pregnancies']+['Glucose']+['BloodPressure']+

                                 ['SkinThickness']+['Insulin']+['BMI']+['DiabetesPedigreeFunction']+['Age']])

# Converting predicted values into classes using threshold

threshold=0.5

predicted\_class1=np.zeros(predicted\_values1.shape)

predicted\_class1[predicted\_values1>threshold]=1

predicted\_class1

cm1 = sk.metrics.confusion\_matrix(d[['Outcome']],predicted\_class1)

print('Confusion Matrix : \n', cm1)

total1=sum(sum(cm1))

# accuracy

accuracy1 = (cm1[0,0]+cm1[1,1])/total1

print ('Accuracy : ', accuracy1)

# sensitivity

sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])

print('Sensitivity : ', sensitivity1 )

# specificity

specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])

print('Specificity : ', specificity1)

#misclassification

print("Missclassification: ", 1-accuracy1)

# Question 2c

# QDA

qda = QDA()

fit2 = qda.fit(trainX, trainY)

# predictions

predicted\_values2=fit2.predict(d[['Pregnancies']+['Glucose']+['BloodPressure']+

                                    ['SkinThickness']+['Insulin']+['BMI']+['DiabetesPedigreeFunction']+['Age']])

# Converting predicted values into classes using threshold

threshold=0.5

predicted\_class2=np.zeros(predicted\_values2.shape)

predicted\_class2[predicted\_values2>threshold]=1

predicted\_class2

qdapred = qda.predict(testX)

pd.crosstab(qdapred, testY, rownames = ['predict'], colnames = ['test'])

# Confusion Matrix

cm2 = sk.metrics.confusion\_matrix(d[['Outcome']],predicted\_class2)

print('Confusion Matrix : \n', cm2)

total2 = sum(sum(cm2))

# accuracy

accuracy2=(cm2[0,0] + cm2[1,1])/total2

print ('Accuracy : ', accuracy2)

# sensitivity

sensitivity2 = cm2[0,0]/(cm2[0,0]+cm2[0,1])

print('Sensitivity : ', sensitivity2)

# specificity

specificity2 = cm2[1,1]/(cm2[1,0]+cm2[1,1])

print('Specificity : ', specificity2)

## Misclassification

print("Missclassification: ", 1 - accuracy2)

# adding an ROC curve for Question 2b and 2c

y = 1 \* (testY == 'Up')

# Question 2B

ldafpr, ldatpr, \_ = roc\_curve(testY, lda.predict\_proba(testX)[:,1])

#Question 2C

qdafpr, qdatpr, \_ = roc\_curve(testY, qda.predict\_proba(testX)[:,1])

print(qda.predict\_proba(testX)[:,1])

#plt.figure(figsize=(8, 6), dpi=80)

plt.plot(ldafpr, ldatpr, 'r-', label='LDA')

plt.plot(qdafpr, qdatpr, 'b-', label='QDA')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend()

plt.show()

# Question 3a

# Define parameters

p = 10

sigma = 1

true\_mean = np.array([1] \* p)

# Generate 1000 observations

np.random.seed(0)  # For reproducibility

N = 1000

observations = np.random.multivariate\_normal(true\_mean, sigma\*\*2 \* np.identity(p), N)

# Initialize arrays to store estimator results

js\_estimators = np.zeros((N, p))

mle\_estimators = np.zeros((N, p))

# Compute JS and MLE estimators for each observation

for i in range(N):

    y\_i = observations[i]

    js\_estimators[i] = (1 - (p - 2) \* sigma\*\*2 / np.linalg.norm(y\_i)\*\*2) \* y\_i

    mle\_estimators[i] = y\_i

# Calculate empirical bias and risk

bias\_js = np.mean(js\_estimators, axis=0) - true\_mean

bias\_mle = np.mean(mle\_estimators, axis=0) - true\_mean

risk\_js = np.mean(np.linalg.norm(js\_estimators - true\_mean, axis=1)\*\*2)

risk\_mle = np.mean(np.linalg.norm(mle\_estimators - true\_mean, axis=1)\*\*2)

print("Empirical Bias (JS):", bias\_js)

print("Empirical Bias (MLE):", bias\_mle)

print("Empirical Risk (JS):", risk\_js)

print("Empirical Risk (MLE):", risk\_mle)

# Question 3b

import matplotlib.pyplot as plt

# Define parameters

p = 10

sigma = 1

a\_values = np.arange(1, 11)  # Values of a from 1 to 10

# Initialize arrays to store Risk values

risk\_js\_values = []

risk\_mle\_values = []

# Generate observations and compute Risk for each value of a

for a in a\_values:

    true\_mean = a \* np.array([1] \* p)

    # Generate 1000 observations

    observations = np.random.multivariate\_normal(true\_mean, sigma\*\*2 \* np.identity(p), N)

    # Initialize arrays to store estimator results

    js\_estimators = np.zeros((N, p))

    mle\_estimators = np.zeros((N, p))

    # Compute JS and MLE estimators for each observation

    for i in range(N):

        y\_i = observations[i]

        js\_estimators[i] = (1 - (p - 2) \* sigma\*\*2 / np.linalg.norm(y\_i)\*\*2) \* y\_i

        mle\_estimators[i] = y\_i

    # Calculate empirical Risk for both estimators

    risk\_js = np.mean(np.linalg.norm(js\_estimators - true\_mean, axis=1)\*\*2)

    risk\_mle = np.mean(np.linalg.norm(mle\_estimators - true\_mean, axis=1)\*\*2)

    risk\_js\_values.append(risk\_js)

    risk\_mle\_values.append(risk\_mle)

# Create a plot to compare the Risk of JS and MLE estimators

plt.figure(figsize=(10, 6))

plt.plot(a\_values, risk\_js\_values, label="James-Stein Estimator", marker='o')

plt.plot(a\_values, risk\_mle\_values, label="MLE", marker='x')

plt.xlabel("Value of a")

plt.ylabel("Risk")

plt.title("Risk Comparison of Estimators vs. Value of a")

plt.legend()

plt.grid(True)

plt.show()

# Question 3c

# Define parameters

p = 10

a = 1

sigma\_values = [0.1, 0.5, 2, 5, 10]  # Values of σ

# Initialize arrays to store Risk values

risk\_js\_values = []

risk\_mle\_values = []

# Generate observations and compute Risk for each value of σ

for sigma in sigma\_values:

    true\_mean = a \* np.array([1] \* p)

    # Generate 1000 observations

    observations = np.random.multivariate\_normal(true\_mean, sigma\*\*2 \* np.identity(p), N)

    # Initialize arrays to store estimator results

    js\_estimators = np.zeros((N, p))

    mle\_estimators = np.zeros((N, p))

    # Compute JS and MLE estimators for each observation

    for i in range(N):

        y\_i = observations[i]

        js\_estimators[i] = (1 - (p - 2) \* sigma\*\*2 / np.linalg.norm(y\_i)\*\*2) \* y\_i

        mle\_estimators[i] = y\_i

    # Calculate empirical Risk for both estimators

    risk\_js = np.mean(np.linalg.norm(js\_estimators - true\_mean, axis=1)\*\*2)

    risk\_mle = np.mean(np.linalg.norm(mle\_estimators - true\_mean, axis=1)\*\*2)

    risk\_js\_values.append(risk\_js)

    risk\_mle\_values.append(risk\_mle)

# Create a plot to compare the Risk of JS and MLE estimators

plt.figure(figsize=(10, 6))

plt.plot(sigma\_values, risk\_js\_values, label="James-Stein Estimator", marker='o')

plt.plot(sigma\_values, risk\_mle\_values, label="MLE", marker='x')

plt.xlabel("Value of σ")

plt.ylabel("Risk")

plt.title("Risk Comparison of Estimators vs. Value of σ")

plt.legend()

plt.grid(True)

plt.show()