Machine Learning Pset 2, Daniel Avila

1.

(a) We would expect for the LDA to perform better whne the Bayes decision boundary is linear because the LDA would better estimate the true underlying decision boundaries/model.

(b) If the Bayes decision boundary is non-linear, then the QDA offers the flexibilty that better matches the underlying decision boundaries. Therefore QDA would be better. Note however that because we dont know the actual decision true values underneath, then we would not truthfully whether QDA or LDA is better except for how our data looks.

(c) The more data that we have, the better the accuracy of our sample means and variances relative to the true means and variances. Therefore would expect that the LDA would continue to get better with more data.

QDA requires lots of data because we are calculating variances and covariances for each of the classes. Therefore, as we have more data, QDA becomes an option and it gives better test predictions.

Regarding which gets better relative to the other, the answer is still dependant on the underlying true Bayes decision boundary. However we would expect that QDA gets better relative to LDA.

(d) False, because we are limited by how much data we have. Therefore, the QDA can be limited by the number of covariates that it can produce, which will eat away at the degrees of freedom of the model. Therefore, if we have too few degrees of freedom, we can be overfitting to the noise of the data, which would actually make our QDA worse relaive to LDA.

2.

(a) Logit model: (receive an A = 1) = -6 + .05(hours studied) + 1(undergrad GPA)

-6 + .05(hours studied = 40) + 1(undergrad GPA = 3.5) = -.5

odds = 1/(1 + e^(-.5)) odds = .3775 or approximately 38% probability of receiving an A in the class.

(b)

.5 = 1/(1 + e^(log odds)) 1 + e^((-6 + .05(hours studied) + 3.5)) = 2 hours studied = 50

3.

mean dividend companies: 10 mean without dividend companies: 0 variance: 36 80% issued dividends 20% did not issue dividends predict binary dividend, profit = 4

discriminant function, dividend = ln(.8) - 10^2/2(36) + 10/36(4) discriminant function = -.5009

discriminant function, no dividend = ln(.2) - 0 + 0 discriminant function, no dividend = -1.6

calculating probabilities:

exp^(discriminant function, dividend = -.5009) / exp^(-.5009 - 1.6) = .75

exp^(discriminant function, no dividend = -1.5) / exp^(-.5009 - 1.6) = .25

A company with percentage profit of x = 4 will issue a dividend with probability 75%.

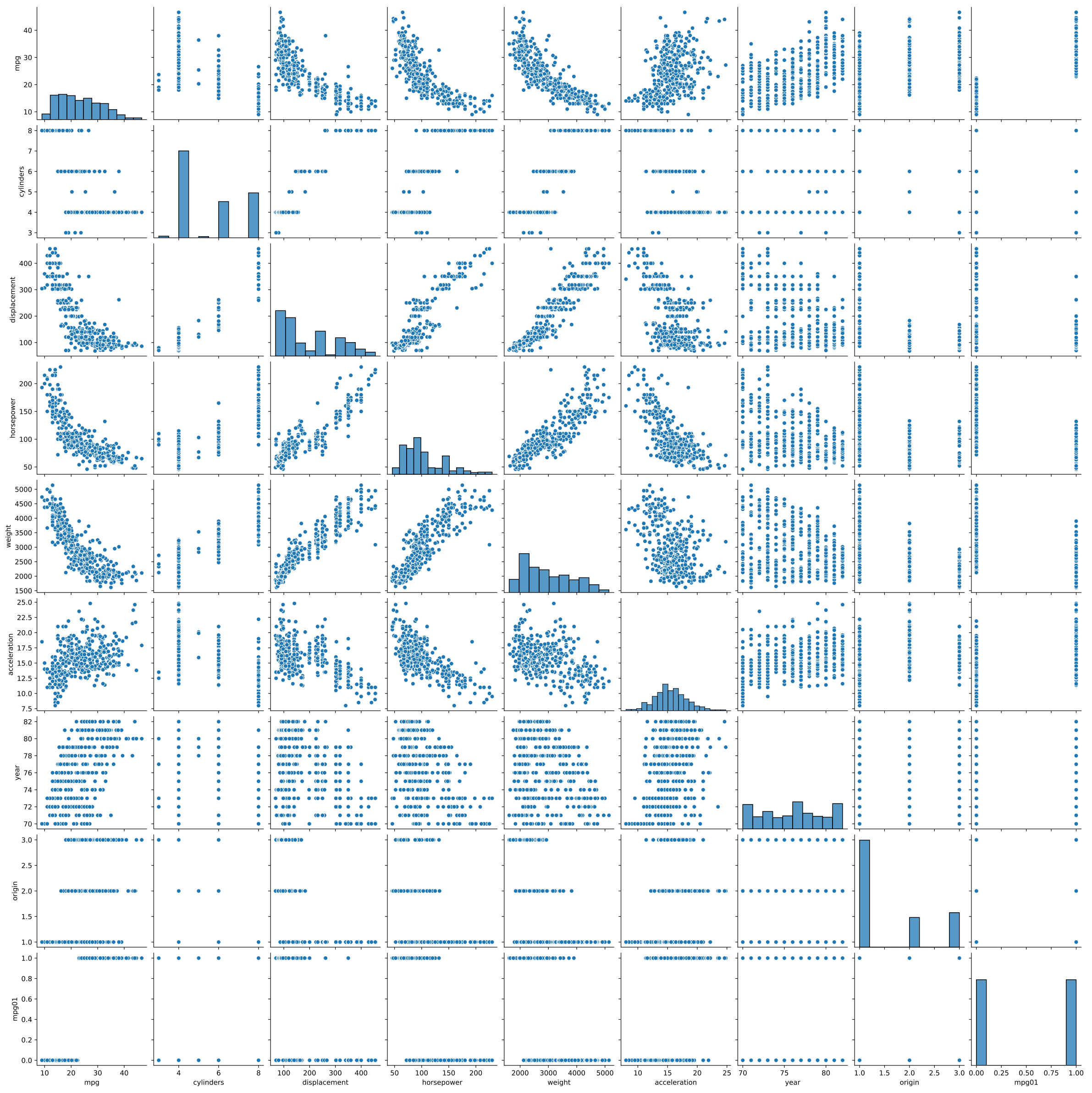
4.

(a)

```
import pandas as pd
import numpy as np
import os
import seaborn as sns

#os.chdir("C:/Users/AVILA/OneDrive/Documents/Github/Machine-Learning--Harris/Problem-Set-2")

auto_df = pd.read_csv("Data-Auto.csv")
auto_df = auto_df.drop("Unnamed: 0", axis = 1)
default_df = pd.read_csv("Data-Default.csv")
```

(b)

```
auto_df.loc[auto_df["mpg"] > np.median(auto_df["mpg"]), "mpg01"] = 1
auto_df["mpg01"] = auto_df["mpg01"].fillna(0)


sns.pairplot(auto_df)
```



mpg01 seems to be correlated with... 1. horsepower 2. weight 3. acceleration, 4. displacement

The reason for thinking that these are the 4 primary predictive features is because, looking at the pairplot graph, we can see that there are sections of mpg01 = 1 that do not overlap with mpg01 = 0 at all. This is essentially a plot of a logit function, which indicates probabilities (visible in the last row of the pairplot).

(c)

```python
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

X = auto_df.loc[:, ["horsepower", "weight", "acceleration", "displacement"]]
X_train, X_test, y_train, y_test = train_test_split(X, auto_df["mpg01"], test_size = .5, ranc
```

(d)

```python
modelLDA = LinearDiscriminantAnalysis()
modelLDA.fit(X_train, y_train)

print(f"model accuracy: {round(modelLDA.score(X_test, y_test), 4) * 100}%")
print(f"model test error: {round(1-modelLDA.score(X_test, y_test), 4) * 100}%")
```

```
model accuracy: 89.8%
model test error: 10.2%
```

(e)

```python
modelQDA = QuadraticDiscriminantAnalysis()
modelQDA.fit(X_train, y_train)

print(f"model accuracy: {round(modelQDA.score(X_test, y_test), 4) * 100}%")
print(f"model test error: {round(1-modelQDA.score(X_test, y_test), 4) * 100}%")
```

```
model accuracy: 88.78%
model test error: 11.219999999999999%
```

(f)

```python
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(X_train, y_train)

print(f"model accuracy: {round(logit.score(X_test, y_test), 4) * 100}%")
print(f"model test error: {round(1-logit.score(X_test, y_test), 4) * 100}%")
```

```
model accuracy: 88.78%
model test error: 11.219999999999999%
```

(g)

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)

print(f"model accuracy: {round(gnb.score(X_test, y_test), 4) * 100}%")
print(f"model test error: {round(1-gnb.score(X_test, y_test), 4) * 100}%")
```

```
model accuracy: 87.76%
model test error: 12.24%
```

5.

(a), (b)

```
from sklearn.metrics import confusion_matrix
logit = LogisticRegression()


def logit_func(random_seed = 42):
    X_train, X_test, y_train, y_test = train_test_split(default_df.loc[:, "balance":], defaul

    logit.fit(X_train, y_train)
    y_prob = logit.predict_proba(X_test)[:, 1]
    y_prob = (y_prob > .5).astype(int)
    y_test = (y_test == "Yes").astype(int)

    matrix = confusion_matrix(y_test, y_prob)
    fn = matrix[0,1]
    fp = matrix[1, 0]
    error_rate = (fp + fn) / y_test.shape[0]

    return f"error rate: {round(error_rate, 4) * 100}%"

logit_func()
```

```
'error rate: 2.67%'
```

(c)

```
print(logit_func(random_seed = 2))

print(logit_func(random_seed = 6))

print(logit_func(random_seed = 9))
```

```
error rate: 2.37%
error rate: 2.4699999999999998%
error rate: 2.17%
```

We can see from the different error rates that, depending on the data used in the training set vs. validation set, we get different outcomes in the error rates of our model. What is happening is that the model is producing slightly different paramaters based on the data that it is trained on. Additionally, once we push the validation data through the fitted model, each validation set has different data points based on the different seed we use. Therefore both the parameters of our logit model, and the validation data will be different, which leads to different error rates. What is good is that that, across the 3 models, our error rate is closely bundled around the 3% value, which indicates that our model is performing well to the data.

(d)

```
default_df["dummy"] = np.where(default_df["student"] == "Yes", 1, 0)
logit_func(random_seed = 42)
```

```
'error rate: 2.67%'
```

Including the dummy variable did not seem to improve our results in anyway. Note that in the first set of code, I have coded the features such that all features past the "balance" feature in the dataset will be included. When we add the "dummy" feature in this round, that variable gets added to the right of the "balance" feature, which means it is included here. Therefore, since the error rate has not decreased relative to the previous models, we can say that including the "dummy" feature is not adding much, if anything, to improving the quality of predictions of our model.