ML_PS2

Clarice Tee

2025-02-06

Part 1

```python
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
import os
import statsmodels.formula.api as smf
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

1.a. QDA is expected to perform better on the training set/ This is because the QDA's greater flexibility yields a closer fit, but does have greater variance.

LDA is expected to do better on the test. While the QDA is flexible, this means it runs into the problem of overfitting the linear Bayes decision boundary.

1.b.(Non-linear) While QDA's flexibility increases its variance compared to LDA, when the Bayes boundary is non-linear, this flexibility is actually aa good thing because it can be offset by a larger reduction in bias, thus better test performance.

1.c. We expect the tesst predition accuracy of the QDA to improve relative to the LDA as the sample size increases since is flexibility will yield a better fit, especially when we have more samples and this also helps deal with the problem of variance. We expect the test prediction accuracy of QDA relative to LDA to improve.

1.d. False.Flexible methods like QDA require more data to prevent overfitting, which happens due to the model's sensitivity to the noise in the training sets. Overfitting would make the QDA have a higher test error rate than the LDA, which already approximates the Bayes decision boundary accurately.

Part 2

2.a. X = [40 hours, 3.5 GPA] From the logistic regression model, we can fill in the formula: P(Y=1)|X = exp(-6 + 0.05 * X1 + X2) / (1 + exp(-6 + 0.05 * X1 + X2))

```
X1 = 40
X2 = 3.5


probability_A = np.exp(-6 + 0.05 * X1 + X2) / (1 + np.exp(-6 + 0.05 * X1 +
 ↪  X2))

print(f"Probability of getting an A is {probability_A * 100:.1f}%")
```

```
Probability of getting an A is 37.8%
```

2.b. Same student, 50%, how many hours (X1)

P(Y=.5)|X is .5 = exp(-6 + 0.05 * X1 + 3.5) / (1 + exp(-6 + 0.05 * X1 + 3.5))

solving for x1 gives us: $\exp(0.05X1 - 2.5) = 1$

```
X1 = 2.5 / 0.05

print(f"Student who wants a  50% probability needs to study {X1} hours")
```

```
Student who wants a  50% probability needs to study 50.0 hours
```

Part 3

Ave profit with dividends(X bar) =10 Ave profit 1/0 dividends(X bar) =0

Variance =36

80% issued dividends.

Using Bayes' theorem:

A. Using normal distribution: Get likelihood for a company with profits X=4 to be in with dividend group

B. Use 80% with dividend and 20% without dividend to wieigh likelihoods.

C. Calculate posterior probability (weighted likelihood of with dividend group divided by the weighted likelihoods of with and without dividend)

```
w_dividend = 0.8
wo_dividend = 0.2
exp_w_dividend = np.exp(-0.5)
exp_wo_dividend = np.exp(-2 / 9)

# Compute posterior probability
posterior_w_dividend = (w_dividend * exp_w_dividend) / (w_dividend *
↪  exp_w_dividend + wo_dividend * exp_wo_dividend)

print(f"Probability of issuing a dividend: {posterior_w_dividend:.4f} or
↪  {posterior_w_dividend * 100:.2f}%")
```

```
Probability of issuing a dividend: 0.7519 or 75.19%
```

Part 4

```
# Load the dataset
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\ps2"
auto_path = os.path.join(directory, "Data-Auto.csv")
auto_df = pd.read_csv(auto_path)
print(auto_df.dtypes)
print(auto_df.shape)
```

```
Unnamed: 0        int64
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year              int64
origin            int64
name             object
```

```
dtype: object
(392, 10)
```

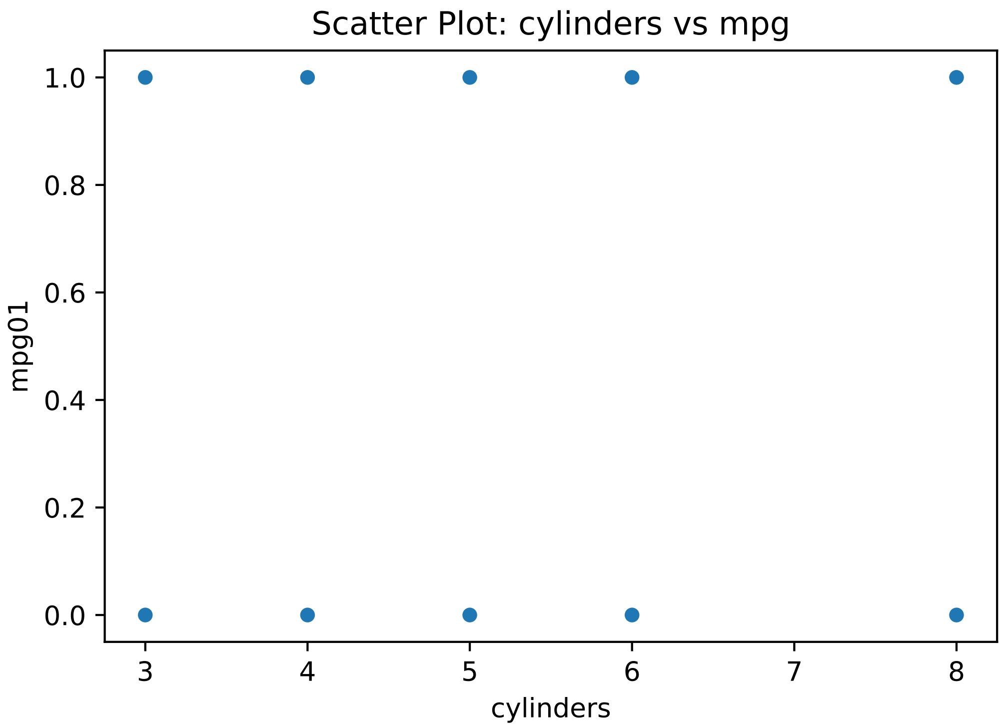
4.a. Making dummy variable

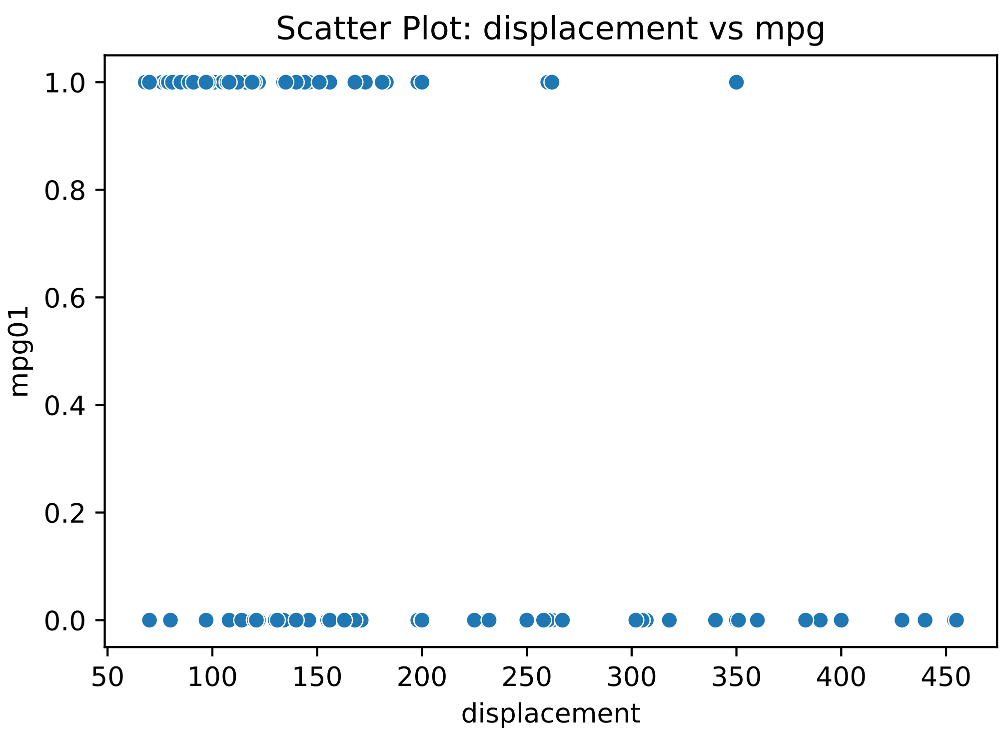
```
auto_df["mpg01"] = np.where(auto_df["mpg"] > auto_df["mpg"].median(), 1, 0)
```

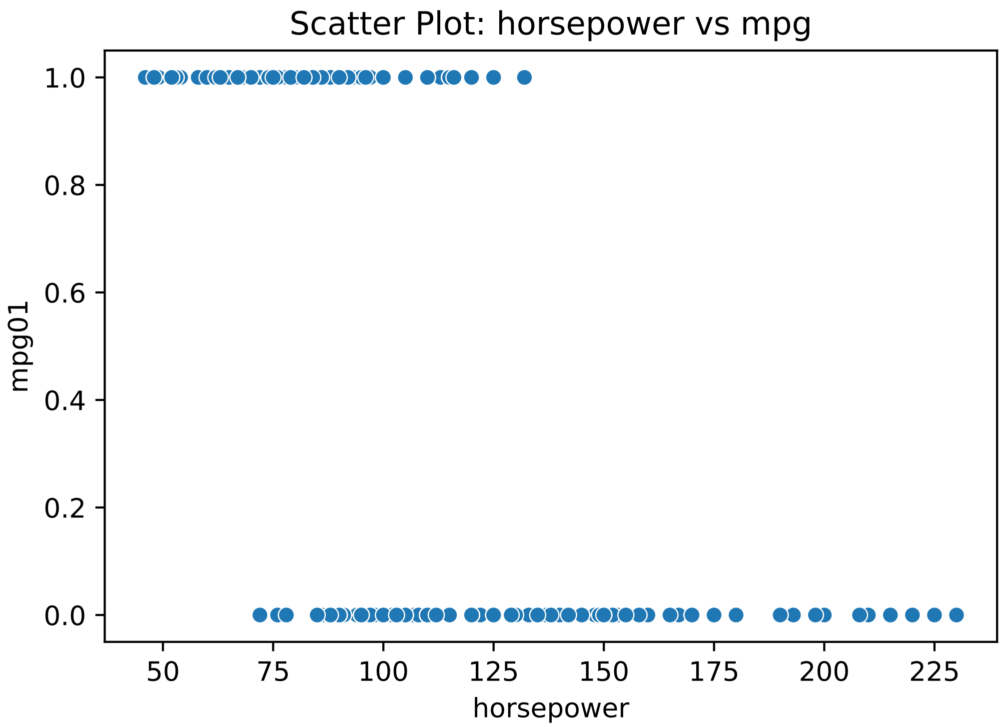
4.b. It looks like horsepower, acceleration, and weight could be useful in predicting mpg01 because the number of observations and the value of the variable tends to increase or decrease based on if mpg01 is 0 or 1 (although there are some overlaps)

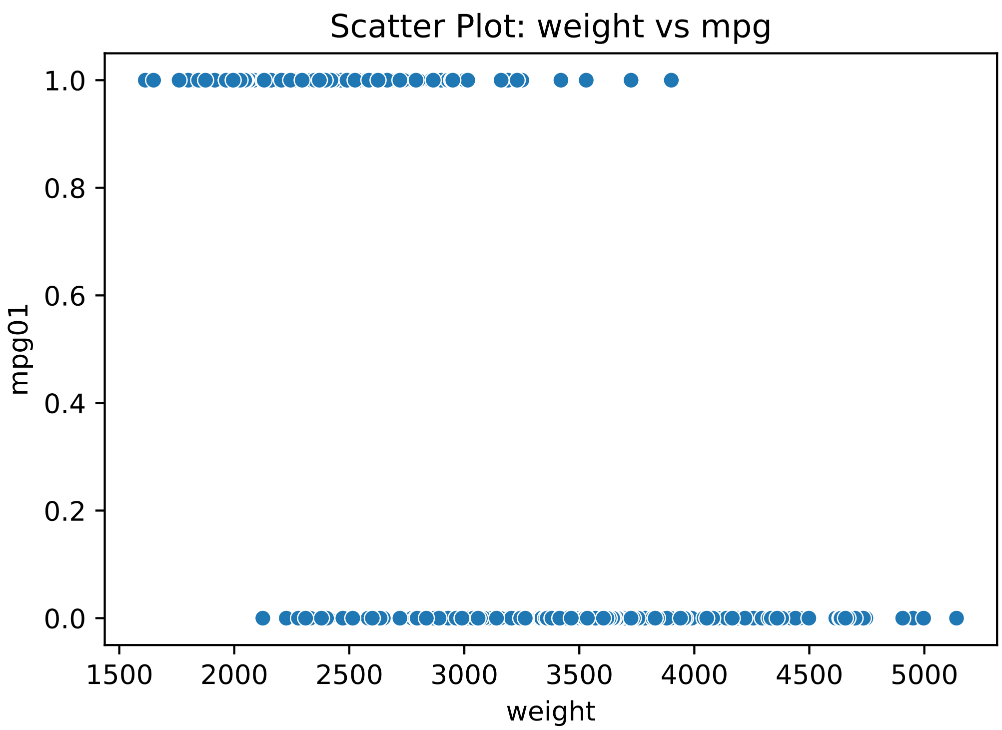
Scatterplot

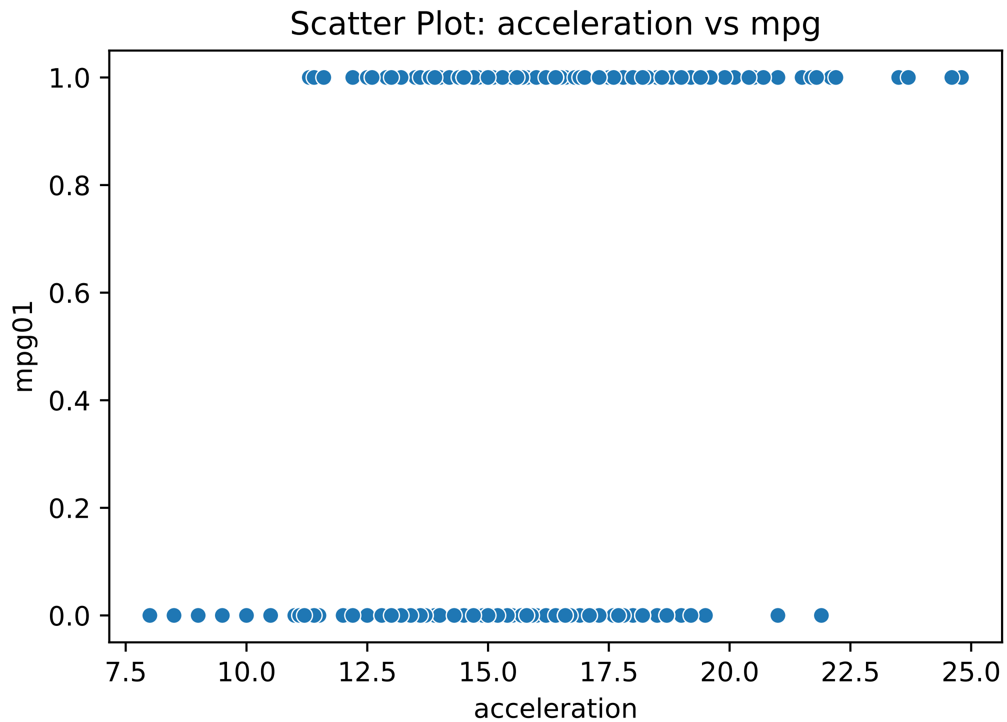
```
# creating a list of variables to loop through
auto_vars = ['cylinders', 'displacement', 'horsepower', 'weight',
 ↪  'acceleration', 'year', 'origin'] #excluding name, which is an object


for var in auto_vars:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(x=var, y='mpg01', data=auto_df)
    plt.title(f'Scatter Plot: {var} vs mpg')
    plt.tight_layout
    plt.show()
```

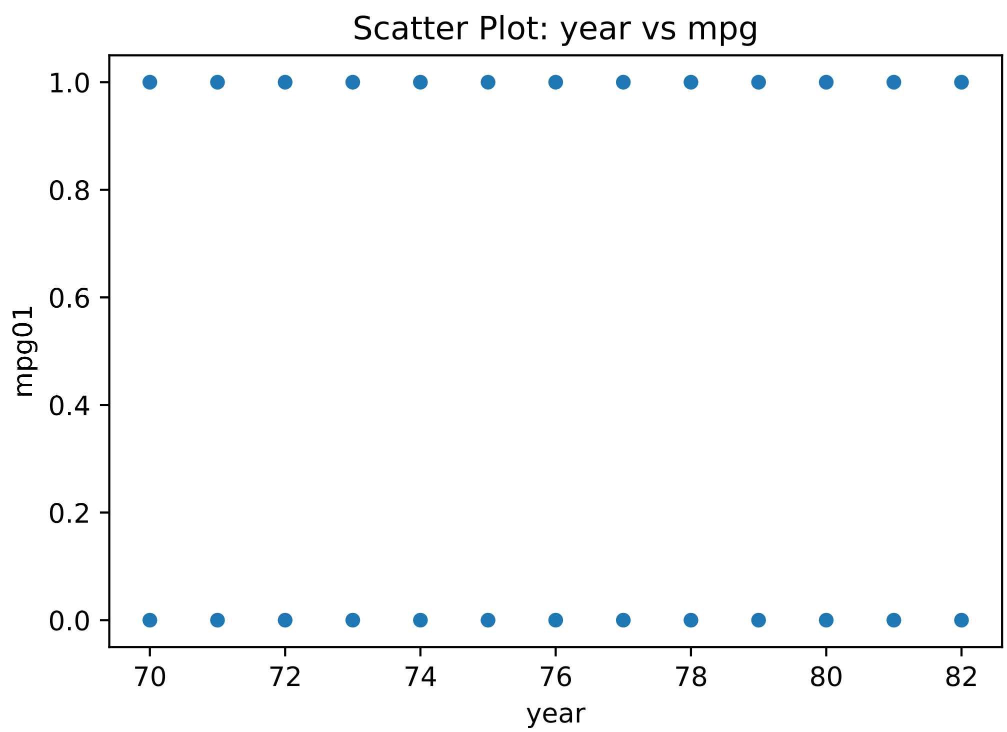

Scatter Plot: cylinders vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8
cylinders

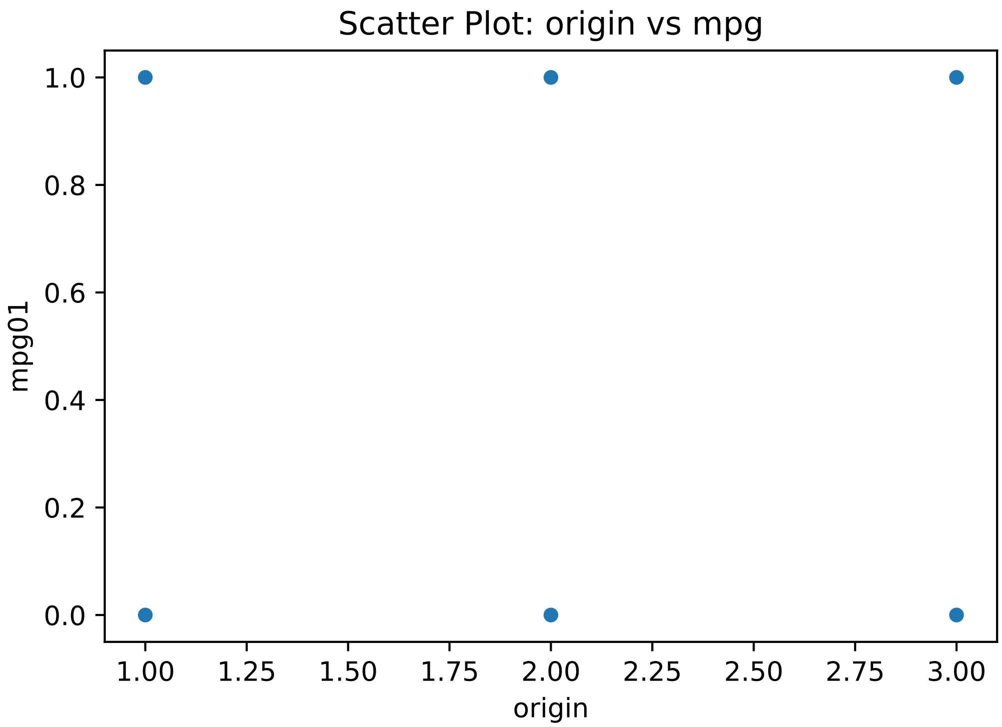

Scatter Plot: displacement vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250
300
350
400
450
displacement


Scatter Plot: horsepower vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
50
75
100
125
150
175
200
225
horsepower


Scatter Plot: weight vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
1500
2000
2500
3000
3500
4000
4500
5000
weight

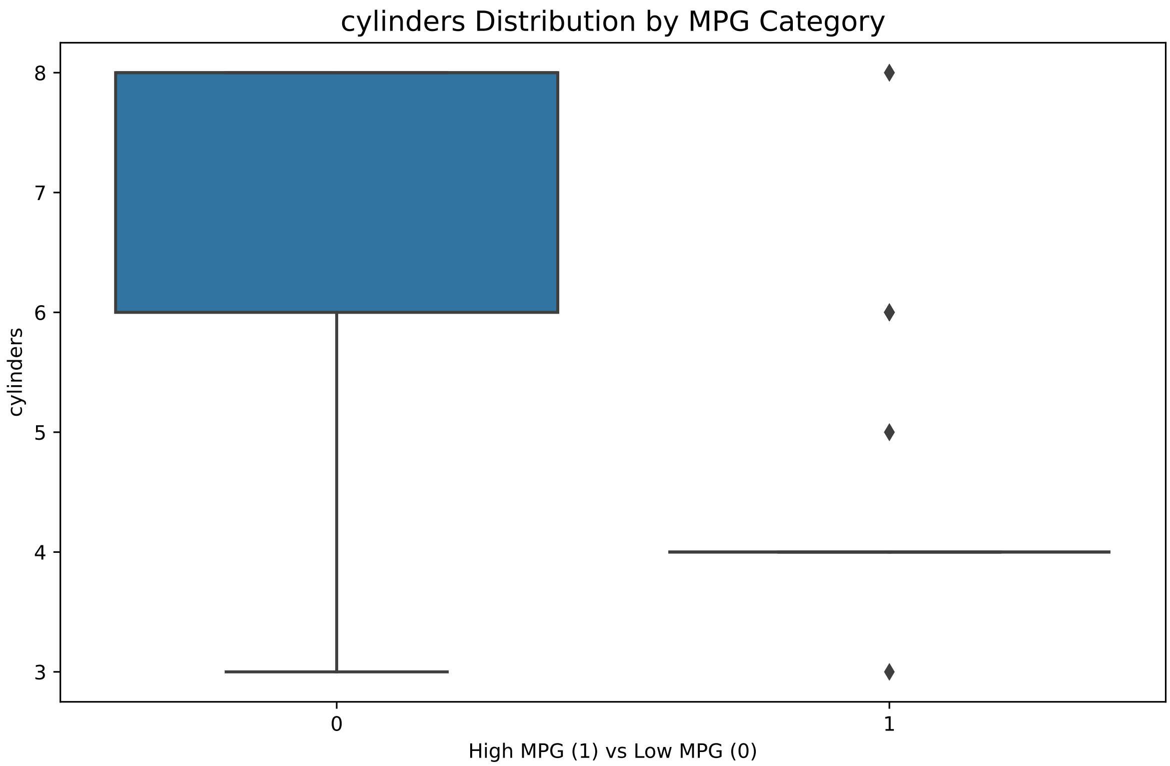

Scatter Plot: acceleration vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
acceleration


Scatter Plot: year vs mpg
mpg01
1.0
0.8
0.6
0.4
0.2
0.0
70
72
74
76
78
80
82
year

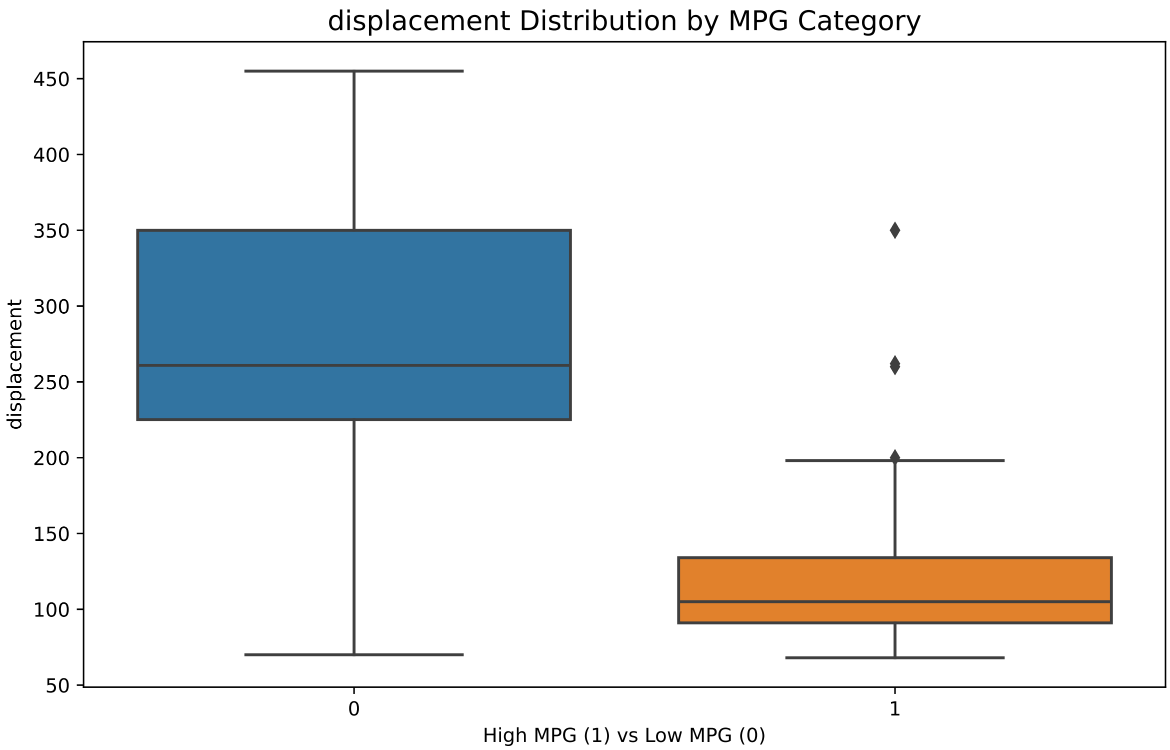


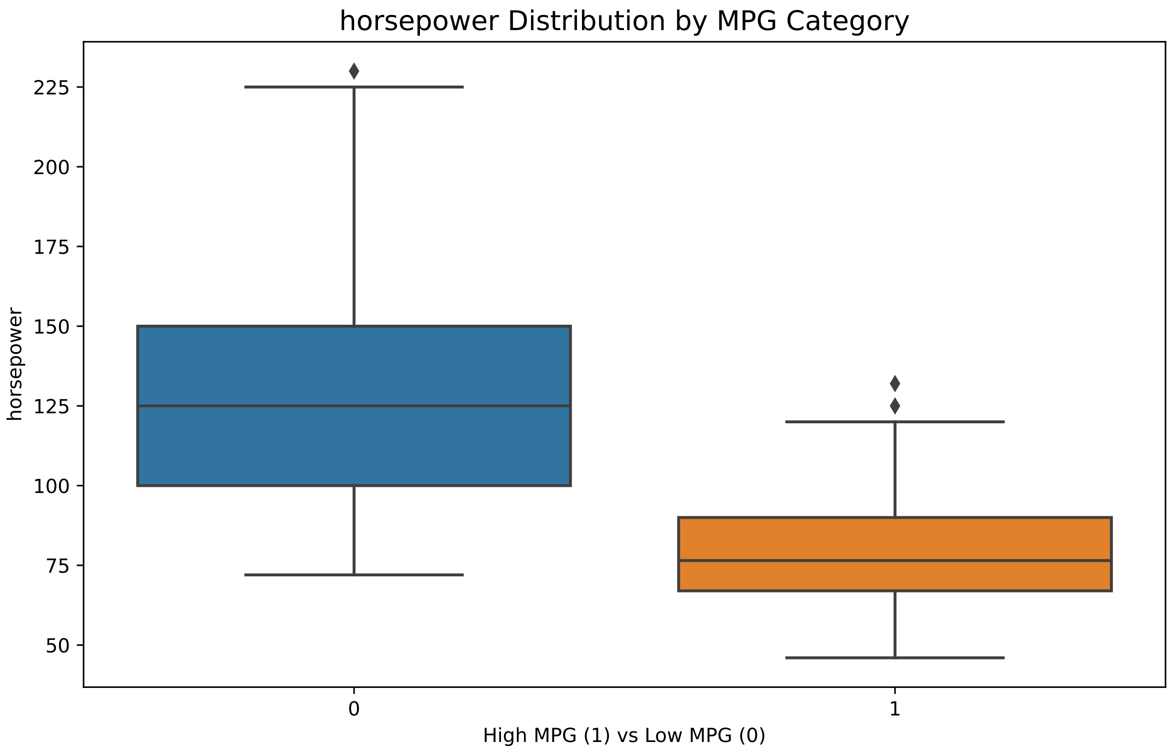
Boxplot

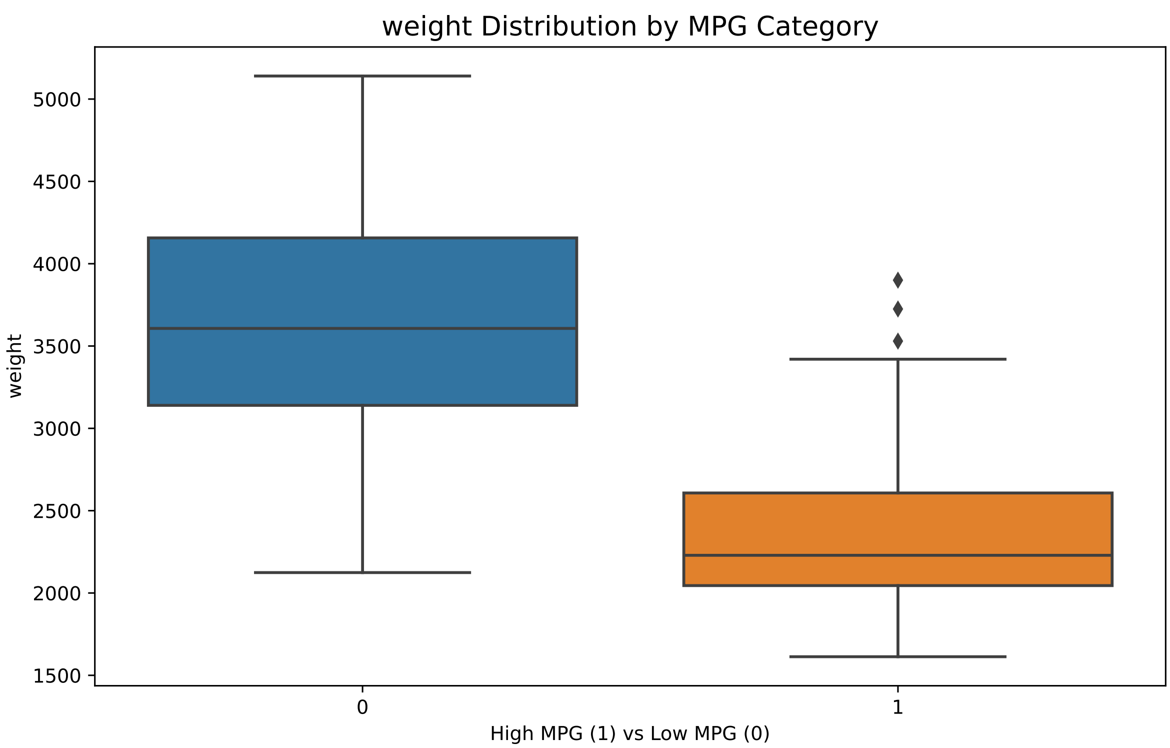
```
for var in auto_vars:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='mpg01', y=var, data=auto_df)
    plt.title(f'{var} Distribution by MPG Category', size=14)
    plt.xlabel('High MPG (1) vs Low MPG (0)')
    plt.ylabel(var)
    plt.show()
```

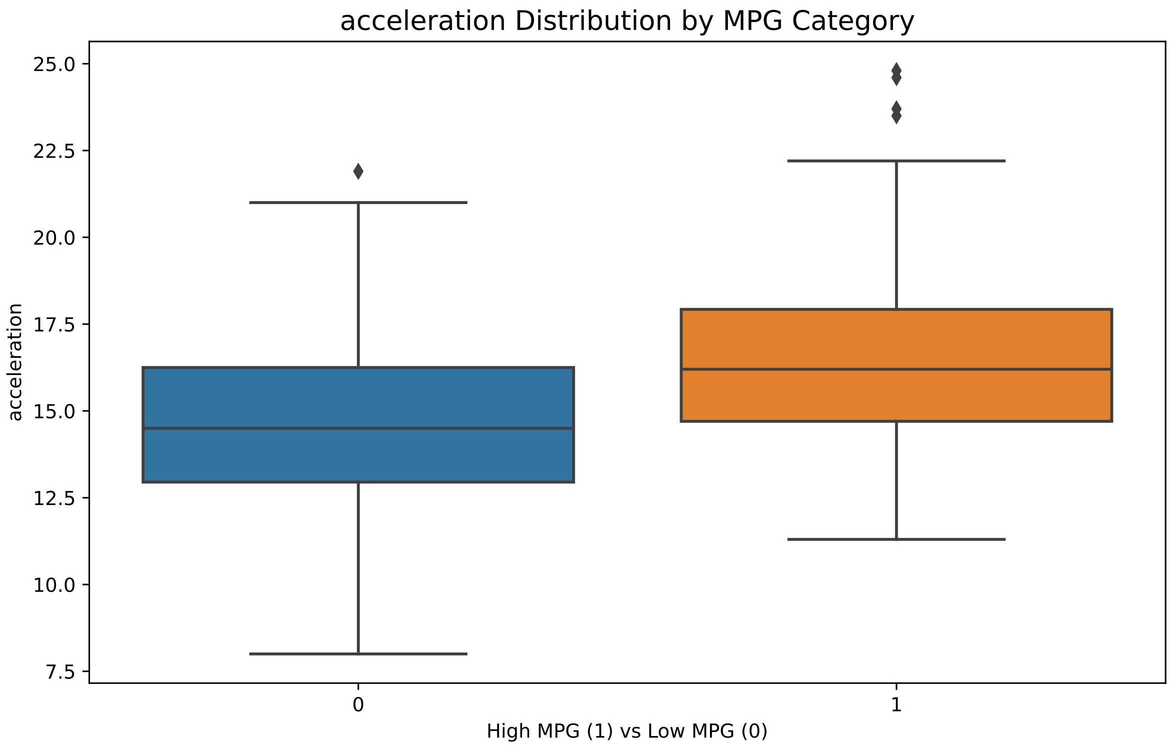


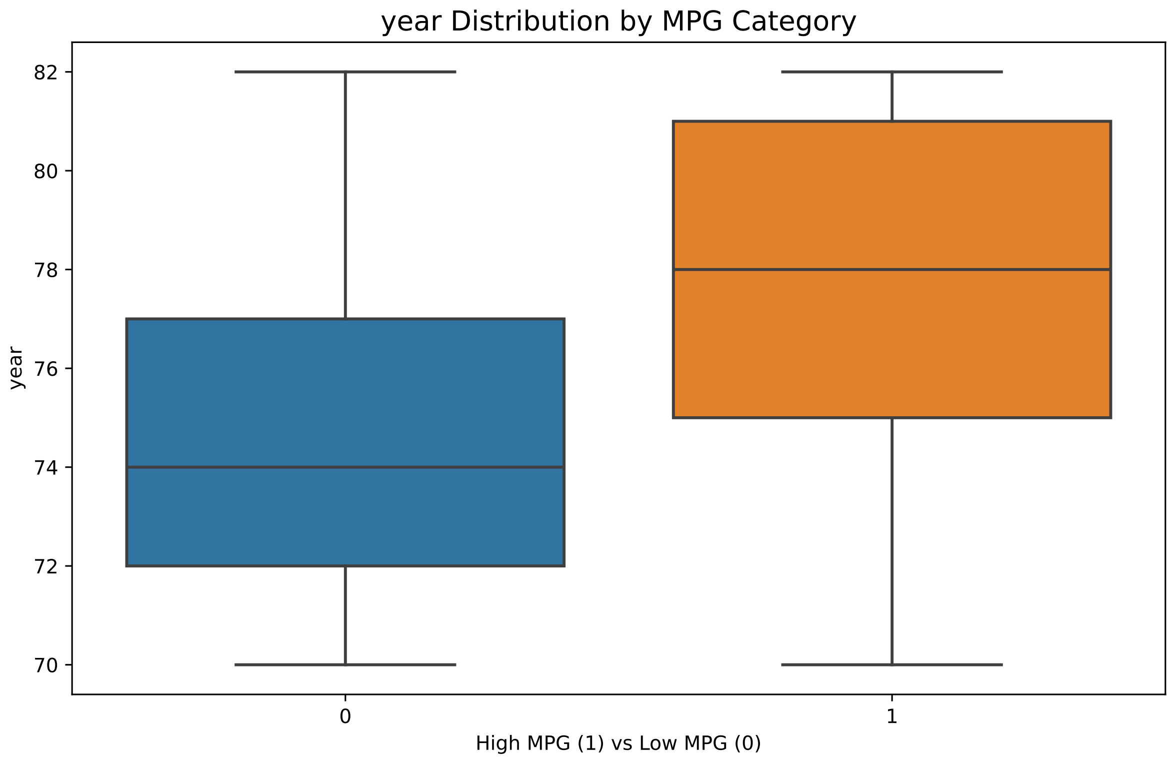
cylinders Distribution by MPG Category
cylinders
8
7
6
5
4
3
0
1
High MPG (1) vs Low MPG (0)

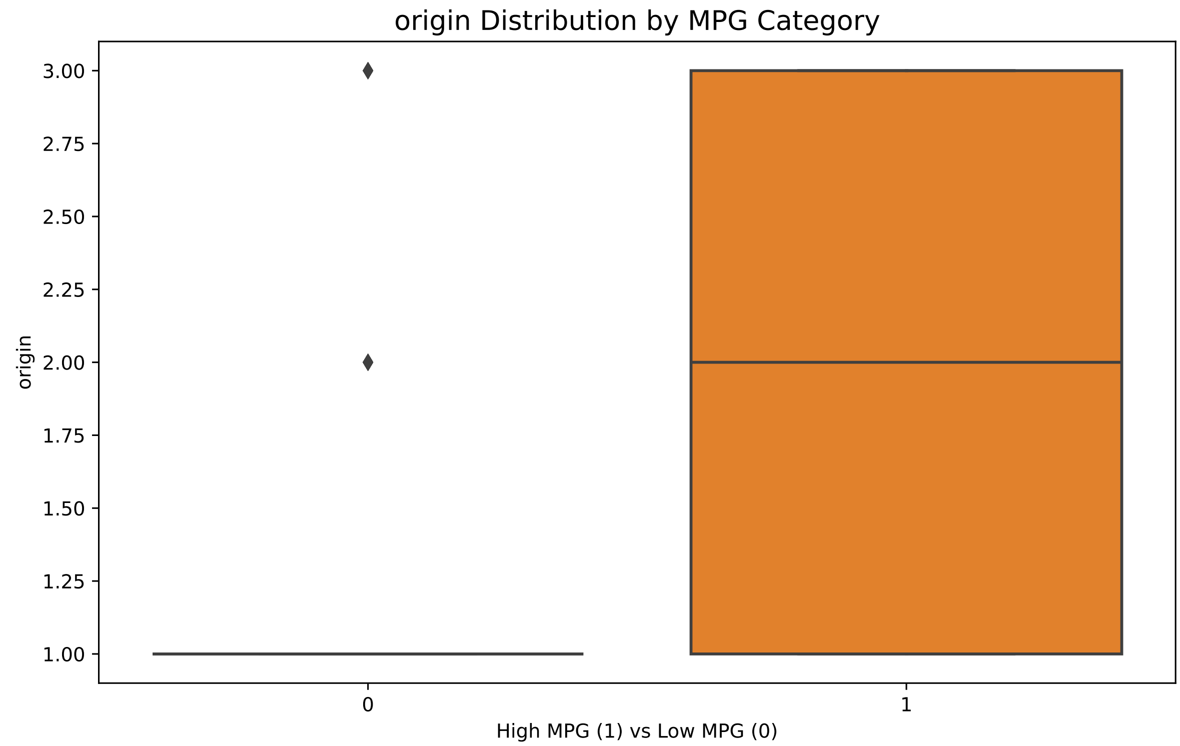

displacement Distribution by MPG Category
displacement
450
400
350
300
250
200
150
100
50
0
1
High MPG (1) vs Low MPG (0)


horsepower Distribution by MPG Category
horsepower
225
200
175
150
125
100
75
50
0
1
High MPG (1) vs Low MPG (0)


weight Distribution by MPG Category
weight
5000
4500
4000
3500
3000
2500
2000
1500
0
1
High MPG (1) vs Low MPG (0)


acceleration Distribution by MPG Category
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
acceleration
0
1
High MPG (1) vs Low MPG (0)


year Distribution by MPG Category
82
80
78
76
74
72
70
year
0
1
High MPG (1) vs Low MPG (0)



From the scatterplots, we see a bit of a patterm from horsepower, acceleration,and weight, where mpg01 being equal to 0 or 1 is more likely based on whether these variables have higher or lower values. ALthough there is some overlap, it at least shows greater distinctions compared to the other variables.

From the boxplots, we can see that the median weight of mgp01 =1 cars much lower than that of mpg01 = 0 cars.So it may suggest heavier cars have lower mpg. We also see that mpg01=1 cars have much lower horsepower as mpg01 = 0 cars (though they have more variance in values)While acceleration shows less clear separation than weight/horsepower in the plots, it still shows a trend where faster acceleration (lower values) tends to be associated with lower mpg (mpg01=0)

Meanwhile, the other vairables don't show as clear patterns or have more overlaps in terms of the values of mpg01=0 or =1.

4.c.Splitting to traiing and test set

```
X = auto_df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
 ↪  'acceleration', 'year', 'origin']]

y = auto_df["mpg01"]
```

```python
# Train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
    random_state=22)


# Sanity check for index matching
display(X_train.head(), X_test.head(), y_train.head(), y_test.head())
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
176	23.0	4	120.0	88	2957	17.0	75	2
307	41.5	4	98.0	76	2144	14.7	80	2
137	14.0	8	302.0	140	4638	16.0	74	1
18	27.0	4	97.0	88	2130	14.5	70	3
285	16.5	8	351.0	138	3955	13.2	79	1

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
280	22.3	4	140.0	88	2890	17.3	79	1
57	25.0	4	97.5	80	2126	17.0	72	1
46	19.0	6	250.0	100	3282	15.0	71	1
223	17.5	6	250.0	110	3520	16.4	77	1
303	28.4	4	151.0	90	2670	16.0	79	1

```
176    1
307    1
137    0
18     1
285    0
Name: mpg01, dtype: int32
```

```
280    0
57     1
46     0
223    0
303    1
Name: mpg01, dtype: int32
```

```python
# Make a copy of the training and test data
X_train_dummy = X_train.copy()
X_test_dummy = X_test.copy()

## Insert the dummy variable in each set.
## df.insert(column #, 'column name', value)
X_train_dummy.insert(0, 'test', 0)
X_test_dummy.insert(0, 'test', 1)


X_full = pd.concat([X_test_dummy, X_train_dummy], axis = 0)

display(X_full)
print(X_full['test'].value_counts())
```

	test	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
280	1	22.3	4	140.0	88	2890	17.3	79	1
57	1	25.0	4	97.5	80	2126	17.0	72	1
46	1	19.0	6	250.0	100	3282	15.0	71	1
223	1	17.5	6	250.0	110	3520	16.4	77	1
303	1	28.4	4	151.0	90	2670	16.0	79	1
...	...	...	...	...	...	...	...	...	...
358	0	22.4	6	231.0	110	3415	15.8	81	1
356	0	25.4	6	168.0	116	2900	12.6	81	3
300	0	34.5	4	105.0	70	2150	14.9	79	1
132	0	16.0	6	258.0	110	3632	18.0	74	1
373	0	36.0	4	98.0	70	2125	17.3	82	1

```
test
1    196
0    196
Name: count, dtype: int64
```

Run regression

```python
result = smf.ols(
    'test ~ mpg + cylinders + displacement + horsepower + weight +
     ↪  acceleration + year + origin',
    data=X_full
).fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   test   R-squared:
0.007
Model:                            OLS   Adj. R-squared:
-0.014
Method:                 Least Squares   F-statistic:
0.3430
Date:                Fri, 07 Feb 2025   Prob (F-statistic):
0.949
Time:                        21:51:49   Log-Likelihood:
-283.11
No. Observations:                 392   AIC:
584.2
Df Residuals:                     383   BIC:
620.0
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025
                   0.975]
--------------------------------------------------------------------------------
Intercept        0.1527      0.716      0.213      0.831      -1.255
1.560
mpg              0.0009      0.008      0.119      0.905      -0.014
0.016
cylinders        0.0277      0.049      0.563      0.574      -0.069
0.124
displacement    -0.0012      0.001     -1.057      0.291      -0.003
0.001
horsepower      -0.0004      0.002     -0.187      0.852      -0.005
0.004
weight           0.0001      0.000      1.165      0.245   -8.87e-05
0.000
acceleration    -0.0067      0.015     -0.448      0.654      -0.036
0.023
year             0.0024      0.010      0.248      0.804      -0.017
0.021
origin          -0.0065      0.044     -0.149      0.882      -0.092
0.079
==============================================================================
Omnibus:                     1741.652   Durbin-Watson:
0.027
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
63.402
Skew:                           0.003   Prob(JB):
1.71e-14
Kurtosis:                       1.030   Cond. No.
8.74e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 8.74e+04. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

Since all the p-values are larger than 0.1 we aren't as worried that the train and test set are significantly different, but we do want to check distribuiton of the training and test sets to see if they are balanced on characteristics we haven't included or are unobservable

```
## Check the  distribution in the training and test sets

print(f'Percentage of positive labels in the test set:
 ↪  {round(y_test.mean()*100, 2)}')
print(f'Percentage of positive labels in the training set:
 ↪  {round(y_train.mean()*100, 2)}')
```

```
Percentage of positive labels in the test set: 48.47
Percentage of positive labels in the training set: 51.53
```

4.d. LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Choosing predictors related to mpg01
X_train_rel = X_train.copy()[["horsepower", "weight", "acceleration"]]
X_test_rel = X_test.copy()[["horsepower", "weight", "acceleration"]]

# Fit the LDA model
lda_model = LinearDiscriminantAnalysis()
lda_model.fit(X_train_rel, y_train)
# view the predicted test values
y_pred_lda = lda_model.predict(X_test_rel)
y_pred_lda
```

```
array([1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1])
```

Testing error rate: #cite how to get this

```python
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve

error_rate_lda = 1 - accuracy_score(y_test, y_pred_lda)

print(f"The error rate is: {round(error_rate_lda, 4)*100}%")
```

```
The error rate is: 11.219999999999999%
```

4.e. QDA model

```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
qda_model = QuadraticDiscriminantAnalysis()
qda_model.fit(X_train_rel, y_train)

# view the predicted test values
y_pred_qda = qda_model.predict(X_test_rel)
y_pred_qda
```

```
array([1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
# Get  the error rate
error_rate_qda = 1 - accuracy_score(y_test, y_pred_qda)

print(f"The QDA model's error rate is: {round(error_rate_qda, 4)*100}%")
```

```
The QDA model's error rate is: 12.76%
```

4.f. Logistic regression

```
# Fit the model
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(max_iter=500)
logisticRegr.fit(X_train_rel, y_train)

# Predict the test set
y_pred_log = logisticRegr.predict(X_test_rel)
```

Get test error

```
error_rate_log = 1 - accuracy_score(y_test, logisticRegr.predict(X_test_rel))

# Print the error rate
print(f"The logistic regression model's error rate is: {round(error_rate_log,
 ↪  4)*100}%")
```

```
The logistic regression model's error rate is: 9.69%
```

4.g.

```
from sklearn.naive_bayes import GaussianNB
bayes_model = GaussianNB()
bayes_model.fit(X_train_rel, y_train)
# view the predicted test values
y_pred_bayes = bayes_model.predict(X_test_rel)
y_pred_bayes
# Compute the error rate
error_rate_bayes = 1 - accuracy_score(y_test, y_pred_bayes)
# Print the error rate
print(f"The Naive Bayes model has an error rate of: {round(error_rate_bayes,
 ↪  4)*100}%")
```

```
The Naive Bayes model has an error rate of: 15.310000000000002%
```

Part 5

5.a.

```
# Load the dataset
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\ps2"
default_path = os.path.join(directory, "Data-Default.csv")
default_df = pd.read_csv(default_path)
print(default_df.dtypes)
print(default_df.shape)
default_df.head(5)
```

```
default     object
student     object
balance    float64
income     float64
dtype: object
(10000, 4)
```

	default	student	balance	income
0	No	No	729.526495	44361.625074
1	No	Yes	817.180407	12106.134700
2	No	No	1073.549164	31767.138947
3	No	No	529.250605	35704.493935
4	No	No	785.655883	38463.495879

Logistic Regression of income and balance on default

```
# Cchange default into a dummy variable
default_df["default01"] = default_df["default"].map({"Yes": 1, "No": 0})

print(default_df["default01"].value_counts(normalize=True) * 100)
# Checking if it worked
yes_rows = default_df[default_df["default"] == "Yes"]
print(yes_rows.head(5))
no_rows = default_df[default_df["default"] == "No"]
print(no_rows.head(5))
```

```
default01
0    96.67
```

```
1     3.33
Name: proportion, dtype: float64
    default student      balance         income  default01
136     Yes     Yes  1486.998122  17854.397028          1
173     Yes     Yes  2205.799521  14271.492253          1
201     Yes     Yes  1774.694223  20359.506086          1
206     Yes      No  1889.599190  48956.171589          1
209     Yes     Yes  1899.390626  20655.200003          1
  default student      balance         income  default01
0      No      No   729.526495  44361.625074          0
1      No     Yes   817.180407  12106.134700          0
2      No      No  1073.549164  31767.138947          0
3      No      No   529.250605  35704.493935          0
4      No      No   785.655883  38463.495879          0
```

```
# Defining X and y
X_default = default_df[["income", "balance"]]
y_default = default_df["default01"]

# Logisitc regression model
default_logit_reg = LogisticRegression(max_iter=500)
default_logit_reg.fit(X_default,y_default)
```

```
LogisticRegression(max_iter=500)
```

5.b.Spit data, random seed 42, .7

```
X_train, X_validation, y_train, y_validation = train_test_split(X_default,
↪   y_default, train_size=0.7, random_state=42)
# Sanity check
display(X_train.head(), X_validation.head(), y_train.head(),
↪   y_validation.head())
```

	income	balance
9069	41239.020510	0.000000
2603	37073.192381	961.999353
7738	19039.168273	655.611221
1579	27690.113535	864.047198
5058	57561.411261	1306.832034

	income	balance
6252	31507.089277	1435.662933
4684	42139.070269	771.789347
1731	21809.218509	0.000000
4742	32803.832648	113.571264
4521	49903.597081	1358.132472

```
9069    0
2603    0
7738    0
1579    0
5058    0
Name: default01, dtype: int64
```

```
6252    0
4684    0
1731    0
4742    0
4521    0
Name: default01, dtype: int64
```

```
# Fit the training data into logistic regression
default_logit_train= LogisticRegression(max_iter=500)
default_logit_train.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=500)
```

```
# Predict the validation set
y_pred_log = default_logit_train.predict_proba(X_validation)[:, 1]
print("Predicted probabilities above 0.5:", y_pred_log[y_pred_log >= 0.5])
print("Count of values >= 0.5:", len(y_pred_log[y_pred_log >= 0.5]))
```

```
Predicted probabilities above 0.5: [0.54364253]
Count of values >= 0.5: 1
```

```
# Classifying to default category if porbablity is > 0.5
y_pred_log = np.array(y_pred_log)
y_pred_log = y_pred_log.astype(float)
y_default_category = np.where(y_pred_log >= 0.5, 1, 0)
print(y_pred_log[:10])  # First 10 predictions
print(type(y_pred_log))  # Type check
print(pd.Series(y_default_category).value_counts())
```

```
[0.03355695 0.00641997 0.05732826 0.01540883 0.00314391 0.02882375
 0.00609724 0.05234981 0.01991224 0.09350953]
<class 'numpy.ndarray'>
0    2999
1       1
Name: count, dtype: int64
```

```
# Compute the error rate
error_valid = 1 - accuracy_score(y_validation, y_default_category)
# Print the error
print(f"The validation set error is {round(error_valid, 4)*100}%")
```

```
The validation set error is 3.17%
```

5.c.

```
X = default_df[["income", "balance"]]
y = default_df["default01"]


random_states = [2, 6, 9]
error_rates = []

for state in random_states:
    # Split data with current random state
    X_train, X_validation, y_train, y_validation = train_test_split(
        X, y,
        train_size=0.7,
        random_state=state
    )

    # Train model
    default_logit_train = LogisticRegression(max_iter=500)
    default_logit_train.fit(X_train, y_train)

    # Predict and calculate error
    y_pred_log = default_logit_train.predict_proba(X_validation)[:, 1]
    y_default_category = np.where(y_pred_log > 0.5, 1, 0)
    error_rate = 1 - accuracy_score(y_validation, y_default_category)

    error_rates.append(error_rate)
    print(f"Random state {state}: validation error = {error_rate:.2%}")
```

```python
# Analyze results
print("\nSummary:")
print(f"Average error rate: {np.mean(error_rates):.2%}")
print(f"Standard deviation: {np.std(error_rates):.2%}")
```

```
Random state 2: validation error = 2.37%
Random state 6: validation error = 2.47%
Random state 9: validation error = 3.07%

Summary:
Average error rate: 2.63%
Standard deviation: 0.31%
```

- Consistency: The error rates across the three relatively close, ranging from 2.37% to 3.07%, meaning it's not overly sensitive to how the data is split
- Low Error Rates: They all have lower error rates than the random state 42 split. Slight Variability: There is some variability in the results, with a standard deviation of 0.31%. This variability is expected due to the random nature of the splits and demonstrates the importance of using multiple splits to assess model performance.

These results give us some level of confidence in the model's performance and its ability to generalize to new data. Butmaybe doing the k-fold cross-validation will get us an even more robust estimate of the model's performance.

5.d.

```python
# Create dummy variable for student
default_df["student01"] = default_df["student"].map({"Yes": 1, "No": 0})

# Checking if it worked
yes_rows = default_df[default_df["student"] == "Yes"]
print(yes_rows.head(5))
no_rows = default_df[default_df["student"] == "No"]
print(no_rows.head(5))
```

```
   default student      balance        income  default01  student01
1       No     Yes   817.180407  12106.134700          0          1
5       No     Yes   919.588530   7491.558572          0          1
7       No     Yes   808.667504  17600.451344          0          1
10      No     Yes     0.000000  21871.073089          0          1
11      No     Yes  1220.583753  13268.562221          0          1
  default student      balance        income  default01  student01
```

```
0       No      No    729.526495  44361.625074          0           0
2       No      No   1073.549164  31767.138947          0           0
3       No      No    529.250605  35704.493935          0           0
4       No      No    785.655883  38463.495879          0           0
6       No      No    825.513331  24905.226578          0           0
```

```
# Deine X, y
X = default_df[["income", "balance", "student01"]]
y = default_df["default01"]

# Logisitc regression model
student_logit_reg = LogisticRegression(max_iter=500)
student_logit_reg.fit(X,y)
```

```
LogisticRegression(max_iter=500)
```

```
X_train, X_validation, y_train, y_validation = train_test_split(X, y,
 ↪  train_size=0.7, random_state=42)
# Sanity check
display(X_train.head(), X_validation.head(), y_train.head(),
 ↪  y_validation.head())
```

	income	balance	student01
9069	41239.020510	0.000000	0
2603	37073.192381	961.999353	0
7738	19039.168273	655.611221	1
1579	27690.113535	864.047198	0
5058	57561.411261	1306.832034	0

	income	balance	student01
6252	31507.089277	1435.662933	0
4684	42139.070269	771.789347	0
1731	21809.218509	0.000000	0
4742	32803.832648	113.571264	0
4521	49903.597081	1358.132472	0

```
9069    0
2603    0
7738    0
```

```
1579    0
5058    0
Name: default01, dtype: int64

6252    0
4684    0
1731    0
4742    0
4521    0
Name: default01, dtype: int64
```

```
# Fit the training data into logistic regression
student_logit_train= LogisticRegression(max_iter=500)
student_logit_train.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=500)
```

```
# Predict the validation set
student_y_pred_log = student_logit_train.predict_proba(X_validation)[:, 1]
student_y_pred_log[:5]
```

```
array([0.21470633, 0.00421135, 0.00278732, 0.00100267, 0.01614738])
```

```
# Classifying to default category if porbablity is > 0/5
student_y_default_category = np.where(student_y_pred_log > 0.5, 1 , 0)
```

```
# Compute the error rate
error_valid = 1 - accuracy_score(y_validation, student_y_default_category)
# Print the error
print(f"The validation set error is {round(error_valid, 4)*100}%")
print(pd.Series(student_y_default_category).value_counts())
```

```
The validation set error is 3.17%
0    2961
1      39
Name: count, dtype: int64
```

Adding in student dummy variable didn't change the test error rate of the validation set. This can be interpreted as: being a student doesn't affect one's probability of default, all else equal. This doesn't match with our expectations because being a student probably affects default. Maybe if we added in the other variable like balance and income into the model, this may lower the error rate.