HW3

Zeshi Feng

Exercise1

(a)

- On the training set, QDA is more flexible and can fit the data more closely, even if the true boundary is linear. Therefore, QDA will generally perform better (lower training error).
- On the test set, Because QDA is more complex, it may overfit the training data when the true decision boundary is linear. As a result, LDA, which correctly assumes linearity, will typically perform better on the test set (lower test error).

(b)

- On the training set, QDA is more flexible and can fit complex, non-linear boundaries. Therefore, it will generally achieve lower training error than LDA.
- On the test set, Since QDA can capture non-linear decision boundaries, it is expected to perform better on the test set as well — provided that there are enough training observations to estimate the model parameters reliably. If the sample size is small, QDA might overfit; but in general, with sufficient data, QDA will outperform LDA on both training and test sets.

(c)

As the sample size n increases, the test prediction accuracy of QDA relative to LDA is expected to improve. QDA is a more flexible model than LDA because it allows each class to have its own covariance matrix, resulting in a quadratic (nonlinear) decision boundary. However, this flexibility comes at the cost of estimating more parameters. When the sample size is small, these additional parameters make QDA prone to high variance and overfitting, so LDA often performs better. As n increases, the covariance estimates in QDA become more reliable,

reducing its variance. Therefore, QDA's ability to capture nonlinear boundaries improves, and its performance relative to LDA tends to improve with larger sample sizes.

(d)

FALSE, Even though QDA is flexible enough to model a linear decision boundary (since a linear boundary is a special case of a quadratic one), it generally does not achieve a lower test error than LDA when the true Bayes boundary is linear. QDA estimates a separate covariance matrix for each class, requiring far more parameters than LDA. This added flexibility increases the model's variance. When the true boundary is linear, LDA's assumptions are correct, so it achieves lower bias and much lower variance. As a result, LDA typically yields better test performance in this scenario, while QDA's extra flexibility only adds estimation noise without improving bias.

Exercise2

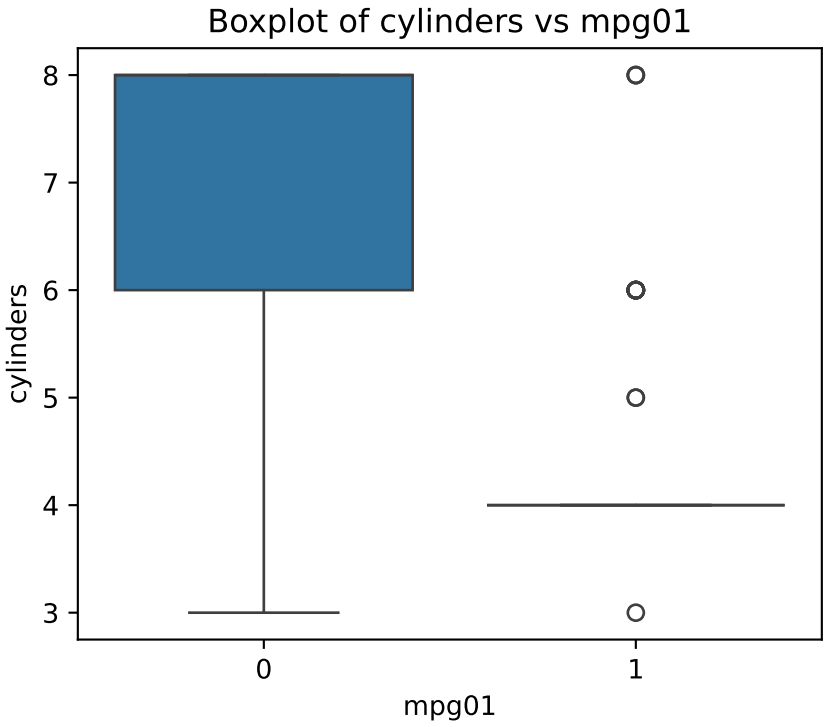
```
import pandas as pd
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantA
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import ISLP as islp
Auto = islp.load_data("Auto")
```

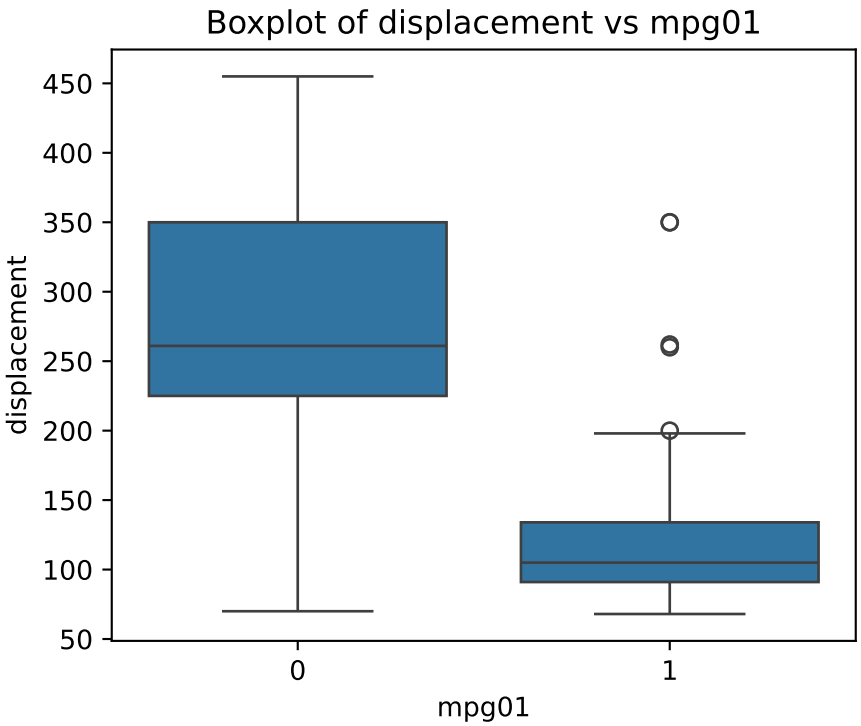
(a)

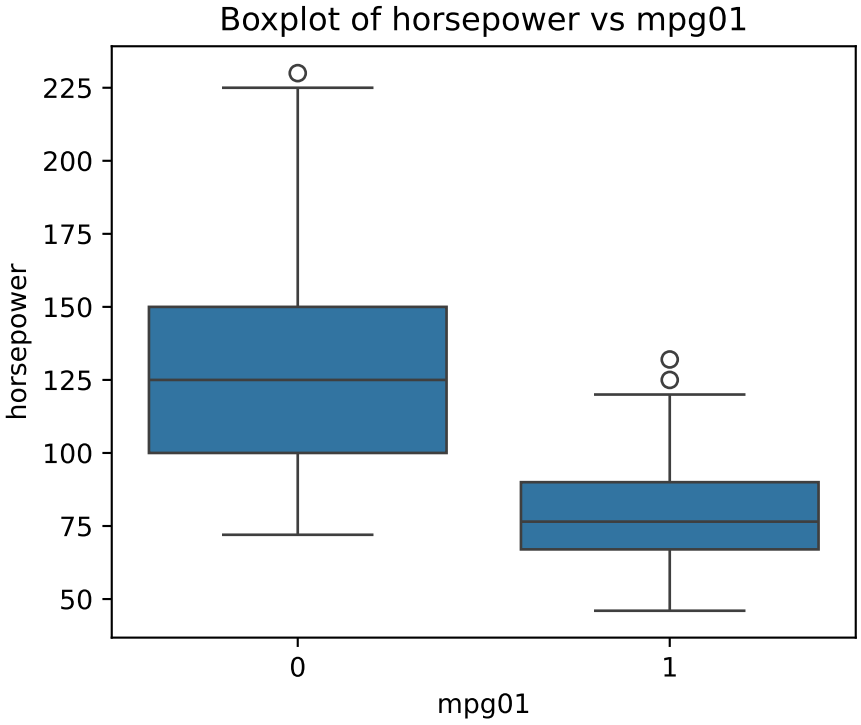
```
mpg_median = Auto['mpg'].median()
Auto['mpg01'] = (Auto['mpg'] > mpg_median).astype(int)
```

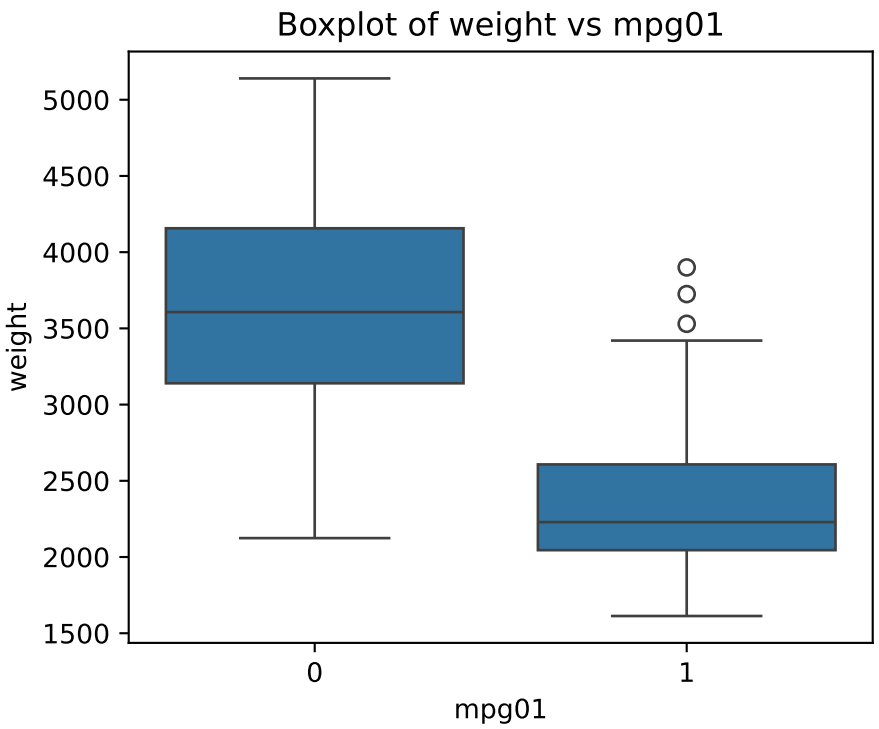
(b)

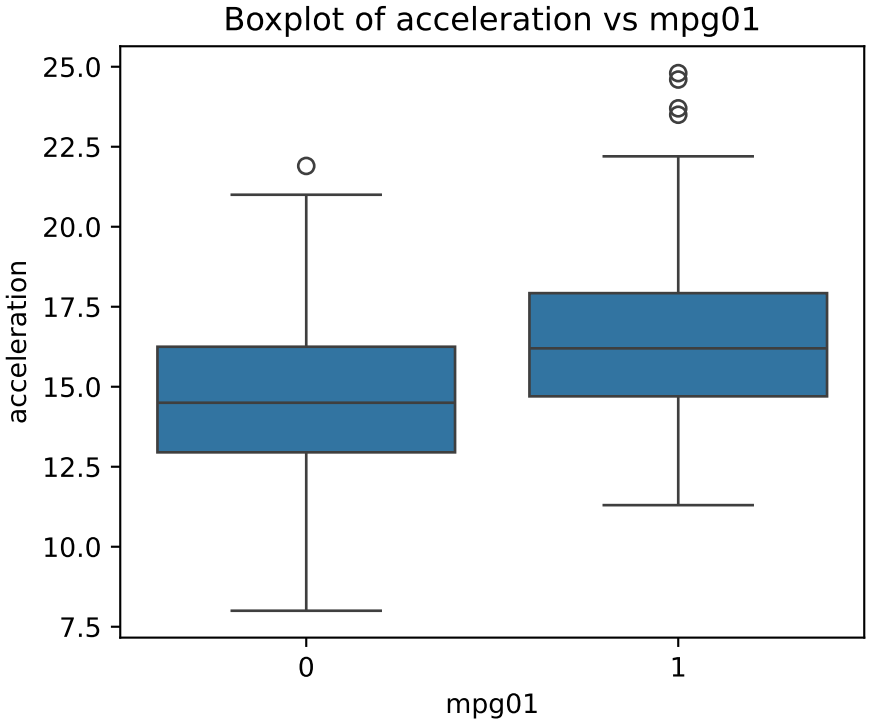
```
features = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'ori

for col in features:
    plt.figure(figsize=(5,4))
    sns.boxplot(x='mpg01', y=col, data=Auto)
    plt.title(f'Boxplot of {col} vs mpg01')
    plt.show()

sns.pairplot(Auto, vars=['displacement', 'horsepower', 'weight', 'acceleration', 'year'], hue
```

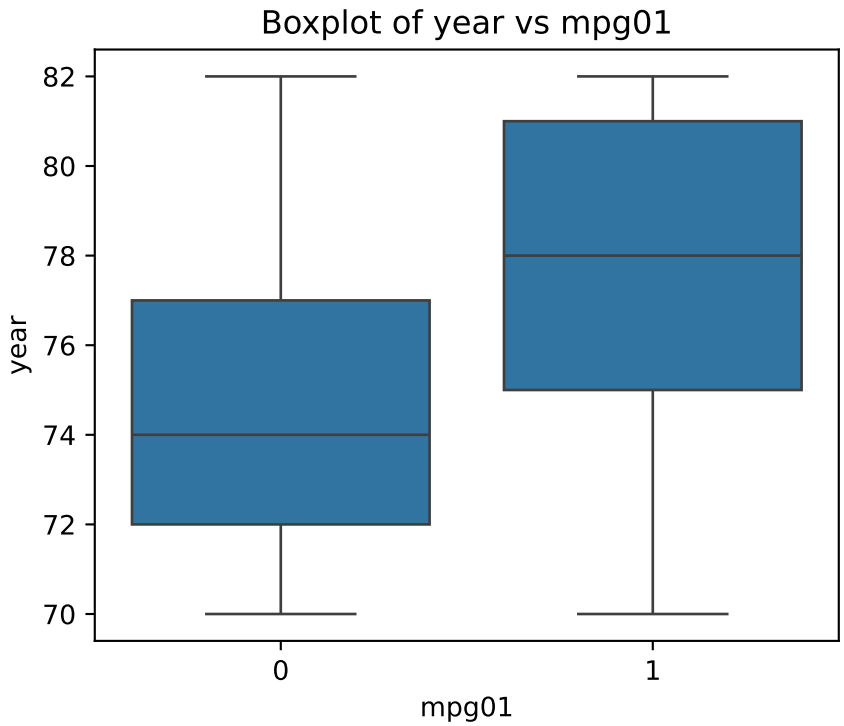


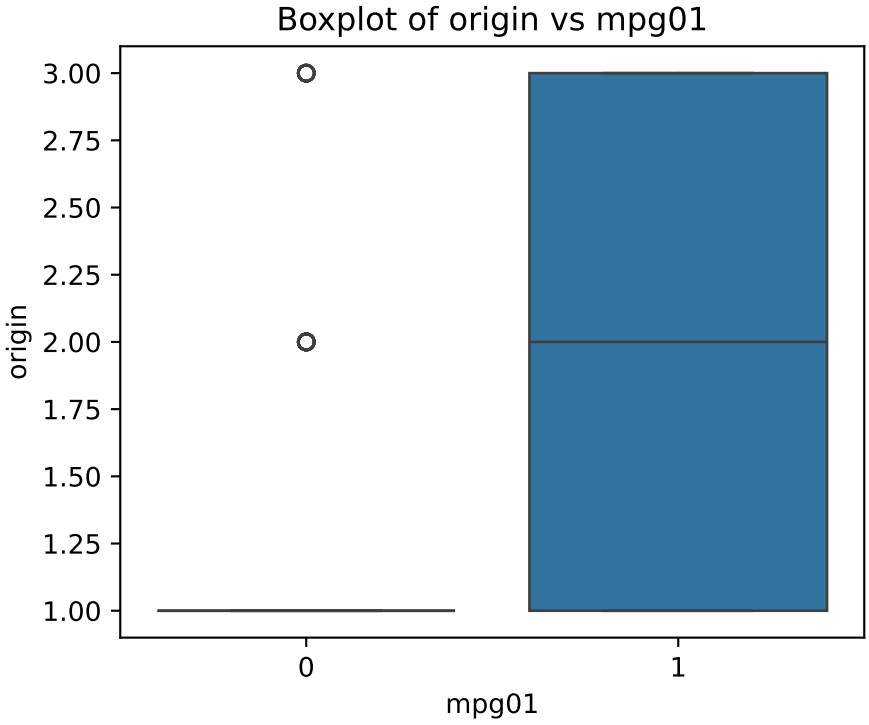

Boxplot of displacement vs mpg01
displacement
450
400
350
300
250
200
150
100
50
0
1
mpg01

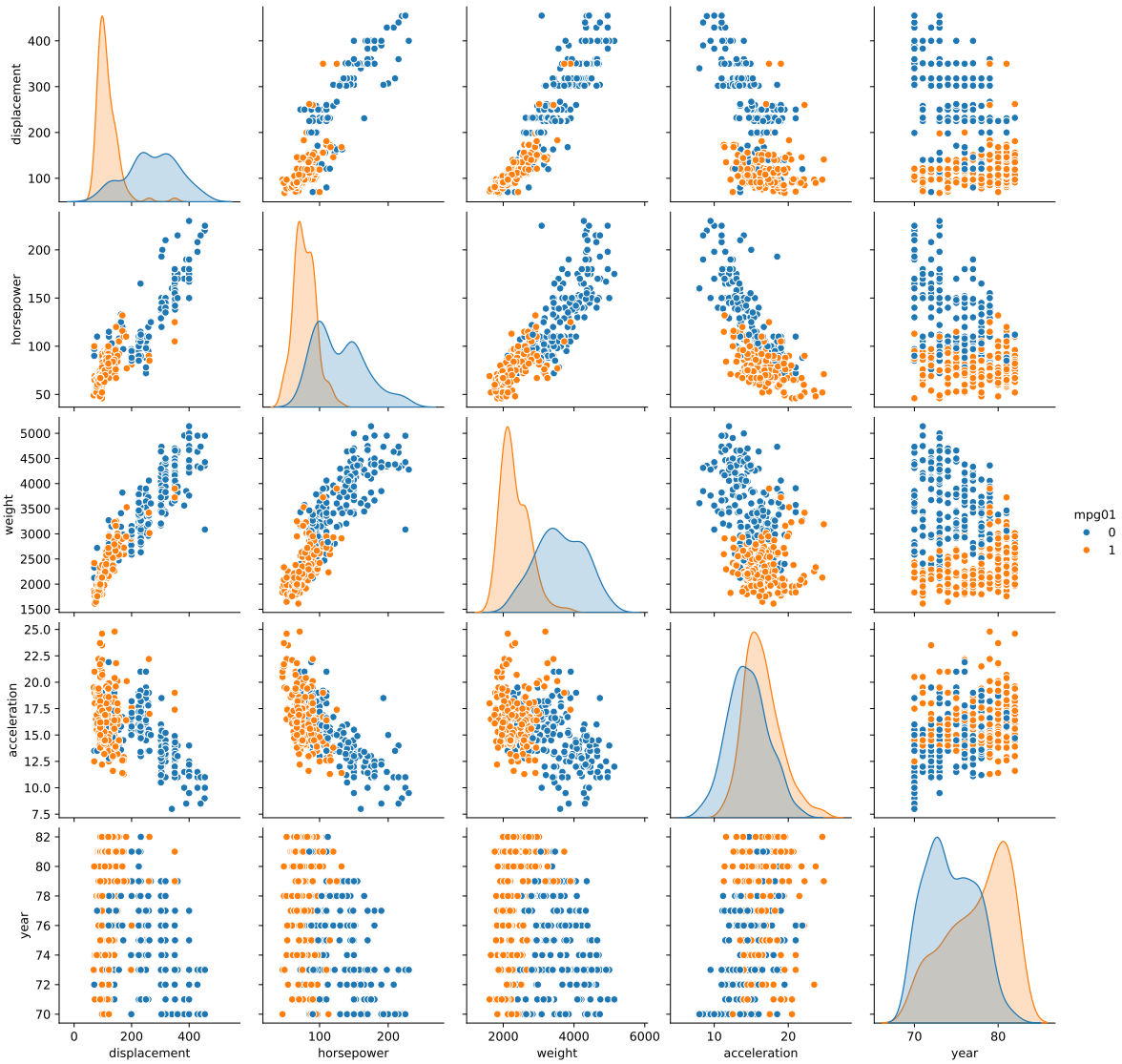

Boxplot of horsepower vs mpg01
horsepower
225
200
175
150
125
100
75
50
0
1
mpg01


Boxplot of weight vs mpg01
weight
5000
4500
4000
3500
3000
2500
2000
1500
0
1
mpg01


Boxplot of acceleration vs mpg01
acceleration
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
0
1
mpg01


Boxplot of year vs mpg01
82
80
78
76
74
72
70
year
0
1
mpg01


Boxplot of origin vs mpg01
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
origin
0
1
mpg01



The pairplot shows clear separation between the two classes (mpg01 = 0 and mpg01 = 1): Displacement, Horsepower, and Weight: Strong negative association with mpg01. High-mileage cars tend to have smaller displacement, lower horsepower, and lighter weight. Year: Positively associated — newer cars are more fuel-efficient. Acceleration: Shows little difference between the two groups. * Therefore, displacement, horsepower, weight, and year are the most useful predictors for classifying cars into high vs. low gas mileage categories.

(c)

```
X = Auto[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'orig
y = Auto['mpg01']

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

(d)

```
features = ['cylinders', 'displacement', 'horsepower', 'weight', 'year']
X_train_lda = X_train[features]
X_test_lda = X_test[features]

lda = LinearDiscriminantAnalysis()
lda.fit(X_train_lda, y_train)

y_pred = lda.predict(X_test_lda)

accuracy = accuracy_score(y_test, y_pred)
test_error = 1 - accuracy

print("Test Accuracy:", round(accuracy, 3))
print("Test Error:", round(test_error, 3))
```

```
Test Accuracy: 0.915
Test Error: 0.085
```

(e)

```
features = ['cylinders', 'displacement', 'horsepower', 'weight', 'year']


qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train[features], y_train)
y_pred_qda = qda.predict(X_test[features])
```

```
accuracy_qda = accuracy_score(y_test, y_pred_qda)
test_error_qda = 1 - accuracy_qda

print("Test Accuracy:", round(accuracy_qda, 3))
print("Test Error:", round(test_error_qda, 3))
```

```
Test Accuracy: 0.915
Test Error: 0.085
```

(f)

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train[features], y_train)

y_pred_log = log_reg.predict(X_test[features])

accuracy_log = accuracy_score(y_test, y_pred_log)
test_error_log = 1 - accuracy_log

print("Test Accuracy:", round(accuracy_log, 3))
print("Test Error:", round(test_error_log, 3))
```

```
Test Accuracy: 0.915
Test Error: 0.085
```

(g)

```
nb = GaussianNB()
nb.fit(X_train[features], y_train)

y_pred_nb = nb.predict(X_test[features])

accuracy_nb = accuracy_score(y_test, y_pred_nb)
test_error_nb = 1 - accuracy_nb

print("Test Accuracy:", round(accuracy_nb, 3))
print("Test Error:", round(test_error_nb, 3))
```

```
Test Accuracy: 0.924
Test Error: 0.076
```

(h)

```
test_errors = []

for k in [1, 3, 5, 7, 9, 11, 15, 20]:
    knn = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=k))
    knn.fit(X_train[features], y_train)
    y_pred_knn = knn.predict(X_test[features])
    accuracy = accuracy_score(y_test, y_pred_knn)
    test_error = 1 - accuracy
    test_errors.append((k, round(test_error, 3)))

for k, err in test_errors:
    print(f"K = {k:2d}, Test Error = {err}")
```

```
K =  1, Test Error = 0.076
K =  3, Test Error = 0.059
K =  5, Test Error = 0.076
K =  7, Test Error = 0.068
K =  9, Test Error = 0.068
K = 11, Test Error = 0.068
K = 15, Test Error = 0.068
K = 20, Test Error = 0.076
```

Exercise3

```
from palmerpenguins import load_penguins
penguins = load_penguins()
cols = ['species', 'island', 'sex']
penguins[cols] = penguins[cols].astype("category")
penguins
```

```
/opt/anaconda3/envs/stat4255/lib/python3.11/site-packages/palmerpenguins/penguins.py:2: UserW
11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	ma
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	fem
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	fem
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	Na
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	fem
...	...	...	...	...	...	...	...
339	Chinstrap	Dream	55.8	19.8	207.0	4000.0	ma
340	Chinstrap	Dream	43.5	18.1	202.0	3400.0	fem
341	Chinstrap	Dream	49.6	18.2	193.0	3775.0	ma
342	Chinstrap	Dream	50.8	19.0	210.0	4100.0	ma
343	Chinstrap	Dream	50.2	18.7	198.0	3775.0	fem

(1)

```
len(penguins)
penguins['species'].nunique()
penguins[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']].isna().sum
```

```
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
dtype: int64
```

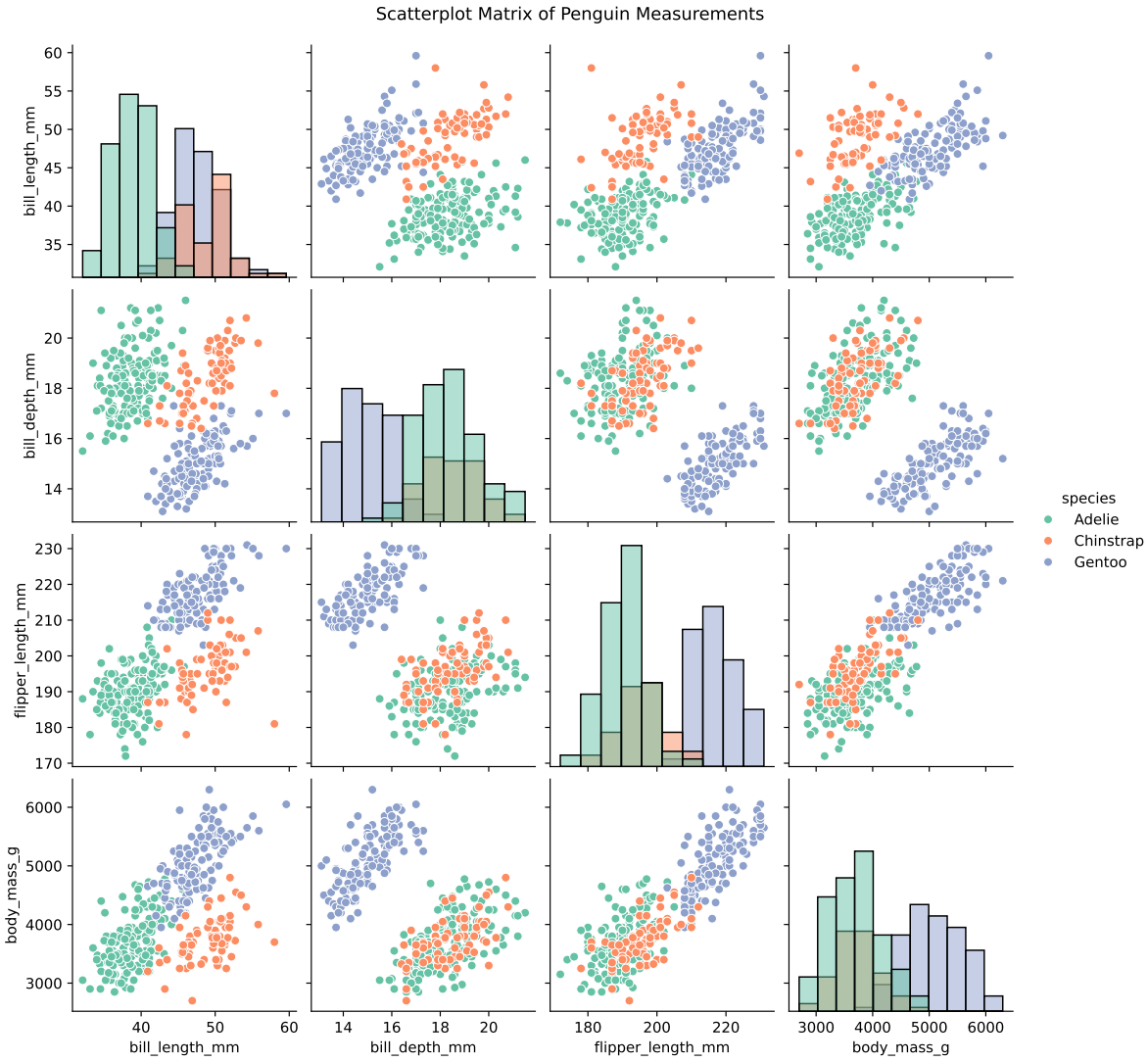
(2)

```
penguins_clean = penguins.dropna(
    subset=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
)
print("Before:", len(penguins))
print("After:", len(penguins_clean))
```

```
Before: 344
After: 342
```

(3)

```
sns.pairplot(penguins_clean,
             vars=['bill_length_mm', 'bill_depth_mm',
                   'flipper_length_mm', 'body_mass_g'],
             hue='species',
             palette='Set2',
             diag_kind='hist')

plt.suptitle("Scatterplot Matrix of Penguin Measurements", y=1.02)
plt.show()
```



The pair (bill_length_mm, bill_depth_mm) clearly separates the three species — Adelie has short/deep bills, Gentoo has long/shallow bills, and Chinstrap lies in between. The pair (flipper_length_mm, body_mass_g) also separates Gentoo (large, heavy) from the other two species. Other variable pairs show more overlap.

(4)

```
features = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
X = penguins_clean[features]
y = penguins_clean['species']
```

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

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

y_pred = lda.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
test_error = 1 - accuracy

print("Test Accuracy:", round(accuracy, 3))
print("Test Error:", round(test_error, 3))
```

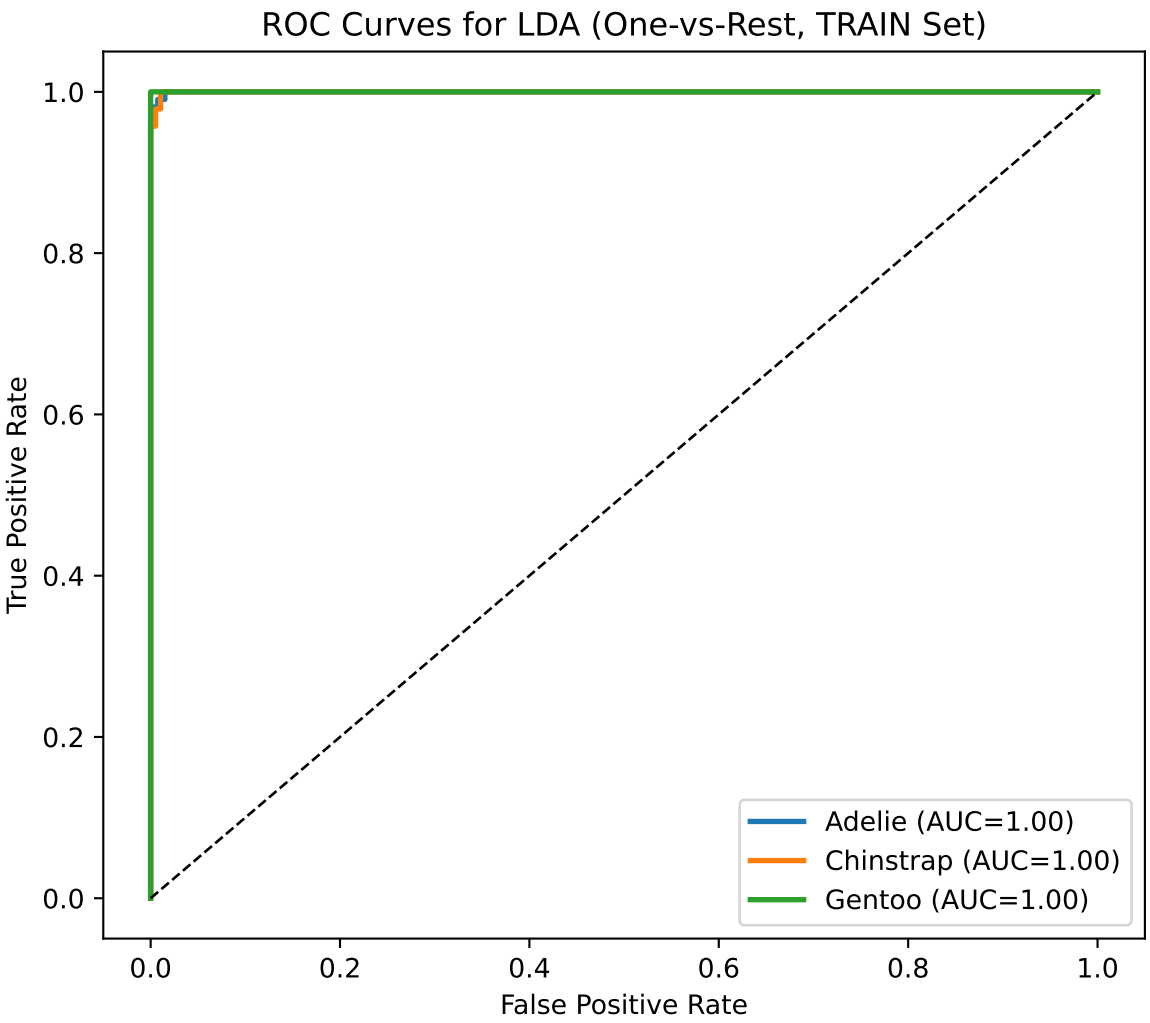
```
Test Accuracy: 0.981
Test Error: 0.019
```

(5)

```
features = ['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']
X = penguins_clean[features]
y = penguins_clean['species']

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

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

classes = lda.classes_
y_train_bin = label_binarize(y_train, classes=classes)


proba_train = lda.predict_proba(X_train)

plt.figure(figsize=(7,6))
for i, sp in enumerate(classes):
```

```
    fpr, tpr, _ = roc_curve(y_train_bin[:, i], proba_train[:, i])
    roc_auc = auc(fpr, tpr)
    print(f"TRAIN {sp} AUC = {roc_auc:.3f}")
    plt.plot(fpr, tpr, lw=2, label=f'{sp} (AUC={roc_auc:.2f})')

plt.plot([0,1],[0,1],'k--',lw=1)
plt.title("ROC Curves for LDA (One-vs-Rest, TRAIN Set)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.show()
```

```
TRAIN Adelie AUC = 1.000
TRAIN Chinstrap AUC = 1.000
TRAIN Gentoo AUC = 1.000
```



On the training set, all one-vs-rest ROC curves lie near the top-left corner with AUCs close to 1, indicating near-perfect separability of the three species using the four morphological measurements under LDA.

Exercise4

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
GLOW = pd.read_csv(
  "http://knightgu.github.io/data/GLOW.data",
  sep = "\\s+"
)
cols = ['PRIORFRAC', 'PREMENO', 'MOMFRAC', 'ARMASSIST',
        'SMOKE', 'RATERISK', 'FRACTURE']
GLOW[cols] = GLOW[cols].astype('category')
GLOW
```

	SUB_ID	SITE_ID	PHY_ID	PRIORFRAC	AGE	WEIGHT	HEIGHT	BMI	PREMEN
0	1	1	14	No	62	70.3	158	28.16055	No
1	2	4	284	No	65	87.1	160	34.02344	No
2	3	6	305	Yes	88	50.8	157	20.60936	No
3	4	6	309	No	82	62.1	160	24.25781	No
4	5	1	37	No	61	68.0	152	29.43213	No
...	...	...	...	...	...	...	...	...	...
495	496	5	287	Yes	79	63.5	157	25.76169	No
496	497	5	296	No	64	48.1	149	21.66569	No
497	498	5	287	Yes	61	70.8	161	27.31376	Yes
498	499	3	181	Yes	81	77.6	153	33.14964	No
499	500	6	317	No	63	74.8	165	27.47475	No

(1)

```
X = GLOW[['AGE', 'WEIGHT', 'PRIORFRAC', 'PREMENO', 'RATERISK']]
y = GLOW['FRACTURE'].cat.codes


num_features = ['AGE', 'WEIGHT']
cat_features = ['PRIORFRAC', 'PREMENO', 'RATERISK']

preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('cat', OneHotEncoder(drop='first'), cat_features),
        ('num', 'passthrough', num_features)
    ]
)

log_reg = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

feature_names = (
    log_reg.named_steps['preprocessor']
    .named_transformers_['cat']
    .get_feature_names_out(cat_features)
)
feature_names = np.concatenate([feature_names, num_features])

coef = log_reg.named_steps['classifier'].coef_[0]

coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coef,
    'OddsRatio': np.exp(coef)
}).sort_values(by='OddsRatio', ascending=False)

coef_df
```

```
Accuracy: 0.7133333333333334

Confusion Matrix:
```

```
 [[102  11]
 [ 32   5]]

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.90      0.83       113
           1       0.31      0.14      0.19        37

    accuracy                           0.71       150
   macro avg       0.54      0.52      0.51       150
weighted avg       0.65      0.71      0.67       150
```

	Feature	Coefficient	OddsRatio
0	PRIORFRAC_Yes	0.891310	2.438321
1	PREMENO_Yes	0.468625	1.597796
4	AGE	0.042424	1.043337
5	WEIGHT	0.000313	1.000313
3	RATERISK_Same	-0.486784	0.614600
2	RATERISK_Less	-0.696077	0.498537

(2)

The coefficient for PRIORFRAC is positive, indicating that women who had a previous fracture are more likely to experience another fracture during the first year of follow-up compared with those who did not. Specifically, holding other variables constant, the odds of having a fracture increase by a factor of $e^{\beta * PRIORFRAC}$, which suggests that a prior fracture history is a strong risk factor. For RATERISK, which has three levels ("Less," "Same," and "Greater"), the model uses "Less" as the reference category. The positive coefficients for "Same" and "Greater" imply that women who perceive their fracture risk as the same or greater than their peers have higher odds of actually experiencing a fracture compared with those who perceive their risk as less. In other words, self-perceived fracture risk is positively associated with true fracture outcomes, indicating that women's self-assessment of risk tends to align with their actual vulnerability.

(3)

```
GLOW = GLOW.dropna(subset=['AGE','PRIORFRAC','RATERISK','FRACTURE'])
GLOW['FRACTURE_NUM'] = GLOW['FRACTURE'].map({'No': 0, 'Yes': 1}).astype(int)
```

```
GLOW['PRIORFRAC'] = GLOW['PRIORFRAC'].astype('category')
GLOW['RATERISK']  = GLOW['RATERISK'].astype('category')


reduced_model = smf.logit(
    formula="FRACTURE_NUM ~ AGE + PRIORFRAC + C(RATERISK, Treatment('Less'))",
    data=GLOW
).fit()

print(reduced_model.summary())


new_data = pd.DataFrame({
    'AGE': [65],
    'PRIORFRAC': ['Yes'],
    'RATERISK': ['Same']
})
pred_prob = float(reduced_model.predict(new_data))
print(f"Predicted probability: {pred_prob:.3f}")
```

```
Optimization terminated successfully.
         Current function value: 0.518899
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:           FRACTURE_NUM   No. Observations:                  500
Model:                          Logit   Df Residuals:                      495
Method:                           MLE   Df Model:                            4
Date:                Mon, 13 Oct 2025   Pseudo R-squ.:                0.07724
Time:                        21:17:44   Log-Likelihood:                -
259.45
converged:                       True   LL-Null:                       -
281.17
Covariance Type:            nonrobust   LLR p-value:                 8.400e-
09
================================================================================================
                                             coef    std err          z      P>|z|      [
------------------------------------------------------------------------------------------------
Intercept                                 -4.9906      0.903     -5.529      0.000      -
6.760      -3.221
```

```
PRIORFRAC[T.Yes]                                0.7002      0.241      2.904      0.004       0
C(RATERISK, Treatment('Less'))[T.Greater]       0.8658      0.286      3.025      0.002       0
C(RATERISK, Treatment('Less'))[T.Same]          0.5486      0.275      1.995      0.046       0
AGE                                             0.0459      0.012      3.690      0.000       0
================================================================================================
Predicted probability: 0.319

/var/folders/4z/6pdkk_910xjdb2pb9453_b6c0000gn/T/ipykernel_71103/4046310637.py:22: FutureWarn
  pred_prob = float(reduced_model.predict(new_data))
```