HW3

Jianan Zhang

Exercise 1

(a)

- **Training performance:**
  QDA is more flexible and typically captures training patterns more closely, even when the actual decision boundary is linear. As a result, it generally achieves **lower training error**.
- **Test performance:**
  When the true decision boundary is linear, LDA's assumptions are correct, while QDA introduces extra variance by estimating separate covariance matrices. Therefore, **LDA usually yields a smaller test error** in this case.

(b)

- **Training:**
  QDA's quadratic boundaries allow it to adapt to nonlinear relationships, giving it **lower training error** than LDA.
- **Testing:**
  If the sample size is large enough to estimate QDA's parameters reliably, it tends to **outperform LDA** on the test set as well. However, with limited data, the extra flexibility of QDA may lead to overfitting.

(c)

As the sample size (n) increases, parameter estimates in QDA (especially the covariance matrices) become more accurate. Consequently, the **variance decreases**, and the relative **test accuracy of QDA improves** compared to LDA.

In small samples, QDA often overfits due to high variance, but with large (n), its flexibility becomes advantageous.

(d)

False.
Although QDA can model a linear decision boundary as a special case, its higher variance means that when the true Bayes boundary is linear, **LDA** is typically better.
LDA's simpler form has lower variance and correct bias, so the additional flexibility of QDA offers no benefit here.

Exercise 2

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

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
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score

import ISLP as islp
Auto = islp.load_data("Auto")
```

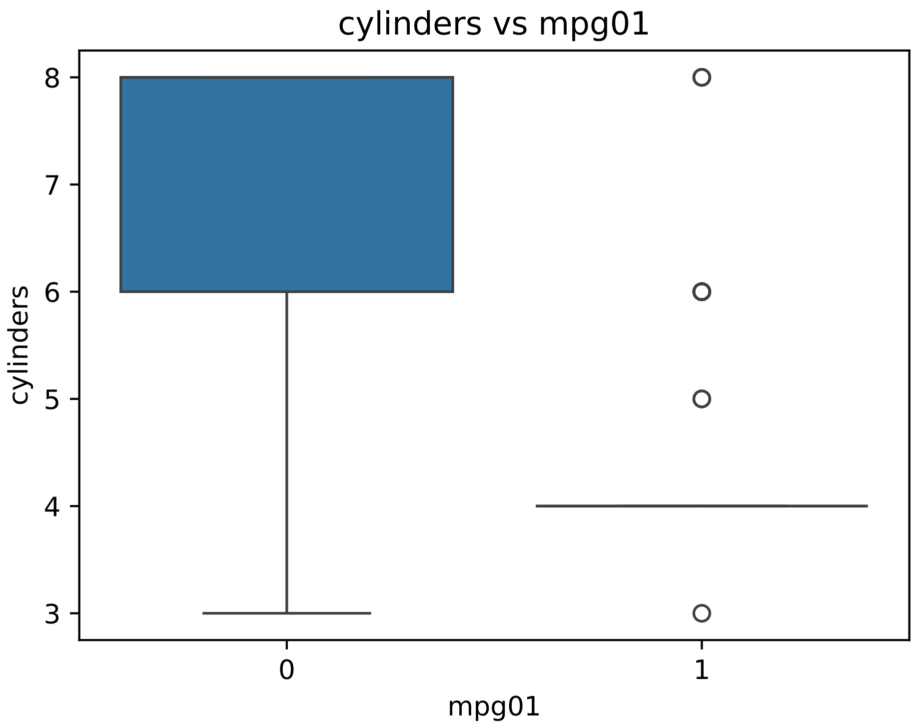
(a)

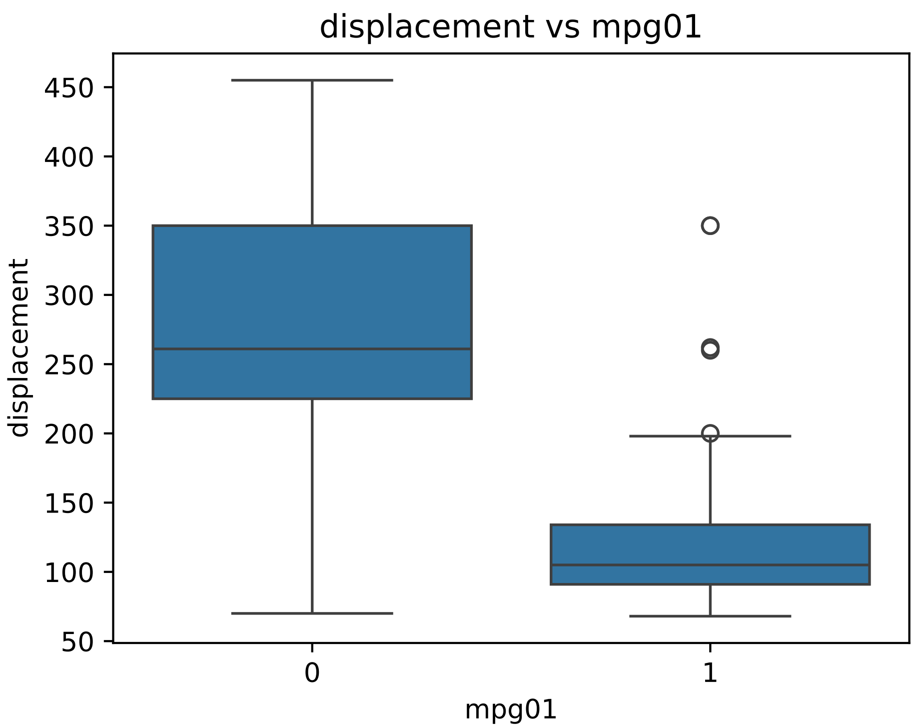
(a)

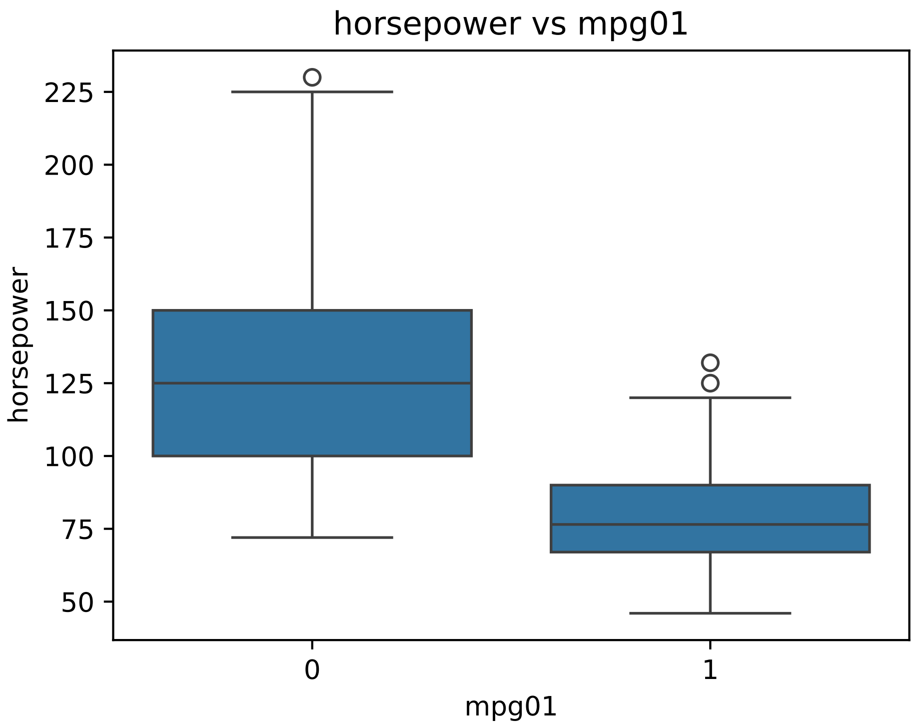
```python
mpg_median = Auto['mpg'].median()
Auto['mpg01'] = (Auto['mpg'] > mpg_median).astype(int)
```

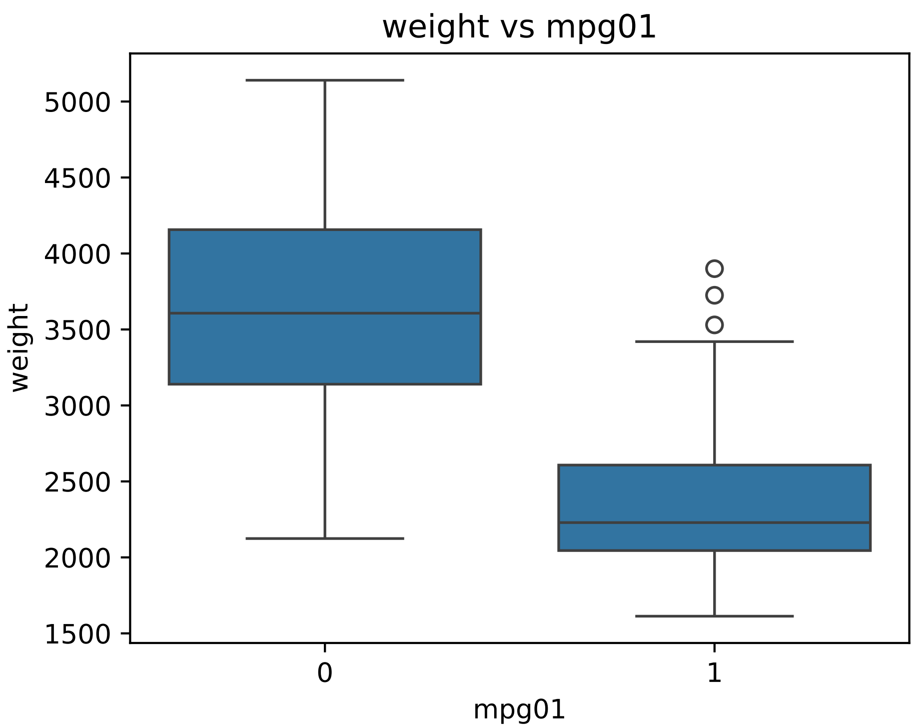
(b)

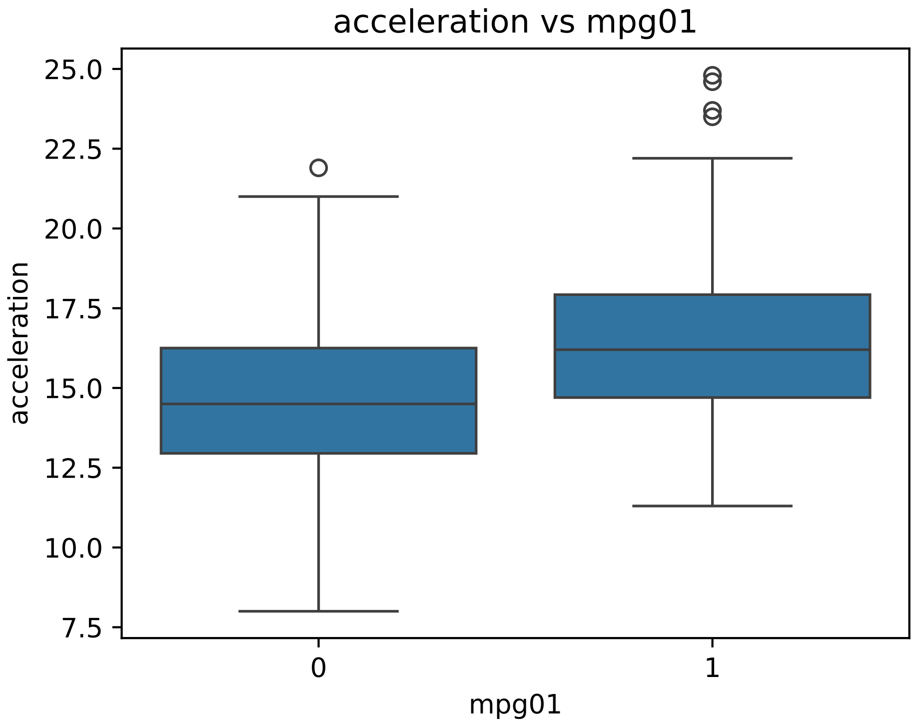
```python
cols = ['cylinders','displacement','horsepower','weight','acceleration','year','origin']

for feature in cols:
    plt.figure(figsize=(5,4))
    sns.boxplot(x='mpg01', y=feature, data=Auto)
    plt.title(f'{feature} vs mpg01')
    plt.tight_layout()
    plt.show()

sns.pairplot(
    Auto,
    vars=['displacement','horsepower','weight','acceleration','year'],
    hue='mpg01',
    diag_kind='hist'
)
plt.show()
```

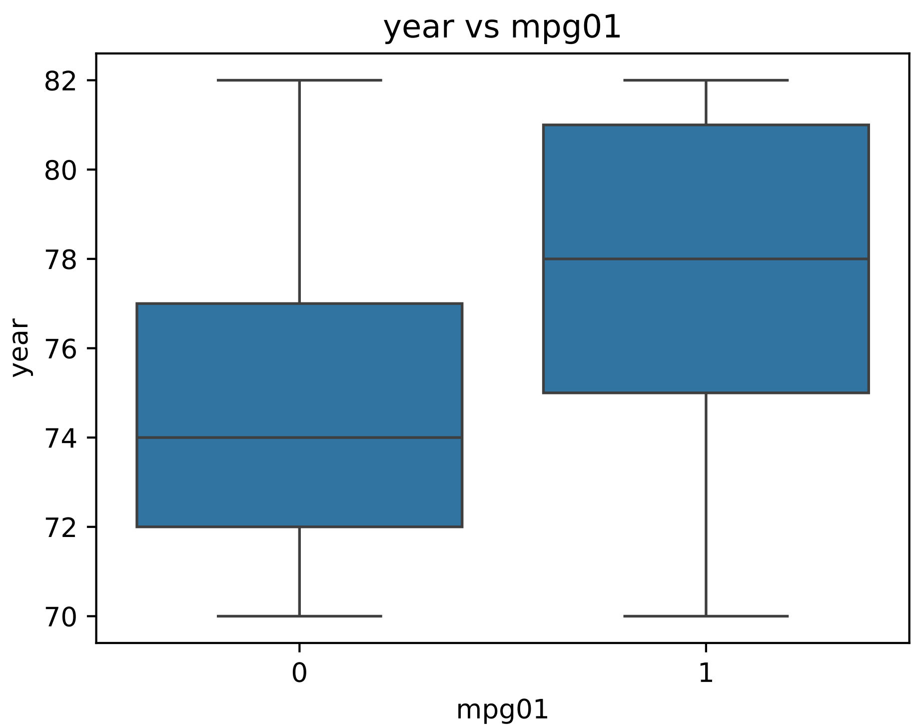

cylinders vs mpg01
cylinders
8
7
6
5
4
3
0
1
mpg01

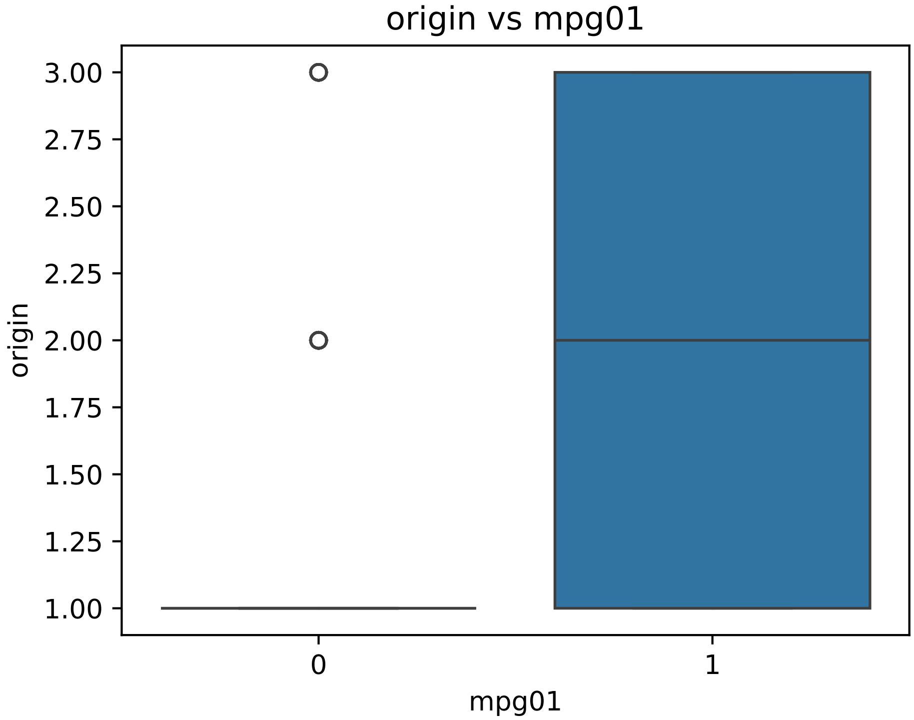

displacement vs mpg01
displacement
450
400
350
300
250
200
150
100
50
0
1
mpg01

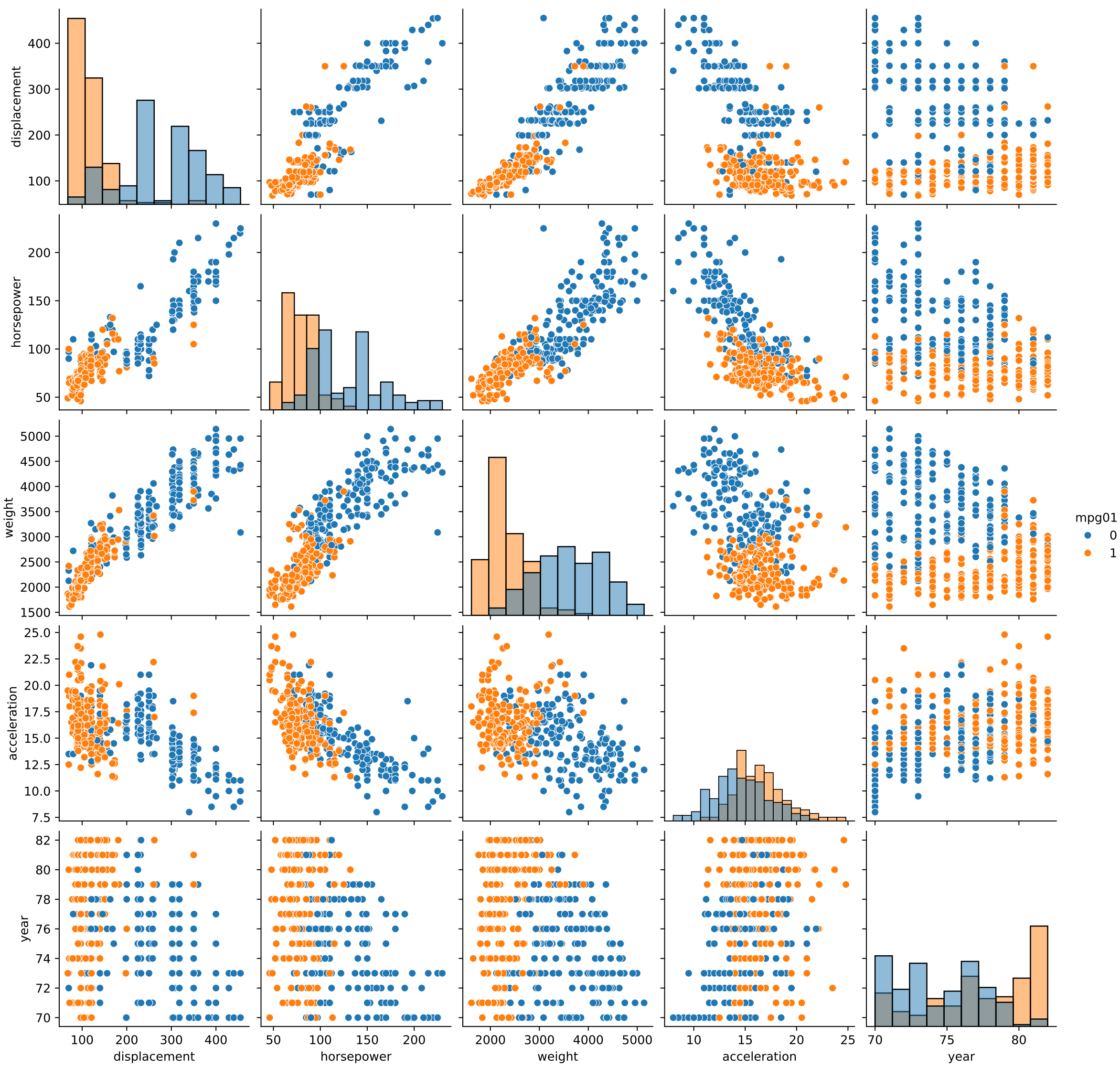

horsepower vs mpg01
horsepower
225
200
175
150
125
100
75
50
0
1
mpg01


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


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


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


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



Interpretation: Cars with higher mpg tend to have smaller displacement, lower horsepower, and lighter weight. Model year is positively associated with mpg, while acceleration differs little. Thus, displacement, horsepower, weight, and year appear most useful for classification.
(c)

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

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

(d)

```
predictors = ['cylinders','displacement','horsepower','weight','year']
lda = LinearDiscriminantAnalysis()
lda.fit(X_train[predictors], y_train)

y_pred = lda.predict(X_test[predictors])
acc = accuracy_score(y_test, y_pred)

print("Test Accuracy:", round(acc,3))
print("Test Error:", round(1-acc,3))
```

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

(e)

```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train[predictors], y_train)

y_pred_qda = qda.predict(X_test[predictors])
acc_qda = accuracy_score(y_test, y_pred_qda)

print("Test Accuracy:", round(acc_qda,3))
print("Test Error:", round(1-acc_qda,3))
```

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

(f)

```
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train[predictors], y_train)

y_pred_log = log_model.predict(X_test[predictors])
acc_log = accuracy_score(y_test, y_pred_log)
```

```python
print("Test Accuracy:", round(acc_log,3))
print("Test Error:", round(1-acc_log,3))
```

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

(g)

```python
nb_model = GaussianNB()
nb_model.fit(X_train[predictors], y_train)

y_pred_nb = nb_model.predict(X_test[predictors])
acc_nb = accuracy_score(y_test, y_pred_nb)

print("Test Accuracy:", round(acc_nb,3))
print("Test Error:", round(1-acc_nb,3))
```

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

(h)

```python
test_errs = []
for k in [1,3,5,7,9,11,15,20]:
    knn = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=k))
    knn.fit(X_train[predictors], y_train)
    pred = knn.predict(X_test[predictors])
    err = 1 - accuracy_score(y_test, pred)
    test_errs.append((k, round(err,3)))

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

```
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

```python
from palmerpenguins import load_penguins
penguins = load_penguins()
for col in ['species','island','sex']:
    penguins[col] = penguins[col].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)

```python
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
```

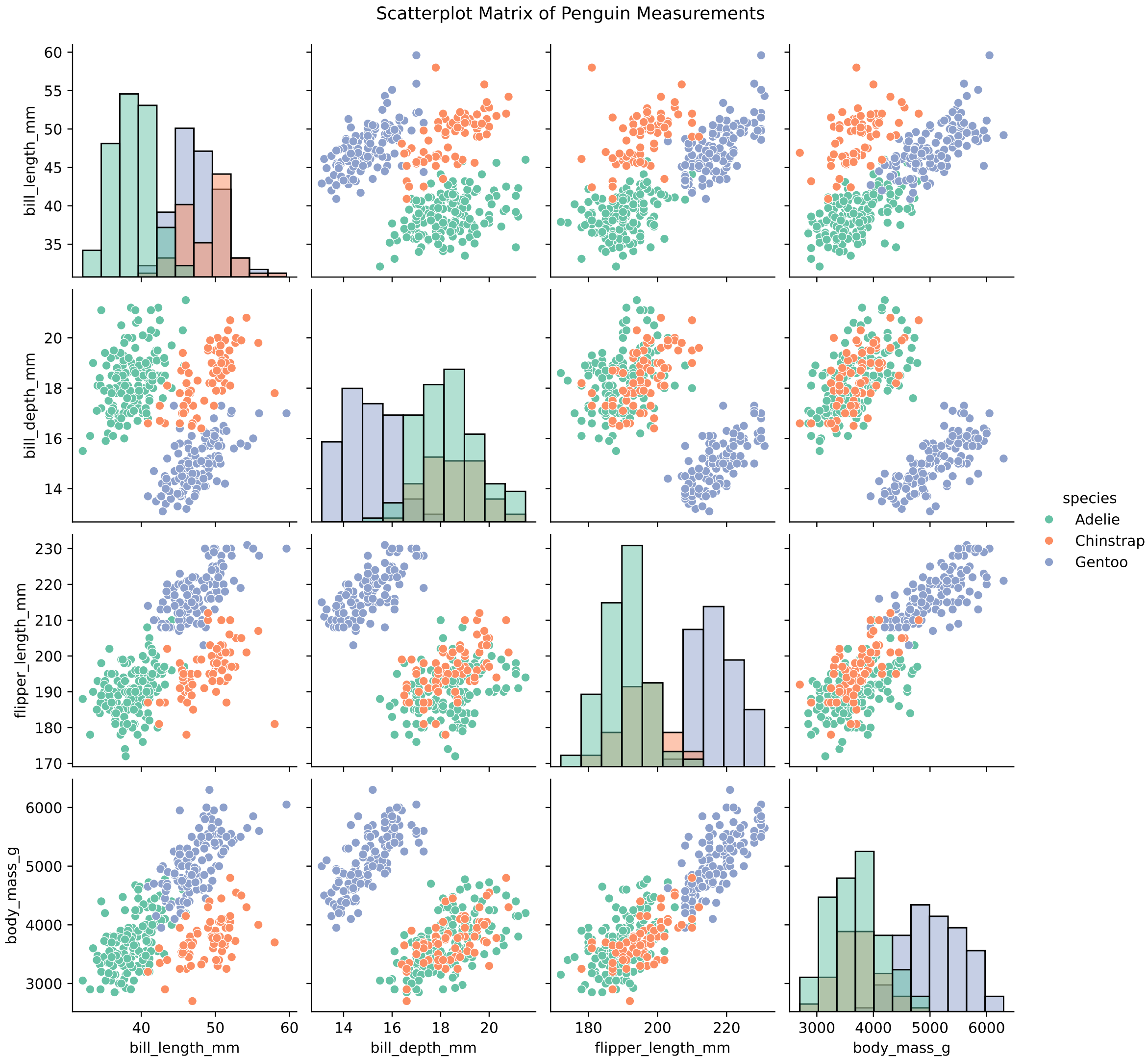
(2)

```
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()
```



Observation: Bill length vs. depth and flipper length vs. body mass effectively separate the three species. Adelie = shorter/deeper bill; Gentoo = longer/shallow bill and heavier body; Chinstrap = intermediate.

(4)

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score

features = ['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']
```

```python
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)

acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", round(acc,3))
print("Test Error:", round(1-acc,3))
```

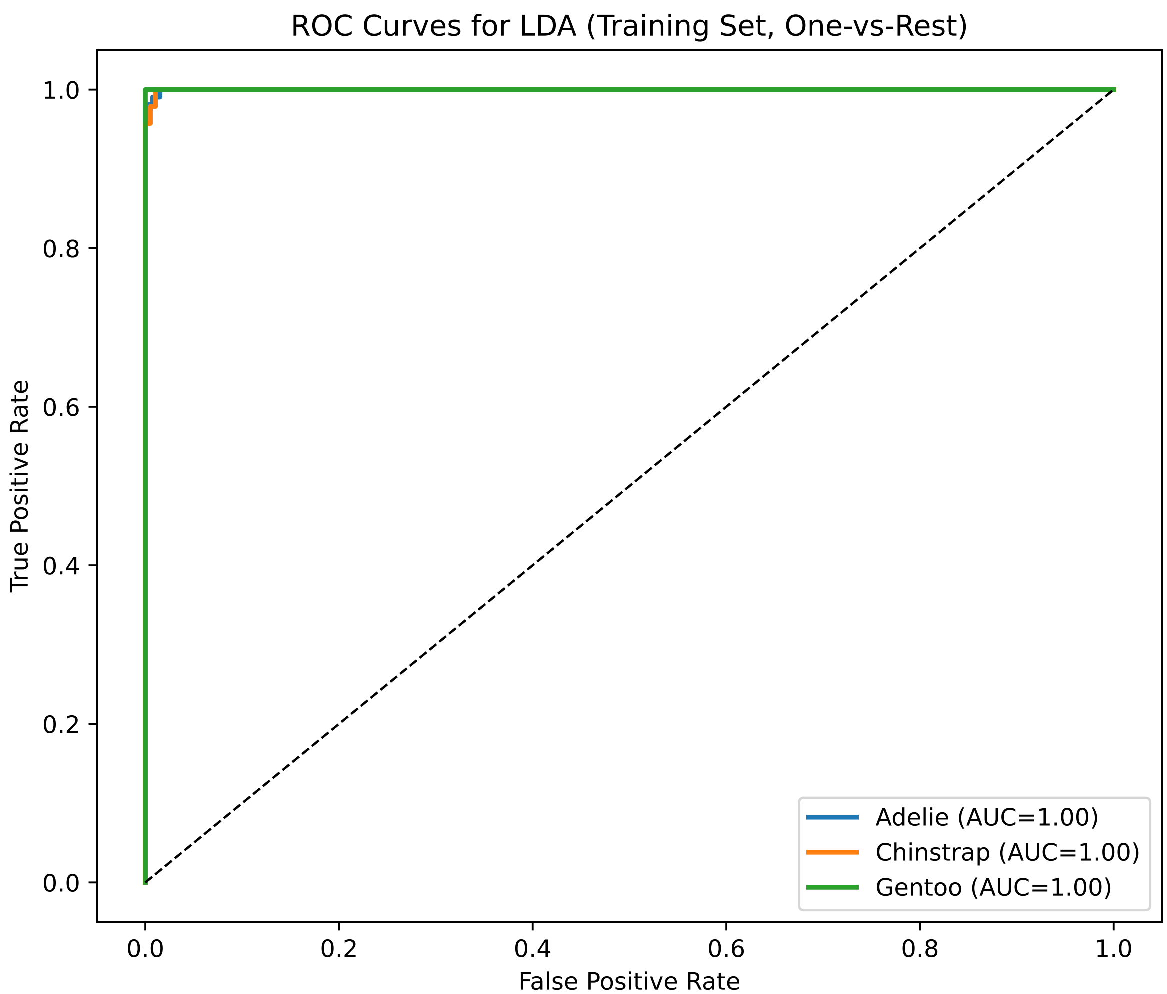
```
Test Accuracy: 0.981
Test Error: 0.019
```

(5)

```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

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 (Training Set, One-vs-Rest)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

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



Interpretation: All ROC curves lie near the top-left, showing almost perfect class separation with AUC   1.

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
```

```python
num_cols = ['AGE','WEIGHT']
cat_cols = ['PRIORFRAC','PREMENO','RATERISK']

preprocessor = ColumnTransformer([
    ('categorical', OneHotEncoder(drop='first'), cat_cols),
    ('numerical', 'passthrough', num_cols)
])

log_reg = Pipeline([
    ('prep', preprocessor),
    ('clf', 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))

feat_names = log_reg.named_steps['prep'].named_transformers_['categorical'].get_feature_names
feat_names = np.concatenate([feat_names, num_cols])

coefs = log_reg.named_steps['clf'].coef_[0]
coef_df = pd.DataFrame({
    'Feature': feat_names,
    'Coefficient': coefs,
    'OddsRatio': np.exp(coefs)
}).sort_values('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)

Women with a previous fracture (PRIORFRAC = Yes) have substantially higher odds of experiencing another fracture. For RATERISK, both "Same" and "Greater" categories show higher odds than "Less," suggesting perceived risk aligns with actual outcomes. ## (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 = smf.logit(
    formula="FRACTURE_NUM ~ AGE + PRIORFRAC + C(RATERISK, Treatment('Less'))",
    data=GLOW
).fit()

print(reduced.summary())

sample = pd.DataFrame({'AGE':[65],'PRIORFRAC':['Yes'],'RATERISK':['Same']})
pred = float(reduced.predict(sample))
print(f"Predicted probability: {pred:.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:                        21:36:40   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      (
C(RATERISK, Treatment('Less'))[T.Greater]     0.8658      0.286      3.025      0.002      (
C(RATERISK, Treatment('Less'))[T.Same]        0.5486      0.275      1.995      0.046      (
AGE                                           0.0459      0.012      3.690      0.000      (
================================================================================
Predicted probability: 0.319

/var/folders/4z/6pdkk_910xjdb2pb9453_b6c0000gn/T/ipykernel_71668/366751199.py:14: FutureWarni
  pred = float(reduced.predict(sample))
```