Problem set 4

Jae Hu

Partner: Duoshu Xu

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score, KFold
from sklearn.cross_decomposition import PLSRegression
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
```

```python
# load the data
file_path = "/Users/kevinxu/Documents/GitHub/ML-PS4/Data-College.csv"
college_df = pd.read_csv(file_path)
```

Question 1a

```python
# remove the first column if it has university names in it
if 'Unnamed: 0' in college_df.columns:
    college_df.drop(columns = ['Unnamed: 0'], inplace = True)

# convert 'Private' into numerical
```

```
college_df['Private'] = college_df['Private'].apply(lambda x: 1 if x == 'Yes'
 ↪  else 0)

# define features and target variable
X = college_df.drop(columns = ['Apps'])
y = college_df['Apps']

# split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5,
 ↪  random_state = 37)

# scale the predictor variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# convert scaled data back into DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns = X.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns = X.columns)
```

Question 1b

```
# fit the linear model
lm = LinearRegression()
lm.fit(X_train_scaled_df, y_train)

# make predictions on the test set
y_pred = lm.predict(X_test_scaled_df)

# calculate the test error
test_mse = mean_squared_error(y_test, y_pred)

# print results
print(f"Test Mean Squared Error: {test_mse:.4f}")
```

```
Test Mean Squared Error: 1222954.0383
```

Question 1c

```
# set random state
random_state = 1
```

```python
# perform PCA on the scaled training data
pca = PCA()
X_train_pca = pca.fit_transform(X_train_scaled_df)
X_test_pca = pca.transform(X_test_scaled_df)

# determine the optimal number of components
cv_errors = []
m_values = range(1, X_train_scaled_df.shape[1] + 1)

for m in m_values:
    # fit linear regression
    pcr_model = LinearRegression()
    scores = cross_val_score(pcr_model, X_train_pca[:, :m], y_train,
                             cv = 10, scoring = 'neg_mean_squared_error')
    cv_errors.append(-scores.mean())

# choose M that minimizes the cross-validation error
optimal_m = m_values[np.argmin(cv_errors)]

# determine the elbow M using
cv_errors_diff = np.diff(cv_errors)
cv_errors_diff2 = np.diff(cv_errors_diff)
elbow_m = m_values[np.argmin(cv_errors_diff2) + 1]

# train the final PCR model
pcr_final = LinearRegression()
pcr_final.fit(X_train_pca[:, :optimal_m], y_train)

# make predictions on the test set
y_pred_pcr = pcr_final.predict(X_test_pca[:, :optimal_m])

# compute test error
test_mse_pcr = mean_squared_error(y_test, y_pred_pcr)

# print results
print(f"Optimal number of principal components (M) by CV error minimization:
 ↪  {optimal_m}")
print(f"Optimal number of principal components (M) by elbow method:
 ↪  {elbow_m}")
print(f"PCR Test Mean Squared Error: {test_mse_pcr:.4f}")
```

```
Optimal number of principal components (M) by CV error minimization: 16
Optimal number of principal components (M) by elbow method: 3
PCR Test Mean Squared Error: 1487304.1279
```

Question 1d

```python
# set random state
random_state = 1

# determine the optimal number of components
cv_errors = []
m_values = range(1, X_train_scaled_df.shape[1] + 1)

for m in m_values:
    # fit PLS model with m components
    pls_model = PLSRegression(n_components = m)
    scores = cross_val_score(pls_model, X_train_scaled_df, y_train,
                             cv = 10, scoring = 'neg_mean_squared_error')
    cv_errors.append(-scores.mean())

# choose M that minimizes the cross-validation error
optimal_m = m_values[np.argmin(cv_errors)]

# determine the elbow M
cv_errors_diff = np.diff(cv_errors)
cv_errors_diff2 = np.diff(cv_errors_diff)
elbow_m = m_values[np.argmin(cv_errors_diff2) + 1]

# train the final PLS model using the optimal M
pls_final = PLSRegression(n_components = optimal_m)
pls_final.fit(X_train_scaled_df, y_train)

# make predictions on the test set
y_pred_pls = pls_final.predict(X_test_scaled_df)

# calculate test error
test_mse_pls = mean_squared_error(y_test, y_pred_pls)

# print results
print(f"Optimal number of components (M) by CV error minimization:
 ↪ {optimal_m}")
print(f"Optimal number of components (M) by elbow method: {elbow_m}")
print(f"PLS Test Mean Squared Error: {test_mse_pls:.4f}")
```

```
Optimal number of components (M) by CV error minimization: 8
Optimal number of components (M) by elbow method: 5
PLS Test Mean Squared Error: 1326021.3478
```

Question 1e The linear regression model had the lowest test error (1,222,954.04), meaning it made the most accurate predictions. This suggests that using all the available predictors worked well, and multicollinearity was not a big problem. The Partial Least Squares model had a test error of 1,326,021.35, while Principal Components Regression had the highest test error (1,487,304.13). PLS performed better than PCR because it takes the response variable into account when reducing dimensions, while PCR only looks at the predictors and might ignore useful information. Both PCR and PLS had higher errors than OLS, meaning that reducing the number of predictors may have removed important information. However, PLS was a good middle ground, simplifying the model while keeping decent accuracy. Overall, OLS was the best at predicting applications, but PLS was a reasonable alternative, especially if multicollinearity were a bigger concern. PCR did not perform as well because it ignored the response variable when selecting components.

```python
oj_file_path = "/Users/kevinxu/Documents/GitHub/ML-PS4/Data-OJ.csv"
oj_df = pd.read_csv(oj_file_path)
```

Question 3a

```python
# define variables
X = oj_df.drop(columns = ['Purchase'])
y = oj_df['Purchase']

# split the dataset into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.3, random_state = 3)
```

Question 3b

```python
# encode the target variable into numeric
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# convert categorical predictors to numeric
X_train_encoded = pd.get_dummies(X_train, drop_first = True)
```

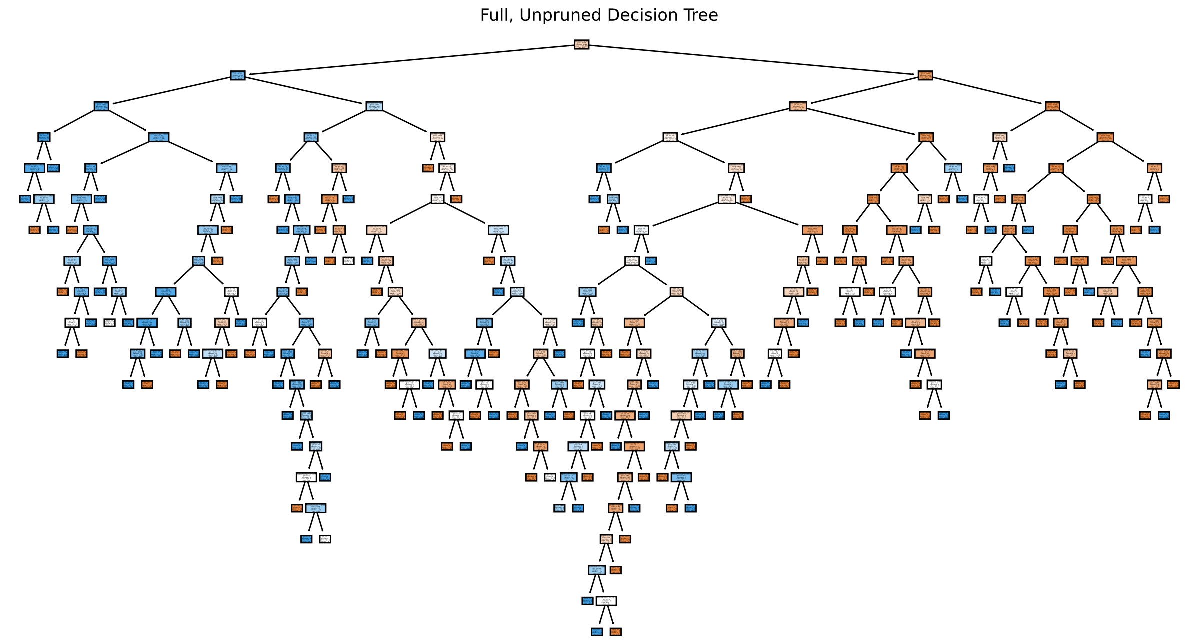
```
X_test_encoded = pd.get_dummies(X_test, drop_first = True)

# align training and test sets
X_train_encoded, X_test_encoded = X_train_encoded.align(
    X_test_encoded, join = 'left', axis = 1, fill_value= 0
)

# fit a decision tree to the training data
tree_clf = DecisionTreeClassifier(random_state = 2)
tree_clf.fit(X_train_encoded, y_train_encoded)

# make predictions on the training set
y_train_pred = tree_clf.predict(X_train_encoded)

# calculate the training error rate
train_accuracy = accuracy_score(y_train_encoded, y_train_pred)
train_error_rate = 1 - train_accuracy

# print results
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Training Error Rate: {train_error_rate:.4f}")
```

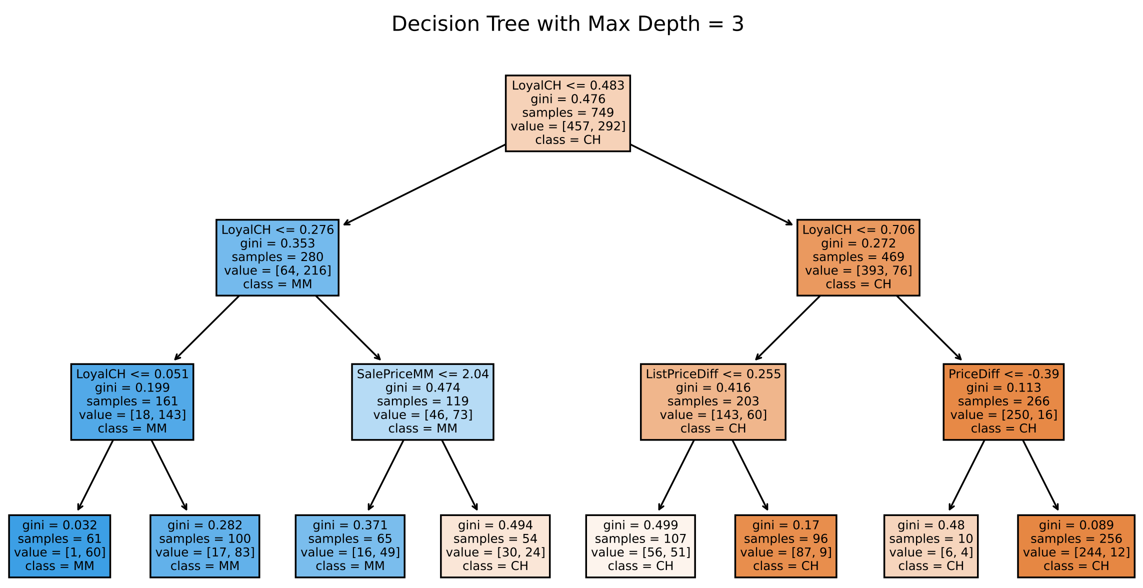
```
Training Accuracy: 0.9933
Training Error Rate: 0.0067
```

Question 3c

```
# plot the tree
plt.figure(figsize = (15, 8))
plot_tree(tree_clf, filled = True, feature_names = X_train_encoded.columns,
 ↪  class_names = label_encoder.classes_)
plt.title("Full, Unpruned Decision Tree")
plt.show()

# fit a pruned decision tree with max depth of 3
tree_clf_pruned = DecisionTreeClassifier(random_state = 2, max_depth = 3)
tree_clf_pruned.fit(X_train_encoded, y_train_encoded)

# plot the pruned tree
plt.figure(figsize = (12, 6))
plot_tree(tree_clf_pruned, filled = True, feature_names =
 ↪  X_train_encoded.columns, class_names = label_encoder.classes_)
```

```python
plt.title("Decision Tree with Max Depth = 3")
plt.show()

# count the number of terminal nodes
num_terminal_nodes = sum(tree_clf_pruned.tree_.children_left == -1)
print(f"Number of terminal nodes: {num_terminal_nodes}")
```





```
Number of terminal nodes: 8
```

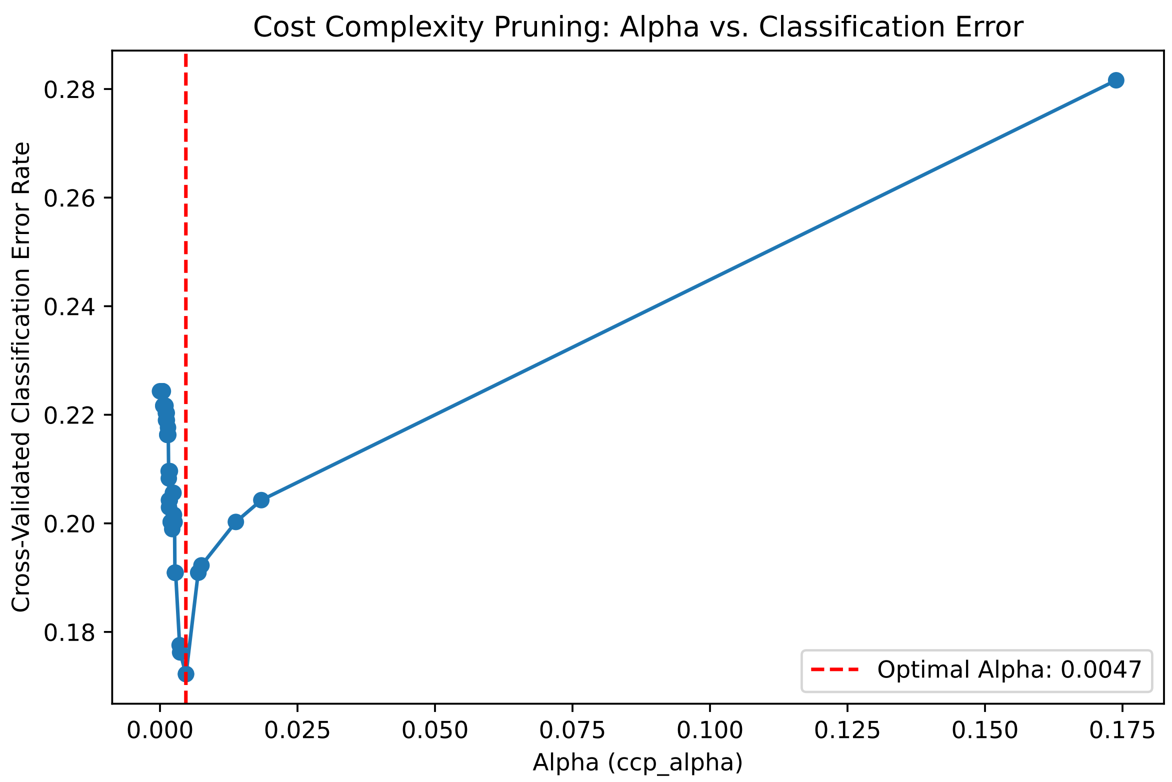
Question 4d

```python
# make predictions on test set
y_test_pred = tree_clf.predict(X_test_encoded)

# generate the confusion matrix
conf_matrix = confusion_matrix(y_test_encoded, y_test_pred)

# compute the test error rate
test_accuracy = accuracy_score(y_test_encoded, y_test_pred)
test_error_rate = 1 - test_accuracy

# print results
print("Confusion Matrix:")
print(conf_matrix)
print(f"Test Error Rate: {test_error_rate:.4f}")
```

```
Confusion Matrix:
[[160  36]
 [ 41  84]]
Test Error Rate: 0.2399
```

Question 4e

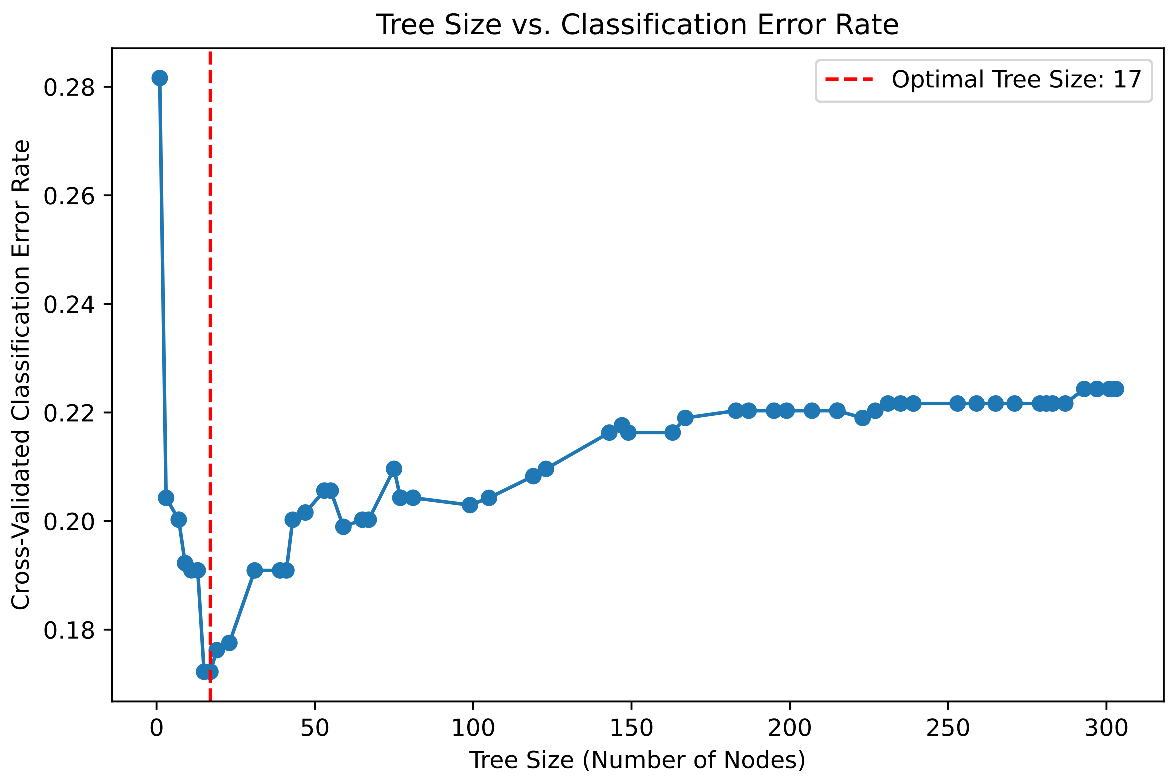
```python
# fit an initial tree
tree_clf = DecisionTreeClassifier(random_state = 2)
tree_clf.fit(X_train_encoded, y_train_encoded)

# get the cost complexity pruning path
path = tree_clf.cost_complexity_pruning_path(X_train_encoded,
 ↪  y_train_encoded)
ccp_alphas = path.ccp_alphas

# perform 5-fold
cv_errors = []

for alpha in ccp_alphas:
    pruned_tree = DecisionTreeClassifier(random_state = 2, ccp_alpha = alpha)
    scores = cross_val_score(pruned_tree, X_train_encoded, y_train_encoded,
                             cv = 5, scoring = 'accuracy')
    cv_errors.append(1 - scores.mean())

# find the optimal alpha
optimal_alpha = ccp_alphas[np.argmin(cv_errors)]

# plot the graph
plt.figure(figsize = (8, 5))
plt.plot(ccp_alphas, cv_errors, marker = 'o', linestyle = '-')
plt.xlabel("Alpha (ccp_alpha)")
plt.ylabel("Cross-Validated Classification Error Rate")
plt.title("Cost Complexity Pruning: Alpha vs. Classification Error")
plt.axvline(x = optimal_alpha, color = 'r', linestyle = '--', label =
 ↪  f'Optimal Alpha: {optimal_alpha:.4f}')
plt.legend()
plt.show()

# print results
print(f"Optimal Alpha: {optimal_alpha:.4f}")
```



```
Optimal Alpha: 0.0047
```

Question 4f

```
# find the number of terminal nodes for each alpha
tree_sizes = []

for alpha in ccp_alphas:
    pruned_tree = DecisionTreeClassifier(random_state = 2, ccp_alpha = alpha)
    pruned_tree.fit(X_train_encoded, y_train_encoded)
    tree_sizes.append(pruned_tree.tree_.node_count)

# plot the graph
plt.figure(figsize = (8, 5))
plt.plot(tree_sizes, cv_errors, marker = 'o', linestyle = '-')
plt.xlabel("Tree Size (Number of Nodes)")
plt.ylabel("Cross-Validated Classification Error Rate")
plt.title("Tree Size vs. Classification Error Rate")
plt.axvline(x = tree_sizes[np.argmin(cv_errors)], color = 'r', linestyle =
 ↪  '--',
            label = f'Optimal Tree Size: {tree_sizes[np.argmin(cv_errors)]}')
plt.legend()
plt.show()

# print results
optimal_tree_size = tree_sizes[np.argmin(cv_errors)]
print(f"Optimal Tree Size: {optimal_tree_size}")
```

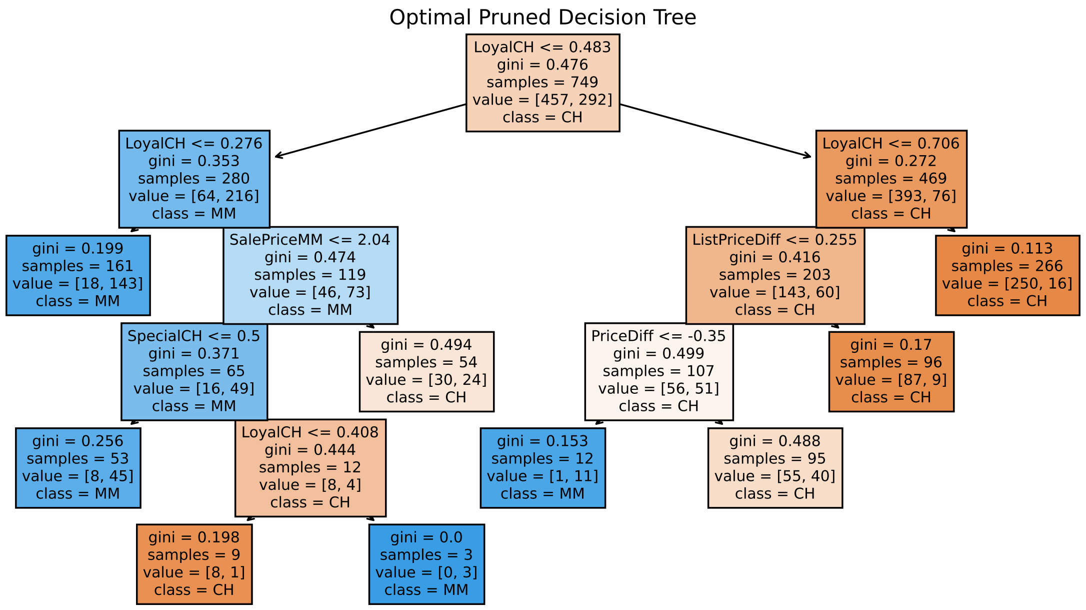


```
Optimal Tree Size: 17
```

Question 4g

```
# fit the optimal pruned tree
optimal_pruned_tree = DecisionTreeClassifier(random_state = 2, ccp_alpha =
↪   optimal_alpha)
optimal_pruned_tree.fit(X_train_encoded, y_train_encoded)

# plot the graph
plt.figure(figsize = (12, 6))
```

```
plot_tree(optimal_pruned_tree, filled = True, feature_names =
 ↪  X_train_encoded.columns,
          class_names = label_encoder.classes_)
plt.title("Optimal Pruned Decision Tree")
plt.show()
```



Question 4h

```
# find training error rate for the pruned tree
y_train_pred_pruned = optimal_pruned_tree.predict(X_train_encoded)
train_accuracy_pruned = accuracy_score(y_train_encoded, y_train_pred_pruned)
train_error_rate_pruned = 1 - train_accuracy_pruned

# print results
print(f"Unpruned Tree Training Error Rate: {train_error_rate:.4f}")
print(f"Pruned Tree Training Error Rate: {train_error_rate_pruned:.4f}")
```

```
Unpruned Tree Training Error Rate: 0.0067
Pruned Tree Training Error Rate: 0.1562
```

The unpruned tree has a training error of 0.67%, while the pruned tree's training error is 15.62%. Pruning removes unnecessary branches, making the tree simpler and better at handling new data. The unpruned tree memorizes the training data, which gives it a very low training error but likely a higher test error. The pruned tree has a higher training error but is expected to perform better on new data because it avoids overfitting.

Question 4i

```
# find test error rate for the pruned tree
y_test_pred_pruned = optimal_pruned_tree.predict(X_test_encoded)
test_accuracy_pruned = accuracy_score(y_test_encoded, y_test_pred_pruned)
test_error_rate_pruned = 1 - test_accuracy_pruned

# print results
print(f"Unpruned Tree Test Error Rate: {test_error_rate:.4f}")
print(f"Pruned Tree Test Error Rate: {test_error_rate_pruned:.4f}")
```

```
Unpruned Tree Test Error Rate: 0.2399
Pruned Tree Test Error Rate: 0.1963
```

The unpruned tree's test error is 24.3%, while the pruned tree's test error is 19.63%. Pruning removes unnecessary splits that fit noise in the training data, helping the tree generalize better to new data. This reduces test errors. The unpruned tree overfits, meaning it performs well on training data but struggles with new data, leading to a higher test error.