question2.py

```python
import numpy as np
import pandas as pd
import math
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score

# The CSV data has no headers.  These headers will be added after import.
column_headers = [
    "age",
    "class_of_worker",
    "detailed_industry_recode",
    "detailed_occupation_recode",
    "education",
    "wage_per_hour",
    "enroll_in_edu_inst_last_wk",
    "marital_stat",
    "major_industry_code",
    "major_occupation_code",
    "race",
    "hispanic_origin",
    "sex",
    "member_of_labor_union",
    "reason_for_unemployment",
    "full_or_part_time_employment_stat",
    "capital_gains",
    "capital_losses",
    "dividends_from_stocks",
    "tax_filer_stat",
    "region_of_previous_residence",
    "state_of_previous_residence",
    "detailed_household_and_family_stat",
    "detailed_household_summary_in_household",
    "unknown_value",
    "migration_code_change_in_msa",
    "migration_code_change_in_reg",
    "migration_code_move_within_reg",
    "live_in_this_house_1_year_ago",
    "migration_prev_res_in_sunbelt",
    "num_persons_worked_for_employer",
    "family_members_under_18",
    "country_of_birth_father",
    "country_of_birth_mother",
    "country_of_birth_self",
    "citizenship",
    "own_business_or_self_employed",
    "fill_inc_questionnaire_for_veterans_admin",
    "veterans_benefits",
    "weeks_worked",
    "year",
    "income_50k_plus"
]

# The names of the files to import.
datafiles = ["data/census-income.data.csv", "data/census-income.test.csv"]

# Initialize an array to hold dataframes for the data and test sets.
dataframes = []

# Load the CSV files.
for f in datafiles:
    # Read the file into a dataframe.  As noted, there are no headers.
    df = pd.read_csv(f, header=None)
    # Add the column headers.
    df.columns = column_headers

    # Data cleaning operations:

    # Handle missing values using forward fill and backward fill.
    df = df.ffill().bfill()

    # Upon analyzying the data, the two values in the "income_50k_plus" column, after
    # import, look like this: [' - 50000.', ' 50000+.'].  I am replacing these values using
    # one-hot encoding so that less than 50k income is a 0, and 50k+ is a 1.

    # Strip whitespace
    df["income_50k_plus"] = df["income_50k_plus"].str.strip()
    # Map the existing values to 0 or 1.
    df["income_50k_plus"] = df["income_50k_plus"].map({"- 50000.": 0, "50000+.": 1})

    # Upon analyzing the data, the the values in the "race" column contain leading
    # and/or trailing spaces.  These will be removed.

    # Strip whitespace
    df["race"] = df["race"].str.strip()

    # Add this dataframe to the array
    dataframes.append(df)

# Assign the data frames as training or test data.
training_data = dataframes[0]
test_data = dataframes[1]

# Define features and target variable
target_col = "income_50k_plus"
feature_cols = [col for col in training_data.columns if col != target_col]
```

```python

# Identify categorical and numerical features
categorical_cols = training_data.select_dtypes(include=["object"]).columns.tolist()
numerical_cols = training_data.select_dtypes(include=["int64", "float64"]).columns.tolist()

# Create a one-hot encoder for categorical features.
# Note: The income_50k_plus column was encoded earlier, becaue that code was carried
# over from Assignment 0.
encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)

# Encode the categorical columns from the test data.
encoded_train = encoder.fit_transform(training_data[categorical_cols])

# Encode the categorical columns from the test data.
encoded_test = encoder.transform(test_data[categorical_cols])

# Convert the encoded columns to dataframes.
encoded_train_df = pd.DataFrame(encoded_train, columns=encoder.get_feature_names_out(categorical_cols))
encoded_test_df = pd.DataFrame(encoded_test, columns=encoder.get_feature_names_out(categorical_cols))

# Assemble the prepared training and test data from the numerical features and the one-hot encoded
# categorical features.  Remove the target column from numerical features before during this process.
X_train = pd.concat([training_data[numerical_cols].drop(columns=[target_col]).reset_index(drop=True), encoded_train_df], axis=1)
X_test = pd.concat([test_data[numerical_cols].drop(columns=[target_col]).reset_index(drop=True), encoded_test_df], axis=1)

# Error checking: The code will exit if the target column is in the training or test data.
assert target_col not in X_train.columns, "Target column should not be in X_train!"
assert target_col not in X_test.columns, "Target column should not be in X_test!"

# Get the target column from the original training and test datasets.
y_train = training_data[target_col]
y_test = test_data[target_col]

# Part A: Train decision trees for depths of 2 - 10 and store accuracies.
depth_accuracies = []
for depth in range(2, 11):
    # The random_state value is arbitrary, as long as the same value is used
    # each time to allow reproducibility.
    model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    model.fit(X_train, y_train)
    train_acc = accuracy_score(y_train, model.predict(X_train))
    depth_accuracies.append({"Depth": depth, "Training Accuracy": train_acc})
```

```python

print(depth_accuracies)
# Find the maximum accuracy and its associated depth.
max_accuracy = max([entry["Training Accuracy"] for entry in depth_accuracies])
# The optimal depth is the depth associated with max_accuracy.
optimal_depth = next(item["Depth"] for item in depth_accuracies if item["Training Accuracy"] == max_accuracy)
print(f"max_accuracy = {max_accuracy}")
print(f"optimal_depth = {optimal_depth}")

# Part B: use the optimal depth found above to classify the test data.
# Re-train the model using the optimal depth found above.
model = DecisionTreeClassifier(max_depth=optimal_depth, random_state=42)
model.fit(X_train, y_train)
# Get the accuracy score for the training data (should be the same as before).
train_acc = accuracy_score(y_train, model.predict(X_train))
# Now get the accuracy score for the test data.
test_acc = accuracy_score(y_test, model.predict(X_test))
print(f"Training accuracy for depth=10: {train_acc}")
print(f"Test accuracy for depth=10: {test_acc}")


```