DCS 211: In-Class Work – ML Workflow (Iris Dataset)

Names: Benjamin Adovasio (Completed independently)

1. Block 3: For **`pd.read_csv`**, what does **`header=0`** do? Refer to API.
   header=0 tells pdf.read_csv that row 0 is the "header" and contains the column labels
2. Block 3: What does **`df.head()`** do?
   It returns the first 5 rows by default, so we can quickly inspect the beginning of DataFrame.
3. Block 5: What information does **`df.info()`** provide? Refer to API.
   It prints a summary of the DataFrame, including the index dtype, columns, non-null counts, and memory usage.
4. Block 6: According to the API for **`drop`**, can it remove rows? Columns? How do you specify what to remove?
   Drop can remove either rows or columns. You specify what to remove with labels plus axis, or directly with index= for rows and columns= for columns.
5. Block 8: What does **`df_clean['irisname'].unique()`** do?
   It returns the distinct values in the irisname column, in order of appearance.
6. Block 9: What does the method **`dropna`** do? Refer to API.
   It removed missing values.
7. Block 10: What code did you write? Justify.
   I had to make several small changes to the code, but the main change was:

```python
df_final = df_clean[df_clean['irisname'] != 'alieniris'].copy()
```

to Block 10, and replacing the default import in Block 3, with the correct file name. My justification for adding the above to Block 10, was after dropna() ther are still bogus rows labeled alieniris, so this removes them. .copy() also makes df_final a real copy. The rest of my changes are in my commit history in git, as well as the uploaded final python file.

8. Block 11: Explain what the following two lines of code are doing:
   ```
   species_names = df_final['irisname'].unique()
   species_name_to_index = { species_names[i]:i for i in range(len(species_names))}
   ```
   It collects the distinct species names still in the cleaned data and builds a dictionary mapping each species name to an integer.
9. Block 12: How did you address the CopyOnWarning message from showing up (see link in code block – specifically noting the explanation of using .loc at that link)?
   I used .loc as so:

```python
df_final.loc[:, 'irisname'] = df_final['irisname'].apply(convertSpecies)
```

10. Block 14: According to the DataFrame API, what is the difference between the following two:

a. **`A = df_final.values`**
b. `A = df_final.to_numpy()`

df_final.values returns a NumPy representation of the DataFrame while df_final.to_numpy() also returns a NumPy array, the api recommends using to_numnpy() instead of values.

11. Block 16: What does **`.shape`** on a numpy array return? What data type is it?
    It returns a tuple of array dimensions. For this cleaned dataset it is (145, 5) with data type of tuple.
12. Block 18: What does **`np.linalg.norm`** do?
    It is being used as the distance between future vectors.
13. Block 18: Inside the **`predictiveModel`** function, what do **`A[i]`** and **`A[i,0:4]`** each represent? Be specific
    A[i] is the full ith flower row and only the 4 feature columns. Y_all = A[:,4] is only the first four entries of that row.
14. Block 20: What information does **`X_all = A[:,0:4]`** give you? **`y_all = A[:,4]`**? How do they relate?
    x_all = A[:,0:4] gives al rows and only the 4 feature columns. y_all = A[:,0:4] gives gives all rows and only the label columns.
15. Block 22: What would **`np.random.permutation([1,2,3,4,5])`** do?
    It would return those same values in random order.
16. Block 23: Explain what **`X_labeled[:test_size]`** and **`X_labeled[test_size:]`** are doing.
    It takes the first part of the permuted data for the test set.
17. Block 25: Explain what **`sum(predicted_labels == actual_labels)`** does.
    It compares and produces a Boolean array, then counts how many entries are true.
18. Blocks 24 & 25: What does **`knn.fit`** do versus **`knn.predict`**? Refer to the API.
    It trains the classifier on the training data and labels, and then uses the trained model to return class labels for the test samples.
19. Block 27: What does **`knn_model.predict`** return, and why do we have to unpack it?
    It returns a NumPy array of predicted labels. We need to unpack the first element with [0], because it still returns an array of shape (1,) when you predict one example.
20. Block 27: For the four different example feature sets, what iris was predicted?
    Veriscolor was predicted for all of them.
21. Block 29: What code did you have to add to the loop, and why?

```python
best_k = 1
best_accuracy = -1

if this_cv_accuracy > best_accuracy:
    best_accuracy = this_cv_accuracy
    best_k = k
```

I had to add the above code because otherwise best_k would just end up equal to the last loop value, not the best one.

22. Blocks 30-31: For scikit-learn's kNN approach, there are three primary steps required. What are they?
    a. Create the model
    b. Train the model
    c. Predict with the model
23. Block 31: How much better did the "best-k model" do? Quantify.
    The first model (k = 84) got 7/29 correct, so 24.14%. The best model got 28/29 so 96.55%. It improved by 21 more correct predictions or 72.41 percentage points.
24. Block 32: For at least three different permutations, indicate the best k identified.
    a. Permutation 1: 2
    b. Permutation 2: 11
    c. Permutation 3: 9
25. Block 34: For the four different example feature sets, what iris was predicted?
    a. [5.8, 2.7, 4.1, 1.0] -> versicolor
    b. [4.6, 3.6, 3.0, 2.2] -> versicolor
    c. [6.7, 3.3, 5.7, 2.1] -> virginica
    d. [4.6, 3.6, 3.0, 1.2] -> versicolor
26. Block 35b: Explain how
    `df_final[ df_final['irisname'] == convertSpecies('versicolor') ]`
    grabs the data corresponding to versicolor.
    It converts the string 'veriscolor' into its numeric label (1), then it creates a boolean mask that is true only for veriscolor rows, then it uses that mask to keep only those rows.
27. Block 35b: What does **`versicolor_data.mean()[:-1]`** give back (identify the data type)? Explain how that line of code computes the mean of each different feature.
    It returns a panas series containing the mean of each numeric column except the last one. [:-1] slices off the label column so only the 4 feature means remain. Then it converts that series to a NumPy array.
28. Block 35b: What code did you add to call the predictive model and print a result?

```python
result = predictiveModel(versicolor_features_means)
print(f"Mean versicolor features {versicolor_features_means} -> {result}")
```

It would print veriscolor.

29. Block 35c: What code did you add to automate the process across all species?

```python
for name in species_names:
    species_data = df_final[df_final['irisname'] == convertSpecies(name)]
    species_features_means = np.asarray(species_data.mean()[:-1])
    result = predictiveModel(species_features_means)
    print(f"Mean {name} features {species_features_means} -> {result}")
```

30. Block 35d: For each mean-of-features, what species was predicted? Did each match the actual species for the mean-of-features?

a. Setosa mean -> setosa
b. Veriscolor mean -> veriscolor
c. Virginica mean -> virginica