Data Models Problem Set II

Jacob Miller, Jacob Shiohira, Reid Gahan

Spring 2018, RAIK370H

All code for this problem set is in Python. The approach for this problem set was to try and divide up the code by creating a function per question. The code corresponding to each question (if applicable) will be included before the question itself. Additionally, output of the functions that serve as answers to the questions will be included where applicable.

```python
def describe_data_frame(dataframe):
    print(">> Data Frame Description\n")

    rows, columns = dataframe.shape
    print("num rows (observations): " + str(rows))
    print("num cols (features): " + str(columns))
    print("\n")

    print(str(dataframe.info()))

    print("\nData Set Statistics")
    print("=====================")

    print(str(dataframe.describe()))
    print("\n")

    print(dataframe.corr())

    print("\nFirst 5 obversvations from the dataset")
    print("=======================================")
    print(dataframe[:5])

    print("\nLast 5 obversvations from the dataset")
    print("======================================")
    print(dataframe[-5:])

    return dataframe
```

1. There are 150 observations in the data set with 5 features. The data, as classified in Python, for the features are as follows: *Sepal.Length*, *Sepal.Width*, *Petal.Length*, and *Petal.Width* are non-null float64; and *Species* is non-null object. The total memory usage from the data set is 5.9 KB.

The explanatory variables are *Sepal.Length*, *Sepal.Width*, *Petal.Length*, and *Petal.Width*. The response variable is *Species*. Additionally, the first and last 5 observations are listed below.

First 5 obversvations from the dataset

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Last 5 obversvations from the dataset

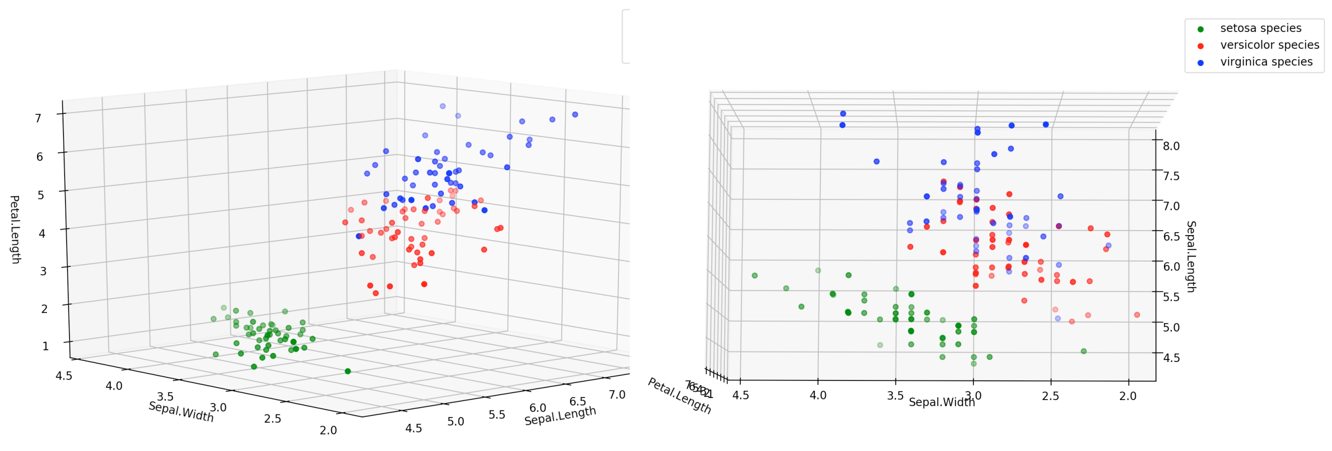
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

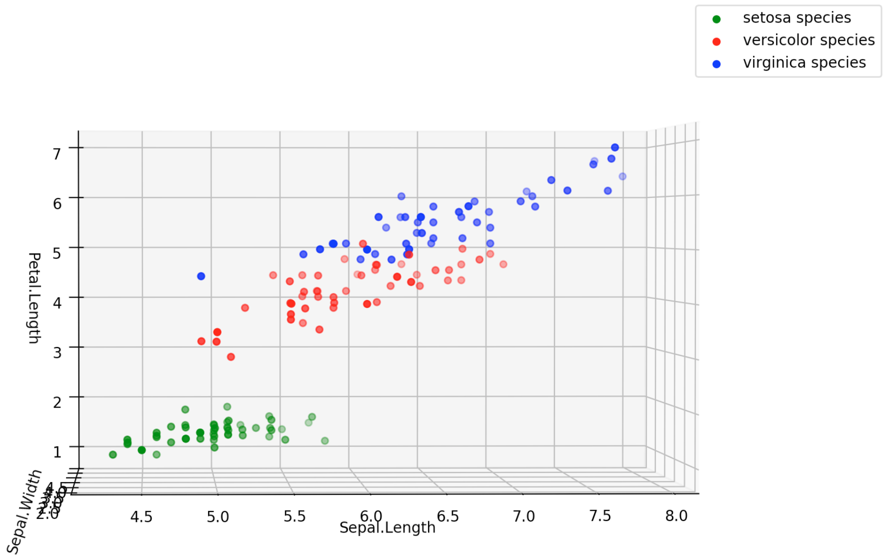
Visualize the Data

```
def visualize_data_3d(dataframe, features, weights=None, plot_flag=False):
    if not plot_flag:
        return

    if len(features) != 3:
        print(">> FAILED. Expecting exactly 3 features in the feature array.")

    fig = plot.figure()
    ax = fig.add_subplot(111, projection='3d')
    species_arr = ['setosa', 'versicolor', 'virginica']
    colors = ['green', 'red', 'blue']

    for i, f in enumerate(features):
        temp_df = dataframe.loc[dataframe['Species'] == species_arr[i]]
        temp_x = temp_df[features[0]]
        temp_y = temp_df[features[1]]
        temp_z = temp_df[features[2]]
        ax.scatter(temp_x, temp_y, temp_z, color=colors[i], label= species_arr[i] + ' sp

    ax.set_xlabel(features[0])
    ax.set_ylabel(features[1])
    ax.set_zlabel(features[2])
    ax.legend()
    plot.show()
```

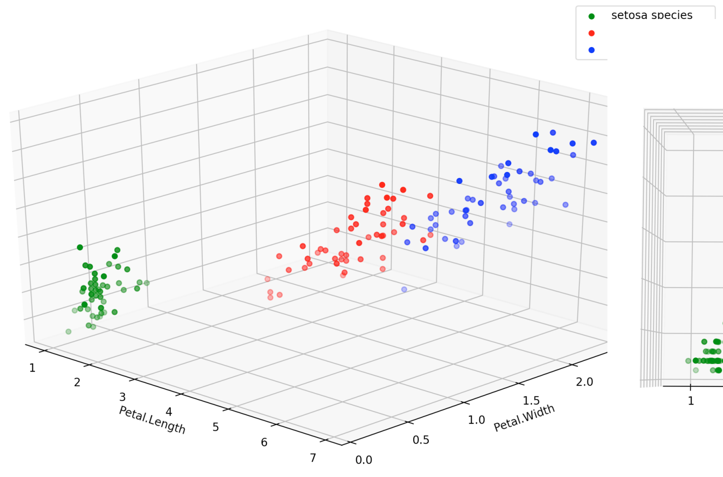
2. Included below is the resulting plot from the *matplotlib* plot output from the function above. Plotting was done for two sets of features: Sepal.Length, Sepal.Width, Petal.Length and Petal.Length, Petal.Width, Sepal.Width. The function aggregates the different types of species and then plots them on the 3D plot. You will see three plots per combination of 3 features.

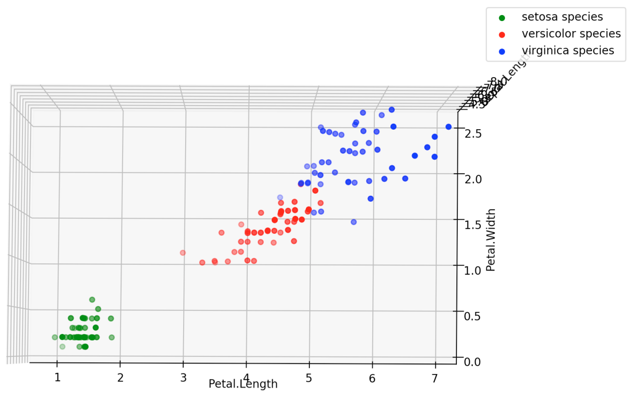
Sepal.Length, Sepal.Width, Petal.Length

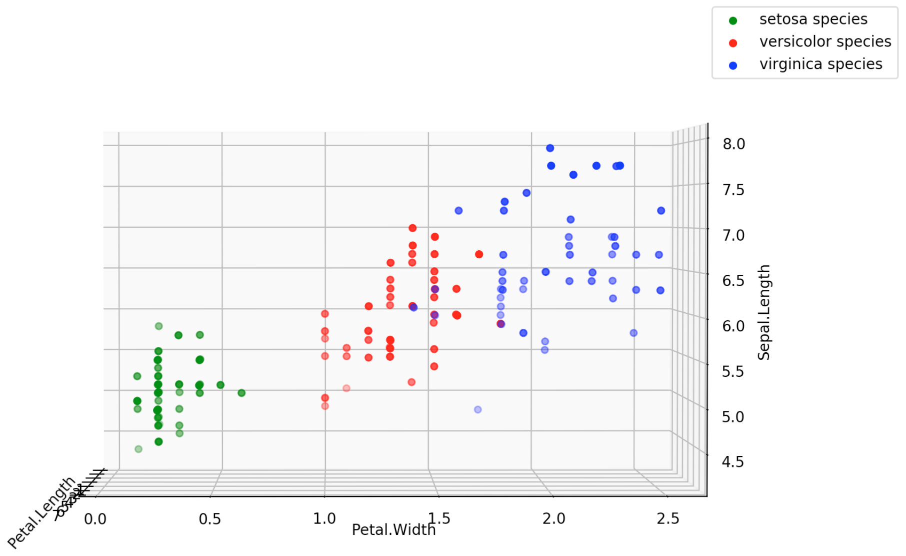




Petal.Length, Petal.Width, Sepal.Width







```
def find_and_plot_correlation(dataframe):
    sepalLW = np.correlate(dataframe['Sepal.Length'], dataframe['Sepal.Width'])
    print(">> -- Correlation between Sepal.Length and Sepal.Width: " +
          str(sepalLW))

    sepalLPetalL = np.correlate(dataframe['Sepal.Length'], dataframe['Petal.Length'])
    print(">> -- Correlation between Sepal.Length and Petal.Length: " +
          str(sepalLPetalL))

    sepalLPetalW = np.correlate(dataframe['Sepal.Length'], dataframe['Petal.Width'])
    print(">> -- Correlation between Sepal.Length and Petal.Width: " +
          str(sepalLPetalW))

    sepalWPetalW = np.correlate(dataframe['Sepal.Width'], dataframe['Petal.Width'])
    print(">> -- Correlation between Sepal.Width and Petal.Width: " +
          str(sepalWPetalW))

    sepalWPetalL = np.correlate(dataframe['Sepal.Width'], dataframe['Petal.Length'])
    print(">> -- Correlation between Sepal.Width and Petal.Length: " +
          str(sepalWPetalL))

    petalLW = np.correlate(dataframe['Petal.Length'], dataframe['Petal.Width'])
    print(">> -- Correlation between Petal.Length and Petal.Width: " +
          str(petalLW))
```

3. Below are all the correlation amongst the explanatory variables.

Explanatory Variables	Correlation
Sepal.Length and Sepal.Width	27.56
Sepal.Length and Petal.Length	39.00
Sepal.Length and Petal.Width	38.35
Sepal.Width and Petal.Width	27.10
Sepal.Width and Petal.Length	27.42
Petal.Length and Petal.Width	45.75

Randomize and Scale Dataset

```
def randomize_and_scale_dataset(dataframe):
    explanatory_variables_df = dataframe[['Sepal.Length', 'Sepal.Width',
                                          'Petal.Length', 'Petal.Width']]
    scaler = MinMaxScaler()
    scaled_data_array = scaler.fit_transform(explanatory_variables_df)
    df_norm = pd.DataFrame(scaled_data_array, index=explanatory_variables_df.index,
                           columns=explanatory_variables_df.columns)
    describe_data_frame(df_norm)
    df_norm = df_norm.sample(frac=1)
    df_norm['Species'] = dataframe['Species']
    return df_norm
```

4. Since the observations are grouped by species, we will randomize the observations for subsequent use. Each feature is scaled so that each observation lies between 0 and 1. Below is a table outlining the scaled data set statistics.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.0	150.0	150.0	150.0
mean	0.43	0.44	0.47	0.46
min	0.0	0.0	0.0	0.0
median	0.42	0.42	0.57	0.50
max	1.0	1.0	1.0	1.0

Splitting test and train data

```
def create_train_test_set(X, Y, training_set_size=130, testing_set_size=20):
    test_size = testing_set_size / (training_set_size + testing_set_size)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
                                                         random_state=42)

    print('Length of X training set: ' + str(len(X_train)))
    print('Length of Y training set: ' + str(len(Y_train)))

    print('Length of X testing set: ' + str(len(X_test)))
    print('Length of Y testing set: ' + str(len(Y_test)))

    Y_train_ohe = generate_integer_encoding(Y_train)
    Y_test_ohe = generate_integer_encoding(Y_test)

    return X_train, X_test, Y_train_ohe, Y_test_ohe
```

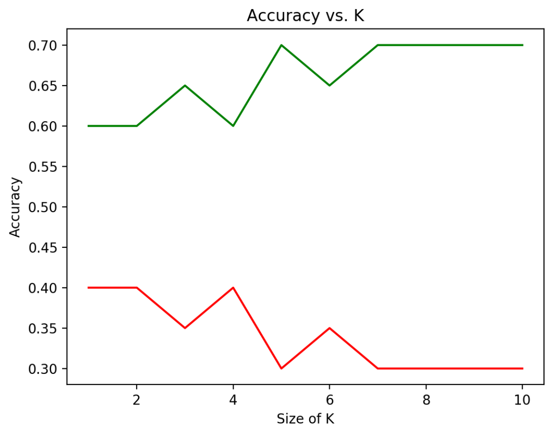
5. The above code creates a training set using 130 observations and a test set with the other 20 observations. It then creates a one hot encoding of the *Y train* and *Y test* set to be used in the ANN model. Here is the output from the previous code:

Length of X training set: 130
Length of Y training set: 130
Length of X testing set: 20
Length of Y testing set: 20

Custom KNN Implementation

```
def minkowski_distance(vector0, vector1, p):
    distance = 0
    for idx in range(len(vector0)):
        distance += (vector0[idx] - vector1[idx]) ** p
        return Math.pow(distance, 1/p)

def get_n_neighbors(vector, point, k):
    distances = []
    for idx in range(len(vector)):
        dist = minkowski_distance(vector[idx], point, 2)
        distances.append((idx, dist))
    distances.sort(key=operator.itemgetter(1))
    return distances[:k]

def get_class_from_n_neighbors(neighbors, vector):
    species = []
    for idx in range(len(neighbors)):
        species.append(vector[neighbors[idx][0]])
    common_species = max(species, key=species.count)
    return common_species

def knn_model(xtrain, ytrain, xtest, k):
    prediction = []
    for idx, element in enumerate(xtest):
        neighbors = get_n_neighbors(xtrain, element, k)
        prediction.append(get_class_from_n_neighbors(neighbors, ytrain))
    return prediction
```

6. Below is a line graph of the accuracy versus error rates as k increases. Accuracy is the green line and error is the red line.



Additionally, here are a few of the confusion matrices that correspond to certain k values from the custom KNN model. The matrices are 3×3 because there are 3 flower species. So, the entries on the diagonal represent instances where the prediction species matched the actual species.

$K = 1$

5	1	0
1	4	2
1	3	3

$K = 5$

5	1	0
2	5	1
0	4	2

$K = 10$

5	1	0
2	5	1
0	2	4

Built-In KNN Model

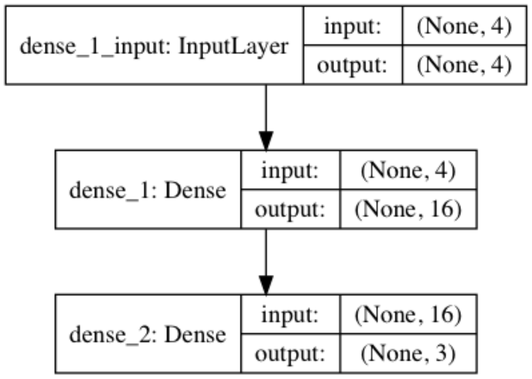
```
def use_knn(X_train, Y_train, X_test, Y_test, n_neighbors=5):
    print(">> Using KNeighborsRegressor from sklearn.neighbors to calculate " +
          str(n_neighbors) + " nearest neighbors\n")

    encoded_y_train = generate_integer_encoding(Y_train)
    encoded_y_test = generate_integer_encoding(Y_test)

    knn = KNeighborsRegressor(n_neighbors=n_neighbors)
    knn.fit(X_train, encoded_y_train)
    predictions = knn.predict(X_test)

    mse = mean_squared_error(predictions, encoded_y_test)
    print("Mean squared error: " + str(mse))
```

7. Now, we will repeat the process from Question 6, but we will use the KNeighborsRegressor function from sklearn.neighbors to calculate 5 nearest neighbors. The resulting mean squared error: 0.0147.

ANN Model

```
def ann_model(X_train, Y_train, X_test, Y_test):
    model = Sequential()

    model.add(Dense(16, input_shape=(4,), activation='sigmoid'))
    model.add(Dense(3, activation='softmax'))
    # Keras does not have an RMSE loss function so categorical_crossentropy is used
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

    model.fit(X_train, Y_train, epochs=100, batch_size=1, verbose=0)

    # Question 8: Plot model network
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    # Question 9: Predict
    prediction = model.predict(X_test)
    print("Predicted Y Values:")
    print(prediction)
    print("Actual Y Values:")
    print(Y_test)

    # Question 9: RMSE Error
    loss, acc = model.evaluate(X_test, Y_test, verbose=1)
    print(model.metrics_names)
    print("Loss: {}. Accuracy: {}".format(loss, acc))

    # Question 9: Display actual and activated values
    print("Raw Predicted Y Values:")
    print(prediction)
    activated_prediction = np.round_(prediction)
    print("Activated Predicted Y Values:")
    print(activated_prediction)

    # Question 9: Confusion table
    print(confusion_matrix(activated_prediction.argmax(axis=1), Y_test.values.argmax(axi

    return
```

8. We developed an ANN model for the iris species prediction using a library call to a neural network modeler as shown in the above code. Below is a graphic displaying the layers of the ANN.



9. We used the ANN to predict the species for each observation in the test set. Displayed below are the comparisons between actual and predicted. *NOTE*: RMSE is not a valid loss function in Keras, so categorical_crossentropy was used in its place. The loss for the ANN was 0.136578.

Setosa

Index	Predicted	Activated	Actual
0	0.000032	0	0
1	0.006592	0	0
2	0.001842	0	0
3	0.000176	0	0
4	0.995508	1	1
5	0.990137	1	1
6	0.003076	0	0
7	0.997090	1	1
8	0.007078	0	0
9	0.002747	0	0
10	0.987949	1	1
11	0.006200	0	0
12	0.003325	0	0
13	0.007235	0	0
14	0.000006	0	0
15	0.993495	1	1
16	0.000003	0	0
17	0.000015	0	0
18	0.000091	0	0
19	0.023174	0	0

Versicolor

Index	Predicted	Activated	Actual
0	0.135769	0	0
1	0.938303	1	1
2	0.826816	1	1
3	0.367354	0	1
4	0.004492	0	0
5	0.009864	0	0
6	0.846047	1	1
7	0.002910	0	0
8	0.934125	1	1
9	0.813009	1	1
10	0.012051	0	0
11	0.895738	1	1
12	0.864639	1	1
13	0.957207	1	1
14	0.067611	0	0
15	0.006505	0	0
16	0.039042	0	0
17	0.106785	0	0
18	0.249678	0	0
19	0.958328	1	1

Virginicia

Index	Predicted	Activated	Actual
0	0.864199	1	1
1	0.055105	0	0
2	0.171342	0	0
3	0.632470	0	1
4	0.00	0	0
5	0.00	0	0
6	0.150877	0	0
7	0.00	0	0
8	0.058796	0	0
9	0.184244	0	0
10	0.00	0	0
11	0.098062	0	0
12	0.132037	0	0
13	0.035558	0	0
14	0.932383	1	1
15	0.00	0	0
16	0.960954	1	1
17	0.893201	1	1
18	0.750231	1	1
19	0.018497	0	0

Here is the confusion table for each species prediction.

Actual

	5	0	0
Predicted	0	9	0
	0	1	5

The accuracy was 19/20 = 95%
The error was 1/20 = 5%

10. Using the ANN model, we experimented with the number of hidden nodes. From the results below, it seems the best accuracy comes from setting the number of nodes in the first layer to around 25. Due to limitations that come from Keras, the second layer had to be set to 3 nodes in order to maintain data shape.

Nodes in First Layer

	5	10	15	20	25	30
Loss:	0.220383	0.224407	0.207841	0.135865	0.108737	0.125834