

## M6-L2 Problem 1 (6 Points)

In this problem you will code a function to perform feature filtering using the Pearson's Correlation Coefficient method.

To start, run the following cell to load the mtcars dataset. Feature names are stored in `feature_names`, while the data is in `data`.

```
import numpy as np

feature_names =
["mpg", "cyl", "disp", "hp", "drat", "wt", "qsec", "vs", "am", "gear", "carb"]
data = np.array([[21, 6, 160, 110, 3.9, 2.62, 16.46, 0, 1, 4, 4],
[21, 6, 160, 110, 3.9, 2.875, 17.02, 0, 1, 4, 4],
[22.8, 4, 108, 93, 3.85, 2.32, 18.61, 1, 1, 4, 1],
[21.4, 6, 258, 110, 3.08, 3.215, 19.44, 1, 0, 3, 1],
[18.7, 8, 360, 175, 3.15, 3.44, 17.02, 0, 0, 3, 2],
[18.1, 6, 225, 105, 2.76, 3.46, 20.22, 1, 0, 3, 1],
[14.3, 8, 360, 245, 3.21, 3.57, 15.84, 0, 0, 3, 4],
[24.4, 4, 146.7, 62, 3.69, 3.19, 20, 1, 0, 4, 2],
[22.8, 4, 140.8, 95, 3.92, 3.15, 22.9, 1, 0, 4, 2],
[19.2, 6, 167.6, 123, 3.92, 3.44, 18.3, 1, 0, 4, 4],
[17.8, 6, 167.6, 123, 3.92, 3.44, 18.9, 1, 0, 4, 4],
[16.4, 8, 275.8, 180, 3.07, 4.07, 17.4, 0, 0, 3, 3],
[17.3, 8, 275.8, 180, 3.07, 3.73, 17.6, 0, 0, 3, 3],
[15.2, 8, 275.8, 180, 3.07, 3.78, 18, 0, 0, 3, 3],
[10.4, 8, 472, 205, 2.93, 5.25, 17.98, 0, 0, 3, 4],
[10.4, 8, 460, 215, 3, 5.424, 17.82, 0, 0, 3, 4],
[14.7, 8, 440, 230, 3.23, 5.345, 17.42, 0, 0, 3, 4],
[32.4, 4, 78.7, 66, 4.08, 2.2, 19.47, 1, 1, 4, 1],
[30.4, 4, 75.7, 52, 4.93, 1.615, 18.52, 1, 1, 4, 2],
[33.9, 4, 71.1, 65, 4.22, 1.835, 19.9, 1, 1, 4, 1],
[21.5, 4, 120.1, 97, 3.7, 2.465, 20.01, 1, 0, 3, 1],
[15.5, 8, 318, 150, 2.76, 3.52, 16.87, 0, 0, 3, 2],
[15.2, 8, 304, 150, 3.15, 3.435, 17.3, 0, 0, 3, 2],
[13.3, 8, 350, 245, 3.73, 3.84, 15.41, 0, 0, 3, 4],
[19.2, 8, 400, 175, 3.08, 3.845, 17.05, 0, 0, 3, 2],
[27.3, 4, 79, 66, 4.08, 1.935, 18.9, 1, 1, 4, 1],
[26, 4, 120.3, 91, 4.43, 2.14, 16.7, 0, 1, 5, 2],
[30.4, 4, 95.1, 113, 3.77, 1.513, 16.9, 1, 1, 5, 2],
[15.8, 8, 351, 264, 4.22, 3.17, 14.5, 0, 1, 5, 4],
[19.7, 6, 145, 175, 3.62, 2.77, 15.5, 0, 1, 5, 6],
[15, 8, 301, 335, 3.54, 3.57, 14.6, 0, 1, 5, 8],
[21.4, 4, 121, 109, 4.11, 2.78, 18.6, 1, 1, 4, 2]])
```

# Filtering

Now define a function `find_redundant_features(data, target_index, threshold)`. Inputs:

- `data`: input feature matrix
- `target_index`: index of column in `data` to treat as the target feature
- `threshold`: eliminate indices with pearson correlation coefficients greater than `threshold`

Return:

- Array of the indices of features to remove.

Procedure:

1. Compute correlation coefficients with `np.corrcoef(data.T)`, and take the absolute value.
2. Find off-diagonal entries greater than `threshold` which are not in the `target_index` row/column.
3. For each of these entries above `threshold`, determine which has a lower correlation with the target feature -- add this index to the list of indices to filter out/remove.
4. Remove possible duplicate entries in the list of indices to remove.

```
def find_redundant_features(data, target_index, threshold):  
    # YOUR CODE GOES HERE  
    redundant_features = set()  
    correlation_matrix = np.abs(np.corrcoef(data.T))  
    features = correlation_matrix.shape[0]  
  
    for i in range(features):  
        for j in range(i+1, features):  
            if i != target_index and j != target_index and  
correlation_matrix[i, j] > threshold:  
                redundant_features.add(i if  
correlation_matrix[target_index, i] < correlation_matrix[target_index,  
j] else j)  
  
    return list(redundant_features)
```

## Testing your function

The following test cases should give the following results: | target\_index | threshold | Indices to remove |  
|---|---|---|---| 0 | 0.9 | [2] | 2 | 0.7 | [0, 3, 4, 5, 6, 7, 8, 9, 10] | 10 | 0.8 | [1, 2, 5] |

Try these out in the cell below and print the indices you get.

```
# YOUR CODE GOES HERE  
test_cases = [(0, 0.9), (2, 0.7), (10, 0.8)]
```

```

for target_index, threshold in test_cases:
    removed_indices = find_redundant_features(data, target_index,
threshold)
    print(f"Target Index: {target_index}, Threshold: {threshold} =>
Removed Indices: {removed_indices}")

```

```

Target Index: 0, Threshold: 0.9 => Removed Indices: [2]
Target Index: 2, Threshold: 0.7 => Removed Indices: [0, 3, 4, 5, 6, 7,
8, 9, 10]
Target Index: 10, Threshold: 0.8 => Removed Indices: [1, 2, 5]

```

## Using your function

Run these additional cases and print the results: | target\_index | threshold | Indices to remove |  
|---|---|---|---|| 4 | 0.9 || ? || 5 | 0.8 || ? || 6 | 0.95 || ? |

```

# YOUR CODE GOES HERE
additional_test_cases = [(4, 0.90), (5, 0.80), (6, 0.95)]

for target_index, threshold in additional_test_cases:
    removed_indices = find_redundant_features(data, target_index,
threshold)
    print(f"Target Index: {target_index}, Threshold: {threshold} =>
Removed Indices: {removed_indices}")

```

```

Target Index: 4, Threshold: 0.9 => Removed Indices: [1]
Target Index: 5, Threshold: 0.8 => Removed Indices: [0, 1, 3, 7]
Target Index: 6, Threshold: 0.95 => Removed Indices: []

```