PART A: CLASSIFICATION

A1. Supervised Learning

1. Explain supervised machine learning, the notion of labelled data, and train and test datasets.

Supervised machine learning is when a model is trained using a labeled dataset; the model would then proceed to make predictions/decisions based on the input and output examples. An example to understand it better would be like how monash students (the model in this case) are given sample exams (training data) by the professors with the questions (input) and the answers to the questions (output/label) so they can learn for the final exam (unseen/test data).

In supervised machine learning, **labelled data** is raw data that has been given a "label" which is the output variable that the model tries to predict. An example of a labelled data type is categorical, for example, in email spam filters, when we are classifying emails as "Spam" or "Not spam", or numerical, like predicting house prices.

Using our dataset, a good idea is to split it into two subsets, the first subset being the **training set** which is used to train our model. The model is trained by trying to find the relationship between the input variables and its output labels in the training dataset. The second subset is the **test set**, After the model has been trained, it is important to see how effective it performs on new and unseen data. The input variables and their output labels aren't used during training, so the test set can be used to evaluate how well the model has learned the relationship between the input variables and output labels, the model's predictions are compared with the test set's actual labels.

2. Read the 'FIT1043-Essay-Features.csv' file and separate the features and the label (Hint: the label, in this case, is the 'score')

Answer Explanation:

1. Read CSV data
2. Extract the score column as our label
3. Drop the score label for our features

In [1]:
```
import pandas as pd
# Read csv file
df = pd.read_csv('FIT1043-Essay-Features.csv')

# Extract only the "score" column as our label
label_df = pd.DataFrame(df['score'])

# Drop the "score" (label) column
features_df = df.drop('score', axis=1)
```

3. Use the sklearn.model_selection.train_test_split function to split your data for training and testing.

Answer Explanation:

1. Import necessary module for the train_test_split function
2. Use the function to split data for training and testing
3. Display each dataset

In [2]:
```
# Import necessary module
from sklearn.model_selection import train_test_split

# Use the train_test_split function to split data for training and testing
features_train_set, features_test_set, label_train_set, label_test_set = train_test_split(features_df, label_df, random_state=7)
```

Training dataset for features:

In [3]:
```
features_train_set
```

Out[3]:

	essayid	chars	words	commas	apostrophes	punctuations	avg_word_length	sentences	questions	avg_word_sentence	POS	POS/total_words	prompt_words	prompt_words/total_words	synonym_words	synonym_words/total_words	unstemmed	stemmed
732	1072	2043	433	5	2	0	4.718245	18	0	24.055556	423.650980	0.978409	169	0.390300	120	0.277136	517	498
231	222	3162	624	10	6	1	5.067308	20	2	31.200000	620.993559	0.995182	266	0.426282	162	0.259615	697	674
147	306	1370	282	8	7	0	4.858156	11	4	25.636364	278.978571	0.989286	136	0.482270	73	0.258865	345	339
167	1066	3579	727	34	16	2	4.922971	32	3	22.718750	716.657382	0.985774	364	0.500688	193	0.265475	750	750
1165	1468	1027	207	4	2	0	4.961353	6	0	34.500000	203.990148	0.985460	98	0.473430	54	0.260870	273	265
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
211	1684	2095	417	4	7	1	5.023981	24	1	17.375000	414.990361	0.995181	194	0.465228	115	0.275779	503	483
502	1146	638	136	3	2	0	4.691176	5	0	27.200000	134.323308	0.987671	60	0.441176	40	0.294118	186	182
537	727	1195	257	1	2	0	4.649805	6	0	42.833333	253.648294	0.986958	110	0.428016	74	0.287938	293	281
1220	1705	672	134	9	2	0	5.014925	7	0	19.142857	133.000000	0.992537	60	0.447761	25	0.186567	169	167
175	5	1769	372	17	8	0	4.755376	24	0	15.500000	368.989160	0.991906	168	0.451613	94	0.252688	417	406

999 rows × 18 columns

Testing dataset for features:

In [4]:
```
features_test_set
```

Out[4]:

	essayid	chars	words	commas	apostrophes	punctuations	avg_word_length	sentences	questions	avg_word_sentence	POS	POS/total_words	prompt_words	prompt_words/total_words	synonym_words	synonym_words/total_words	unstemmed	stemmed
51	570	2018	392	9	4	0	5.147959	21	1	18.666667	389.989744	0.994872	181	0.461735	110	0.280612	472	457
285	1444	2029	401	17	13	0	5.059850	15	2	26.733333	398.324958	0.993329	179	0.446384	92	0.229426	473	459
530	204	995	196	9	2	0	5.076531	15	0	13.066667	193.989637	0.989743	83	0.423469	42	0.214286	257	255
712	552	2501	505	32	11	0	4.952475	17	2	29.705882	501.326680	0.992726	238	0.471287	123	0.243564	567	550
571	623	3208	615	28	11	0	5.216260	27	1	22.777778	611.324619	0.994024	296	0.481301	169	0.274797	702	676
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
462	247	1318	268	10	4	0	4.917910	12	0	22.333333	265.649123	0.991228	141	0.526119	78	0.291045	311	306
691	1153	2346	487	13	4	3	4.817248	27	4	18.037037	482.987603	0.991761	192	0.394251	81	0.166324	584	572
1271	439	2219	455	4	10	0	4.876923	14	0	32.500000	450.651917	0.990444	200	0.439560	123	0.270330	486	466
593	1060	3664	746	34	26	1	4.911528	32	3	23.312500	742.994616	0.995971	317	0.424933	142	0.190349	750	750
713	357	1575	318	8	2	0	4.952830	15	0	21.200000	312.658120	0.983202	165	0.518868	95	0.298742	353	342

333 rows × 18 columns

Training dataset for label:

In [5]:
```
label_train_set
```

Out[5]:

	score
732	2
231	4
147	4
167	5
1165	3
...	...
211	3
502	2
537	2
1220	2
175	4

999 rows × 1 columns

Testing dataset for label:

In [6]:
```
label_test_set
```

Out[6]:

	score
51	3
285	4
530	2
712	4
571	4
...	...
462	4
691	3
1271	3
593	4
713	4

333 rows × 1 columns

A2. Classification (training)

1. Explain the difference between binary and multi-class classification.

Only two outcomes/classes are possible in **binary classification**. Each instance will be assigned one of these two classes by the algorithm. The most basic classification type is binary classification, where you can see examples of it in E-mail spam filtering, Classifying if a student passes or fails a class, etc.

However, in **multi-class classification**, there are more than two outcomes/classes and each instance will be assigned one of these three or more classes by the algorithm. Multi-class classification is more complex in general than binary classification because of the higher number of outcomes/classes. A popular example of multi-class classification is image recognition, where we need to classify a hand-written number.

2. In preparation for classification, your data should be normalised/scaled.

a. Describe what you understand from this need to normalise data (this is in your Week 7 applied session).

Normalization or scaling is necessary for machine learning algorithms, especially in classification tasks, to prevent features with varying units and ranges of numerical values from negatively affecting the model's performance. This is crucial for algorithms that use distances or gradients since other features can be overpowered by features with greater numeric ranges. For algorithms like k-NN, SVM, and k-means clustering, which depend on distance measures like Euclidean distance, normalizing data is essential. Additionally, scaling generates coefficients that are easy to understand and guarantees that each characteristic contributes equally to the calculation of distances or calculations.

b. Choose and use the appropriate normalisation functions available in sklearn.preprocessing and scale the data appropriately.

Answer Explanation:

1. Import StandardScaler
2. Perform feature scaling using StandardScaler on training and testing set

In [7]:
```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
sc = StandardScaler()

# Use StandardScaler for feature scaling
scaled_features_train_set = sc.fit_transform(features_train_set)
scaled_features_test_set = sc.transform(features_test_set)
```

3. Use the Support Vector Machine algorithm to build the model.

a. Describe SVM.

The supervised ML algorithm, **Support Vector Machine (SVM)** can be applied to classification as well as regression issues. It is, however, more frequently applied in classification issues. Finding a hyperplane (line in 2D or plane in 3D space) that most effectively splits the data into different classes is the objective of SVM.

b. In SVM, there is something called the kernel. Explain what you understand from it.

The kernel in SVM is a set of mathemathical functions that takes two input data points and transforms it into a higher-dimensional space so that the points will be easily more seperated by a hyperplane. SVMs become more powerful by being able to create hyperplanes that perform better classification as the problem is translated into higher-dimensional space. It is basically used to handle non-linearly separable data.

There are many types of kernels, like the linear kernel, polynomial kernel, Radial Basis Function kernel, etc. Linear kernels are the simplest form of kernel and the RBF kernel is one of the most commonly used kernels.

c. Write the code to build a predictive SVM model using your training dataset.

Answer Explanation:

1. Import necessary modules
2. Initialize SVM model
3. Train the model using scaled features training set

In [8]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize SVM model
kernel_SVM = SVC(kernel='linear', C=1)

# Train the model using scaled features training set
kernel_SVM.fit(scaled_features_train_set, label_train_set.values.ravel())
```

Out[8]:
```
SVC
SVC(C=1, kernel='linear')
```

4. Repeat Task A2.3.c by using another classification algorithm such as Decision Tree or Random Forest algorithms instead of SVM.

Answer Explanation:

1. Choose random forest algorithm
2. Import necessary modules
3. Initialize random forest module
4. Train random forest model using scaled training set

In [9]:
```
# I will be using the Random forest algorithm here

from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest model
model_random_forest_algo = RandomForestClassifier(random_state=7)

# Train Random Forest model using scaled training set
model_random_forest_algo.fit(scaled_features_train_set, label_train_set.values.ravel())
```

Out[9]:
```
RandomForestClassifier
RandomForestClassifier(random_state=7)
```

A3. Classification (prediction)

1. Using the testing dataset you created in Task A1.3 above, conduct the prediction for the 'score' (label) using the two models built by SVM and your other classification algorithm in A.2.4.

Answer Explanation:

1. Make predictions on SVM outcomes
2. Evaluate SVM accuracy
3. Make predicitons on random forest outcomes
4. Evaluate Random Forest accuracy
5. Print out SVM and Random forest accuracies

In [10]:
```
# Make predictions on SVM outcomes
pred_outcomes = kernel_SVM.predict(scaled_features_test_set)

# Evaluate SVM accuracy
accuracy_SVM = accuracy_score(label_test_set, pred_outcomes)

# Make predictions on Random forest outcomes
pred_outcomes_forest = model_random_forest_algo.predict(scaled_features_test_set)

# Evaluate Random Forest accuracy
accuracy_forest = accuracy_score(label_test_set, pred_outcomes_forest)

print(f'Accuracy_SVM: {accuracy_SVM}')
print(f'Random Forest Accuracy: {accuracy_forest}')
```

```
Accuracy_SVM: 0.6546546546546547
Random Forest Accuracy: 0.6816816816816816
```

2. Display the confusion matrices for both models (it should look like a 6x6 matrix)

Answer Explanation:

1. Import confision_matrix function from sklearn.metrics module
2. Create confusion matrix for SVM model using confision_matrix function
3. Create confusion matrix for random forest model using confision_matrix function

In [11]:
```
from sklearn.metrics import confusion_matrix

# Create confusion matrix for SVM model
confusion_matrix_SVM = confusion_matrix(label_test_set, pred_outcomes)

# Create confusion matrix for Random Forest model
confusion_matrix_RandomForest = confusion_matrix(label_test_set, pred_outcomes_forest)

print(f"SVM Confusion Matrix: \n{confusion_matrix_SVM}")
print(f"Random Forest Confusion Matrix: \n{confusion_matrix_RandomForest}")
```

```
SVM Confusion Matrix:
[[  0   7   0   0   0   0]
 [  1  13  13   0   0   0]
 [  0   3 102  41   0   0]
 [  0   0  37 102   1   1]
 [  0   0   0  10   1   0]
 [  0   0   0   1   0   0]]
Random Forest Confusion Matrix:
[[  3   3   1   0   0   0]
 [  0  10  17   0   0   0]
 [  0   4 106  36   0   0]
 [  0   0  33 105   3   0]
 [  0   0   0   8   3   0]
 [  0   0   0   1   0   0]]
```

3. Compare the performance of SVM and your other classifier and provide your justification of which one performed better.

When reading a confusion matrix, we know that the diagonal values in the matrix show how many correct predictions for each class, where a "correct prediction" means that the predicted label matches the true label. In this case, the first diagonal value of the matrix tells us how many essays were accurately classified with a score of 1, the second diagonal value tells us how many were accurately classified with a score of 2, etc.

When comparing the two matrices, we can observe as below:

First diagonal value: Random forest accurately classifies 3 instances while SVM didn't correctly classify any. **So, Random forest is slightly superior.**

Second diagonal value: Random forest accurately classifies 10 instances while SVM correctly classifies 13 instances. **So, SVM is slightly superior.**

Third diagonal value: Random forest accurately classifies 106 instances while SVM correctly classifies 102 instances. **So, Random forest is slightly superior.**

Fourth diagonal value: Random forest accurately classifies 105 instances while SVM correctly classifies 102 instances. **So, Random forest is slightly superior.**

Fifth diagonal value: Random forest accurately classifies 3 instances while SVM only correctly classified 1. **So, Random forest is slightly superior.**

Sixth diagonal value: Both Random forest and SVM didn't correctly classify any. **So, we can consider them equal.**

From our observations above, we can conclude that **Random Forest outperforms SVM**, as Random forest diagonal values 1, 3, 4, and 5 has higher counts than SVM's. This observation is also backed up by the fact that the Random Forest model has a slightly higher accuracy (68.2% vs 65.5%) from what we calculated in A3.1.

A4. Independent evaluation

1. Read the 'FIT1043-Essay-Features-Submission.csv' file and use the best model you built earlier to predict the 'score' for the essays in this file.

Answer Explanation:

1. Read Essay CSV data
2. Perform feature scaling on data
3. As the random forest model was the best model that was built earlier, we

use it to Make score predictions on the data 4. Then we add a new column named 'score' with the score predictions

In [12]:
```
# Read the essay data
essay_sub_df = pd.read_csv('FIT1043-Essay-Features-Submission.csv')

# Feature Scaling
submission_features_scaled = sc.transform(essay_sub_df)

# Make score predictions on the submission data
new_scores = model_random_forest_algo.predict(submission_features_scaled)

# Add the predictions to the submission DataFrame in a new column 'Score'
essay_sub_df['Score'] = new_scores

essay_sub_df
```

Out[12]:

	essayid	chars	words	commas	apostrophes	punctuations	avg_word_length	sentences	questions	avg_word_sentence	POS	POS/total_words	prompt_words	prompt_words/total_words	synonym_words	synonym_words/total_words	unstemmed	stemmed	Sco
0	1623	4332	900	28	13	0	4.813333	39	1	23.076923	893.988852	0.993321	392	0.435556	196	0.217778	750	750	
1	1143	1465	280	11	3	1	5.232143	14	3	20.000000	278.321343	0.994005	131	0.467857	51	0.182143	339	316	
2	660	1696	325	17	2	0	5.218462	19	1	17.105263	321.316770	0.988667	178	0.547692	92	0.283077	352	337	
3	1596	2640	555	20	17	0	4.756757	28	0	19.821429	551.989150	0.994575	228	0.410811	107	0.192793	632	605	
4	846	2844	596	33	4	1	4.771812	24	9	24.833333	593.658810	0.996072	279	0.468121	138	0.231544	626	607	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
194	1226	1208	242	8	8	0	4.991736	13	0	18.615385	237.327684	0.980693	135	0.557851	58	0.239669	244	242	
195	862	4039	817	24	11	1	4.943696	47	2	17.382979	812.656033	0.994683	386	0.472460	210	0.257038	750	750	
196	1562	2448	468	22	7	0	5.230769	22	0	21.272727	465.656652	0.994993	224	0.478632	101	0.215812	540	526	
197	1336	1081	214	14	5	0	5.051402	11	0	19.454545	212.990566	0.995283	114	0.532710	63	0.294393	259	256	
198	1171	2094	433	11	12	0	4.836028	19	0	22.789474	426.651090	0.985337	221	0.510393	121	0.279446	501	478	

199 rows × 19 columns

2a. Output of your predictions should be submitted in a CSV file format

Answer Explanation:

1. Extract only the 'essayid' and 'score' columnsExtract only the 'essayid' and 'score' columns
2. Use the .to_csv function to save dataframe into a csv file

In [13]:
```
# Extract only the 'essayid' and 'score' columns
df_output = essay_sub_df[['essayid', 'Score']]

# Save this DataFrame to a CSV file
df_output.to_csv('Prediction_Essay_Features_Submission.csv', index=False)
print(f"length of df: {len(df_output)}")
```

```
length of df: 199
```

Part B: Selection of Dataset, Clustering and Video Preparation

B1. Selection of a Dataset with missing data, Clustering

1. Select a suitable dataset that contains some missing data and at least two numerical features.

Housing in California Dataset: https://drive.google.com/file/d/1lsn2uv6DwiEr2Th4Hn6-TwhAlqO9hUW8/view?usp=drive_link

Objective: Cluster houses based on economic indicators median_income and median_house_value to identify and group areas with similar economic status.

2. Perform wrangling on the dataset to handle the missing data and explain your procedure

Answer Explanation:

1. Import pandas module
2. Load california_housing data
3. Check for missing data in each column
4. Display

In [14]:
```
import pandas as pd

# Load data
california_df = pd.read_csv("california_housing.csv")

# Check for missing values
print(california_df.isnull().sum())
print(len(california_df))
```

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           8
ocean_proximity         0
median_house_value     10
dtype: int64
20640
```

As we are doing economic clustering on the dataset, doing data cleaning on median_income and median_house_value columns are sufficient for the predictions. There are missing data in the 'total_bedrooms' column, which isn't relevant to the median_income and median_house_value columns, so we can actually ignore it or drop it.

Answer Explanation:

1. Import KMeans and StandardScaler
2. Drop total_bedrooms column
3. Remove rows where median_income is missing
4. Remove rows where median_house_value is missing
5. Display data to make sure there are no missing data
6. Select Features for Economic Clustering
7. Normalize Data

In [15]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Drop total_bedrooms column
california_df = california_df.drop('total_bedrooms', axis=1)

# Remove rows where median_income is missing
california_df = california_df.dropna(subset=['median_income'])

# Remove rows where median_house_value is missing
california_df = california_df.dropna(subset=['median_house_value'])
print(california_df.isnull().sum())
print(len(california_df))

# Select Features for Economic Clustering
economic_features = ['median_income', 'median_house_value']
economic_df = california_df[economic_features]

# Normalize Data
sc = StandardScaler()
scaled_economic_df = sc.fit_transform(economic_df)
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
population            0
households            0
median_income         0
ocean_proximity       0
median_house_value    0
dtype: int64
20622
```

3. Perform k-means clustering, choosing two numerical features in your dataset, and apply k-means clustering to your data to create k clusters in Python (k>=2)

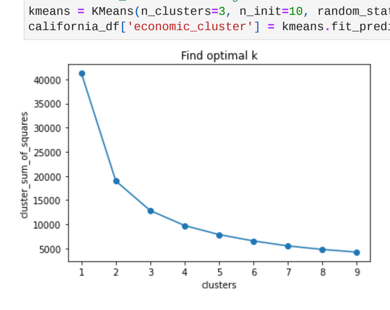
1. Import pyplot
2. Find the most optimal number of clusters using the elbow method
3. Plot the cluster_sum_of_squares list
4. Use the most optimal number of clusters for k means clustering
5. Create a new column 'economic_cluster' so that every row is labelled with a "0", "1", or "2" that represents its economic cluster

In [16]:
```
import matplotlib.pyplot as plt

# Choosing value for n_clusters with elbow method
cluster_sum_of_squares = []
range_vals = range(1, 10)
for i in range_vals:
    kmeans = KMeans(n_clusters=i, random_state=7, n_init=10)
    kmeans.fit(scaled_economic_df)
    cluster_sum_of_squares.append(kmeans.inertia_)

plt.plot(range_vals, cluster_sum_of_squares, marker='o')
plt.title('Find optimal k')
plt.xlabel('clusters')
plt.ylabel('cluster_sum_of_squares')
plt.show()

# K-Means Clustering
# We select n_clusters=3 using the elbow method above
kmeans = KMeans(n_clusters=3, n_init=10, random_state=7)
california_df['economic_cluster'] = kmeans.fit_predict(scaled_economic_df)
```



Answer Explanation:

1. Summarize Clusters by grouping california_df by the economic_cluster column values with the median_income and median_house_value features and using its mean.

In [17]:
```
# Summarize Clusters
summary = california_df.groupby('economic_cluster')[economic_features].mean()
print(summary)
```

```
                  median_income  median_house_value
economic_cluster
0                      4.403672       244329.642724
1                      7.249343       428052.396794
2                      2.609578       122237.767460
```

4. Visualise the data as well as the results of the k-means clustering, and describe your findings about the identified clusters.

Answer Explanation:

1. Import necessary modules
2. From the summary we can see that the economic_cluster "1" is has the highest economic feature values so we can say that it's the upper economic cluster, etc. So we map all the cluster numbers to names in the cluster_names dictionary.
3. Create custom_pallete with red, blue, and green.
4. Create a scatterplot where the x-axis is the median_income, and y-axis is the median_house_value and the hue is the ecnomic_group column

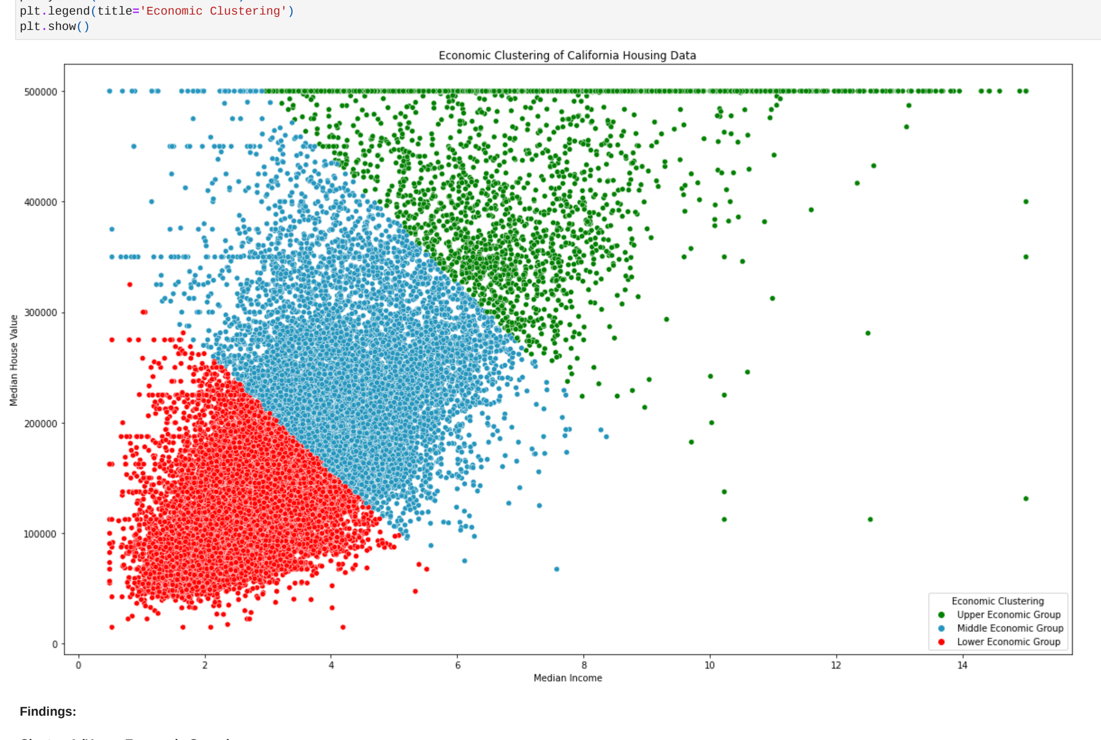
In [18]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Mapping clusters to economic group names based on the cluster_summary above
cluster_names = {
    0: 'Middle Economic Group',
    1: 'Upper Economic Group',
    2: 'Lower Economic Group'
}

custom_palette = ["#008000", "#2596be", "#ff0000"] #Red, Blue, and Green

california_df['economic_group'] = california_df['economic_cluster'].map(cluster_names)

# Visualizing the clusters with new labels
plt.figure(figsize=(20, 12))
sns.scatterplot(data=california_df, x='median_income', y='median_house_value', hue='economic_group', palette=custom_palette)
plt.title('Economic Clustering of California Housing Data')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.legend(title='Economic Clustering')
plt.show()
```



Findings:

Cluster_1 (Upper Economic Group):

Median Income is around 7.24, This cluster has the highest median income among the three clusters. Median House Value is about 428,052, Corresponding to the higher income, the median house value is the highest in this cluster, indicating these are upper-middle-class areas.

Cluster_0 (Middle Economic Group):

Median Income is around 4.40, The median income for this cluster is more than the lower economic cluster but less than the upper economic cluster, making it be the middle-class areas. Median House Value is about 244,329, The median house value is also moderate.

Cluster_2 (Lower Economic Group):

Median Income is around 2.60, This cluster has the lowest median income, so it represents lower-income areas. Median House Value is about 122,237, The low median house value is consistent with the lower income levels, suggesting that it is more affordable but probably less preferable areas.

The clusters **shows 3 economic statuses within California**, that is lower, middle, to upper-income areas. These findings can be **useful for real estate agents/professionals and investors**, where they can use these findings to make informed investments, like how the upper economic areas might offer more stable and elite investment opportunities while investors could look for much cheaper real estate in the lower economic cluster and start investing/building on a long-term housing project there. It can also be **used by the Department of Social Services**, so that the distribution of social services to the people can be directed to where they are most needed.

In [ ]: