FinalProject

December 6, 2024

1 Keiran Berry

2 Data Science Final Project (Undergraduate)

Spotify Song Genre Classification

2.0.1 Data Loading and Cleaning

[3]:
```
import kagglehub
import pandas as pd
import numpy as np

# loading in the dataset from kagglehub
path = kagglehub.dataset_download("maharshipandya/-spotify-tracks-dataset")
data = pd.read_csv(path + "/dataset.csv")

# splitting up the features and targets
X = data.drop(columns=["track_genre"])
y = data["track_genre"]

# need to map genres to integers for models
mapping = {genre: idx for idx, genre in enumerate(y.unique())}
y = y.map(mapping)

# printing the mapping for our use later
print("Genre Mapping:")
print(mapping)

print("Shape of uncleaned data: ", X.shape)
print("Shape of uncleaned targets: ", y.shape)

cleanX = X.dropna()  # drop observations with missing features
cleanY = y.loc[cleanX.index]  # drop target values that had their observations␣
↪dropped

# making sure the boolean column will work even if any model we select does not␣
↪automatically handle it
```

```python
X['explicit'] = X['explicit'].astype(int)

# dropping columns we don't want to train on (feature selection)
X.drop(columns=["Unnamed: 0", "track_id", "artists", "album_name",␣
↪"track_name"], inplace=True) # inplace -> put it back in X

numXMissing = X.isnull().sum().sum() # same method as in project 4 to check for␣
↪missing values
numYMissing = y.isnull().sum().sum()
print("\n\nNumber of missing values in features:", numXMissing)
print("Number of missing values in target:", numYMissing)

print("Missing values:") # print list of where values are missing (if they are␣
↪missing)
print(X.isnull().sum())

print("\n\nShape of cleaned data: ", X.shape)
print("Shape of cleaned targets: ", y.shape)
uniqueLabels = np.unique(y)
print("\nTotal classes: ", len(uniqueLabels))

print("\nObservations per class: ", y.value_counts())
```

```
Genre Mapping:
{'acoustic': 0, 'afrobeat': 1, 'alt-rock': 2, 'alternative': 3, 'ambient': 4,
'anime': 5, 'black-metal': 6, 'bluegrass': 7, 'blues': 8, 'brazil': 9,
'breakbeat': 10, 'british': 11, 'cantopop': 12, 'chicago-house': 13, 'children':
14, 'chill': 15, 'classical': 16, 'club': 17, 'comedy': 18, 'country': 19,
'dance': 20, 'dancehall': 21, 'death-metal': 22, 'deep-house': 23, 'detroit-
techno': 24, 'disco': 25, 'disney': 26, 'drum-and-bass': 27, 'dub': 28,
'dubstep': 29, 'edm': 30, 'electro': 31, 'electronic': 32, 'emo': 33, 'folk':
34, 'forro': 35, 'french': 36, 'funk': 37, 'garage': 38, 'german': 39, 'gospel':
40, 'goth': 41, 'grindcore': 42, 'groove': 43, 'grunge': 44, 'guitar': 45,
'happy': 46, 'hard-rock': 47, 'hardcore': 48, 'hardstyle': 49, 'heavy-metal':
50, 'hip-hop': 51, 'honky-tonk': 52, 'house': 53, 'idm': 54, 'indian': 55,
'indie-pop': 56, 'indie': 57, 'industrial': 58, 'iranian': 59, 'j-dance': 60,
'j-idol': 61, 'j-pop': 62, 'j-rock': 63, 'jazz': 64, 'k-pop': 65, 'kids': 66,
'latin': 67, 'latino': 68, 'malay': 69, 'mandopop': 70, 'metal': 71,
'metalcore': 72, 'minimal-techno': 73, 'mpb': 74, 'new-age': 75, 'opera': 76,
'pagode': 77, 'party': 78, 'piano': 79, 'pop-film': 80, 'pop': 81, 'power-pop':
82, 'progressive-house': 83, 'psych-rock': 84, 'punk-rock': 85, 'punk': 86,
'r-n-b': 87, 'reggae': 88, 'reggaeton': 89, 'rock-n-roll': 90, 'rock': 91,
'rockabilly': 92, 'romance': 93, 'sad': 94, 'salsa': 95, 'samba': 96,
'sertanejo': 97, 'show-tunes': 98, 'singer-songwriter': 99, 'ska': 100, 'sleep':
101, 'songwriter': 102, 'soul': 103, 'spanish': 104, 'study': 105, 'swedish':
106, 'synth-pop': 107, 'tango': 108, 'techno': 109, 'trance': 110, 'trip-hop':
111, 'turkish': 112, 'world-music': 113}
```

```
Shape of uncleaned data:  (114000, 20)
Shape of uncleaned targets:  (114000,)


Number of missing values in features: 0
Number of missing values in target: 0
Missing values:
popularity          0
duration_ms         0
explicit            0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
time_signature      0
dtype: int64


Shape of cleaned data:  (114000, 15)
Shape of cleaned targets:  (114000,)

Total classes:  114

Observations per class:  track_genre
0      1000
85     1000
83     1000
82     1000
81     1000
       …
34     1000
33     1000
32     1000
31     1000
113    1000
Name: count, Length: 114, dtype: int64
```

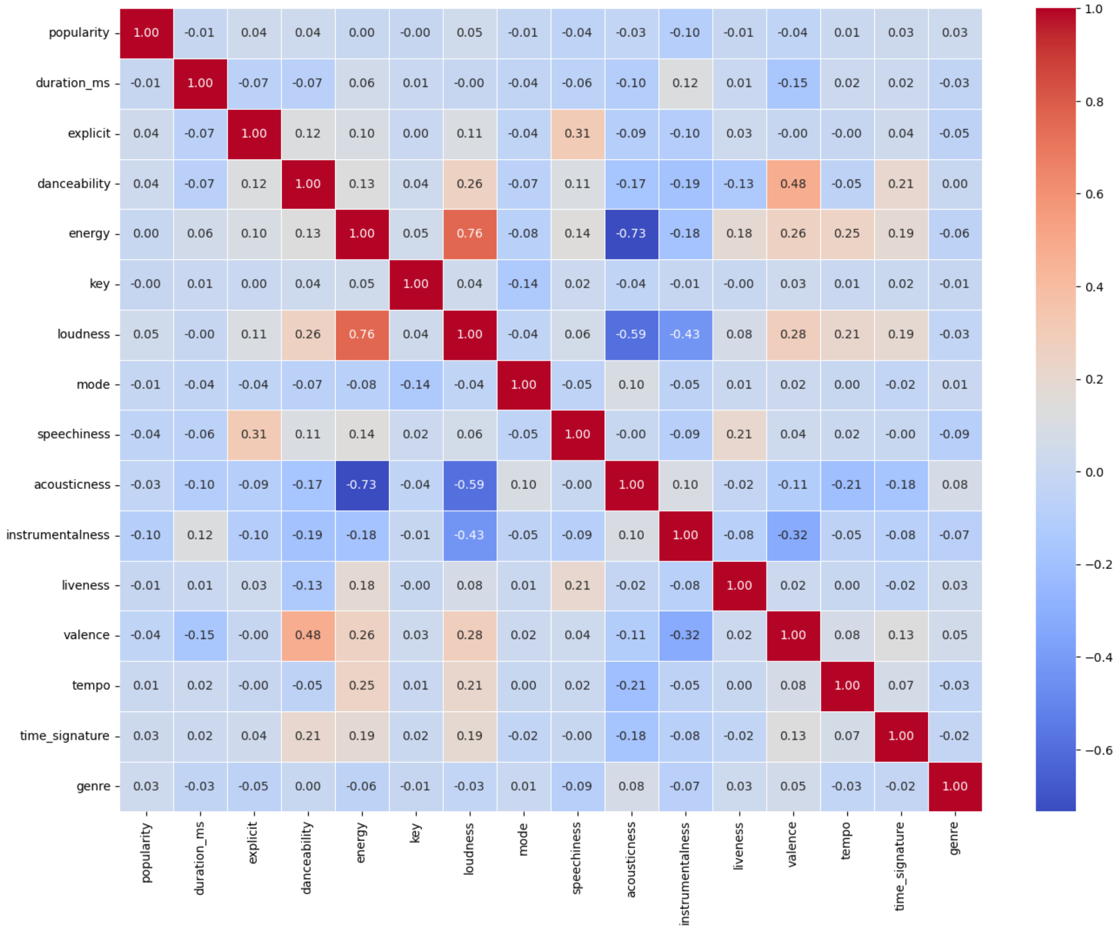
2.0.2 Data Loading and Cleaning Discussion:

- There are 114,000 observations in the dataset. There are a total of 114 targets, each representing a different genre which the given song can fall into.

- Each of these genres was mapped from a string to an integer, so that classifiers can work with them. I have printed the mapping above, for reference as we look at the data.
- In my efforts to process the data, I found that there were a total of three missing values in the dataset. Each one of these was in a column which we dropped during feature selection anyway, so we were able to keep all of the data by doing the cleaning after dropping the columns which we don't care about.
- As we can see in the printed observations per class, there are 1000 observations per genre. This lines up with our 114,000 observation count. The classes are very balanced here, more so than any data set I have used in the past! Each class has the same exact number of observations.

2.0.3 Other Data Analysis

[5]:
```
from mlxtend.plotting import heatmap
import matplotlib.pyplot as plt
import seaborn as sns

# making a new variable so we don't mess with x
allData = X
allData["genre"] = y

plt.figure(figsize=(16, 12))

correlationMatrix = allData.corr()

# using seaborn to specify more things so that we can actually read the matrix
sns.heatmap(correlationMatrix, annot=True, fmt=".2f", cmap='coolwarm',␣
↪linewidths=0.5, xticklabels=allData.columns, yticklabels=allData.columns)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=10)

plt.show()
```



2.0.4 Feature Correlation Matrix Discussion

- I am surprised at how little correlation all of these features have with the genre of the song. I hope that the models are able to predict well, as this looks like a tough dataset.

2.0.5 Splitting Data and Training Models

[7]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# want 20% testing data, want to make sure that class distribution is even so␣
↪we will use stratify = y
trainX, testX, trainY, testY = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 50, stratify = y)
```

```python
scaler = StandardScaler()

# scaling the training and test sets
scaledTrainX = scaler.fit_transform(trainX)
scaledTestX = scaler.transform(testX)

logisticRegression = LogisticRegression(max_iter = 1000)
logisticRegression.fit(scaledTrainX, trainY)

randomForest = RandomForestClassifier(criterion = 'gini', n_estimators = 100,
 ↪max_depth = 17, n_jobs = 7)
randomForest.fit(scaledTrainX, trainY)


# doing predictions
logRegPredictions = logisticRegression.predict(scaledTestX)

randomForestPredictions = randomForest.predict(scaledTestX)

# evaluating performance
print("Logistic Regression Performance:")
print("Accuracy:", accuracy_score(testY, logRegPredictions))
print("\nClassification Report:")
print(classification_report(testY, logRegPredictions))

print("Random Forest Classifier Performance:")
print("Accuracy:", accuracy_score(testY, randomForestPredictions))
print("\nClassification Report:")
print(classification_report(testY, randomForestPredictions))
```

```
Logistic Regression Performance:
Accuracy: 0.501140350877193

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.74      0.70       200
           1       0.62      0.75      0.68       200
           2       0.38      0.37      0.37       200
           3       0.42      0.38      0.40       200
           4       0.66      0.77      0.71       200
           5       0.38      0.33      0.35       200
           6       0.76      0.81      0.78       200
           7       0.57      0.69      0.63       200
           8       0.33      0.20      0.25       200
           9       0.46      0.41      0.44       200
          10       0.68      0.62      0.65       200
          11       0.33      0.20      0.25       200
```

12	0.47	0.57	0.52	200
13	0.69	0.79	0.74	200
14	0.56	0.65	0.60	200
15	0.52	0.57	0.55	200
16	0.80	0.81	0.81	200
17	0.54	0.33	0.41	200
18	0.90	0.85	0.87	200
19	0.49	0.64	0.55	200
20	0.43	0.47	0.45	200
21	0.49	0.51	0.50	200
22	0.72	0.85	0.78	200
23	0.41	0.48	0.44	200
24	0.67	0.82	0.74	200
25	0.40	0.48	0.44	200
26	0.71	0.67	0.69	200
27	0.57	0.64	0.60	200
28	0.31	0.14	0.19	200
29	0.37	0.45	0.40	200
30	0.31	0.40	0.35	200
31	0.16	0.08	0.11	200
32	0.38	0.20	0.27	200
33	0.30	0.34	0.31	200
34	0.40	0.38	0.39	200
35	0.51	0.79	0.62	200
36	0.32	0.21	0.25	200
37	0.35	0.34	0.34	200
38	0.40	0.32	0.35	200
39	0.32	0.14	0.20	200
40	0.48	0.70	0.57	200
41	0.25	0.15	0.19	200
42	0.75	0.91	0.82	200
43	0.37	0.17	0.23	200
44	0.33	0.47	0.39	200
45	0.63	0.66	0.65	200
46	0.48	0.53	0.50	200
47	0.38	0.29	0.33	200
48	0.32	0.23	0.27	200
49	0.42	0.45	0.43	200
50	0.39	0.56	0.46	200
51	0.50	0.47	0.49	200
52	0.64	0.83	0.73	200
53	0.38	0.49	0.43	200
54	0.65	0.62	0.63	200
55	0.35	0.44	0.39	200
56	0.29	0.18	0.22	200
57	0.18	0.05	0.08	200
58	0.51	0.40	0.45	200
59	0.70	0.81	0.75	200

60	0.54	0.48	0.51	200
61	0.45	0.70	0.55	200
62	0.28	0.14	0.18	200
63	0.27	0.21	0.24	200
64	0.48	0.54	0.50	200
65	0.38	0.45	0.41	200
66	0.53	0.63	0.58	200
67	0.42	0.60	0.50	200
68	0.25	0.18	0.21	200
69	0.46	0.35	0.40	200
70	0.42	0.60	0.49	200
71	0.44	0.39	0.41	200
72	0.56	0.71	0.63	200
73	0.82	0.89	0.85	200
74	0.42	0.28	0.34	200
75	0.69	0.72	0.71	200
76	0.67	0.65	0.66	200
77	0.57	0.73	0.64	200
78	0.57	0.65	0.60	200
79	0.60	0.39	0.47	200
80	0.45	0.57	0.50	200
81	0.20	0.14	0.16	200
82	0.42	0.59	0.49	200
83	0.52	0.66	0.58	200
84	0.35	0.31	0.33	200
85	0.27	0.17	0.21	200
86	0.32	0.24	0.28	200
87	0.36	0.12	0.19	200
88	0.40	0.33	0.36	200
89	0.41	0.47	0.44	200
90	0.46	0.43	0.45	200
91	0.45	0.56	0.50	200
92	0.33	0.28	0.30	200
93	0.67	0.87	0.76	200
94	0.60	0.72	0.66	200
95	0.52	0.69	0.59	200
96	0.52	0.47	0.49	200
97	0.55	0.69	0.61	200
98	0.48	0.46	0.47	200
99	0.39	0.32	0.35	200
100	0.45	0.55	0.49	200
101	0.86	0.87	0.86	200
102	0.49	0.41	0.45	200
103	0.42	0.48	0.45	200
104	0.43	0.29	0.35	200
105	0.83	0.94	0.89	200
106	0.44	0.21	0.29	200
107	0.47	0.57	0.52	200

	precision	recall	f1-score	support
108	0.72	0.85	0.78	200
109	0.52	0.44	0.48	200
110	0.51	0.62	0.56	200
111	0.59	0.40	0.47	200
112	0.61	0.70	0.65	200
113	0.66	0.81	0.73	200
accuracy			0.50	22800
macro avg	0.48	0.50	0.48	22800
weighted avg	0.48	0.50	0.48	22800

Random Forest Classifier Performance:
Accuracy: 0.8930701754385965

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.99	1.00	200
1	1.00	0.99	0.99	200
2	0.95	0.95	0.95	200
3	0.94	0.94	0.94	200
4	0.97	0.94	0.95	200
5	0.95	0.93	0.94	200
6	0.97	0.97	0.97	200
7	0.97	0.97	0.97	200
8	0.88	0.85	0.87	200
9	0.86	0.92	0.89	200
10	0.97	0.95	0.96	200
11	0.94	0.81	0.87	200
12	0.96	0.97	0.96	200
13	0.98	0.97	0.98	200
14	0.94	0.92	0.93	200
15	0.96	0.92	0.94	200
16	0.94	0.93	0.93	200
17	0.89	0.73	0.80	200
18	0.99	0.92	0.96	200
19	0.97	0.77	0.86	200
20	0.92	0.96	0.94	200
21	0.83	0.96	0.89	200
22	0.93	0.98	0.96	200
23	0.86	0.92	0.89	200
24	0.98	0.99	0.99	200
25	0.83	0.86	0.84	200
26	0.87	0.88	0.87	200
27	0.96	0.88	0.92	200
28	0.77	0.79	0.78	200
29	0.82	0.85	0.84	200
30	0.86	0.86	0.86	200

31	0.88	0.87	0.87	200
32	0.78	0.69	0.73	200
33	0.79	0.69	0.73	200
34	0.70	0.72	0.71	200
35	0.74	0.99	0.85	200
36	0.80	0.78	0.79	200
37	0.86	0.86	0.86	200
38	0.78	0.71	0.75	200
39	0.88	0.70	0.78	200
40	0.69	0.96	0.80	200
41	0.79	0.71	0.75	200
42	0.99	0.98	0.99	200
43	0.73	0.67	0.70	200
44	0.80	0.86	0.83	200
45	0.75	0.91	0.82	200
46	0.91	0.91	0.91	200
47	0.82	0.84	0.83	200
48	0.92	0.87	0.89	200
49	0.93	0.92	0.92	200
50	0.95	0.96	0.96	200
51	0.86	0.93	0.89	200
52	0.94	0.95	0.95	200
53	0.77	0.94	0.84	200
54	0.93	0.89	0.91	200
55	0.54	0.85	0.66	200
56	0.68	0.56	0.61	200
57	0.65	0.45	0.53	200
58	0.89	0.85	0.87	200
59	0.94	0.99	0.97	200
60	0.95	0.92	0.93	200
61	0.96	0.94	0.95	200
62	0.82	0.74	0.78	200
63	0.83	0.76	0.79	200
64	0.90	0.76	0.82	200
65	0.89	0.89	0.89	200
66	0.97	0.93	0.95	200
67	0.94	0.90	0.92	200
68	0.88	0.92	0.90	200
69	0.79	0.80	0.79	200
70	0.76	0.90	0.82	200
71	0.93	0.94	0.93	200
72	0.94	0.94	0.94	200
73	0.96	0.98	0.97	200
74	0.77	0.96	0.86	200
75	0.98	0.94	0.96	200
76	0.89	0.93	0.91	200
77	0.91	0.98	0.94	200
78	0.92	0.94	0.93	200

79	0.98	0.78	0.87	200
80	0.91	0.94	0.92	200
81	0.88	0.94	0.91	200
82	0.95	0.87	0.91	200
83	0.86	0.94	0.90	200
84	0.93	0.84	0.88	200
85	0.90	0.85	0.88	200
86	0.86	0.82	0.84	200
87	0.81	0.85	0.83	200
88	0.94	0.86	0.90	200
89	0.93	0.94	0.94	200
90	0.85	0.91	0.88	200
91	0.92	0.89	0.90	200
92	0.93	0.89	0.91	200
93	0.98	0.98	0.98	200
94	0.98	0.97	0.98	200
95	0.99	0.98	0.99	200
96	0.99	0.97	0.98	200
97	0.97	0.99	0.98	200
98	0.96	0.91	0.93	200
99	0.89	0.88	0.88	200
100	0.94	0.94	0.94	200
101	0.98	1.00	0.99	200
102	0.82	0.89	0.85	200
103	0.93	0.92	0.92	200
104	0.94	0.91	0.92	200
105	0.97	0.99	0.98	200
106	0.96	0.91	0.94	200
107	0.94	0.94	0.94	200
108	0.98	0.98	0.98	200
109	0.99	0.99	0.99	200
110	0.99	0.99	0.99	200
111	0.99	0.99	0.99	200
112	1.00	1.00	1.00	200
113	1.00	1.00	1.00	200
accuracy			0.89	22800
macro avg	0.90	0.89	0.89	22800
weighted avg	0.90	0.89	0.89	22800

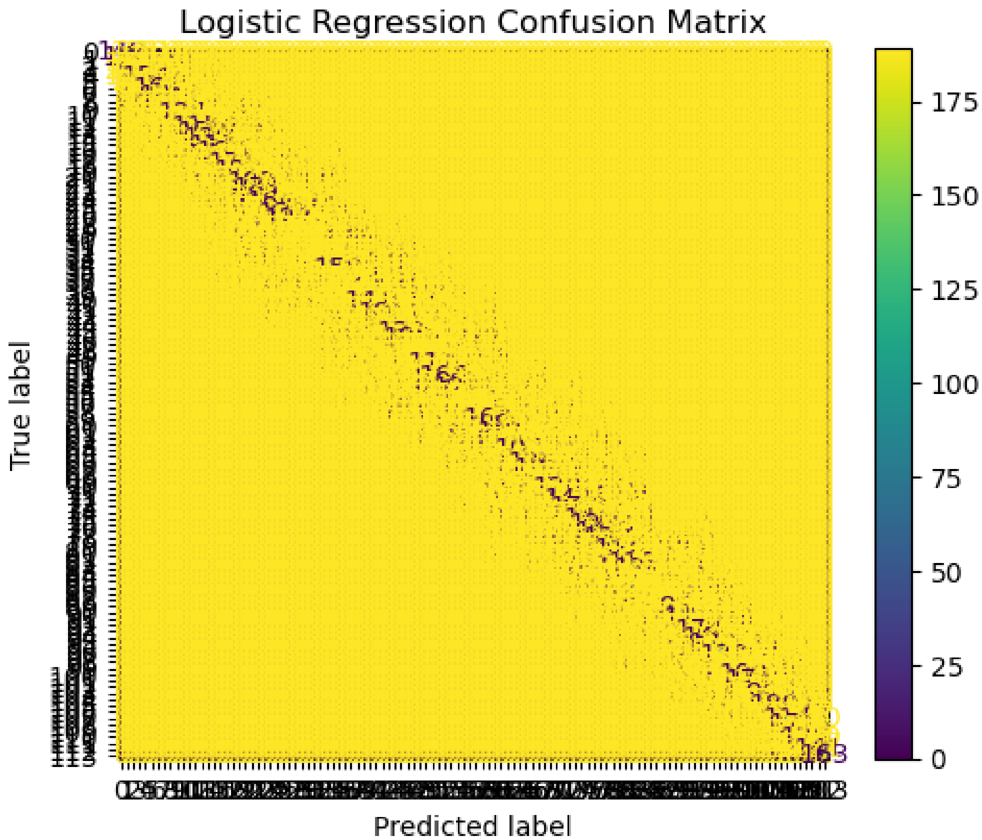
2.0.6 Prediction Discussion

- The Logistic Regression offered an accuracy score of about 50%, which is honestly better than I thought it would have. While it is not an amazing accuracy score, with 114 targets I am pretty impressed.
- The Random Forest Classifier offered the most opportunity to optimize hyperparameters with tweaking the max depth. A max depth of 10 resulted in about 70%, depth of 15 resulted in

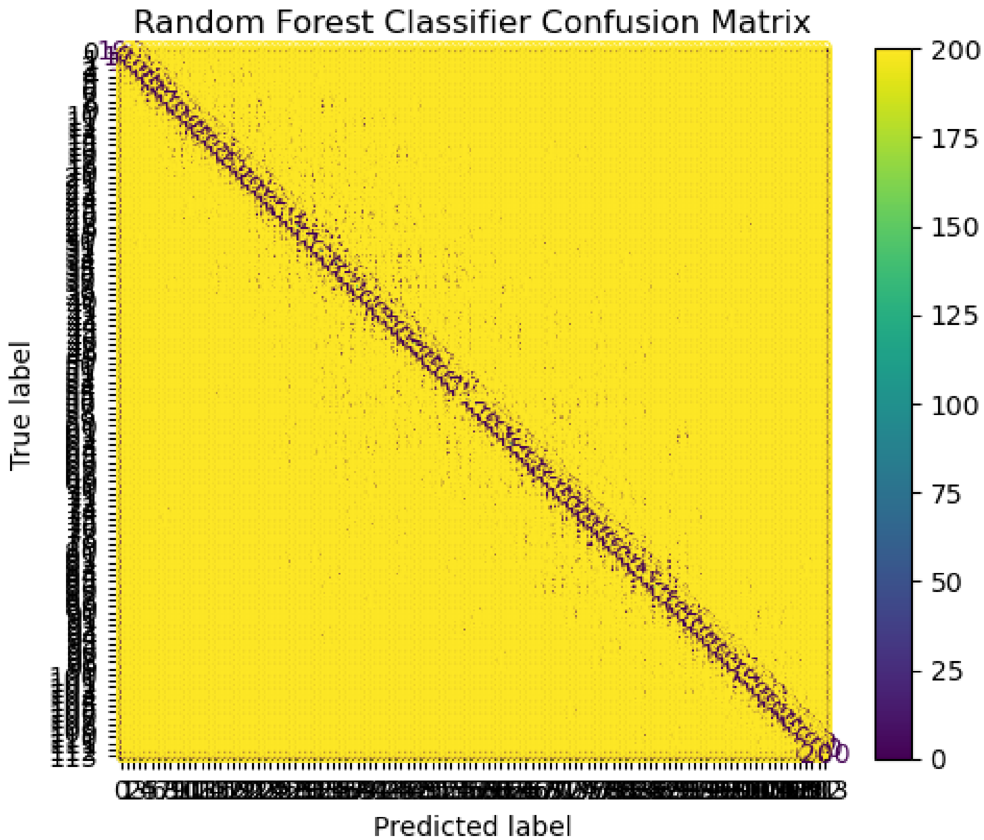
about 80%, and a depth of 20 resulted in about 90% classification accuracy. Depth of 17 and 18 both resulted in 89% classification accuracy, so diminishing returns begin at about depth 17. I am very impressed with the performance of the Random Forest Classifier, as there was so little correlation between any of the features and the target that I did not have much faith in any classifier.

2.0.7 Confusion Matrices

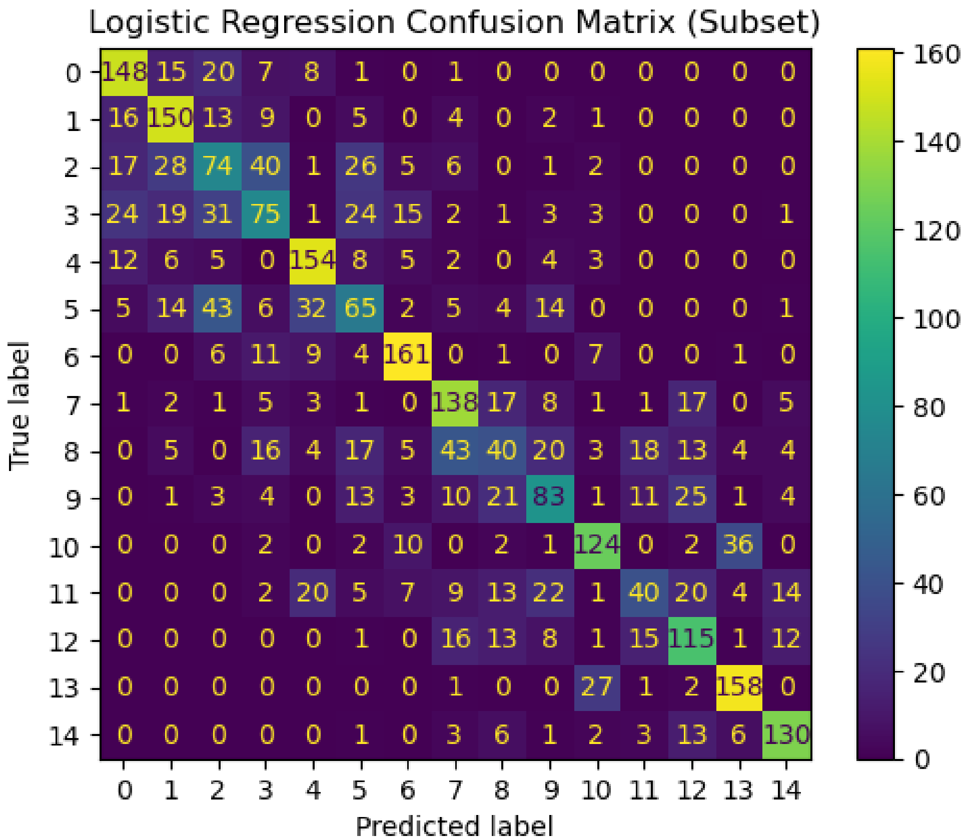
[12]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

lrMatrix = confusion_matrix(testY, logRegPredictions)
rfMatrix = confusion_matrix(testY, randomForestPredictions)

ConfusionMatrixDisplay(confusion_matrix = lrMatrix).plot()
plt.title("Logistic Regression Confusion Matrix")
plt.show()

ConfusionMatrixDisplay(confusion_matrix = rfMatrix).plot()
plt.title("Random Forest Classifier Confusion Matrix")
plt.show()

subsetNum = 15 # messing with this to try and get as many readable results as␣
 ↪possible

# getting subset of the confusion matrix
lrMatrixSub = lrMatrix[:subsetNum, :subsetNum]
rfMatrixSub = rfMatrix[:subsetNum, :subsetNum]

ConfusionMatrixDisplay(confusion_matrix = lrMatrixSub, display_labels = np.
 ↪arange(subsetNum)).plot()
plt.title("Logistic Regression Confusion Matrix (Subset)")
plt.show()

ConfusionMatrixDisplay(confusion_matrix= rfMatrixSub, display_labels = np.
 ↪arange(subsetNum)).plot()
plt.title("Random Forest Classifier Confusion Matrix (Subset)")
plt.show()
```

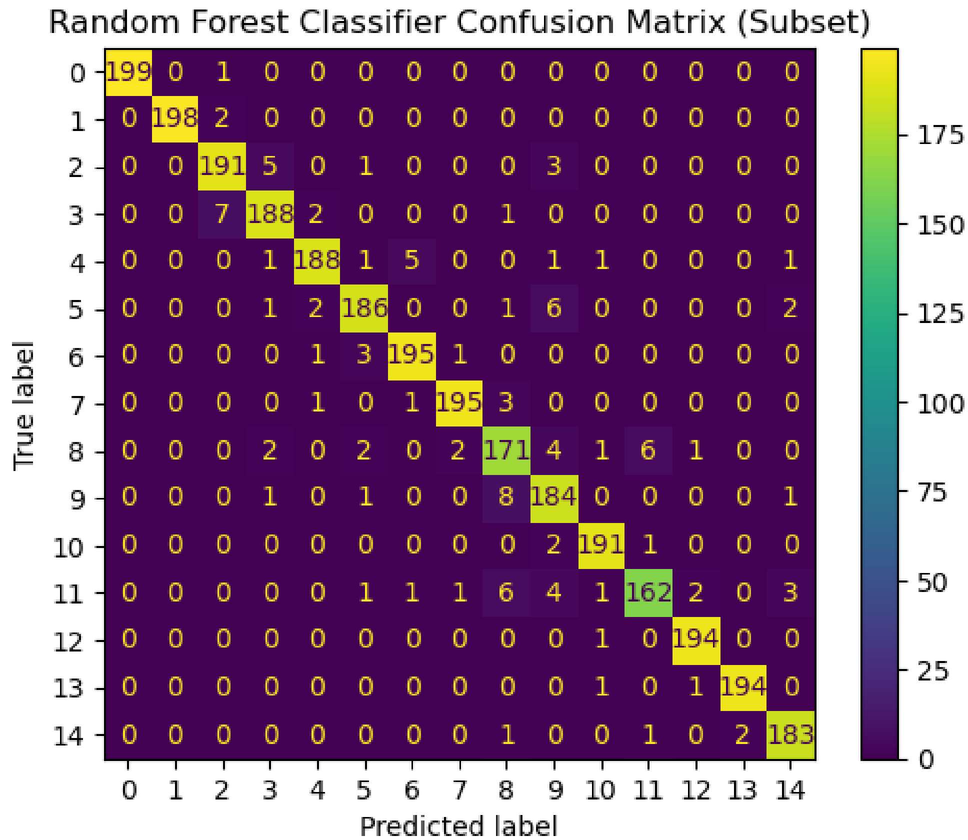


Logistic Regression Confusion Matrix
True label
Predicted label
175
150
125
100
75
50
25
0



Random Forest Classifier Confusion Matrix
True label
Predicted label
200
175
150
125
100
75
50
25
0


Logistic Regression Confusion Matrix (Subset)
True label
Predicted label
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 20 40 60 80 100 120 140 160



3 Confusion Matrices Discussion

- I left the original confusion matrices in, so that my reasoning for displaying a subset is clear. The 114 by 114 confusion matrix is unfortunately unreadable, so I displayed a subset as well. This subset number had to be tweaked, since I wanted the confusion matrix to have as much information as possible while still being readable.
- Simply looking at the difference between the logistic regression and random forest classifiers is striking. While the logistic regression has scattered guesses in some cases, with classifications being incorrect in some batches, the random forest classifier has mostly zeroes off of the main diagonal, with all of the incorrect classifications for each permutation being in the single digits. I am thoroughly impressed with how well the random forest classifier did, because when looking at the correlation matrix I thought that this dataset may be rough for any classifier I chose.