Introduction

With coronavirus and quarantine, companies in at-home fitness like Peloton experienced explosive growth. I was one of these people who invested in a Peloton bike and became a part of their online community. A common post is expression of excitement for their bike delivery coming soon and asking the group: **"What class and/or instructor should I take first?"**

A big part in selecting classes comes down to perceived difficulty based on the user. There isn't a UI feature to filter by this, but at a class level you can see the difficulty level based on user ratings they are prompted for after completing a ride. Additionally, there is the capability to search class by a song or artist based on what is on the playlist to pull classes with the type of music the person wants to ride to.

This notebook contains the winning Decision Tree model that predicts the perceived difficulty of the class based on Peloton class data including the Spotify features of the artists and tracks from the class playlists. As this was created with those receiving their Peloton bike in mind, the dataset and model only considers Cycling classes.

The goal metric was the F1 score (both micro-averaged and macro-averaged were tested for/considered)** as a False Positive like a class is labeled easy, but it isn't and a False Negative of a class is not labeled easy, but it is are of equal risk. They both impact a riders first experience with the bike. This is especially important for confidence building and creating a routine in fitness. If a class labelled easy is too difficult, the rider will less likely be back as completing the ride was not an achievable outcome.

Notebook Summary

Models

Initiating

- Creating Dataset

Modeling

- Manual Preprocessing
- Winning Decision Tree and Viz

Analysis

- Low Impact Rides
- Class Duration
- Interval Rides
- Hannah Corbin Rides
- Top Artists and Tracks per Difficulty
    - Beginner (Full Data and Model Results)
    - Advanced (Full Data and Model Results)
    - Intermediate (Full Data)
- Plotting

Conclusion

- Recommendations
- Future Work

Results

The final Decision Tree resulted in an accuracy and f1 score of 67-69%. From the modeling perspective, this is a net positive result as it is an improvement from the baseline of a 50% accuracy guessing Intermediate difficulty as it is the majority class.

The top features the drove classification were:

The top features the drove classification were:

- whether a class was a Low Impact ride
- the class duration
- whether the class was an Intervals ride

The Intermediate class has highest f1 and recall scores, which makes sense as Intermediate is the majority class.

- **Recall: What proportion of Intermediate classes were identified correctly?**

The Beginner class has the highest precision.

- **Precision: What proportion of classes the model identified as "Beginner" difficulty were actually correct?**

When applied to the business context, it is also positive result as Peloton is introducing cycling and spin to those who were not cyclists before ([Source]). This would signal that the majority of people's first ride would be in the Beginner or Intermediate space; thus it is acceptable for Advanced class difficulty to not be a labelling priority in comparison.

In [1]:

```
# Basics
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
import math

# # Importing dfs with heavy processing
# import pickle

# Imports Modeling
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Feat Imp
from yellowbrick.model_selection import FeatureImportances

# Decision Tree Plotting
import graphviz
from sklearn import tree
import pylab
```

In [2]:

```
# import custom functions, associated packages and pickle files
%run '/Users/amandagaeta/Documents/Flatiron/capstone/project/Peloton-Class-Difficulty-Cla
ssification-With-Playlists/data/peloton_spotify_functions.py'
```

Creating base dataset

The steps below get the data to match Data V6 from previous notebooks where instructors and class categories were condensed to account for those with low counts.

In [3]:

```
df = pd.read_pickle("../../data/pickled_dfs/master_first_classes_with_stats.pkl")
df.head()
```

Out[3]:

	classId	className	classDescription	classDifficulty	classDuration	classType	classLength	c
0	9680a817bf2149d2b91990c87166a400	20 min Pop Ride	We dare you not to dance as you ride to all th...	7.4000	20	Cycling	24	

	classId	className	classDescription	classDifficulty	classDuration	classType	classLength	c
1	932f15ed407f46049988ba4c46e3ee3b	20 min HIIT Ride	Powerful and efficient, this high-intensity in...	7.7203	20	Cycling	24	
2	9319eb174dee4cb081f6491cc81e7c7e	30 min Advanced Beginner Ride	Still warming up but ready for more? Build on ...	8.0000	30	Cycling	33	
3	8a8c181b523b430487f6a23bb0436178	30 min Pop Ride	We dare you not to dance as you ride to all th...	7.6487	30	Cycling	34	p
4	8903dfb7bae742a9bd00bf3afd718afa	20 min 80s Ride	Take a ride through the classic tracks of the ...	7.1325	20	Cycling	23	p

In [4]:

```
df['classDifficulty'].describe()
```

Out[4]:

```
count    6983.000000
mean        7.858991
std         0.778619
min         4.272800
25%         7.554450
50%         8.047400
75%         8.378250
max         9.653600
Name: classDifficulty, dtype: float64
```

In [5]:

```
# Create classDifficulty_cat for categories of Class Difficulty for classification target
df['classDifficulty_cat'] = df['classDifficulty'].apply(label_class_diff_cat)
```

In [6]:

```
# Create numerical target, can use classDifficulty_cat for labels
df['classDifficulty_num'] = df['classDifficulty_cat'].map({'Beginner': 0,
                                                           'Intermediate': 1,
                                                           'Advanced': 2})
```

In [7]:

```
# Group instructors will low class count
other_instructors = ['Irène Scholz', 'Erik Jäger', 'Christian Vande Velde', 'Cliff Dweng
er',
                     'Alex & Tunde', 'Mayla Wedekind', 'Ally & Emma', 'Cycling Instructo
rs 2019']

df['instructorName'] = df['instructorName'].replace(other_instructors, 'Other')
```

In [8]:

```
# Recategorize low count classes
# Create Other category for those <50
# Put Groove Cody classes into Theme - not a music genre or year, but a programmatic type
class like XOXO
df['classCategory'] = df['classCategory'].replace({'Heart Rate Zone' : 'Other',
                                                   'Live DJ' : 'Other',
                                                   'Pro Cyclist' : 'Other',
```

```
                                                          'Groove': 'Theme'})
```

Modeling

In [9]:

```
# Define X and y
# Drop Peloton class features not used in model
X = df.drop(columns=['classId', 'classDifficulty', 'classDifficulty_cat', 'classDifficul
ty_num',
                     'classOriginalAirdate', 'classType', 'classLength', 'instructorBio'
, 'classEquipment',
                     'classUrl', 'classRating', 'classRatingCount', 'classDescription',
'classSongs',
                     'classArtists','className', 'classLocation'])

# Drop target
y = df['classDifficulty_num']
```

Manual Preprocessing for Modeling

Purposefully not in pipeline for easier data access in analysis

In [10]:

```
# Replace className with classCategory in ohe_cols
num_cols = ['classDuration', 'popularity_song', 'explicit', 'danceability', 'energy', 'ke
y',
            'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'live
ness',
            'valence', 'tempo', 'time_signature', 'followers', 'popularity_artist', 'dur
ation_mins']

ohe_cols = ['classCategory', 'instructorName']
```

Numeric Treatment - Scale

In [11]:

```
# Copy df for manipulation
scaled_features = df.copy()
```

In [12]:

```
# Scale num_col features
features = scaled_features[num_cols]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)

# Put into DF for concatenation
scaled_features[num_cols] = features
scaled = scaled_features[num_cols]

# Check work
scaled.head()
```

Out[12]:

	classDuration	popularity_song	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	i
0	-0.925958	0.889340	-0.26515	0.343782	-0.733255	0.999161	0.884929	0.690826	-0.164187	1.428143	
1	-0.925958	-1.671140	-0.26515	0.283662	0.263357	-0.427414	0.094531	0.690826	-0.295639	-0.832514	
2	-0.053947	0.808055	-0.26515	-0.203303	0.733554	-0.189652	0.856351	-1.671005	-0.596627	0.082597	

	classDuration	popularity_song	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	i
3	-0.053947	0.482915	-0.26515	-1.357593	0.334909	0.761399	1.101792	0.690826	-0.606455	-0.841938	
4	-0.925958	-0.736362	-0.26515	-0.612114	-0.533932	0.761399	-2.961995	0.690826	-0.640854	-0.627213	

Categorical Treatment (OHE)

In [13]:

```
# Copy df for manipulation
ohe_features = df.copy()
```

In [14]:

```
# Filter down to just ohe_cols
ohe_features = ohe_features[ohe_cols]

# OHE/Get Dummies
ohe_features = pd.get_dummies(ohe_features)

# Preview, check work
ohe_features.head()
```

Out[14]:

	classCategory_Beginner	classCategory_Climb	classCategory_Intervals	classCategory_Low Impact	classCategory_Music	classCate
0	0	0	0	0	1	
1	0	0	1	0	0	
2	1	0	0	0	0	
3	0	0	0	0	1	
4	0	0	0	0	1	

In [15]:

```
ohe_features = pd.get_dummies(ohe_features)
```

Combine

In [16]:

```
# Combine scaled numerical, OHE categoricals, and target into one df
preprocessed = pd.concat([scaled, ohe_features, y], axis=1)
```

In [17]:

```
# Review available columns, check work
preprocessed.columns
```

Out[17]:

```
Index(['classDuration', 'popularity_song', 'explicit', 'danceability', 'energy', 'key', '
loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valenc
e', 'tempo', 'time_signature', 'followers', 'popularity_artist', 'duration_mins', 'classC
ategory_Beginner', 'classCategory_Climb', 'classCategory_Intervals', 'classCategory_Low I
mpact', 'classCategory_Music', 'classCategory_Other', 'classCategory_Power Zone', 'classC
ategory_Theme', 'instructorName_Alex Toussaint', 'instructorName_Ally Love', 'instructorN
ame_Ben Alldis', 'instructorName_Christine D'Ercole', 'instructorName_Cody Rigsby', 'inst
ructorName_Denis Morton', 'instructorName_Emma Lovewell', 'instructorName_Hannah Corbin',
'instructorName_Hannah Frankson', 'instructorName_Jenn Sherman', 'instructorName_Jess Kin
g', 'instructorName_Kendall Toole', 'instructorName_Leanne Hainsby', 'instructorName_Matt
Wilpers', 'instructorName_Olivia Amato', 'instructorName_Other', 'instructorName_Robin Ar
```

```
zon',
       'instructorName_Sam Yo', 'instructorName_Tunde Oyeneyin', 'classDifficulty_num'],
      dtype='object')
```

In [18]:

```
preprocessed.head()
```

Out[18]:

	classDuration	popularity_song	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	i
0	-0.925958	0.889340	-0.26515	0.343782	-0.733255	0.999161	0.884929	0.690826	-0.164187	1.428143	
1	-0.925958	-1.671140	-0.26515	0.283662	0.263357	-0.427414	0.094531	0.690826	-0.295639	-0.832514	
2	-0.053947	0.808055	-0.26515	-0.203303	0.733554	-0.189652	0.856351	-1.671005	-0.596627	0.082597	
3	-0.053947	0.482915	-0.26515	-1.357593	0.334909	0.761399	1.101792	0.690826	-0.606455	-0.841938	
4	-0.925958	-0.736362	-0.26515	-0.612114	-0.533932	0.761399	-2.961995	0.690826	-0.640854	-0.627213	

Winning Decision Tree Model

In [19]:

```
# X and y split of preprocessed
X = preprocessed.drop(columns=['classDifficulty_num'], axis=1)
y = preprocessed['classDifficulty_num']
```

In [20]:

```
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4
2)
```

In [21]:

```
# Instantiate
tuned_dt_ent = DecisionTreeClassifier(criterion = 'entropy',
                                       max_depth = 9,
                                       max_features = 14,
                                       min_samples_leaf = 1,
                                       min_samples_split = 25,
                                       random_state=42)


tuned_dt_ent.fit(X_train, y_train)
```

Out[21]:

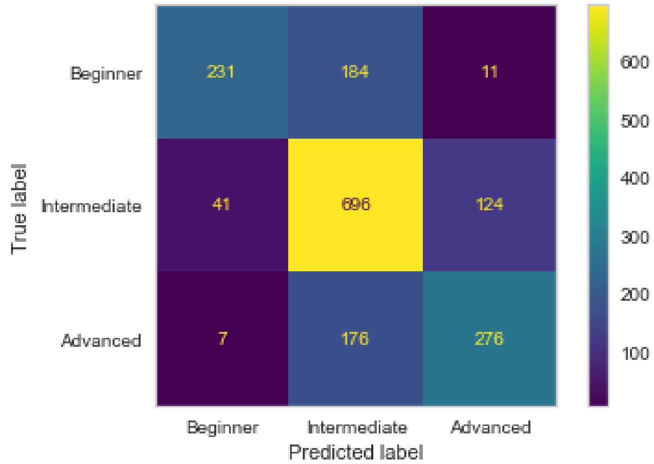
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=9, max_features=14, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=25,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [22]:

```
# F1 Micro
eval_model(tuned_dt_ent, X_train, X_test, y_train, y_test, ['Beginner', 'Intermediate',
'Advanced'], average='micro')
# less overfit by another 1%
# accuracy/f1 only down 0.002
```

```
              precision    recall  f1-score   support
```

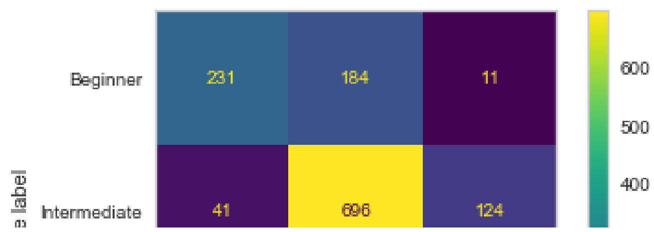
```
              precision    recall  f1-score   support

    Beginner       0.83      0.54      0.66       426
Intermediate       0.66      0.81      0.73       861
    Advanced       0.67      0.60      0.63       459

    accuracy                           0.69      1746
   macro avg       0.72      0.65      0.67      1746
weighted avg       0.70      0.69      0.68      1746

Train Scores
------------
Accuracy: 0.7196868436127554
F1 Score: 0.7196868436127553
--------------------
Test Scores
-----------
Accuracy: 0.6890034364261168
F1 Score: 0.6890034364261168
F1 Score Mean Cross Val 3-Fold: 0.6849357406915083
```

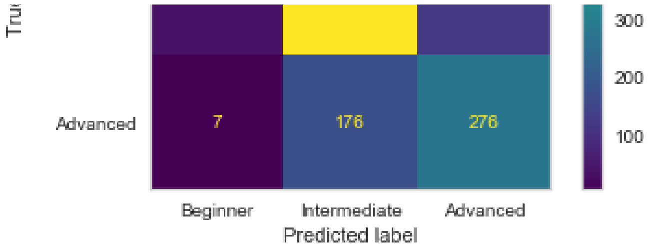


In [23]:

```python
# F1 Macro
eval_model(tuned_dt_ent, X_train, X_test, y_train, y_test, ['Beginner', 'Intermediate',
'Advanced'], average='macro')
# accuracy/f1 down 0.02 from micro
```

```
              precision    recall  f1-score   support

    Beginner       0.83      0.54      0.66       426
Intermediate       0.66      0.81      0.73       861
    Advanced       0.67      0.60      0.63       459

    accuracy                           0.69      1746
   macro avg       0.72      0.65      0.67      1746
weighted avg       0.70      0.69      0.68      1746

Train Scores
------------
Accuracy: 0.7196868436127554
F1 Score: 0.7063475753551763
--------------------
Test Scores
-----------
Accuracy: 0.6890034364261168
F1 Score: 0.6719788309487916
F1 Score Mean Cross Val 3-Fold: 0.6707176813375108
```





Visualization

In [24]:

```
# Feature names and class names for decision tree plotting
fn= X_train.columns
cn=['Beginner', 'Intermediate', 'Advanced']
```

In [25]:

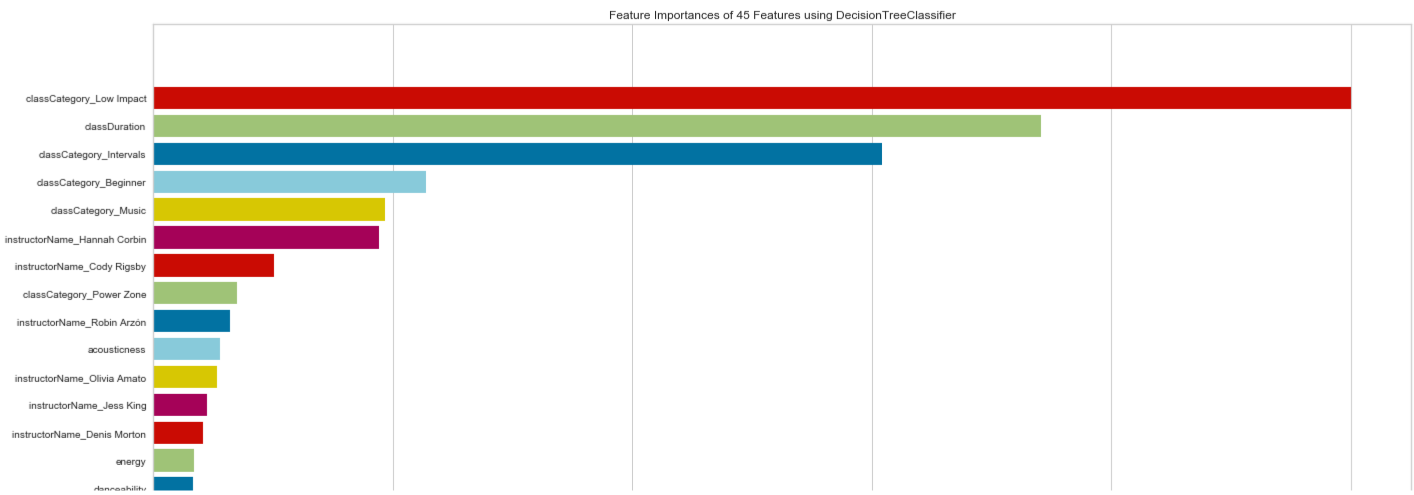
```
# DOT data
dot_data = tree.export_graphviz(tuned_dt_ent, out_file=None,
                                feature_names=fn,
                                class_names=cn,
                                max_depth = 10,
                                filled=True)

# Draw graph
graph = graphviz.Source(dot_data, format="png")

# save graph
pel_tree = graph.render(filename='../../images/man_pel_tree')

# show graph
graph

# Source:  https://mljar.com/blog/visualize-decision-tree/
```

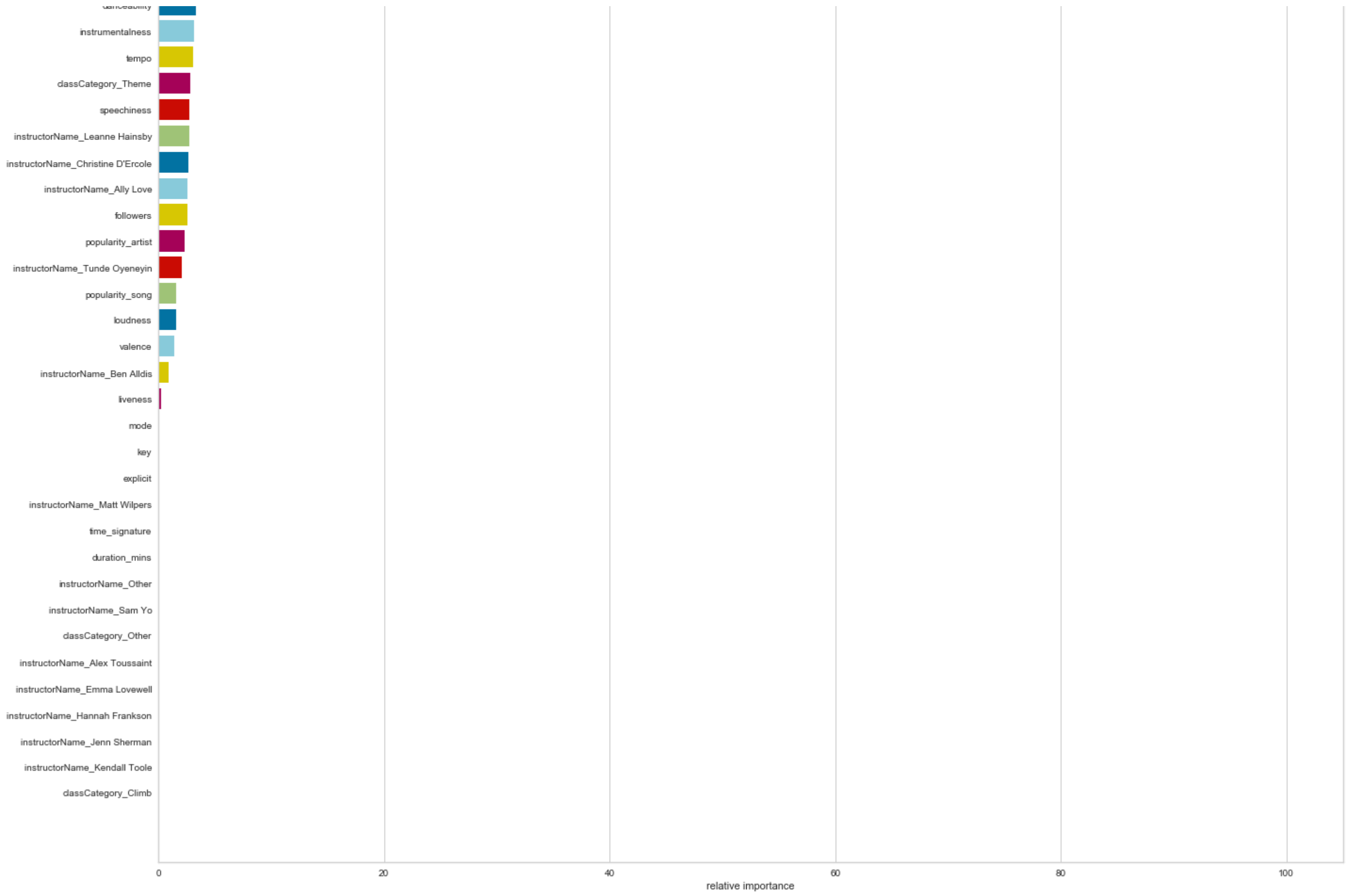
Out[25]:

In [26]:

```
# Feature importance plotting
plt.figure(figsize=(20,20))
viz = FeatureImportances(tuned_dt_ent)
viz.fit(X, y)
viz.show()
```

```
/opt/anaconda3/envs/learn-env/lib/python3.6/site-packages/sklearn/base.py:197: FutureWarn
ing: From version 0.24, get_params will raise an AttributeError if a parameter cannot be
retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```





Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa1fc79da58>
```

Learning:

Top drivers of class difficulty level are:

- **whether the class is a Low Impact class**
- **the Class Duration or length of the class**
- **whether the class is an Intervals class**

Analysis

Top Features

Low Impact

In [27]:

```
# Filter down dataset to Low Impact rides
preprocessed[preprocessed['classCategory_Low Impact'] == 1]['classDifficulty_num'].value
_counts(normalize=True).mul(100).round(1).astype(str) + '%'
# 99.3% of Low Impact rides are Beginner, thus easy for the model to categorize with the
majority
```

Out[27]:

```
0    99.3%
1     0.7%
Name: classDifficulty_num, dtype: object
```

Class Duration

In [28]:

```
# Tree split on classDuration <= -0.49. If False, goes to Intervals then Instructors
# Check out if false subset to see what durations fall under this scaled value for Advanc
ed
classdur_subset = preprocessed[preprocessed['classDuration'] > -0.49]

# create variable for original df classDuration Series
og_classdur = df['classDuration']

# Rename for merging
og_classdur.rename('og_classDur', inplace=True)

# Merge original information to subset of scaled classDuration values
ClassDuration_traceback = classdur_subset.merge(og_classdur, how='left',
                                                 left_index=True, right_index=True)

# filter down columns for comparison
only_classdurs = ClassDuration_traceback[['classDuration', 'og_classDur']]

# overview of original classDurations included in subset
class_durations_pie = pd.DataFrame(only_classdurs['og_classDur'].value_counts())

# Can see classDuration <= -0.49 traces back to classDuration <= 30 minutes
```

In [29]:

```
# Remove outliers (75 min+ classes, only 13) for charting
class_durations_pie = class_durations_pie[:-2]
class_durations_pie
```

Out[29]:

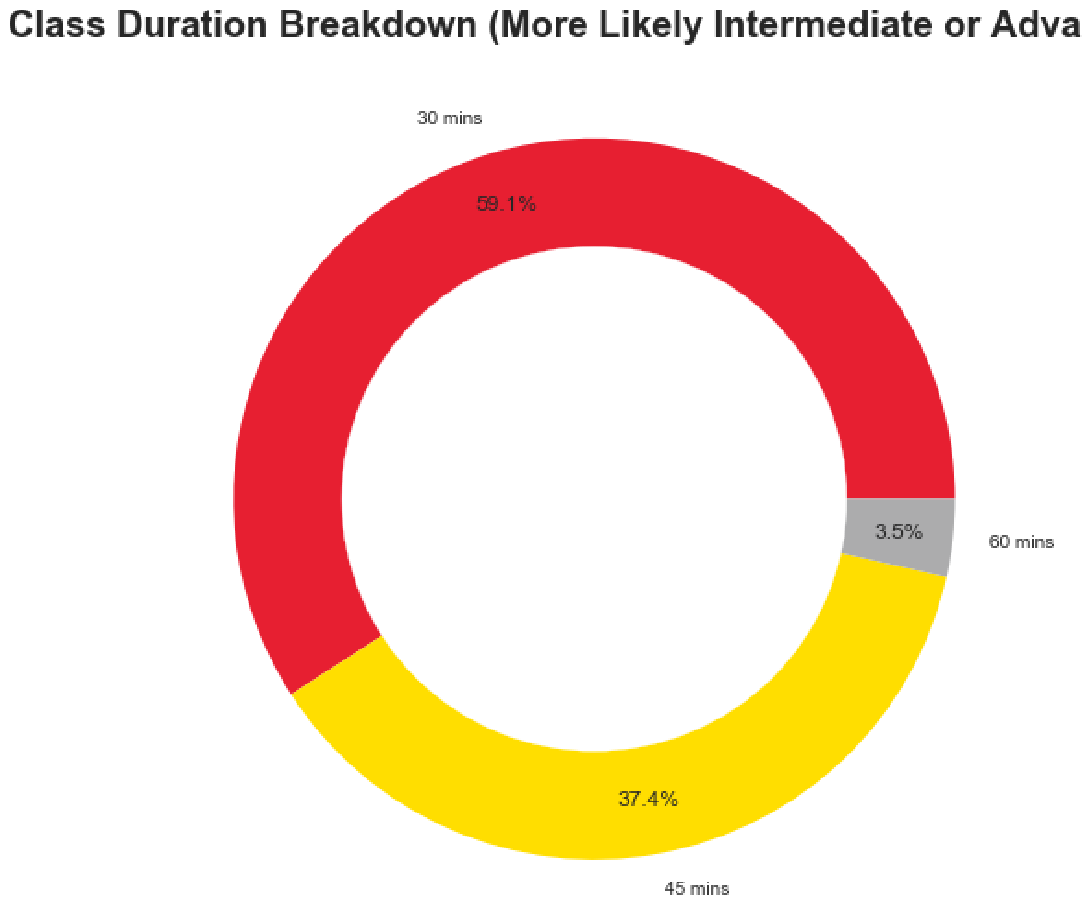
	og_classDur
30	2750
45	1740
60	161

In [30]:

```
# Plot pie chart

# Set font size
plt.rcParams['font.size'] = 12.0

# Set pie chart size
fig = plt.gcf()
fig.set_size_inches(9, 9)

#Create pie chart
plt.pie(class_durations_pie, labels = ['30 mins', '45 mins', '60 mins'],
       labeldistance=1.1, pctdistance=0.85, autopct='%1.1f%%', colors = ['#e71f31','#ffd
e00','#acacad','#000000'])

# add a circle at the center to transform it in a donut chart
my_circle=plt.Circle( (0,0), 0.7, color='white')
p=plt.gcf()
p.gca().add_artist(my_circle)

# Formatting
plt.title('Class Duration Breakdown (More Likely Intermediate or Advanced)', fontsize=20,
fontweight="bold")
plt.tight_layout()

# Save
plt.savefig("../../images/Int_Adv_Class_Duration")
plt.show()
```

```
/opt/anaconda3/envs/learn-env/lib/python3.6/site-packages/ipykernel_launcher.py:12: Matpl
```

```
otlibDeprecationWarning: Non-1D inputs to pie() are currently squeeze()d, but this behavi
or is deprecated since 3.1 and will be removed in 3.3; pass a 1D array instead.
  if sys.path[0] == '':
```



In [31]:

```
# Compare with all original ClassDurations. Can see all 30 min+ classes are included in t
he classDuration > -0.49
df['classDuration'].value_counts()
```

Out[31]:

```
30    2750
20    1749
45    1740
15     318
10     252
60     161
90       7
75       6
Name: og_classDur, dtype: int64
```

Intervals Classes and Instructors

In [32]:

```
# Filter down dataset to Interval rides
preprocessed[preprocessed['classCategory_Intervals'] == 1]['classDifficulty_num'].value_
counts(normalize=True).mul(100).round(1).astype(str) + '%'
# Most Advanced classes, nearly tied with Intermediate
# Nearly 0 Beginner classes
```

Out[32]:

```
2    51.0%
1    47.8%
0     1.2%
Name: classDifficulty_num, dtype: object
```

In [33]:

```
# Filter down dataset to Interval rides
df[df['classCategory'] == 'Intervals']['instructorName'].value_counts(normalize=True).mu
l(100).round(1).astype(str) + '%'

interval_instrs = pd.DataFrame(df[df['classCategory'] == 'Intervals']['instructorName'].
value_counts())
interval_instrs
# Tree shows that Robin and Olivia Interval classes are more Advanced than other instruct
ors.
# They make up 11.4% of Intervals classes
```

Out[33]:

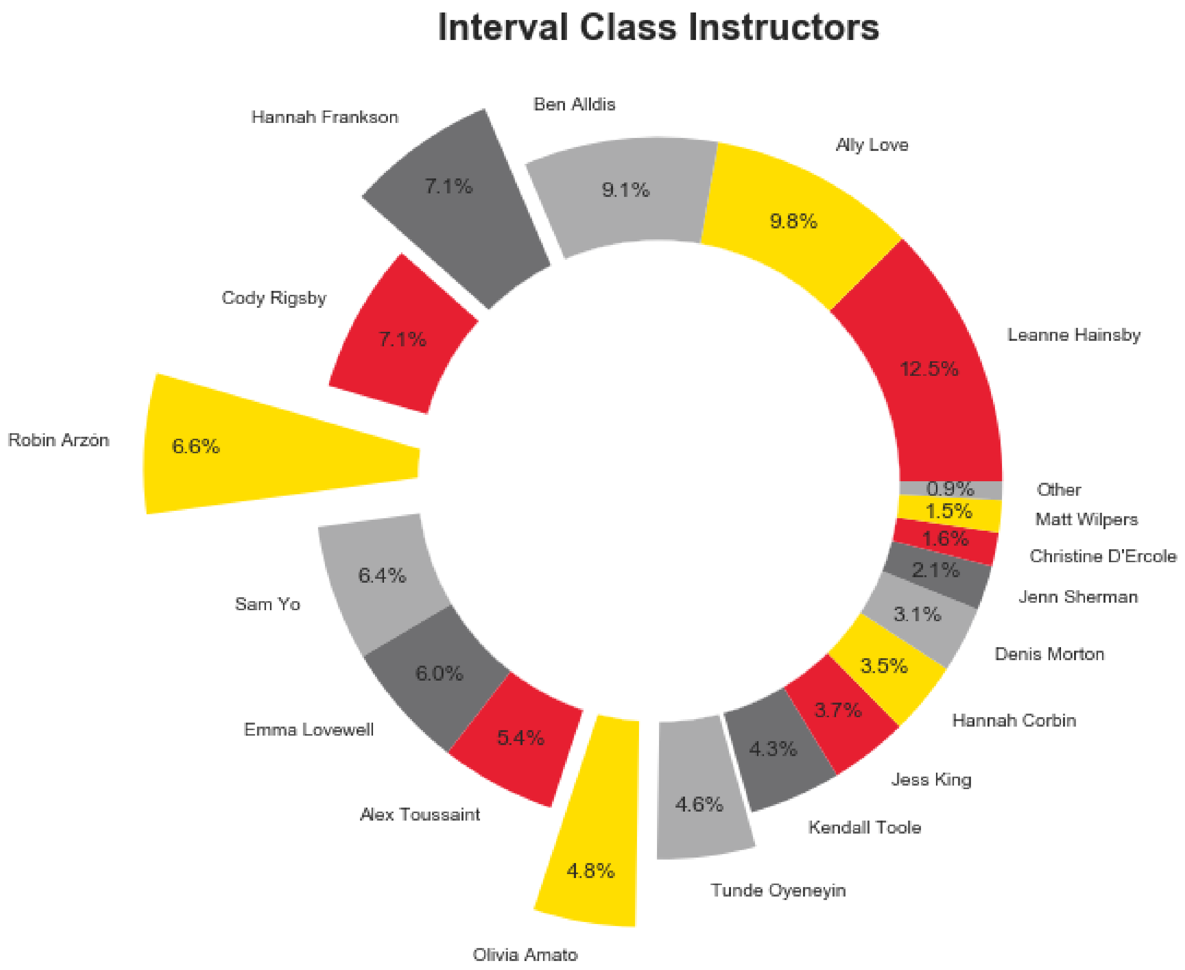
	instructorName
Leanne Hainsby	213
Ally Love	167
Ben Alldis	156
Hannah Frankson	121
Cody Rigsby	121
Robin Arzón	113
Sam Yo	109
Emma Lovewell	102
Alex Toussaint	93
Olivia Amato	82
Tunde Oyeneyin	79
Kendall Toole	74
Jess King	63
Hannah Corbin	59
Denis Morton	53
Jenn Sherman	36
Christine D'Ercole	27
Matt Wilpers	26
Other	15

In [34]:

```
# Plot pie chart
plt.rcParams['font.size'] = 12.0

# Set size
fig = plt.gcf()
fig.set_size_inches(9, 9)


# only "explode" Robin and Olivia as most advanced instructors, then Cody and Tunde in In
tervals
explode = (0, 0, 0, 0.2, 0,
          0.5, 0, 0, 0, 0.3,
          0.1, 0, 0, 0, 0,
          0, 0, 0, 0)

#Create pie chart
plt.pie(interval_instrs, labels = list(interval_instrs.index), explode=explode,
       labeldistance=1.1, pctdistance=0.85, autopct='%1.1f%%',
        colors = ['#e71f31','#ffde00','#acacad','#6f6f71'])

# add a circle at the center to transform it in a donut chart
my_circle=plt.Circle( (0,0), 0.7, color='white')
```

```
p=plt.gcf()
p.gca().add_artist(my_circle)

plt.title('Interval Class Instructors', fontsize=20, fontweight="bold")

# Equal aspect ratio ensures that pie is drawn as a circle
# plt.axis('equal')
plt.tight_layout()
plt.savefig("../../images/Int_Instructors")
plt.show()
```

```
/opt/anaconda3/envs/learn-env/lib/python3.6/site-packages/ipykernel_launcher.py:18: MatplotlibDeprecationWarning: Non-1D inputs to pie() are currently squeeze()d, but this behavior is deprecated since 3.1 and will be removed in 3.3; pass a 1D array instead.
```



Hannah Corbin

In [35]:

```
# Filter down dataset to Hannah Corbin rides
preprocessed[preprocessed['instructorName_Hannah Corbin'] == 1]['classDifficulty_num'].value_counts(normalize=True).mul(100).round(1).astype(str) + '%'
# Mostly Beginner classes, close with Intermediate though
# Rarely Advanced
```

Out[35]:

```
0    58.4%
1    39.4%
2     2.2%
Name: classDifficulty_num, dtype: object
```

Top Artists and Songs in Difficulties

Look at full lists of top artist and song representations in difficulties. See if there are any general patterns.

This was also used to grab samples of artist and song samples for presentation purposes.

Beginner Classes (Total labeled in modeling data)

In [36]:

```
# Create data subset of beginner classes for analysis
beg_subset_all = preprocessed[preprocessed['classDifficulty_num'] == 0]
```

In [37]:

```
# Run diff_top_artists_and_songs on data set to get top artists and songs for analysis
beg_top_artists_all, beg_top_songs_all = diff_top_artists_and_songs(beg_subset_all, df)
```

In [38]:

```
# Preview artists
beg_top_artists_all[:10]
```

Out[38]:

	name	classCount	Class Count	id	class_per	followers	genres	popularity
122	LP	23	40	0J7U24vlOOIeMpuaO6Q85A	0.57	2034354.0	['la pop', "women's music"]	73.0
67	Gavin DeGraw	33	76	5DYAABs8rkY9VhwtENoQCz	0.43	1098820.0	['acoustic pop', 'neo mellow', 'pop', 'pop rock']	68.0
153	Barry White	19	48	3rfgbfpPSfXY40lzRK7Syt	0.40	1257407.0	['adult standards', 'disco', 'funk', 'motown',...	70.0
77	The Temptations	31	89	3RwQ26hR2tJtA8F9p2n7jG	0.35	2019971.0	['brill building pop', 'classic soul', 'funk',...	72.0
156	The Fray	19	62	0zOcE3mg9nS6l3yxt1Y0bK	0.31	3287667.0	['neo mellow', 'piano rock', 'pop', 'pop rock']	74.0
154	Jack Garratt	19	63	1Zp054Jc86WVKCxKEqZGOA	0.30	345251.0	['uk alternative pop']	55.0
167	Matchbox Twenty	19	65	3Ngh2zDBRPEriyxQDAMKd1	0.29	1824581.0	['neo mellow', 'pop rock', 'post-grunge']	71.0
21	Dave Matthews Band	64	220	2TI7qyDE0QfyOlnbtfDo7L	0.29	1504809.0	['jam band', 'neo mellow', 'pop rock']	70.0
61	Jackson 5	35	123	2iE18Oxc8YSumAU232n4rW	0.28	NaN	NaN	NaN
99	Wind & Fire	28	105	NaN	0.27	NaN	NaN	NaN

In [39]:

```
# Preview songs
beg_top_songs_all[:10]
# Top songs are not all matching up to top artists
# There is more variety in artists within songs
```

Out[39]:

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popul
135	Cop Stop	8	spotify:track:2PZo1LXIOMAYGwr0pYn4mT	8	1.00	Gavin DeGraw	NaN	
194	Worry	7	spotify:track:2UingDwvZlskY59ehcc2iE	8	0.88	NaN	['Bill Anderson']	

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popul
239	I'm Not A Saint	6	spotify:track:0CKPLoYW0nsAnjnr00HRWV	7	0.86	Billy Raffoul	NaN	
255	In the Summertime	6	spotify:track:22dGwFDrsk4JmMvpY9kkzV	8	0.75	NaN	['Bill Anderson']	
228	Crystals	6	spotify:track:5wU6jk9kxYzFGUpeE6T2Q5	8	0.75	NaN	['Of Monsters and Men']	
125	Work It Out (Album Version) (feat. Dave Matthe...	8	None	11	0.73	NaN	NaN	
21	Jackie and Wilson (Album Version)	15	spotify:track:7mxNte0lD8HOLwLS4TvmdZ	21	0.71	The Karaoke Party Poppers	NaN	
244	White Houses	6	spotify:track:6UtKnVSvYhNGO0n8SWdpVn	9	0.67	NaN	['Eric Burdon & the Animals']	
242	Skeletons	6	spotify:track:3w2kXOCa1Kain0vJTnGknC	10	0.60	NaN	['Stevie Wonder']	
66	This Too Shall Pass	10	spotify:track:3l9eeEd5NABrvYRzXIVqwK	17	0.59	OK Go	NaN	

Beginner Classes (Model Results - Low Impact Classes)

In [40]:

```
# Create data subset of low impact classes for analysis
beg_subset_lowim = preprocessed[(preprocessed['classCategory_Low Impact'] == 1)]
```

In [41]:

```
# Run diff_top_artists_and_songs on data set to get top artists and songs for analysis
beg_top_artists_lowim, beg_top_songs_lowim = diff_top_artists_and_songs(beg_subset_lowim,
df)
```

In [42]:

```
# Preview artists
beg_top_artists_lowim[:10]
```

Out[42]:

	name	classCount	Class Count	id	class_per	followers	genres	popularity
10	Gavin DeGraw	26	76	5DYAABs8rkY9VhwtENoQCz	0.34	1098820.0	['acoustic pop', 'neo mellow', 'pop', 'pop rock']	68.0
28	The Fray	18	62	0zOcE3mg9nS6l3yxt1Y0bK	0.29	3287667.0	['neo mellow', 'piano rock', 'pop', 'pop rock']	74.0
64	Gin Blossoms	12	45	6kXp61QMZFPcKMcRPqoiVj	0.27	456493.0	['alternative rock', 'neo mellow', 'permanent ...	60.0
49	Andy Grammer	14	51	2oX42qP5ineK3hrhBECLmj	0.27	930554.0	['dance pop', 'modern rock', 'neo mellow', 'po...	77.0
55	Barry White	12	48	3rfgbfpPSfXY40lzRK7Syt	0.25	1257407.0	['adult standards', 'disco', 'funk', 'motown',...	70.0
71	George Ezra	11	53	2ysnwxxNtSgbb9t1m2Ur4j	0.21	3421047.0	['folk-pop', 'modern rock', 'neo mellow', 'neo...	76.0
17	Jackson 5	23	123	2iE18Oxc8YSumAU232n4rW	0.19	NaN	NaN	NaN

	name	classCount	Class Count	id	class_per	followers	genres	popularity
34	The Temptations	17	89	3RwQ26hR2tJtA8F9p2n7jG	0.19	2019971.0	['brill building pop', 'classic soul', 'funk',...	72.0
53	Ray LaMontagne	13	68	6DoH7ywD5BcQvjloe9Oclj	0.19	846154.0	['acoustic pop', 'indie folk', 'neo mellow', '...	69.0
18	Train	23	122	3FUY2gzHeliaesXtOAdB7A	0.19	3650908.0	['dance pop', 'neo mellow', 'pop', 'pop rock',...	78.0

In [43]:

```
# Preview songs
beg_top_songs_lowim[:10]
```

Out[43]:

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	pop
17	Cop Stop	8	spotify:track:2PZo1LXIOMAYGwr0pYn4mT	8	1.00	Gavin DeGraw	NaN	
62	Hysteric	5	spotify:track:3zH9RnVnw2v4TemWU5j6Qz	5	1.00	Yeah Yeah Yeahs	NaN	
77	Are You Happy Now?	5	spotify:track:5B7XIcS5T76NJZFOHX30Io	6	0.83	NaN	['Michelle Branch']	
66	Soldier	5	spotify:track:5QD28FqaM3jTfsqWwvRZwv	7	0.71	NaN	['Eminem']	
46	I'm Not A Saint	5	spotify:track:0CKPLoYW0nsAnjnr00HRWV	7	0.71	Billy Raffoul	NaN	
53	I'm Good.	5	None	7	0.71	NaN	NaN	
3	Jackie and Wilson (Album Version)	12	spotify:track:7mxNte0lD8HOLwLS4TvmdZ	21	0.57	The Karaoke Party Poppers	NaN	
49	White Houses	5	spotify:track:6UtKnVSvYhNGO0n8SWdpVn	9	0.56	NaN	['Eric Burdon & the Animals']	
4	You Are the Best Thing	12	spotify:track:1jyddn36UN4tVsJGtaJfem	26	0.46	NaN	['Ray LaMontagne']	
60	Rain (feat. Nicky Jam)	5	spotify:track:5od0xFdCUrt6vHz6ejEfwn	11	0.45	The Script	NaN	

Advanced Classes (Total labeled in modeling data)

In [44]:

```
# Create data subset of advanced classes for analysis
adv_subset_all = preprocessed[preprocessed['classDifficulty_num'] == 2]
```

In [45]:

```
# Run diff_top_artists_and_songs on data set to get top artists and songs for analysis
adv_top_artists_all, adv_top_songs_all = diff_top_artists_and_songs(adv_subset_all, df)
```

In [46]:

```
# Preview artists
adv_top_artists_all[:10]
```

Out[46]:

	name	classCount	Class Count	id	class_per	followers	genres	popularity
							['alabama rap'	

	name	classCount	Class Count	id	class_per	followers	genres	popularity
184	Yelawolf	30	54	68DWke2VjdDmA75aJX5C57	0.56	1685398.0	['alabama rap', 'hip hop', 'pop rap', 'rap', '...	72.0
186	Knife Party	30	59	2DuJi13MWHjRHrqRUwk8vH	0.51	858221.0	['australian dance', 'brostep', 'complextro', ...	61.0
81	Ace Hood	54	118	31HjiqargV4NAw4GZqUale	0.46	1371482.0	['dirty south rap', 'gangster rap', 'hip hop',...	67.0
72	Rage Against The Machine	60	135	2d0hyoQ5ynDBnkvAbJKORj	0.44	4176893.0	['alternative metal', 'alternative rock', 'con...	75.0
112	Nine Inch Nails	43	99	0X380XXQSNBYuleKzav5UO	0.43	1766783.0	['alternative metal', 'alternative rock', 'cyb...	69.0
109	Darude	44	104	0LhHRmSd1EYM5QdNeNnCoQ	0.42	133345.0	['eurodance', 'europop', 'finnish edm']	62.0
130	Twista	40	98	6vbY3hOaCAhC7VjucswgdS	0.41	699514.0	['chicago rap', 'dirty south rap', 'gangster r...	71.0
123	Limp Bizkit	40	101	165ZgPILkK7bf5bDoFc6Sb	0.40	3931967.0	['alternative metal', 'funk metal', 'nu metal'...	76.0
157	Flux Pavilion	32	85	7muzHifhMdnfN1xncRLOqk	0.38	674503.0	['bass trap', 'brostep', 'classic dubstep', 'e...	66.0
108	Tujamo	45	119	26F2Hcdv4iKv9i9vlE5coT	0.38	5.0	[]	0.0

In [47]:

```
# Preview songs
adv_top_songs_all[:10]
```

Out[47]:

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popul
61	The Longest Yard	16	spotify:track:0Fi5GPyXr7c2vxdw9rJ8xF	16	1.00	Various Artists	NaN	
32	Football Fanatic	19	spotify:track:4DjkbfhdhfmzKNOUPtv9Js	19	1.00	Music Beyond	NaN	
29	Hustle Hard Remix - Album Version (Edited)	20	None	24	0.83	NaN	NaN	
56	Let's Go (feat. Yelawolf	17	spotify:track:19xSVTy7c5jBGmHHgVXNbg	22	0.77	Travis Barker	NaN	
57	Busta Rhymes & Lil Jon)	17	None	22	0.77	NaN	NaN	
279	Murda Something (feat. Waka Flocka Flame)	9	spotify:track:4awpwf3TeFWOtLiswRbKfr	12	0.75	A$AP Ferg	NaN	
51	Twista	17	None	23	0.74	NaN	NaN	
347	Raise Up (feat. Petey Pablo) [VIP]	8	spotify:track:4Y8oWpT44AVDGg5Ytq3OPP	11	0.73	ETC!ETC!	NaN	
274	WTF (feat. Amber Van Day) [Tujamo Remix]	9	spotify:track:31NDAHPCyqEIPKd3kUV6yA	13	0.69	HUGEL	NaN	

237	No Problem	9	spotify:track:0EgigrGFGb4PHaVNb7fgK7	13	0.69	NaN	['Lil Scrappy']	
	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popul

Advanced Classes (Model Results - 30 Min+ Intervals Classes)

In [48]:

```
# Create data subset of 30+ min intervals classes for analysis
adv_subset_int = preprocessed[(preprocessed['classDuration'] > -0.49) &
                              (preprocessed['classCategory_Intervals'] == 1)]
```

In [49]:

```
# Run diff_top_artists_and_songs on data set to get top artists and songs for analysis
adv_top_artists_int, adv_top_songs_int = diff_top_artists_and_songs(adv_subset_int, df)
```

In [50]:

```
# Preview artists
adv_top_artists_int[:10]
```

Out[50]:

	name	classCount	Class Count	id	class_per	followers	genres	popularity
151	I See MONSTAS	27	60	3yWCAtesP5BFtJnBbgfv8b	0.45	10439.0	[]	45.0
66	Chase & Status	46	121	3jNkaOXasoc7RsxdchvEVq	0.38	564006.0	['drum and bass', 'liquid funk']	65.0
76	Tujamo	42	119	26F2Hcdv4iKv9i9vlE5coT	0.35	5.0	[]	0.0
116	Sub Focus	32	98	0QaSiI5TLA4N7mcsdxShDO	0.33	282216.0	['drum and bass', 'house', 'liquid funk', 'uk ...	66.0
132	Keys N Krates	29	98	6c1pBXHYjFcGQQNO5MMsdd	0.30	181860.0	['bass trap', 'edm', 'electronic trap', 'livet...	57.0
17	John Newman	80	267	34v5MVKeQnIo0CWYMbbrPf	0.30	638402.0	[]	71.0
152	Otto Knows	26	90	5fahUm8t5c0GIdeTq0ZaG8	0.29	367948.0	['edm', 'electro house', 'pop dance', 'progres...	63.0
115	Matthew Koma	32	123	1mU61l2mcjEFraXZLpvVMo	0.26	99247.0	['pop edm']	65.0
71	Jack Ü	44	170	1HxJeLhIuegM3KgvPn8sTa	0.26	1050819.0	['edm', 'electro house', 'pop dance']	66.0
101	AlunaGeorge	35	134	2VAnyOxzJuSAj7XIuEOT38	0.26	409236.0	['electropop', 'house', 'pop', 'tropical house']	69.0

In [51]:

```
# Preview songs
adv_top_songs_int[:10]
```

Out[51]:

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popular
160	Would I Lie To You (Cash Cash Remix)	8	spotify:track:5ui8aAJN6W7dvE8hR2OhZS	11	0.73	Various Artists	NaN	
4	Habits (Stay High) (The Chainsmokers	19	spotify:track:7lxKUx67JrEoxjayHCb0Xr	28	0.68	Tove Lo	NaN	

	Extended ... peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	popular
155	Open Wide (feat. Big Sean)	8	spotify:track:64j3Bd62HTe0pclk8Aq9BE	12	0.67	NaN	['Calvin Harris', 'Big Sean']	
89	Uptown Funk (Will Sparks Remix) (feat. Bruno M...	10	spotify:track:5MpKzeXvOBFiZpQWV9iP5O	15	0.67	Mark Ronson	NaN	
55	Drop That Low (When I Dip) [Extended Mix]	11	None	17	0.65	NaN	NaN	N
217	Shape of You (Galantis Remix)	7	spotify:track:5H7CwzYZ60e7w69tX4ivQN	12	0.58	Ed Sheeran	NaN	
145	Blame (R3HAB Club Remix) (feat. John Newman)	8	spotify:track:0ZdJ0MBlyrwvuLNtv3cbKM	14	0.57	Calvin Harris	NaN	
102	Heads Will Roll (A-Trak Remix)	9	spotify:track:2idmlkd8oUaQvYEtINpLBX	16	0.56	Yeah Yeah Yeahs	NaN	
1	Messiah (Dirty South Remix)	23	spotify:track:4ocOSkLYp68DFrmPuRMvyA	44	0.52	I See MONSTAS	NaN	
100	Plain Jane	9	spotify:track:4dVpf9jZjcORqGTLUaeYj9	18	0.50	NaN	['A$AP Ferg']	

Intermediate Classes (Total labeled in modeling data)

In [52]:

```
# Create data subset of intermediate classes for analysis
inter_subset_all = preprocessed[preprocessed['classDifficulty_num'] == 1]
```

In [53]:

```
# Run diff_top_artists_and_songs on data set to get top artists and songs for analysis
inter_top_artists_all, inter_top_songs_all = diff_top_artists_and_songs(inter_subset_all,
df)
```

In [54]:

```
# Preview artists
inter_top_artists_all[:10]
```

Out[54]:

	name	classCount	Class Count	id	class_per	followers	genres	popularity
238	Take That	38	66	1XgFuvRd7r5g0h844A5ZUQ	0.58	979482.0	['boy band', 'dance pop', 'europop']	68.0
229	Whitesnake	39	67	3UbyYnvNIT5DFXU4WgiGpP	0.58	1963119.0	['album rock', 'british blues', 'classic rock'...	69.0
253	Foreigner	36	66	6IRouO5mvvfcyxtPDKMYFN	0.55	1948134.0	['album rock', 'classic rock', 'hard rock', 'h...	73.0
47	Little Mix	114	226	3e7awlrlDSwF3iM0WBjGMp	0.50	8958639.0	['dance pop', 'girl group', 'pop', 'post-teen ...	84.0
112	Bryan Adams	64	129	3Z02hBLubJxuFJfhacLSDc	0.50	2068570.0	['album rock', 'canadian pop', 'canadian singe...	79.0
48	Backstreet Boys	113	229	5rSXSAkZ67PYJSvpUpkOr7	0.49	3795212.0	['boy band', 'dance pop']	79.0

	name	classCount	Class Count	id	class_per	followers	genres	popularity
213	Tate	40	82	2nbTrndWzGHdI3jDqCmTrj	0.49	1562.0	[]	28.0
137	The Who	54	113	67ea9eGLXYMsO2eYQRui3w	0.48	3907800.0	['album rock', 'art rock', 'blues rock', 'brit...	74.0
165	Fifth Harmony	47	97	1l8Fu6IkuTP0U5QetQJ5Xt	0.48	12505767.0	['dance pop', 'electropop', 'girl group', 'pop...	75.0
176	Stormzy	45	98	2SrSdSvpminqmStGELCSNd	0.46	2232326.0	['grime', 'uk hip hop']	78.0

In [55]:

```
# Preview songs
inter_top_songs_all[:10]
```

Out[55]:

	peloton_song_name	classCount	song_id	Class Count	class_per	artist	artists	po
172	Black Magic	16	spotify:track:4cJhBmeJ7KiBeuy7oxRnZ3	17	0.94	NaN	['Slayer']	
384	La Isla Bonita	11	spotify:track:6r8k1vznHrzlEKYxL4dZEe	13	0.85	NaN	['Madonna']	
475	There You Go	10	spotify:track:3A3kg1SK2gpHNPZJmXpfTv	12	0.83	NaN	['Johnny Cash']	
442	Eternal Flame	11	spotify:track:5g3ZD7PmrEQlQZKDW91yGG	14	0.79	NaN	['The Bangles']	
374	Knock on Wood (7" Edit)	11	spotify:track:7vMRj40Rw8PPgZiDcbyUdj	14	0.79	Amii Stewart	NaN	
430	Locked out of Heaven	11	spotify:track:3w3y8KPTfNeOKPiqUTakBh	15	0.73	NaN	['Bruno Mars']	
365	One Kiss (R3HAB Extended Remix)	12	None	17	0.71	NaN	NaN	
449	Love Come Down	11	spotify:track:17EVu5b0l5uo2gErSfqEt1	16	0.69	NaN	['Evelyn "Champagne" King']	
436	Somebody To Love	11	spotify:track:3rLlv187BhjyweFe89SgLn	16	0.69	NaN	['Justin Bieber']	
426	Heartbreaker (Remastered)	11	spotify:track:49cjn3MnwvoNe31RqvWMK5	16	0.69	Various Artists	NaN	

Plotting

In [56]:

```
# Create list of dfs
df_by_difficulty = [beg_top_artists_all, beg_top_artists_lowim,
                    inter_top_artists_all,
                    adv_top_artists_all, adv_top_artists_int]

df_names = ['All Beginner', 'Beginner Low Impact',
            'All Intermediate',
            'All Advanced', 'Advanced Intervals']

# Name is a column in all df's rename column, so df name can be obtained
[single_df.rename(columns={"name": "artist"}, inplace = True) for single_df in df_by_dif
ficulty]
```

Out[56]:

```
[None, None, None, None, None]
```

In [57]:

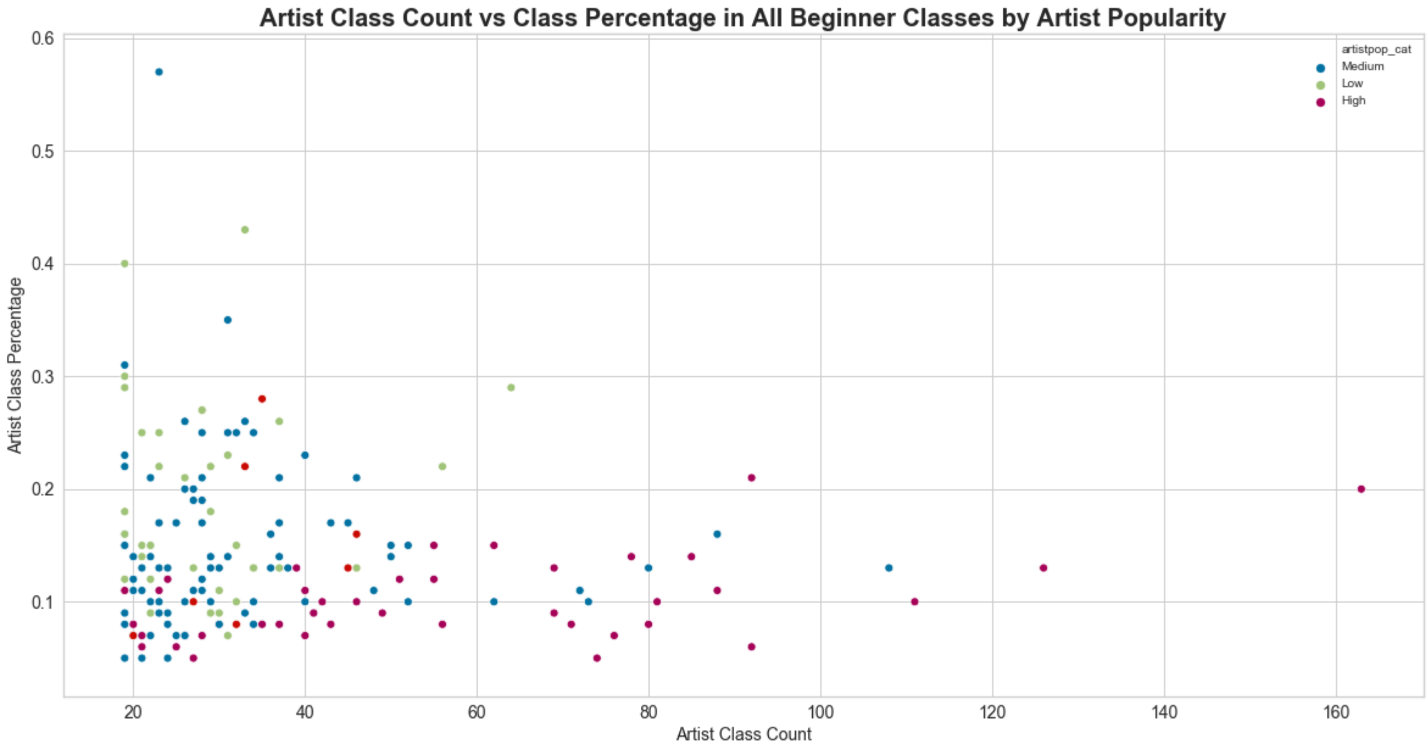
```
for i, single_df in enumerate(df_by_difficulty):
```

```
for i, single_df in enumerate(df_by_difficulty):
    print(df_names[i])
```

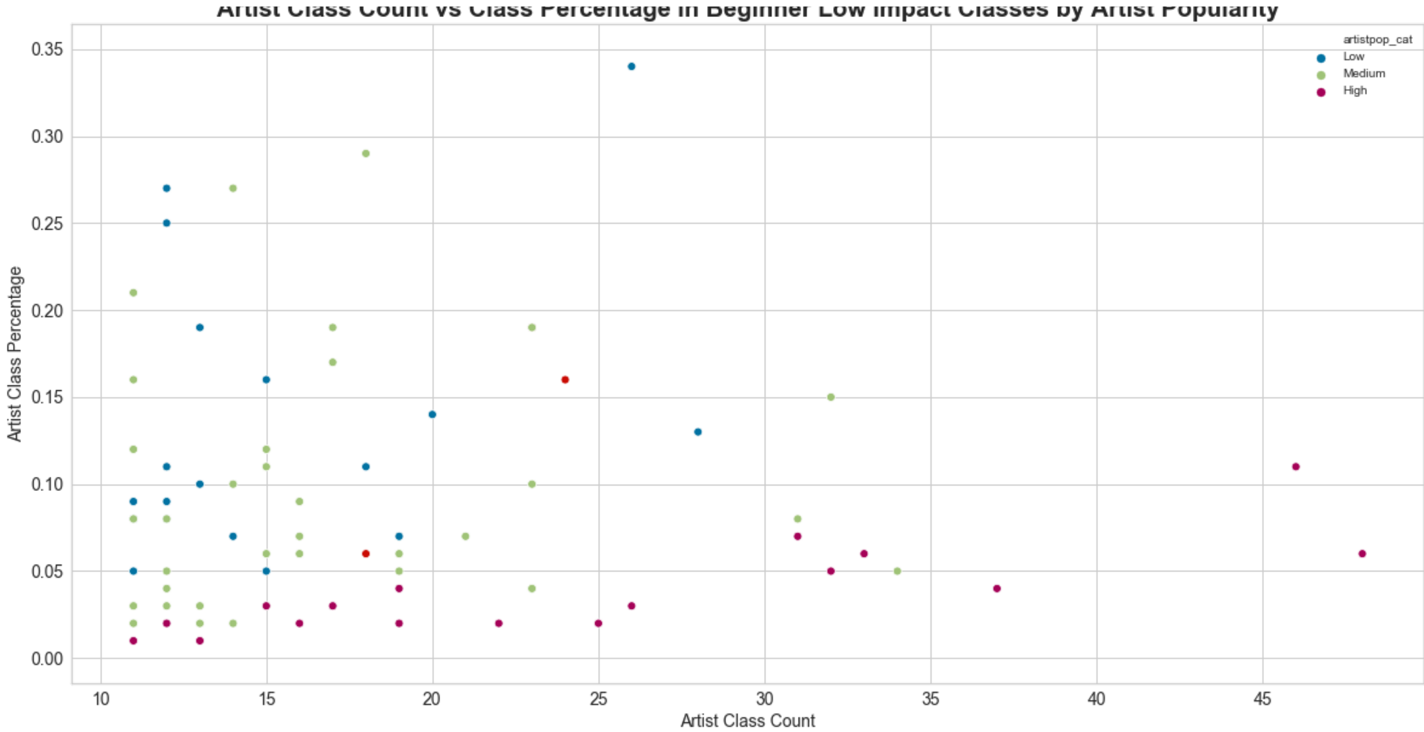
```
All Beginner
Beginner Low Impact
All Intermediate
All Advanced
Advanced Intervals
```

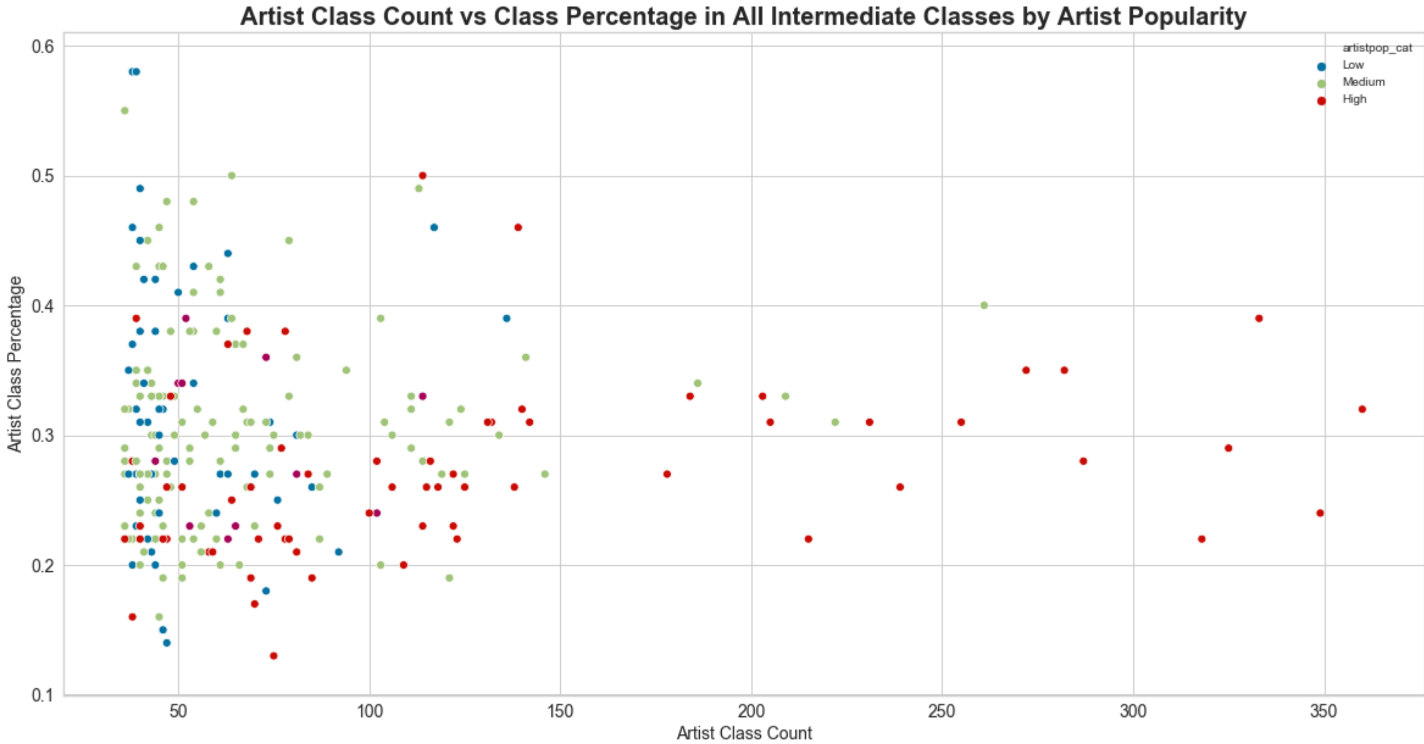
In [58]:

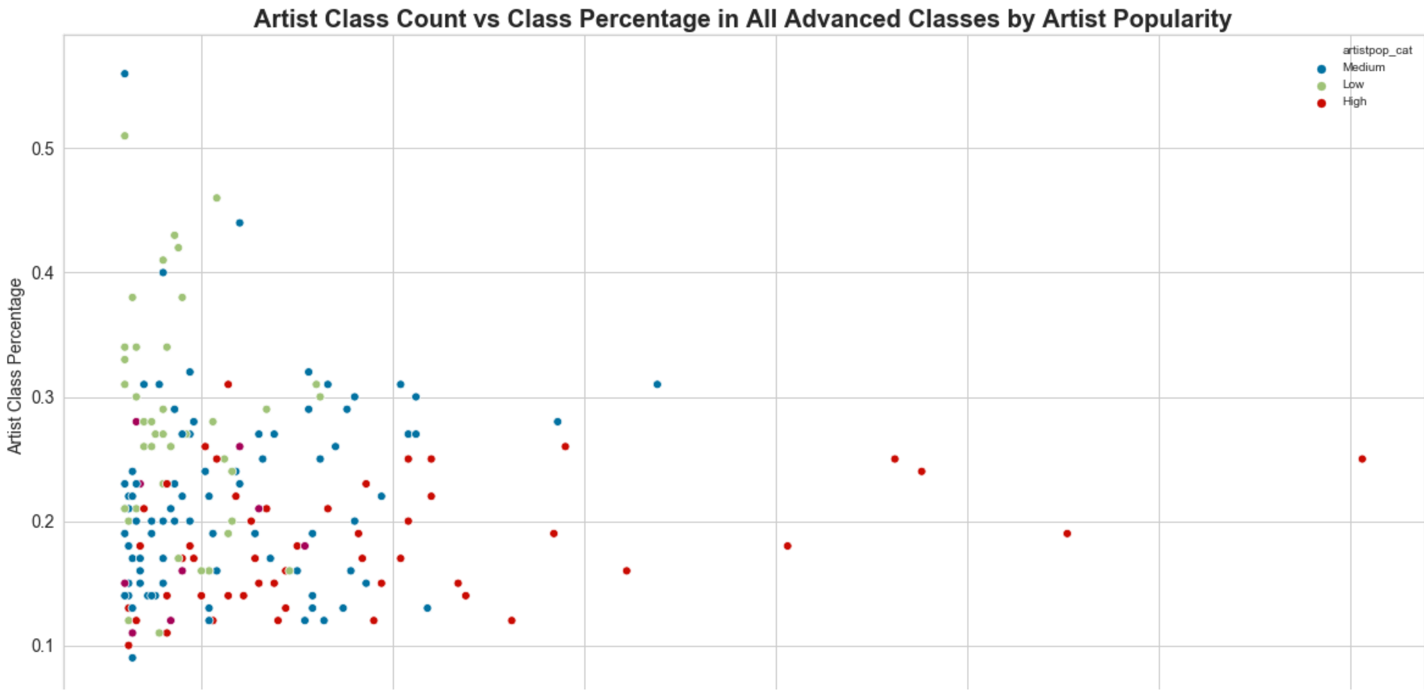
```
# Plot Artist Class Count versus Class Percentage for each df by difficulty with color co
ding
# based on whether the artist has Low, Medium, or High popularity based on the Spotify in
dex.
for i, single_df in enumerate(df_by_difficulty):
    # Create new series in df
    single_df ['artistpop_cat'] = ""

    # Label Artist Popularity Low Medium High
    low_med_high_labels(series = single_df['popularity'],
                        new_column_name = 'artistpop_cat',
                        df = single_df)

    # Plot new categories on classCount versus class %
    plot_scatter(single_df, x="classCount", y="class_per",
             hue='artistpop_cat',  by='Artist Popularity', legend='full',
             artist_or_song='Artist', subset_title=(f'{df_names[i]} Classes'))

# Across all
## As expected, Low/Medium popularity artists are lower class count and have higher class
percentages
## High popularity have range across difficulties of counts and total percentages; but ca
n see these are
## correlated to the original breakdown of 25-50-25 across the classed
# Beginner
## Beginner classes make up 5-22% of total classes high popularity artists are featured i
n
# Intermediate
## Intermediate classes make up 15-40% of total classes high popularity artists are featu
red in
## -- have greater range
## Intermediate is bulk of classes so this makes sense
# Advanced
## Advanced classes make up 10-25% of total classes high popularity artists are featured
in
```

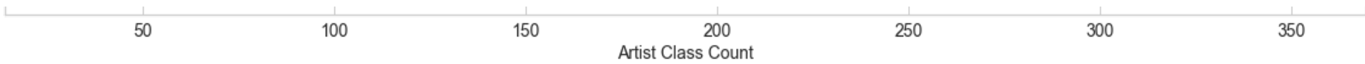


Artist Class Count vs Class Percentage in Beginner Low Impact Classes by Artist Popularity


Artist Class Count vs Class Percentage in Beginner Low Impact Classes by Artist Popularity
artistpop_cat
Low
Medium
High
Artist Class Percentage
Artist Class Count
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
10
15
20
25
30
35
40
45


Artist Class Count vs Class Percentage in All Intermediate Classes by Artist Popularity
artistpop_cat
Low
Medium
High
Artist Class Percentage
Artist Class Count
0.1
0.2
0.3
0.4
0.5
0.6
50
100
150
200
250
300
350


Artist Class Count vs Class Percentage in All Advanced Classes by Artist Popularity
artistpop_cat
Medium
Low
High
Artist Class Percentage
0.1
0.2
0.3
0.4
0.5



Data Analysis Learnings

Learnings from tree and analysis above

1. **If class is Low Impact, it is Beginner.**

1. **If a class is Intervals, difficulty will depend on the duration, instructor and/or the music.**

1. **If the class is < 30 mins it will more likely be Beginner/Intermediate, but could be Intermediate or Advanced if high danceability (> 1.811) and a Robin class.**
    - **Next layer towards Beginner class is if Christine is teaching a non-Intervals class < 30 mins.**
        - **Her playlists are more instrumental, so the decision below her is instrumentalness <=5.184. If true, Beginner.**

1. **If classDuration is 30 mins or more will lean towards Advanced. Tree goes:**
    - **If class is > 30 mins, then still intermediate**
    - **Is the class an Intervals class? If yes, it will be Advanced**
        - **Next driver is if instructor is Robin? If yes, then next driver is it the instructor is Olivia? Then Cody, then Tunde.**
        - **If not Robin, music danceability <= 0.765 (if TRUE, more likely Advanced -- but Advanced either way)**
    - **If class is not Intervals, then if classCategory is Music, the class difficulty will lean Intermediate (rare that these classes are Advanced)**

Conclusion

Recommendations On Model Utilization:

In my initial search for class or instructor recommendations for first time riders, I found that there was a lack of information available. There were "top lists" of instructors with general recommendations, but all of them were different. This leads me to believe these are completely based on the publication and the writer's opinions. **There was a lack of any data backed recommendations; thus I confirmed there is an opportunity in the market for a model like this to be implemented.**

As this is an early iteration, this model could be made available using a simple service like Streamlit to test user engagement and satisfaction.

Future Work

Updated Peloton Class Data Ongoing via API

This model was created utilizing Peloton data from March 2021. As Peloton refreshes their classes a regular basis, it would be a priority to utilize the Peloton API in supporting this model ongoing. The evolution of the library may also bring revisitation to the model as the make up and audience of Peloton also evolves.

Peloton Playlist Data Formatting and Accuracy

There is also opportunity in how the Peloton data is formatted, which impacts the gathering of artist and track related data. In the Peloton class data, the song and artist list for a single class' playlist is separate, which made it difficult to find the exact song. Zipping the two features together did not work as multiple artists can be listed for the same song (see song: "California Girls" artist: Katy Perry, Snoop Dogg). I made the assumption that popular songs users would recognize would be used on playlists; thus searching for the song title on Spotify will return the correct song data. This same assumption applies to searching and obtaining the artist level features. The next iteration should have songs paired with their artists for the most accuracy Spotify search results.

Potential Expanded Approach

Prior notebooks outside of the "final notebooks" folder document other paths I considered in creating a base dataset, especially in regards to the music related data. Editing in this approach could potentially impact the feature importance of music and playlist is for various levels of difficulty.

Model Improvement and Beta Testing

With each of the above considered, there is exponential opportunity to advance this work and model performance to the point where it can be implemented at Peloton and beta tested with users. As Peloton is very much carried by a positive customer experience overall, the first ride on the bike is a key moment in the users relationship with the brand and fitness overall.