Final Project - Forecasting Sporadic Demand with Recurring Neural Networks (RNN)

Description

The objective of this project is to assess the ability of an RNN to forecast demand based on historical data. There are many studies on how RNNs are well suited for forecasting sales. However, this project will focus on demand profiles of products that exhibit sporatic demand patterns. A brief discussion of sporatic demand is provided below.

At a high level, product families to be forecast will be based on high revenue generating families and their associated demand patterns. The data set will consist of 45 months (Jan 2020 - Sept 2023) of demand history for the selected product families. The first three years will be used for training and testing and the last year used for model evaluation. Given the propritary nature of the data some specifics of the data will be ommitted.

Motivation / Problem OVerview

The ability to accurately forecast future demand is critical to any business. Forecasting future demand in mixed model businesses requires different forecasting models depending on the business model’s historical demand profiles. Using a single model across all business units or product lines can lead to excellent results in some areas and very poor results in others. The ability to classify the various demand profiles allows for application of the most appropriate forecasting model and improved forecasting accuracy.

Additionally, demand inevitably changes over time, therefore identifying and understanding how demand is changing is critical to accurate forecasting. The ability to identify profile changes and reclassify the demand patterns allows for a pivot to a more appropriate forecasting model based on current demand attributes. Forecasts have impacts across the business. Inventory levels, long and short-term capacity, strategic plans, headcount, and profit are all intimately linked to understanding future demand.

When future demand is overstated the business will be left with slow moving or obsolete inventory, excess capacity and human resources leading to negative impacts on profits and other financial performance metrics. When future demand is understated the business will not be prepared from a materials, capacity or resource position to meet the actual demand. The end result being additional costs in expediting material and unplanned overtime costs. Again, eroding profits and negatively impacting financial performance. Additionally, not meeting customer demand often leads to lost business as customers find alternate suppliers.

In the end, accurate forecasting is crucial. And critical to accurate forecasting is correct demand pattern classification so that the optimal forecasting model is deployed.

Recurring Neural Networks

Model Architecture - Long-Short Term Memory (LSTM)

The LSTM model is built around the recurring neural networks (RNN) architecture. RNNs are neural networks that are capable of analyzing sequential or time series data. The LSTM is based on a network of gates and feedback loops between nodes and layers. These gates and feedback loops result in the ability of the network to maintain information over time, where the time frames can vary.

We will use the LSTM model to analyze historical demand. The LSTM's ability to maintain both long and shorterm memory will enable it to generate sequential forecasts based on it past experience. At its core, the LSTM is made up of memory blocks who's connections are linked between the layers. In very basic terms, sequences of inputs a passed into the RNN, the RNN though its memory blocks attempts to learn the behavior of the sequence. It then takes what it learned from the sequence and generates a sequence prediction.

The core of the LSTM memory blocks are the input, output and forget gates. These gates employ sigmoid activation to generate a state for each block. The forget gate's primary function is to determine what to keep and what to throw away based on the sequential input. The output gate defines an output based on the input from the input and memory gates. As a sequence is proccesed the block establishes a state, the linkages of the blocks between layers generates the model. Based on what the model has retained in its collective memory a predictive output series is produced.

This project will compare two LSTM model architectures, based on Root Mean Square Error (RMSE), to see which architecture provides the better performance. The specific model architectures are defined in the Model section.

EDA / Data Summary

Sporadic Demand

Since the demand pattern detailed analysis is beyond the primary scope of the project and was only used as a selection tool for the specific product families to evaluate, an abbreviated explanation of the EDA and demand pattern analysis is provided.

Traditional forecasting models and evaluation metrics break down when past demand is not “smooth”. Sporatic demand is when past demand is intermittent (many periods or consecutive periods with zero demand) or erratic (demand levels with large variability). When attempting to forecast sporadic demand traditional models break down. They tend to bias towards the zero periods or demand level extremes.

In 1972, J. Croston developed a more suitable forecasting model to accommodate intermittent demand. As part of the model development, Croston defined a new classification scheme for demand patterns.

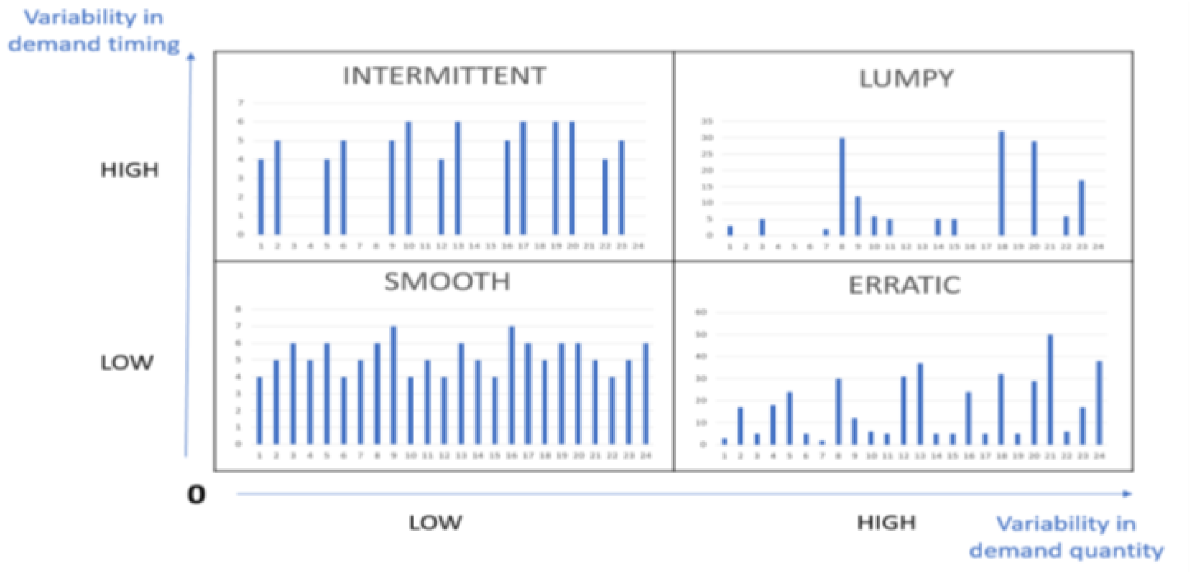
The classification model is comprised of two attributes. First is the Average Demand Interval (ADI), which is the average of demand over total periods with demand. This attribute provides the average demand interval in periods, when there is actual demand.

The second attribute, the Coefficient of Variance Squared (CV2) captures the variability in the level of demand. This is the standard deviation (σ) of the demand divided by the mean (m) of the demand squared. Again, the population is for the periods with actual demand.

Cronston identified four distinct demand patterns based on the ADI-CV2 pair. The patterns are Erratic, Intermittent, Lumpy and Smooth. See Image 1, provided by FREPPLE, for a graphical representation. The four patterns are characterized as follows:

- Erratic – Regular Demand in Time with High Variability Demand Quantities
- Intermittent - Irregular Demand in Time with Low Variability Demand Quantities
- Lumpy - Irregular Demand in Time with High Variability Demand Quantities
- Smooth - Regular Demand in Time and Demand Quantities

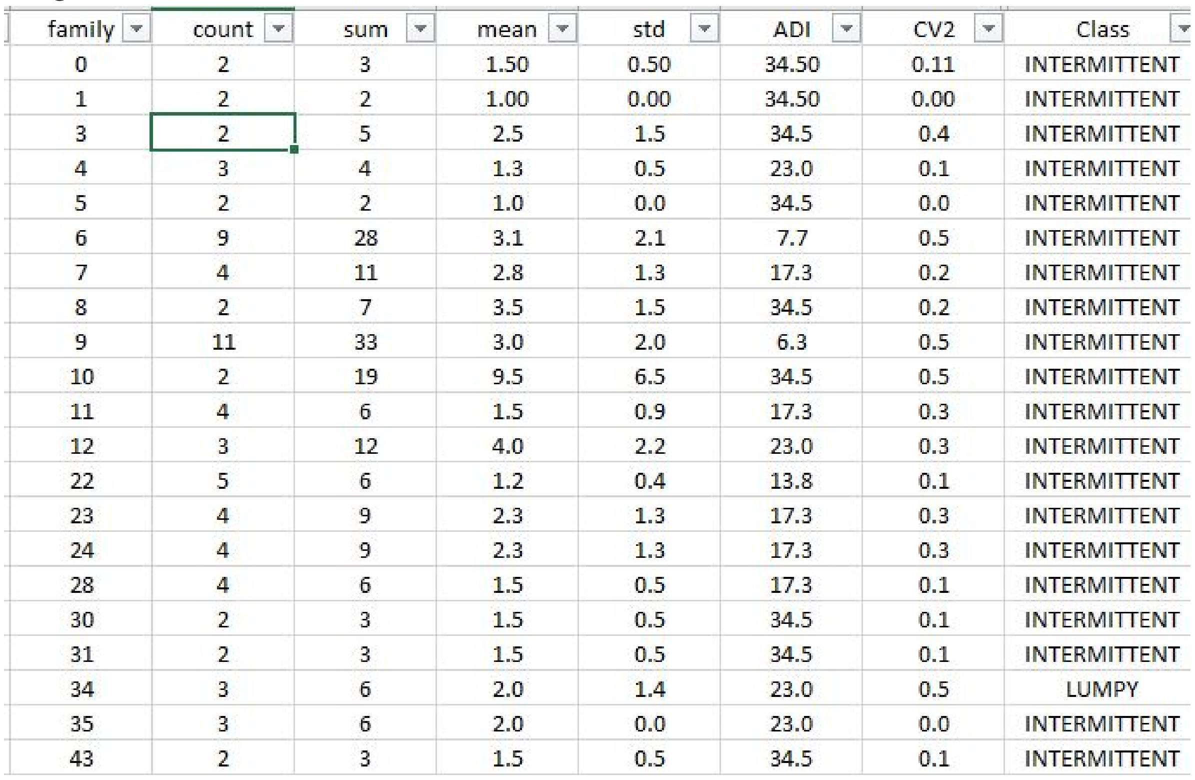
Image 1



Reprinted from "Demand Classification: Why Forecastability Matters," by FREPPLE). 2021 (https://frepple.com/blog/demand-classification/). Copyright 2021 by FREPPLE.

For simplicity and expediancy sake, the EDA and processsing of the demand pattern data was performed in Excell. The intial data set consisted of 588 products whose demand history from 2017 to 2021 was analyzed. The definitions for ADI and CV2 were applied via a simple spreadsheet. Image 2 is a snapshot of the spreadsheet used to generate the demand profile distribution showed in Image 4.

Image 2



family	count	sum	mean	std	ADI	CV2	Class
0	2	3	1.50	0.50	34.50	0.11	INTERMITTENT
1	2	2	1.00	0.00	34.50	0.00	INTERMITTENT
3	2	5	2.5	1.5	34.5	0.4	INTERMITTENT
4	3	4	1.3	0.5	23.0	0.1	INTERMITTENT
5	2	2	1.0	0.0	34.5	0.0	INTERMITTENT
6	9	28	3.1	2.1	7.7	0.5	INTERMITTENT
7	4	11	2.8	1.3	17.3	0.2	INTERMITTENT
8	2	7	3.5	1.5	34.5	0.2	INTERMITTENT
9	11	33	3.0	2.0	6.3	0.5	INTERMITTENT
10	2	19	9.5	6.5	34.5	0.5	INTERMITTENT
11	4	6	1.5	0.9	17.3	0.3	INTERMITTENT
12	3	12	4.0	2.2	23.0	0.3	INTERMITTENT
22	5	6	1.2	0.4	13.8	0.1	INTERMITTENT
23	4	9	2.3	1.3	17.3	0.3	INTERMITTENT
24	4	9	2.3	1.3	17.3	0.3	INTERMITTENT
28	4	6	1.5	0.5	17.3	0.1	INTERMITTENT
30	2	3	1.5	0.5	34.5	0.1	INTERMITTENT
31	2	3	1.5	0.5	34.5	0.1	INTERMITTENT
34	3	6	2.0	1.4	23.0	0.5	LUMPY
35	3	6	2.0	0.0	23.0	0.0	INTERMITTENT
43	2	3	1.5	0.5	34.5	0.1	INTERMITTENT

Image 3 defines the category mapping for theCronston demand pattern model.

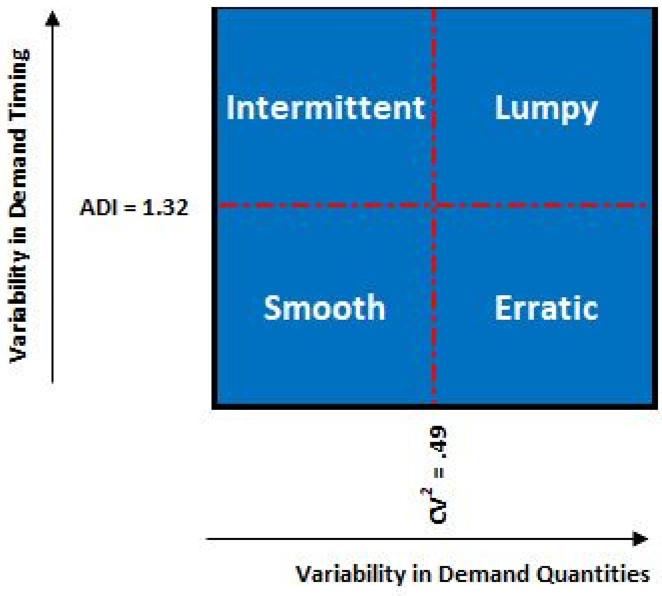

Variability in Demand Timing
ADI = 1.32
Intermittent
Lumpy
Smooth
Erratic
$CV^2 = .49$
Variability in Demand Quantities

Image 3

Image 4 shows the results of the demand pattern analysis. Image 2 confirms that the demand patterns of the business unit (Business Unit 1) meets the criterial of sporatic demand.

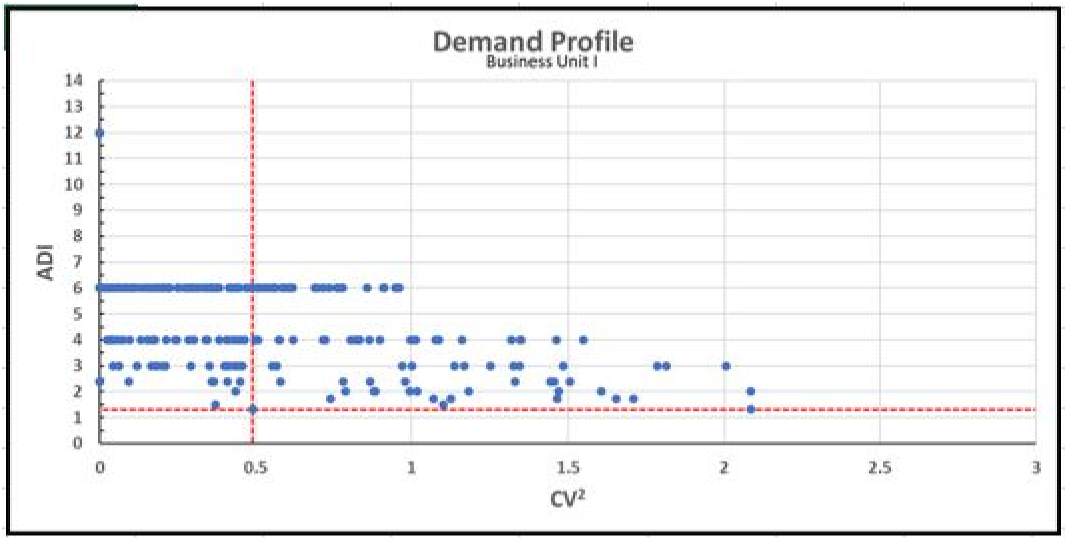


Image 4

With the nature of the demand patterns identified and confirmed, selection of the specific product familes needs to be determined. The determination of the specific products is based on the revenue generated for each of the product families. It was determined that 49 families generated ~90% of the revenue. The demand history of these 49 families will be used in the forecasting model.

In [1]:
```
#!pip
```

In [1]:
```
#Set Page Width to 100%
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [3]:
```
#Load Required Resources
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import math
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

import tensorflow as tf
from tensorflow.keras import models
from tensorflow.python.keras.layers import LSTM, Dense
from tensorflow.python.keras.models import Sequential
from keras.optimizers import Adam
from tensorflow.keras.layers import*
```

In [4]:
```
## Import Data
demand_df = pd.read_csv("demand.csv")
demand_pattern_df = pd.read_csv("demand_patterns.csv ")

print(demand_df.head(), '\n')
print(demand_df.info(), '\n')
print('Train Shape: ', demand_df.shape, '\n')
print(demand_pattern_df.head(), '\n')
print(demand_pattern_df.info())
print('Test Shape: ', demand_pattern_df.shape, '\n')
```

```
   family_id fabricated  hardware  business_unit_id  period  demand
0          1          Y         1                 1       1     355
1          1          Y         1                 1       2     205
2          1          Y         1                 1       3    1303
3          1          Y         1                 1       4     202
4          1          Y         1                 1       5     303

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2205 entries, 0 to 2204
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   family_id         2205 non-null   int64 
 1   fabricated        2205 non-null   object
 2   hardware          2205 non-null   int64 
 3   business_unit_id  2205 non-null   int64 
 4   period            2205 non-null   int64 
 5   demand            2205 non-null   int64 
dtypes: int64(5), object(1)
memory usage: 103.5+ KB
None

Train Shape:  (2205, 6)

   family_id  count    sum    mean     std   ADI   CV2    class
0          1     44  13394  304.41  296.55  1.57  0.95    LUMPY
1          2     69  10831  156.97  141.68  1.00  0.81  ERRATIC
2          3     45   1324   29.42   31.88  1.53  1.17    LUMPY
3          4     66   4589   69.53   87.69  1.05  1.59  ERRATIC
4          5     19    499   26.26   30.92  3.63  1.39    LUMPY

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   family_id  49 non-null     int64  
 1   count      49 non-null     int64  
 2   sum        49 non-null     int64  
 3   mean       49 non-null     float64
 4   std        49 non-null     float64
 5   ADI        49 non-null     float64
 6   CV2        49 non-null     float64
 7   class      49 non-null     object 
dtypes: float64(4), int64(3), object(1)
memory usage: 3.2+ KB
None
Test Shape:  (49, 8)
```

Model EDA

EDA will be performed as follows:

```
1. Extract data and preprocess in Excell
2. Remove Unnecessary Columns
3. Check for Outliers
3. Verify Family and Period Count
4. Family Pattern Distribution
5. Pattern Time Series Examples
```

Extract and Preprocess

The data was extracted directly from the Enterprise Resource System (ERP). Due to the extraction method and format of the data Excell is the prefered tool for data cleanse and structuring given its ease of use. The general steps for data cleanse and structuring are as follows.

```
1. Remove Unused Business Unit Lines (3 BUs)
2. Remove Service Lines (non-demand lines)
3. Remove Unnecessary Columns (18)
4. Remove Unused Product Family Lines (202 Families)
5. Remove Non-Fabricated Product Lines
6. Assign Product Family Ids
7. Pivot to Monthly View to Add 0 Quantity Months. (populate time
series with 0 quantity months)
8. Convert to Tidy format
```

Drop fabricated, hardware and business_unit_id

Drop count, sum, mean, std

In [5]:
```
# Drop fabricated, hardware  and business_unit_id

demand_df = demand_df.drop(['fabricated', 'hardware', 'business_unit_id'], axis=1)
demand_pattern_df = demand_pattern_df.drop(['count', 'sum', 'mean', 'std'], axis=1)

print(demand_df.head(), '\n')
print(demand_pattern_df.head())
```

```
   family_id  period  demand
0          1       1     355
1          1       2     205
2          1       3    1303
3          1       4     202
4          1       5     303

   family_id   ADI   CV2    class
0          1  1.57  0.95    LUMPY
1          2  1.00  0.81  ERRATIC
2          3  1.53  1.17    LUMPY
3          4  1.05  1.59  ERRATIC
4          5  3.63  1.39    LUMPY
```

NA Check

```
In [6]: #Check for NaNs and Nulls
        print('Demand NaNs / Null Count: ', demand_df.isna().sum(), '\n')
        print('Demand Patterns NaNs / Null Count: ', demand_pattern_df.isna().sum(), '\n')
```

```
Demand NaNs / Null Count:  family_id    0
period       0
demand       0
dtype: int64

Demand Patterns NaNs / Null Count:  family_id    0
ADI          0
CV2          0
class        0
dtype: int64
```

NA Check For data frames as expected.

Check for Outliers

```
In [7]: demand_df.describe()
```

Out[7]:

	family_id	period	demand
count	2205.000000	2205.000000	2205.000000
mean	25.000000	23.000000	62.194558
std	14.145344	12.990119	171.657773
min	1.000000	1.000000	-966.000000
25%	13.000000	12.000000	0.000000
50%	25.000000	23.000000	10.000000
75%	37.000000	34.000000	52.000000
max	49.000000	45.000000	4187.000000

Negative quantities have been identified in the demand column.

This is indicitive of returned product and should not be considered part of the demand.

Convert negative demand values to 0.

```
In [8]: demand_df[demand_df < 0] = 0
        demand_df.describe()
```

Out[8]:

	family_id	period	demand
count	2205.000000	2205.000000	2205.000000
mean	25.000000	23.000000	62.772336
std	14.145344	12.990119	170.186030
min	1.000000	1.000000	0.000000
25%	13.000000	12.000000	0.000000
50%	25.000000	23.000000	10.000000
75%	37.000000	34.000000	52.000000
max	49.000000	45.000000	4187.000000

Negative demand quantities have been removed

Verify Family and Period Count

In [9]:
```
print('Demand Unique Family ID Count: ', demand_df['family_id'].nunique(), '\n')
print('Demand Unique Period Count: ', demand_df['period'].nunique(), '\n')
print('Demand Pattern Unique Family ID Count: ', demand_pattern_df['family_id'].nuniqu
print('Demand Pattern Unique Class Count: ', demand_pattern_df['class'].nunique(), '\r
```

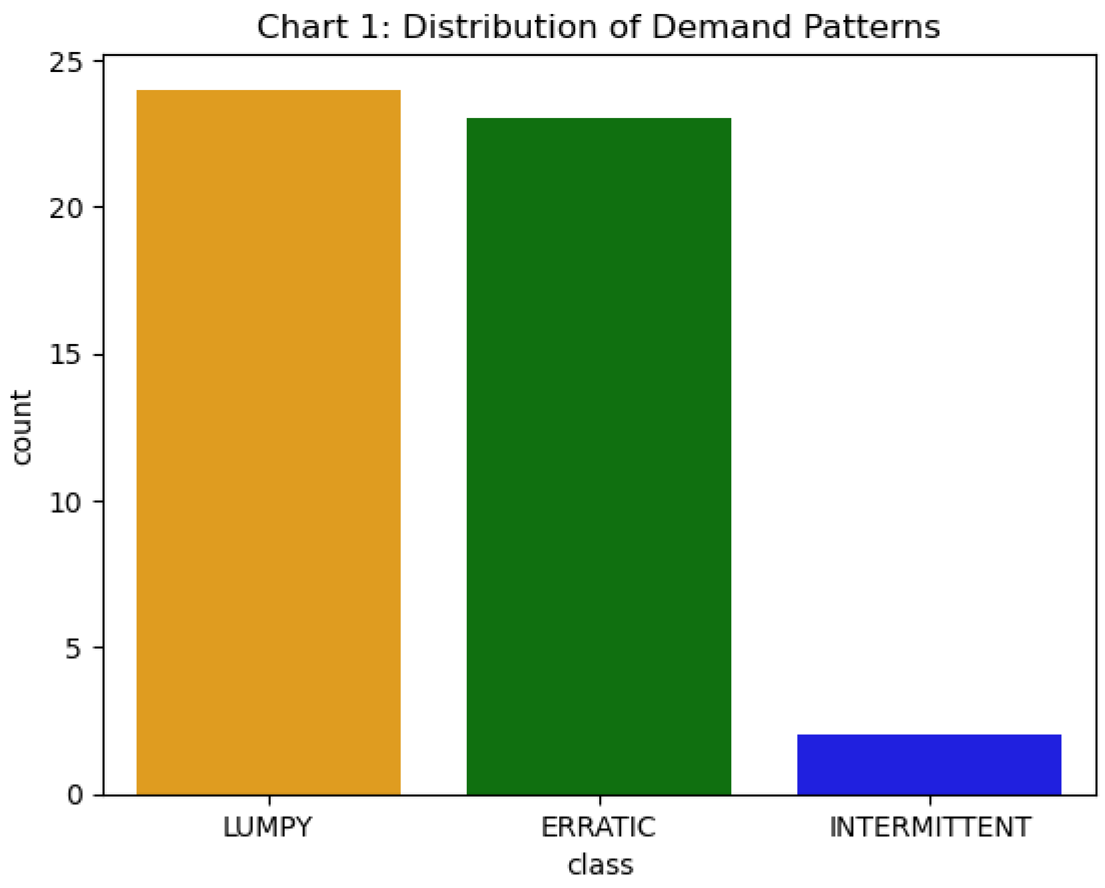
```
Demand Unique Family ID Count:  49

Demand Unique Period Count:  45

Demand Pattern Unique Family ID Count:  49

Demand Pattern Unique Class Count:  3
```

Unique Value Check For data frames as expected.

Family Pattern Distribution

As can be seen in Chart 1, all of the families are classified with a sporatic demand pattern. None of the familes being evaluated fall into the smooth pattern class.

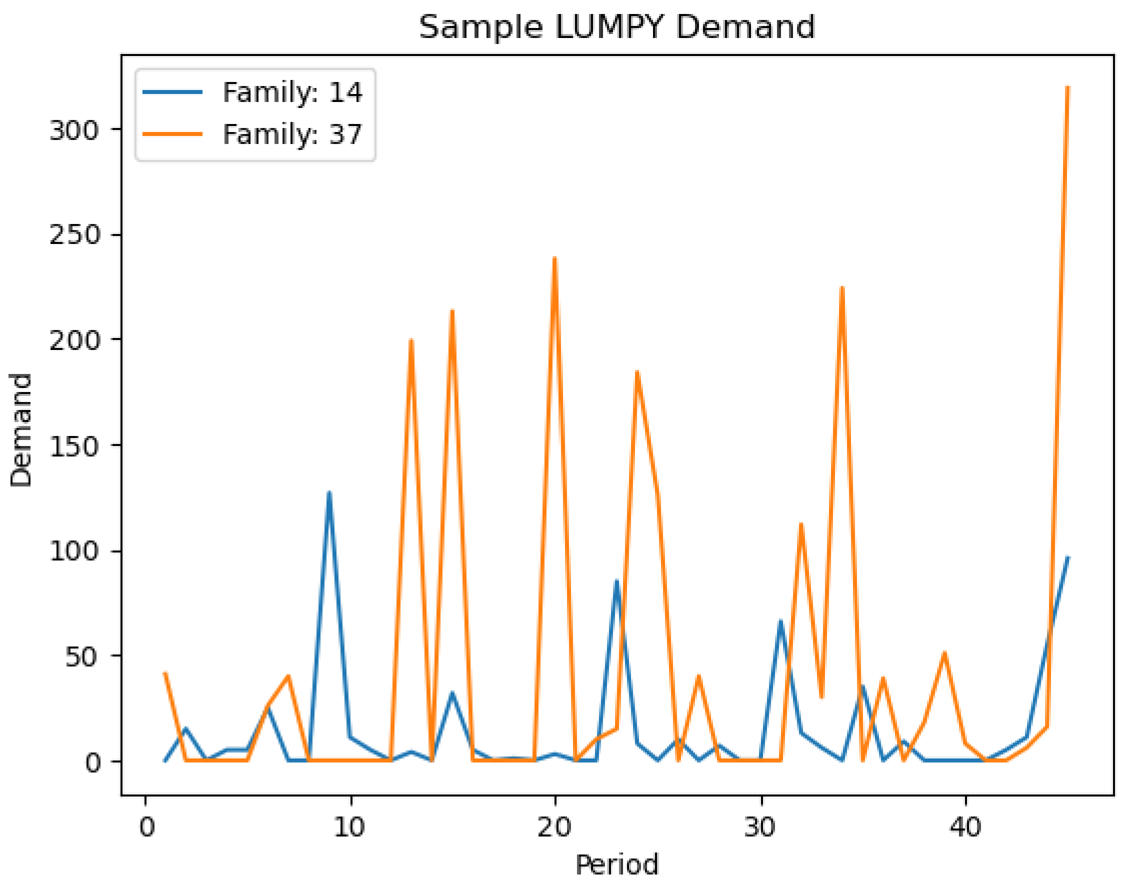
In [114…
```
plt.figure()
sns.countplot(data=demand_pattern_df, x='class', palette=['orange',"#008000", 'blue'])
plt.title('Chart 1: Distribution of Demand Patterns')
plt.show()
```

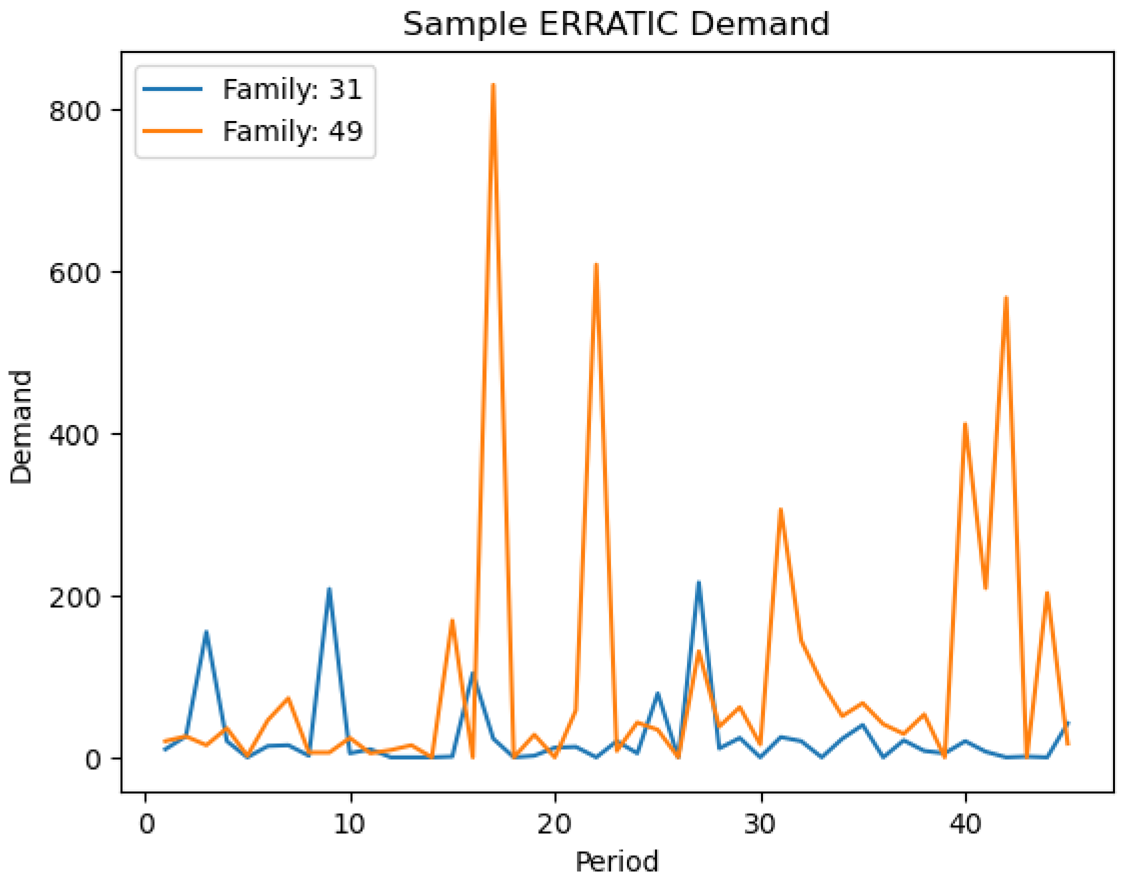


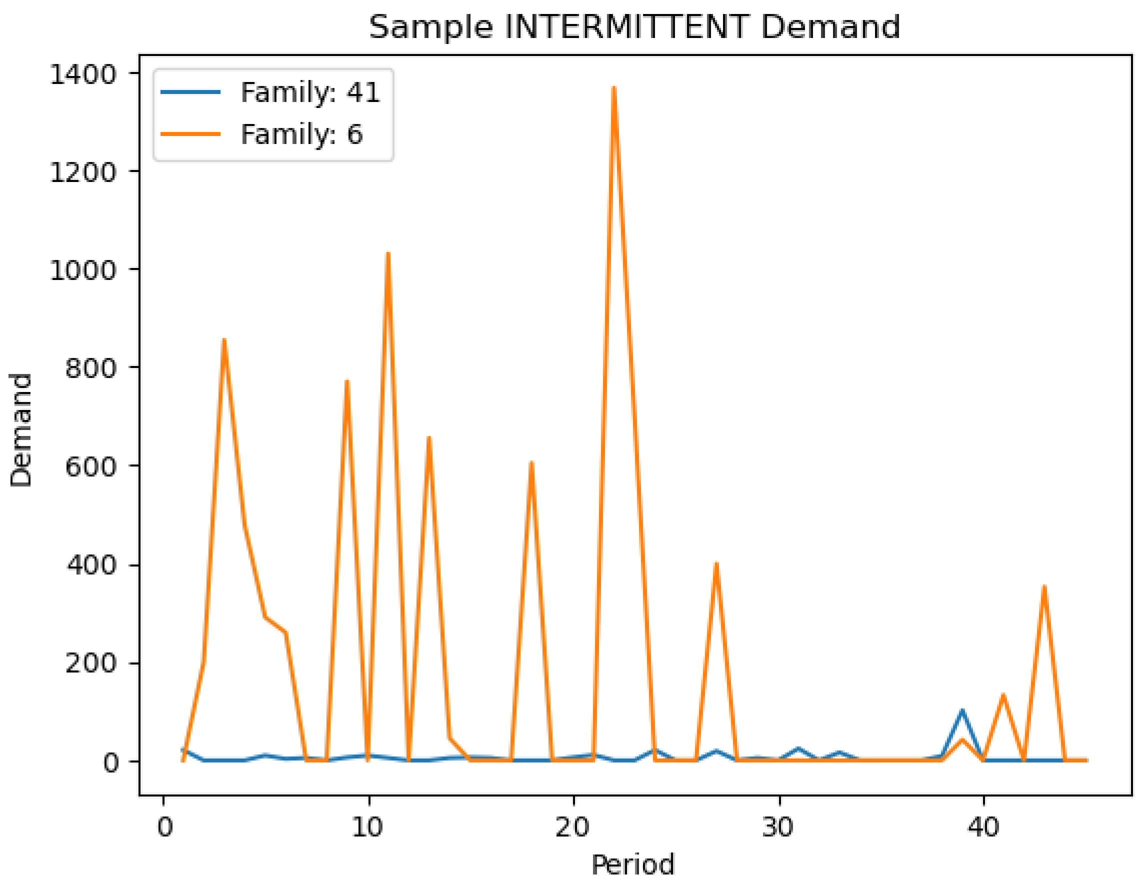
Sample Time Series Plots for Each Demand Pattern

In [123…

```
pattern = ['LUMPY', 'ERRATIC', 'INTERMITTENT']
sample_size = 2
chart_num = 1
for pat in pattern:

    lumpy_family_id = list(demand_pattern_df[demand_pattern_df['class'] == pat].sample
    for p in range(sample_size):
        lumpy_series1 = demand_df[demand_df['family_id'] == lumpy_family_id[p]]
        plt.plot(lumpy_series1['period'], lumpy_series1['demand'], label = 'Family: '
        plt.title ('Sample '+pat+' Demand')


    plt.xlabel('Period')
    plt.ylabel('Demand')
    plt.legend(loc = 'upper left')

    plt.show()
```


Sample LUMPY Demand
Family: 14
Family: 37
Demand
Period
0
50
100
150
200
250
300
0
10
20
30
40


Sample ERRATIC Demand
Family: 31
Family: 49
Demand
Period
0
200
400
600
800
0
10
20
30
40



The three charts above provide illustrations of the three resident demand patterns. Review of the charts shows that the associated demand series align with the pattern descriptions. The lumpy profiles exhibit highly varible demand levels and frequency. The erratic profiles show fairly constant demand frequency with high variablility in quantity. The intermittent profiles show relatively stable demand quanties with highly variable frequency.

EDA Summary

Through the EDA and data cleansing process it has been established that the demand patterns of the product familes are indeed sporatic per the Cronston defitions. It has also been verified that the product family time series data is complete with 45 periods containing demand in all periods, including zero demand. Sample time series plots also confirm that the data is in the proper order for the analysis.

Models

Model Architecture - Long-Short Term Memory (LSTM)

This project will compare two LTSM model architectures, based on RMSE, to see which architecture provides the better performance. The first model will be a basic LSTM model with a single LSTM layer (tanh activation), followed by a dense layer with linear activation. This is the most simplistic model to be used as a baseline.

The second model will layer three sequential LSTM layers, followed by an intermediate Dense layer, and then the final Dense layer for output. The thought process being building an environment where the most simple basic model can be compared to and much more complex model. It is expected that the increased comlexity of the second model will provide the RNN with an increased ability to forecast the sporatic demand patterns.

For each of the 49 product families, the models will be trained and then a prediction generated. The RMSE between the prediction and actuals will be calculated and analyzed. A comparison of the average RMSE across families will also be provided. This will provide insight into how well the models respond to the three different demand pattern classes.

Various epoch counts were reviewed with 100 being the sweet spot for observing the model differences, while maintaining a reasonable run time.

In [51]:
```
#Model Builing and Training helper functions

def get_family_demand(fam_id):
    demand_profile_family_df = demand_df[demand_df['family_id'] == fam_id]
    demand_profile_df = demand_profile_family_df.drop(['family_id','period'], axis=1)
    demand_profile_nparray = demand_profile_df.values  #convert to numpy array
    return demand_profile_nparray

#Based on the fact that demand pattern defintions explicitly address high variability
def scale_profile(demand_profile):

    scaler = MinMaxScaler()
    demand = scaler.fit_transform(demand_profile)

    return demand

def train_test_data(profile, train_percent = .75):

    train_set_size = int(train_percent * len(profile))
    test_set_size = len(profile) - train_set_size
    train, test = profile[0:train_set_size,:], profile[train_set_size:len(profile),:]

    return train, test

#tranform data to matrix format as input into RNN
def data_to_input_matrix(input_array, period_lag=1):
    lag_matrix = []
    period_matrix = []
    input_array_size = len(input_array)
    for i in range(input_array_size-period_lag-1):
        lag_matrix.append(input_array[i:(i+period_lag), 0])
        period_matrix.append(input_array[i + period_lag, 0])

    return np.array(lag_matrix), np.array(period_matrix)


def format_input(array_to_format):
    return (np.reshape(array_to_format, (array_to_format.shape[0], 1, array_to_format.

def build_model1(train_input, train_output, epoch_count, loss_function, optimizer_type
    dropout_rate = .2
    recurrent_dropout_rate = .2
    model = Sequential()
    model.add(LSTM(32, input_shape=(1, lag_periods)))
    model.add(Dense(1))
    model.compile(loss=loss_function, optimizer=optimizer_type)
    model.fit(train_input, train_output, epochs=epoch_count, batch_size=batch, verbose
    return model

def build_model2(train_input, train_output, epoch_count, loss_function, optimizer_type
    dropout_rate = .2
    recurrent_dropout_rate = .2
    model = Sequential()
    model.add(LSTM(32, return_sequences = True, input_shape=(1, lag_periods)))
    model.add(LSTM(32, return_sequences = True))
    model.add(LSTM(32, return_sequences = False))
    model.add(Dense(16))
    model.add(Dense(1))
    model.compile(loss=loss_function, optimizer=optimizer_type)
    model.fit(train_input, train_output, epochs=epoch_count, batch_size=batch, verbose
    return model

def unscale_profile(original_demand_profile, scaled_profile):
    scaler = MinMaxScaler()
    scaler.fit_transform(original_demand_profile)
    return scaler.inverse_transform(scaled_profile)

def write_list_to_csv(l1, l2, l3, filename):
    import csv
    rows = zip(l1, l2,l3)
    with open(filename, "w") as f:
        writer = csv.writer(f)
        for row in rows:
            writer.writerow(row)
```

In [52]:
```
#Initalize variables
lag_periods = 1
batch_size = 1
train_test_split = .65

lumpy_family = []
lumpy_train_rmse = []
lumpy_test_rmse = []

erratic_family = []
erratic_train_rmse = []
erratic_test_rmse = []

intermittent_family = []
intermittent_train_rmse = []
intermittent_test_rmse = []

#loop through all families, build models, getr RMSE
#model build and train routine based on Bowleen example
for i in demand_pattern_df.index:
    print(i)
    family_id = demand_pattern_df['family_id'][i]
    pattern_class = demand_pattern_df['class'][i]
    print(family_id)
    print(pattern_class )
    demand_profile = get_family_demand(family_id)

    scaled_demand_profile = scale_profile(demand_profile)

    train, test = train_test_data(scaled_demand_profile, train_test_split)

    trainX, trainY = data_to_input_matrix(train, lag_periods) #trainX, trainY
    testX, testY = data_to_input_matrix(test, lag_periods)  #testX, testY

    trainX = format_input(trainX) #trainX
    testX = format_input(testX) #testX

#    m1 = build_model1(trainX, trainY, 100, 'mean_squared_error', 'adam', lag_periods,
    m2 = build_model2(trainX, trainY, 100, 'mean_squared_error', 'adam', lag_periods,

    # make predictions
#     trainPredict = m1.predict(trainX)
#     testPredict = m1.predict(testX)

    trainPredict = m2.predict(trainX)
    testPredict = m2.predict(testX)

    # convert predictions to unscalled values
    trainPredict = unscale_profile(demand_profile, trainPredict)
    trainY = unscale_profile(demand_profile, [trainY])
    testPredict = unscale_profile(demand_profile, testPredict)
    testY = unscale_profile(demand_profile, [testY])

    # calculate root mean squared error
    trainRMSE = np.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
    print('Train RMSE: %.2f' % (trainRMSE))
    testScRMSE = np.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
    print('Test Score RMSE: %.2f' % (testScRMSE))

    if pattern_class == 'LUMPY':
        lumpy_family.append(family_id)
        lumpy_train_rmse.append(trainRMSE)
        lumpy_test_rmse.append(testScRMSE)

    elif pattern_class == 'ERRATIC':
        erratic_family.append(family_id)
        erratic_train_rmse.append(trainRMSE)
        erratic_test_rmse.append(testScRMSE)

    else:
        intermittent_family.append(family_id)
        intermittent_train_rmse.append(trainRMSE)
        intermittent_test_rmse.append(testScRMSE)

print(lumpy_family)
print(lumpy_train_rmse)
print(lumpy_test_rmse)
```

```
0
1
LUMPY
Train RMSE: 426.38
Test Score RMSE: 423.46
1
2
ERRATIC
Train RMSE: 123.85
Test Score RMSE: 190.13
2
3
LUMPY
Train RMSE: 49.03
Test Score RMSE: 37.56
3
4
ERRATIC
Train RMSE: 192.26
Test Score RMSE: 87.25
4
5
LUMPY
Train RMSE: 12.16
Test Score RMSE: 25.98
5
6
INTERMITTENT
Train RMSE: 373.91
Test Score RMSE: 290.09
6
7
ERRATIC
Train RMSE: 187.33
Test Score RMSE: 80.68
7
8
LUMPY
Train RMSE: 56.75
Test Score RMSE: 174.35
8
9
ERRATIC
Train RMSE: 242.01
Test Score RMSE: 231.56
9
10
LUMPY
Train RMSE: 8.41
Test Score RMSE: 28.99
10
11
LUMPY
Train RMSE: 74.08
Test Score RMSE: 39.15
11
12
ERRATIC
Train RMSE: 39.11
Test Score RMSE: 31.02
```

```
12
13
ERRATIC
Train RMSE: 128.31
Test Score RMSE: 138.83
13
14
LUMPY
Train RMSE: 26.55
Test Score RMSE: 22.14
14
15
LUMPY
Train RMSE: 15.63
Test Score RMSE: 12.53
15
16
ERRATIC
Train RMSE: 36.56
Test Score RMSE: 29.71
16
17
LUMPY
Train RMSE: 6.23
Test Score RMSE: 5.16
17
18
LUMPY
Train RMSE: 11.20
Test Score RMSE: 23.56
18
19
LUMPY
Train RMSE: 12.24
Test Score RMSE: 33.25
19
20
LUMPY
Train RMSE: 7.28
Test Score RMSE: 6.14
20
21
LUMPY
Train RMSE: 7.89
Test Score RMSE: 5.10
21
22
LUMPY
Train RMSE: 38.47
Test Score RMSE: 63.44
22
23
ERRATIC
Train RMSE: 105.72
Test Score RMSE: 175.42
23
24
ERRATIC
Train RMSE: 117.93
Test Score RMSE: 182.18
```

```
24
25
LUMPY
Train RMSE: 11.19
Test Score RMSE: 9.48
25
26
LUMPY
Train RMSE: 3.46
Test Score RMSE: 1.86
26
27
ERRATIC
Train RMSE: 102.36
Test Score RMSE: 115.72
27
28
ERRATIC
Train RMSE: 102.73
Test Score RMSE: 88.93
28
29
ERRATIC
Train RMSE: 63.13
Test Score RMSE: 46.97
29
30
ERRATIC
Train RMSE: 49.38
Test Score RMSE: 32.08
30
31
ERRATIC
Train RMSE: 60.00
Test Score RMSE: 26.36
31
32
LUMPY
Train RMSE: 17.39
Test Score RMSE: 29.78
32
33
ERRATIC
Train RMSE: 49.45
Test Score RMSE: 31.97
33
34
ERRATIC
Train RMSE: 48.32
Test Score RMSE: 23.68
34
35
ERRATIC
Train RMSE: 770.44
Test Score RMSE: 570.05
35
36
ERRATIC
Train RMSE: 195.37
Test Score RMSE: 216.53
```

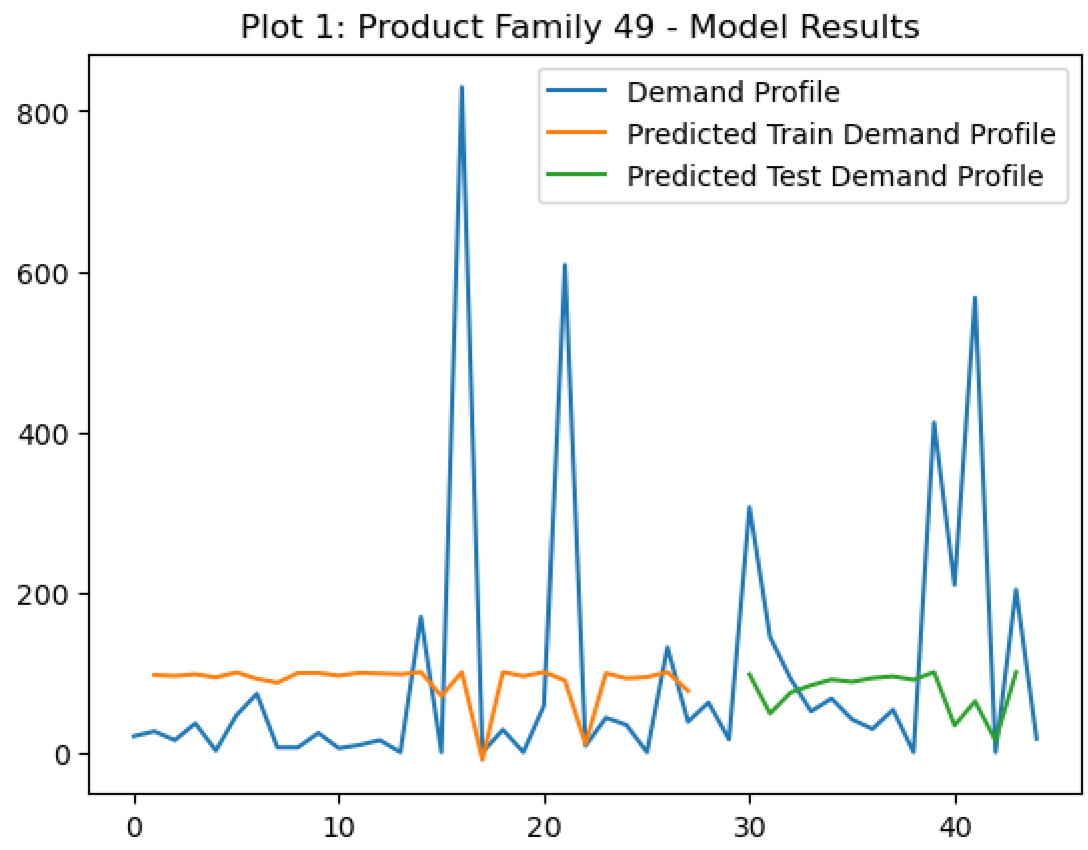
```
36
37
LUMPY
Train RMSE: 74.35
Test Score RMSE: 59.68
37
38
LUMPY
Train RMSE: 68.49
Test Score RMSE: 20.06
38
39
ERRATIC
Train RMSE: 50.18
Test Score RMSE: 44.48
39
40
ERRATIC
Train RMSE: 47.45
Test Score RMSE: 45.86
40
41
INTERMITTENT
Train RMSE: 5.48
Test Score RMSE: 31.71
41
42
ERRATIC
Train RMSE: 30.16
Test Score RMSE: 53.58
42
43
LUMPY
Train RMSE: 60.49
Test Score RMSE: 37.67
43
44
LUMPY
Train RMSE: 30.93
Test Score RMSE: 16.76
44
45
LUMPY
Train RMSE: 6.39
Test Score RMSE: 24.44
45
46
ERRATIC
Train RMSE: 81.90
Test Score RMSE: 75.45
46
47
LUMPY
Train RMSE: 14.75
Test Score RMSE: 15.03
47
48
LUMPY
Train RMSE: 37.87
Test Score RMSE: 41.25
```

```
48
49
ERRATIC
Train RMSE: 185.14
Test Score RMSE: 181.79
[1, 3, 5, 8, 10, 11, 14, 15, 17, 18, 19, 20, 21, 22, 25, 26, 32, 37, 38, 43, 44, 45,
47, 48]
[426.3767654484809, 49.02703839803618, 12.159117644063256, 56.74908511568625, 8.40696
8693949054, 74.07509028690764, 26.552090651252875, 15.632710243441931, 6.234948902348
876, 11.20317778539066, 12.239740941312652, 7.280426876128401, 7.888769055822997, 38.
46534152396223, 11.188116159597648, 3.463012498573144, 17.390181962319858, 74.3519254
056973, 68.49147760650634, 60.491110833298556, 30.93296822689112, 6.391866616919129,
14.74804612491077, 37.87142792373811]
[423.4574279785156, 37.555650665006794, 25.979866322273402, 174.34749450323704, 28.98
6654898185993, 39.14545545578223, 22.140988547171165, 12.530353201734615, 5.163889103
512966, 23.557904117139245, 33.246831543419376, 6.142621888722936, 5.103290372587008,
63.43626932854053, 9.476689250801556, 1.864715692002856, 29.77637218071447, 59.676413
29816673, 20.059422156136336, 37.66638884185679, 16.764681367516143, 24.4359784820927
93, 15.027964386205987, 41.248251118989884]
```

In [53]:
```
#write arrays to CSV for offline access
# write_list_to_csv(lumpy_family, lumpy_train_rmse, lumpy_test_rmse, 'lumpy_rmse1.csv
# write_list_to_csv(erratic_family, erratic_train_rmse, erratic_test_rmse, 'erratic_rm
# write_list_to_csv(intermittent_family, intermittent_train_rmse, intermittent_test_rm

write_list_to_csv(lumpy_family, lumpy_train_rmse, lumpy_test_rmse, 'lumpy2_rmse.csv')
write_list_to_csv(erratic_family, erratic_train_rmse, erratic_test_rmse, 'erratic_rmse
write_list_to_csv(intermittent_family, intermittent_train_rmse, intermittent_test_rmse
```

Model Results

In [127…
```
#Plotting Functions
#routine based on Bowleen example

# shift train predictions for plotting
trainPredictPlot = np.empty_like(scaled_demand_profile)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[lag_periods:len(trainPredict)+lag_periods, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(scaled_demand_profile)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(lag_periods*2)+1:len(scaled_demand_profile)-1, :] =
# plot baseline and predictions

scaler = MinMaxScaler()
demand = scaler.fit_transform(demand_profile)
plt.plot(scaler.inverse_transform(scaled_demand_profile), label = 'Demand Profile')
plt.plot(trainPredictPlot, label = 'Predicted Train Demand Profile')
plt.plot(testPredictPlot, label = 'Predicted Test Demand Profile')
plt.title('Plot 1: Product Family 49 - Model Results ')
plt.legend(loc='upper right')
plt.show()
```



Example Prediction Results

Plot 4 above provides a sample view of the the orginal demand profile and complex model results for Product Family 49. As can be seen in the chart, the resultanf forcast is less than desireable. Results plots for all three demand classes were reviewed and for brevity sake a single sample is provided. Plot 4 is indicitive of the results for both models and all three demand classes. The results show that both models had a very difficult time learning and predicting demand patterns of a sporadic nature.

This is not surprising given the sparcity of the demand profiles. Profiles of this nature are made up of many periods of zero demand. The sporadic zero demand periods make it very difficult to model. As can be seen in the Plot 1, the end result being more of a moving average type of response. The models could not pick up the sporadic patterns resulting in the model's essentially establishing an average type demand patten. This can be seen in both the demand quantity and the period tracking. It is safe to say, the LSTM models as defined above, exhibit poor performance against sporadic demand patterns.

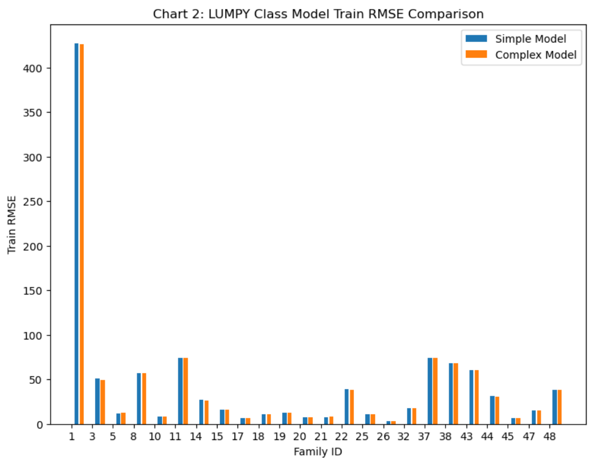
Model Comparison

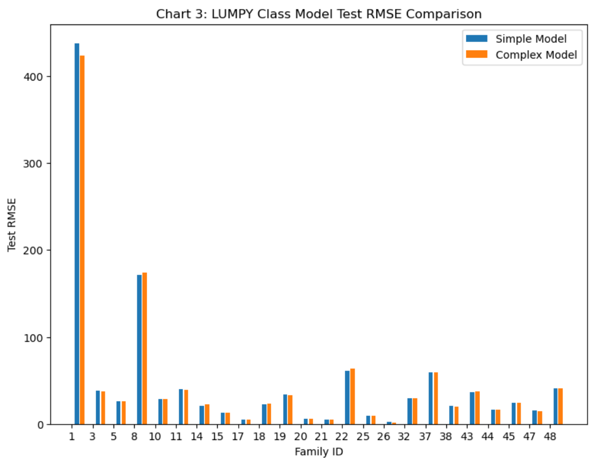
Root Mean Square Error Results

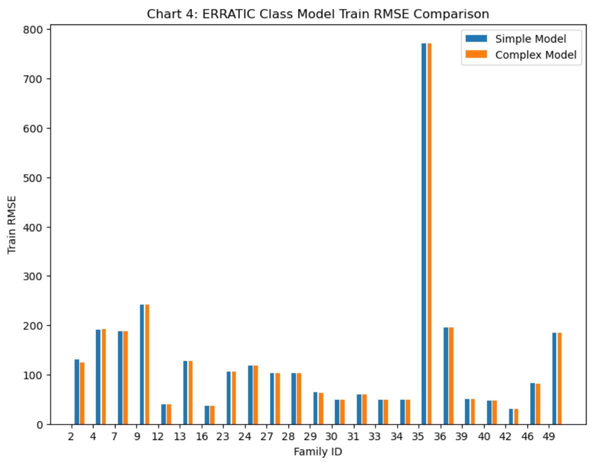
In [125…

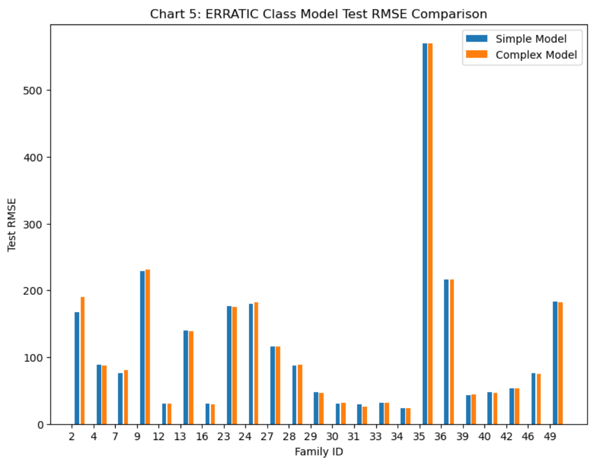
```
#Simple Model Pull in CSV data files

# reading CSV file
lumpy_simple_model = pd.read_csv("lumpy_rmse1.csv", header=None)
lumpy_complex_model = pd.read_csv("lumpy2_rmse.csv", header=None)
```

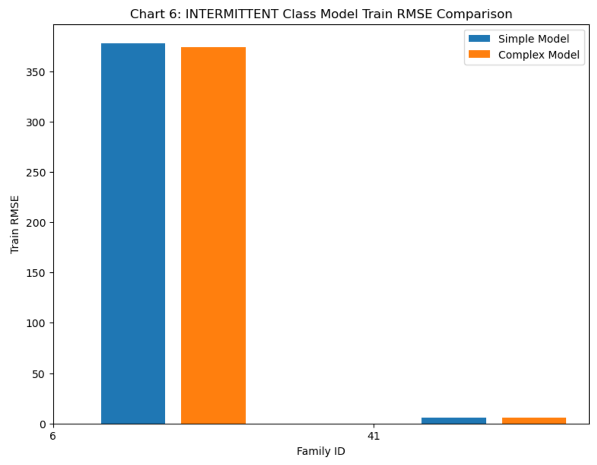
```
erratic_simple_model = pd.read_csv("erratic_rmse1.csv", header=None)
erratic_complex_model = pd.read_csv("erratic_rmse2.csv", header=None)
intermittent_simple_model = pd.read_csv("intermittent_rmse1.csv", header=None)
intermittent_complex_model = pd.read_csv("intermittent_rmse2.csv", header=None)

#calculate average test and train RMSE by model for each class
lumpy_simple_model_train_ave_rmse = lumpy_simple_model.iloc[:,1].mean()
lumpy_simple_model_test_ave_rmse = lumpy_simple_model.iloc[:,2].mean()
lumpy_complex_model_train_ave_rmse = lumpy_simple_model.iloc[:,1].mean()
lumpy_complex_model_test_ave_rmse = lumpy_simple_model.iloc[:,2].mean()

erratic_simple_model_train_ave_rmse = erratic_simple_model.iloc[:,1].mean()
erratic_simple_model_test_ave_rmse = erratic_simple_model.iloc[:,2].mean()
erratic_complex_model_train_ave_rmse = erratic_simple_model.iloc[:,1].mean()
erratic_complex_model_test_ave_rmse = erratic_simple_model.iloc[:,2].mean()

intermittent_simple_model_train_ave_rmse = intermittent_simple_model.iloc[:,1].mean()
intermittent_simple_model_test_ave_rmse = intermittent_simple_model.iloc[:,2].mean()
intermittent_complex_model_train_ave_rmse = intermittent_simple_model.iloc[:,1].mean(
intermittent_complex_model_test_ave_rmse = intermittent_simple_model.iloc[:,2].mean()


#Lumpy Class Plots
plt.figure(figsize=(20, 15))
plt.subplot(2, 2, 1)
lumpy_axis = np.arange(len(lumpy_simple_model.iloc[:,0].tolist()))
plt.bar(lumpy_axis +0.25, lumpy_simple_model.iloc[:,1].tolist(), width=0.2, label = '
plt.bar(lumpy_axis +0.25*2, lumpy_complex_model.iloc[:,1].tolist(), width=0.2, label 
plt.xticks(lumpy_axis,lumpy_family)
plt.xlabel("Family ID")
plt.ylabel("Train RMSE")
plt.title('Chart 2: LUMPY Class Model Train RMSE Comparison')
plt.legend()

plt.subplot(2, 2, 2)
plt.bar(lumpy_axis +0.25, lumpy_simple_model.iloc[:,2].tolist(), width=0.2, label = '
plt.bar(lumpy_axis +0.25*2, lumpy_complex_model.iloc[:,2].tolist(), width=0.2, label 
plt.xticks(lumpy_axis,lumpy_family)
plt.legend()
plt.xlabel("Family ID")
plt.ylabel("Test RMSE")
plt.title('Chart 3: LUMPY Class Model Test RMSE Comparison')
plt.show()

#Erratic Class Plots
plt.figure(figsize=(20, 15))
plt.subplot(2, 2, 1)
erratic_axis = np.arange(len(erratic_simple_model.iloc[:,0].tolist()))
plt.bar(erratic_axis +0.25, erratic_simple_model.iloc[:,1].tolist(), width=0.2, label
plt.bar(erratic_axis +0.25*2, erratic_complex_model.iloc[:,1].tolist(), width=0.2, lab
plt.xticks(erratic_axis,erratic_family)
plt.xlabel("Family ID")
plt.ylabel("Train RMSE")
plt.title('Chart 4: ERRATIC Class Model Train RMSE Comparison')
plt.legend()

plt.subplot(2, 2, 2)
plt.bar(erratic_axis +0.25, erratic_simple_model.iloc[:,2].tolist(), width=0.2, label
plt.bar(erratic_axis +0.25*2, erratic_complex_model.iloc[:,2].tolist(), width=0.2, lab
plt.xticks(erratic_axis,erratic_family)
plt.legend()
plt.xlabel("Family ID")
plt.ylabel("Test RMSE")
plt.title('Chart 5: ERRATIC Class Model Test RMSE Comparison')
plt.show()

#Imtermittent Class Plots
plt.figure(figsize=(20, 15))
plt.subplot(2, 2, 1)
intermittent_axis = np.arange(len(intermittent_simple_model.iloc[:,0].tolist()))
plt.bar(intermittent_axis +0.25, intermittent_simple_model.iloc[:,1].tolist(), width=0
plt.bar(intermittent_axis +0.25*2, intermittent_complex_model.iloc[:,1].tolist(), widt
plt.xticks(intermittent_axis,intermittent_family)
plt.xlabel("Family ID")
plt.ylabel("Train RMSE")
plt.title('Chart 6: INTERMITTENT Class Model Train RMSE Comparison')
plt.legend()

plt.subplot(2, 2, 2)
plt.bar(intermittent_axis +0.25, intermittent_simple_model.iloc[:,2].tolist(), width=0
plt.bar(intermittent_axis +0.25*2, intermittent_complex_model.iloc[:,2].tolist(), widt
plt.xticks(intermittent_axis,intermittent_family)
plt.legend()
plt.xlabel("Family ID")
plt.ylabel("Test RMSE")
plt.title('Chart 7: INTERMITTENT Class Model Test RMSE Comparison')
plt.show()
```

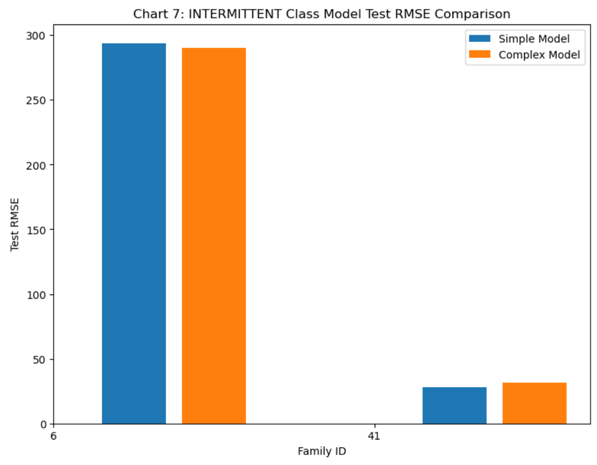












In [113…

```
#print average test and train RMSE by model for each class
print('Average RMSE (Lumpy, Simple, Train): ', round(lumpy_simple_model.iloc[:,1].mear
print('Average RMSE (Lumpy, Simple, Test): ', round(lumpy_simple_model.iloc[:,2].mean(
print('Average RMSE (Lumpy, Complex, Train): ', round(lumpy_complex_model.iloc[:,1].me
print('Average RMSE (Lumpy, Complex, Test): ', round(lumpy_complex_model.iloc[:,2].mea

print('Average RMSE (Erratic, Simple, Train): ', round(erratic_simple_model.iloc[:,1].
print('Average RMSE (Erratic, Simple, Test): ', round(erratic_simple_model.iloc[:,2].r
print('Average RMSE (Erratic, Complex, Train): ', round(erratic_complex_model.iloc[:,1
print('Average RMSE (Erratic, Complex, Test): ', round(erratic_complex_model.iloc[:,2]

print('Average RMSE (Intermittent, Simple, Train): ', round(intermittent_simple_model.
print('Average RMSE (Intermittent, Simple, Test): ', round(intermittent_simple_model.i
print('Average RMSE (Intermittent, Complex, Train): ', round(intermittent_complex_mode
print('Average RMSE (Intermittent, Complex, Test): ', round(intermittent_complex_model
```

```
Average RMSE (Lumpy, Simple, Train):  45.12
Average RMSE (Lumpy, Simple, Test):  48.55
Average RMSE (Lumpy, Complex, Train):  44.9
Average RMSE (Lumpy, Complex, Test):  48.2
Average RMSE (Erratic, Simple, Train):  131.29
Average RMSE (Erratic, Simple, Test):  116.14
Average RMSE (Erratic, Complex, Train):  130.83
Average RMSE (Erratic, Complex, Test):  117.4
Average RMSE (Intermittent, Simple, Train):  191.65
Average RMSE (Intermittent, Simple, Test):  161.03
Average RMSE (Intermittent, Complex, Train):  189.69
Average RMSE (Intermittent, Complex, Test):  160.9
```

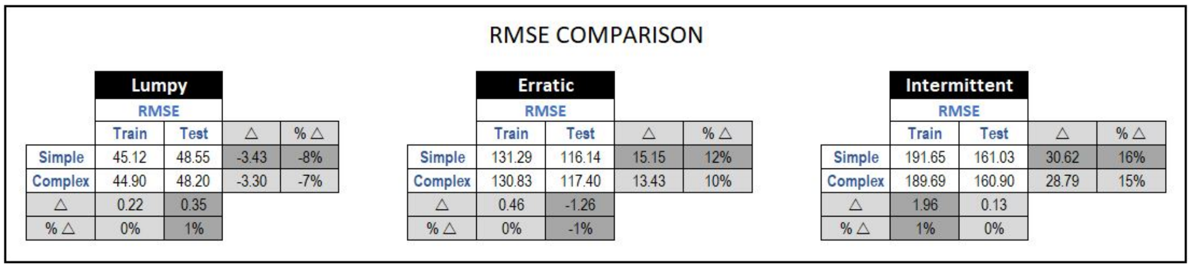
Results and Analysis

Table 1: Model Complexity vs. Root Mean Square Error

Demand Class	Model	Train RMSE	Test RMSE
Lumpy	Simple	45.12	48.55
Lumpy	Complex	44.90	48.2
Erratic	Simple	131.29	116.14
Erratic	Complex	130.83	117.4
Intermittent	Simple	191.65	161.03

Demand Class	Model	Train RMSE	Test RMSE
Intermittent	Complex	189.69	160.9

Table 2: Model Complexity vs. Root Mean Square Error - Change and Percent Change



RMSE COMPARISON

Lumpy	RMSE			
	Train	Test	△	% △
Simple	45.12	48.55	-3.43	-8%
Complex	44.90	48.20	-3.30	-7%
△	0.22	0.35		
% △	0%	1%		

Erratic	RMSE			
	Train	Test	△	% △
Simple	131.29	116.14	15.15	12%
Complex	130.83	117.40	13.43	10%
△	0.46	-1.26		
% △	0%	-1%		

Intermittent	RMSE			
	Train	Test	△	% △
Simple	191.65	161.03	30.62	16%
Complex	189.69	160.90	28.79	15%
△	1.96	0.13		
% △	1%	0%		

Plot 1 depicts a graphical representation of the training and testing prediction results for Product Family 49. Plots of multiple families from all three demand classes were reviewd and they all showed a similiar pattern. The predicted values appear to be tracking a moving average of the actual demand quantity. The average nature of the predictions against some of the larger spikes in the demand profiles are considered to be the large contibutors to the RMSE.

The averaging nature of the models combined with the larger spikes seem to be driving the poor performance of the models. It was proposed that the additional layers and sequencing of the complex model would help absorb some of this variability. Clearly it did not, as the average RMSE for both models tracked closely on a product family basis. It appears that these models suffer from the same bias as traditional models. As noted in the introduction, traditional models tend to bias towards the zero periods or demand level extremes.

The results shown in Charts 2-6 reveal that the RMSE values across both families and demand pattern classes vary widely. This is not necessarily unexpected as each family within a class could have wildly different CV2 and ADI values. As can be seen in Images 3 and 4, the Intermittent, Lumpy and Erratic demand patterns contain CV2 and ADI lower bounds. Additionally the Intermittent pattern has no ADI upper bound, the Erratic pattern has no CV2 upper bound, and the Lumpy pattern has neither an ADI or CV2 upper bound.

Given the boundless nature of the classification structure, the demand profiles for the classes can surely vary dramatically within a class, as well as, across classes. It is not unexpected to see wide ranges in RMSE depending on where and how the individual demand profile's variabilities exsist. The difficulty then becomes how to make comparisons between classes. For the comparison, a simple average will be used. This will provide a global sense of how the models for the different classes compare to each other.

From both the charts and tables it is evident that the family level training and test RSME results track each other quite well. This is a positive sign for the model training. What is interesting is that the Lumpy class (no upper and lower bounds) had the lowest averge RSME. While the classes with a single bound had higher average RSME. The Erratic and Intermittent patterns showed a 2.9X and 7.9X greater RMSE than the Lumpy pattern. This is somewhat perplexing given the bounded nature of the classes. The case of the intermittenet class is somewhat understandable given only two families reside in the intermittent class. A more detailed study is necessary to better understand this.

From Tables 1 and 2 there are a few patterns that result. For all three classes, the complex models performed marginally better than the simple model on training. This falls in lines with the initial hypothesis that a more complex model should perform better. However, given the marginal difference, a maximum of 2% for the Intermittent class, it is hard to say that one is statistically better than the other. In short the complexity of the modles had no impact across the demand pattern classes. The complexity of the model did not favor one class over another.

What is the rational for the poor performance of both models. In retrospect, it is believed that scaling of the time series' may be driving the poor performance, as well as, contibuting to results appearing as moving averages. Scaling the demand, with a considerable amount of zero periods results in a smoothing of the data, hence the moving average appearance.

A more appropriate scaling, or 'transformation' of the data to a more continous space might have been more appropriate. Similiar to the way a fourier transform is used on spectral time series data. The erratic spectoral patterns are moved into the time domain. The time domain space contains a more well behaved pattern to be analyzed. The model could then train and predict on a well behaved series. The resuls then transformed back possibly resulting in much improved performance.

Conclusion

Performance

From the discussion above, the overall performance of both models was poor. The analysis showed that performance of the training and test sets provided similiar results. However, the prediction results were far from desired. The results showed marginal performance improvements in the complex model, which aligned with the initial hypothesis. However, the minimal improvements hardly warrant the addtional resources required by the complex model.

The models also showed dramatically different in-family RMSE results. This falls in line with expectations as CV2 and ADI can vary widely within a family.

The models provided the least error on the Lumpy patterns. The error was ~3 times worse for the Erratic patterns and ~8 times worse on the Intermittent patterns. All in all, the current model and approach is deemed ill-advised. Improvments or an alternate approach is required with potential options provided below.

Improvments

As noted in the analysis section above, defining some type of transformation of the original demand profiles into a more behaved space could provide benefits. The resulting of the training, testing and predictions in the well behaved profiles could then be transformed back into the sporatic space, possibly generating improved performance.

Future Work

Given the poor performace of the models based on the current dataset, one wonders if looking for a more simplistic output would provide improved results. The current ask from the models is to provide a prediction of quantites in a time series. An interesting approach would be to decrease the ask and just look for predictions of future periods of demand, regardless of quantity. The model would analyze a binary times series of demand periods, 1 for periods with demand, 0 for periods without. The model could then be constucted to simply predict future periods with or without demand.

If the binary model provides acceptable results, it could then be used to build a hibrid model that could in some way integrate a quantity element, resulting in a combined model that performs as desired?

References

Croston, J. (1972). Forecasting and stock control for intermittent demands. , 23, 289-304. Opl Res Q, 289-304.

Brownlee, Jason (2016). Deep Learning for Time SeriesTime Series Prediction with LSTM Recurrent Neural Networks in Python with Keras., https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.

```
In [ ]:
```