sales_predictions_EDA

June 10, 2022

1 Sales Predictions using Time Series Data

- https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data

1.1 Overview of Problem

"You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge." *(src: competition page)*

- The data set contain information on historical sales for software stores in Russia
- The items can change every month, which can make things interesting
- Even though the items change, they can be grouped into categories
- There are already some categories in the training data, but there are still some gaps that we need to filled in by clustering items.

1.2 Imports

[3]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

1.3 Load Data and Data Description

File descriptions

- sales_train.csv - the training set. Daily historical data from January 2013 to October 2015.
- test.csv - the test set. You need to forecast the sales for these shops and products for November 2015.
- sample_submission.csv - a sample submission file in the correct format.
- items.csv - supplemental information about the items/products.
- item_categories.csv - supplemental information about the items categories.
- shops.csv- supplemental information about the shops. #### Data fields
- ID - an Id that represents a (Shop, Item) tuple within the test set
- shop_id - unique identifier of a shop
- item_id - unique identifier of a product

- item_category_id - unique identifier of item category
- item_cnt_day - number of products sold. You are predicting a monthly amount of this measure
- item_price - current price of an item
- date - date in format dd/mm/yyyy
- date_block_num - a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33
- item_name - name of item
- shop_name - name of shop
- item_category_name - name of item category

(from competition description)

[ ]:
```
# Load the data
```

[4]:
```
dfSales = pd.read_csv("../input/future-sales/sales_train.csv")
```

[7]:
```
# Quick Stats on the data
dfSales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2935849 entries, 0 to 2935848
Data columns (total 6 columns):
 #   Column          Dtype
---  ------          -----
 0   date            object
 1   date_block_num  int64
 2   shop_id         int64
 3   item_id         int64
 4   item_price      float64
 5   item_cnt_day    float64
dtypes: float64(2), int64(3), object(1)
memory usage: 134.4+ MB
```

[11]:
```
dfSales.isnull().sum()
```

[11]:
```
date              0
date_block_num    0
shop_id           0
item_id           0
item_price        0
item_cnt_day      0
dtype: int64
```

There are not any null values, but the instructions mention that there are holes in data for stores and items within some months. I am going to transform the data into a dataframe with item and store as the key and the sums for each month in the columns.

[12]:
```
dfSales.head()
```

[12]:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.00	1.0
1	03.01.2013	0	25	2552	899.00	1.0
2	05.01.2013	0	25	2552	899.00	-1.0
3	06.01.2013	0	25	2554	1709.05	1.0
4	15.01.2013	0	25	2555	1099.00	1.0

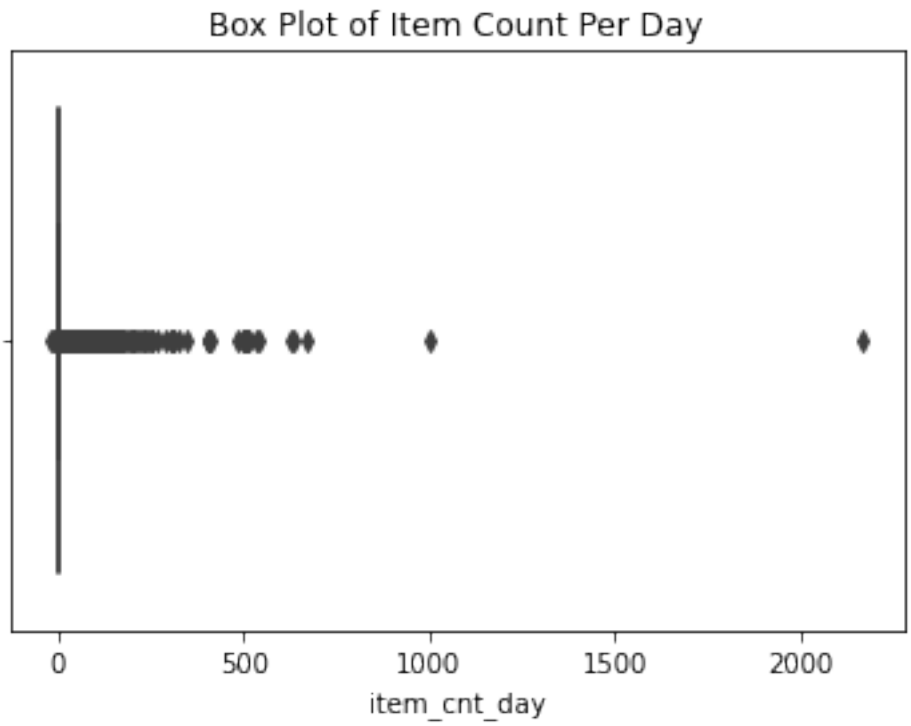
Much of the text of the data is in Russian. This is not really relevant because we do not need to look at item descriptions and category names. We can work with just IDs. I don't see any need to load the item and shops files unless I want to include item descriptions in my report.

1.4 Exploratory Data Analysis and Data Cleaning

[16]:
```
print("Sales Count Mean:",dfSales['item_cnt_day'].mean())
print("Sales Count Standard Deviation:",dfSales['item_cnt_day'].std())
print("Sales Count Max:",dfSales['item_cnt_day'].max())
```

```
Sales Count Mean: 1.242640885140891
Sales Count Standard Deviation: 2.618834430895425
Sales Count Max: 2169.0
```

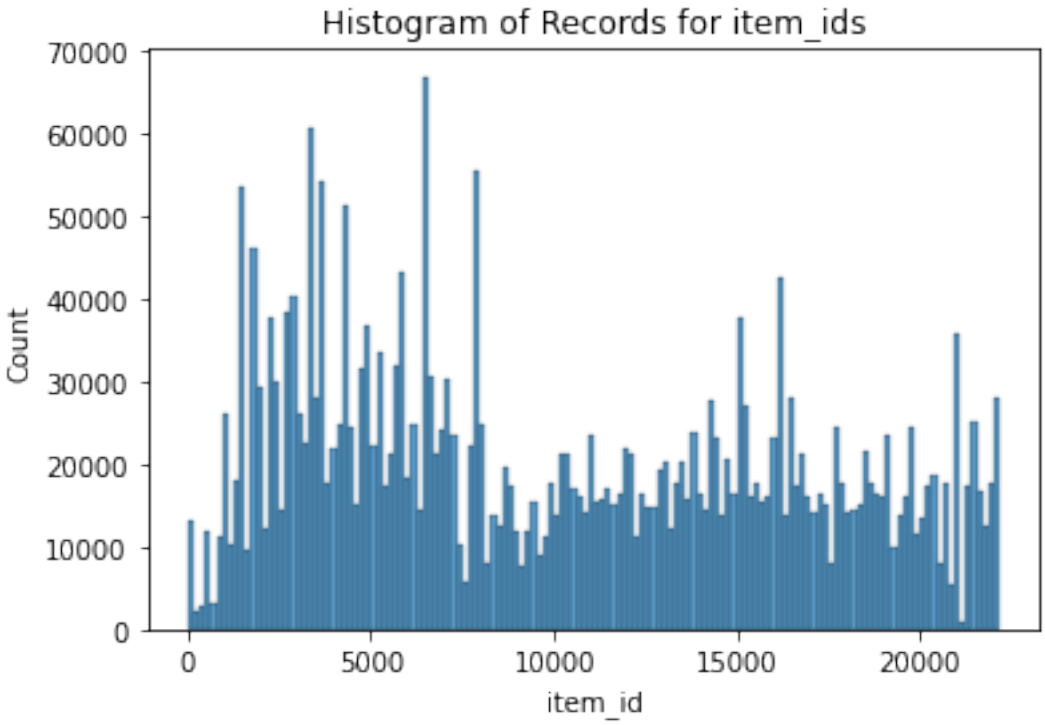
[21]:
```
# It looks like there may be some outliers
sns.boxplot(x=dfSales['item_cnt_day'])
plt.title("Box Plot of Item Count Per Day");
```



[ ]:
```
# Before transforming the data, I want to grab some histograms by store and item
# This will not take the daily count into consideration
# The mean is close to 1 so a simple histogram will be a decent visualization␣
 ↪to get started
# As we can see from the boxplot three are some outliers and they will be␣
 ↪treated the same in the histogram as a single sale
```

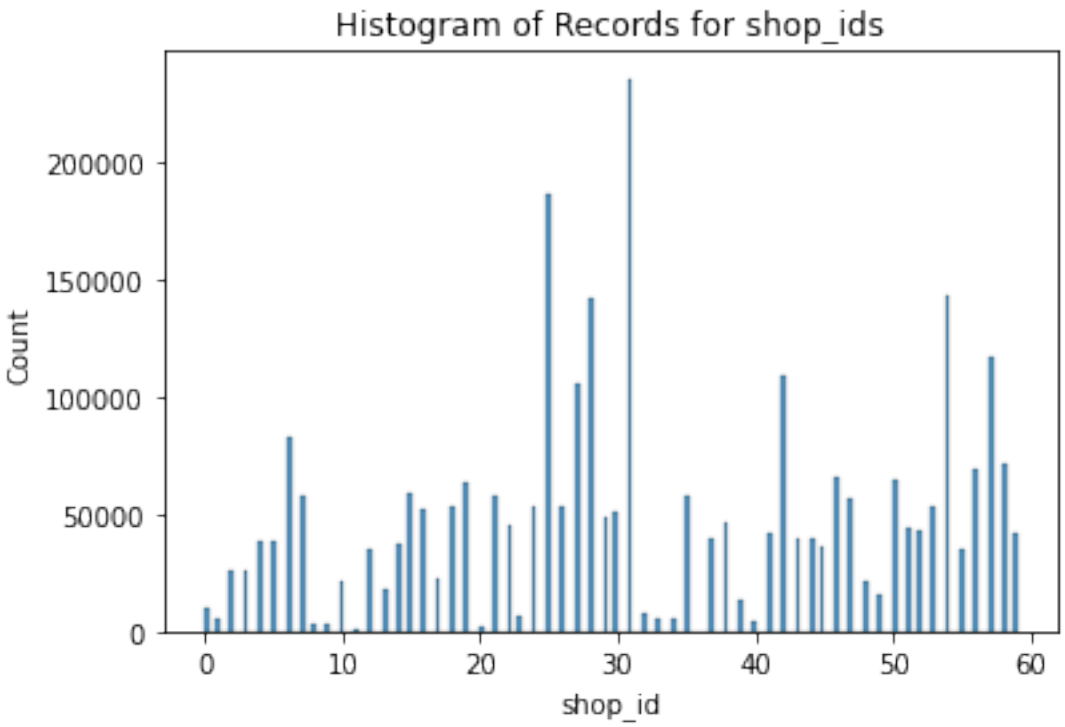
[25]:
```
sns.histplot(dfSales['item_id'])
plt.title("Histogram of Records for item_ids")
```

[25]:
```
Text(0.5, 1.0, 'Histogram of Records for item_ids')
```



[26]:
```
sns.histplot(dfSales['shop_id'])
plt.title("Histogram of Records for shop_ids")
```

[26]:
```
Text(0.5, 1.0, 'Histogram of Records for shop_ids')
```

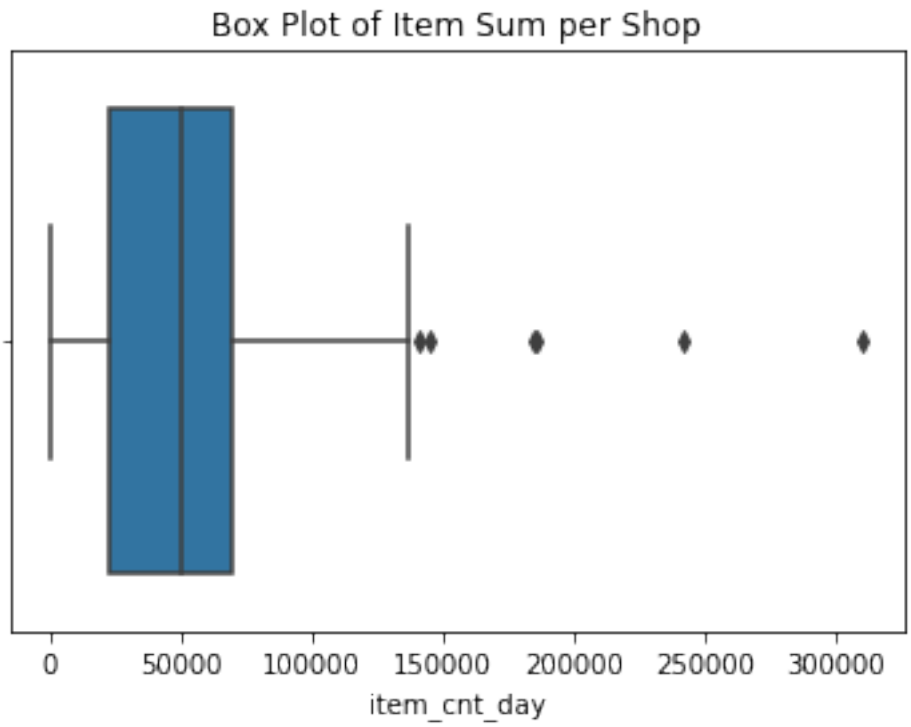


[36]:
```
# There are clearly shops with fewer sales, I am curious if the ones with fewer␣
 ↪sales have bigger numbers per sale
# Lets group by shop ID
dfShopItems = dfSales.groupby(by='shop_id').sum()['item_cnt_day'].reset_index()
dfShopItems.head()
```

[36]:

	shop_id	item_cnt_day
0	0	11705.0
1	1	6311.0
2	2	30620.0
3	3	28355.0
4	4	43942.0

[37]:
```
sns.boxplot(x=dfShopItems['item_cnt_day'])
plt.title("Box Plot of Item Sum per Shop");
# Nice outliers
```



[41]:
```
dfShopItems.describe()
```

[41]:

	shop_id	item_cnt_day
count	60.000000	60.000000
mean	29.500000	60803.433333
std	17.464249	57992.901750
min	0.000000	330.000000
25%	14.750000	23333.000000
50%	29.500000	50176.000000
75%	44.250000	69562.250000
max	59.000000	310777.000000

[45]:
```
dfShopItems[dfShopItems['item_cnt_day'] > 140000]
```

[45]:

	shop_id	item_cnt_day
25	25	241920.0
28	28	184557.0
31	31	310777.0
42	42	144934.0
54	54	185790.0
57	57	141107.0

[47]:
```
# I want to take a deeper look at shop 31
dfSales[dfSales['shop_id'] == 31].describe()
# It looks like shop 31 just has high volume
# The max item cnt per day is not huge
# The mean and std is about the same as the general population
# I did not do a t-test to verify that distribution was the same
```

[47]:

	date_block_num	shop_id	item_id	item_price	item_cnt_day
count	235636.000000	235636.0	235636.000000	235636.000000	235636.000000
mean	14.935341	31.0	11093.120033	724.691619	1.318886
std	9.544102	0.0	6176.734564	1383.986135	2.199277
min	0.000000	31.0	26.000000	0.100000	-2.000000
25%	7.000000	31.0	5612.000000	199.000000	1.000000
50%	14.000000	31.0	11268.000000	374.250000	1.000000
75%	23.000000	31.0	16205.000000	790.000000	1.000000
max	33.000000	31.0	22167.000000	35991.000000	288.000000

[53]:
```
# I will probably toss many of these high quanity items
# but lets investigate a bit more
display(dfSales[dfSales['item_cnt_day'] > 500])
item_list = set(dfSales[dfSales['item_cnt_day'] > 500]['item_id'])
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
1573253	22.04.2014	15	27	8057	1200.000000	502.0
1708207	28.06.2014	17	25	20949	5.000000	501.0
2048518	02.10.2014	21	12	9242	1500.000000	512.0
2067669	09.10.2014	21	55	19437	899.000000	508.0
2326930	15.01.2015	24	12	20949	4.000000	1000.0
2608040	14.04.2015	27	12	3731	1904.548077	624.0
2626181	19.05.2015	28	12	11373	155.192950	539.0
2851073	29.09.2015	32	55	9249	1500.000000	533.0
2851091	30.09.2015	32	55	9249	1702.825746	637.0
2864235	30.09.2015	32	12	9248	1692.526158	669.0
2864260	29.09.2015	32	12	9248	1500.000000	502.0
2909818	28.10.2015	33	12	11373	0.908714	1000.0

```
2864260          504.0
2909818         2169.0
```

[49]:
```
# I previously said I didn't want to look at item descriptions,
# but I want to see what is going on with come off these high cost and high␣
 ↪volume items
dfItemDescription = pd.read_csv("../input/future-sales/items.csv")
```

[52]:
```
dfItemDescription[dfItemDescription['item_id'].isin(item_list)]
```

[52]:
```
                                                item_name  item_id  \
3731           Grand Theft Auto V [PC,           ]     3731
8057                                      iTunes 1500   .     8057
9242       "     2014" (         (      -…     9242
9248       "     2015" - 3      2015 (   ) […     9248
9249       "     2015" - 3      2015 (   ) …     9249
11373                                (Boxberry)    11373
19437          :          [PC,           ]    19437
20949               1          (34*42)…    20949

       item_category_id
3731                 30
8057                 32
9242                  8
9248                 80
9249                  8
11373                 9
19437                31
20949                71
```

- In the above output, we can see some interesting trends such as GameWorld tickets being sold only on 2 days
- Some of the other items may be due to game releases
- For example Middle-earth: Shadows of Mordo was released in Russia on 3 Oct 2014. The spike for that product is on 9 Oct 2014
- The purpose of this exercise is to predict trends
- Major game releases happen but I would like to remove them for this analysis since predicting those trends is out of scope of this analysis

[54]:
```
# Look for other ticket sales
dfItemDescription[dfItemDescription['item_name'].str.contains("   ")]
```

[54]:
```
                                          item_name  item_id  \
9240                              "          "     9240
9241       "     2014" (         (      -…     9241
9242       "     2014" (         (      -…     9242
9243       "     2014" -         4-      …     9243
9244       "     2015" - 2      2015 (   ) […     9244
```

```
9245      "     2015" - 2      2015 (  ) …      9245
9246      "     2015" - 2      2015 [    …     9246
9247      "     2015" - 2      2015    [  …     9247
9248      "     2015" - 3      2015 (  ) […      9248
9249      "     2015" - 3      2015 (  ) …      9249
9250      "     2015" - 3      2015 [    …     9250
9251      "     2015" - 3      2015    [  …     9251
9252      "     2015" - 4      2015 (  ) […      9252
9253      "     2015" - 4      2015 (  ) …      9253
9254      "     2015" - 4      2015 [    …     9254
9255      "     2015" - 4      2015    [  …     9255

      item_category_id
9240                73
9241                 8
9242                 8
9243                 8
9244                80
9245                 8
9246                80
9247                 8
9248                80
9249                 8
9250                80
9251                 8
9252                80
9253                 8
9254                80
9255                 8
```

[61]:
```
# Drop ticket sales
# These seem to be in October and they may skew the Novemeber estimate to be␣
 ↪too hight
ticket_ids = set(dfItemDescription[dfItemDescription['item_name'].str.
 ↪contains("   ")]['item_id'])
dfSales.drop(dfSales[dfSales['item_id'].isin(ticket_ids)].index,inplace=True)
dfSales.drop(dfSales[dfSales['item_id'].isin(item_list)].index,inplace=True)
```

1.4.1 Data Transformation

- We completed initial cleaning
- Now we will transform the data to a frame that contains shop id and item id and sum for each month

[116]:
```
# I may be able to simplify this using a join
# Drop price
try:
```

```
    dfSales.drop(['item_price'],axis=1,inplace=True)
except:
    print("I probably already did this")
dfShopItems = dfSales.groupby(by=['date_block_num', 'shop_id', 'item_id']).
 ↪sum().reset_index()
dfShopItems
```

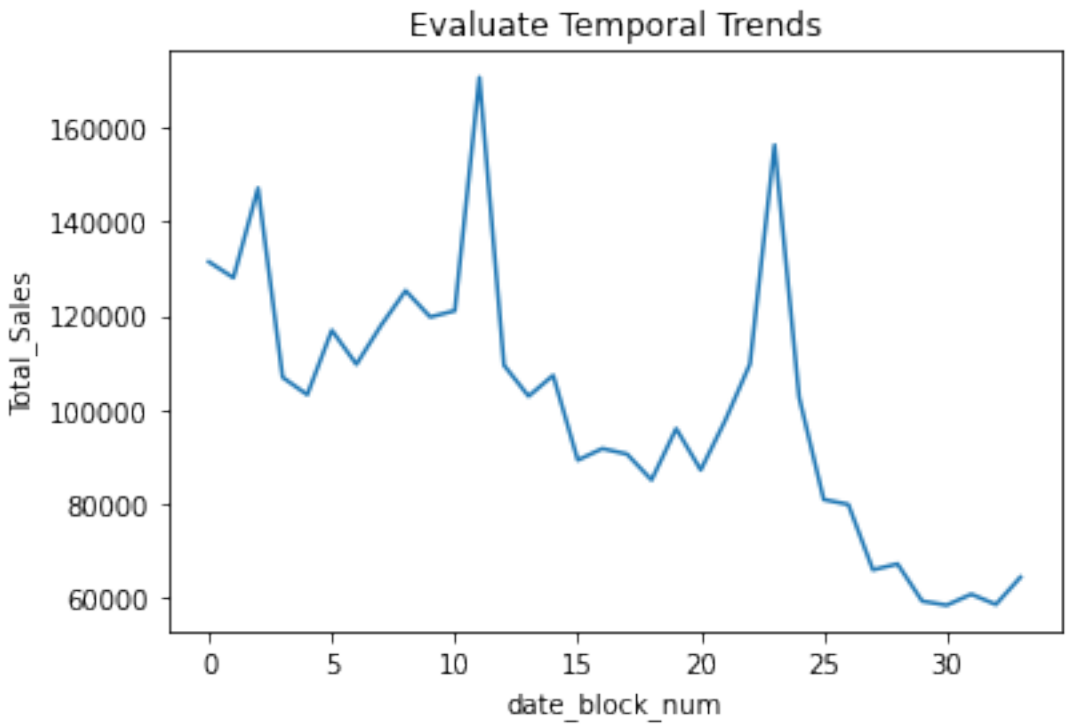
```
I probably already did this
```

[116]:

	date_block_num	shop_id	item_id	item_cnt_day
0	0	0	32	6.0
1	0	0	33	3.0
2	0	0	35	1.0
3	0	0	43	1.0
4	0	0	51	2.0
…	…	…	…	…
1606703	33	59	22087	6.0
1606704	33	59	22088	2.0
1606705	33	59	22091	1.0
1606706	33	59	22100	1.0
1606707	33	59	22102	1.0

[1606708 rows x 4 columns]

[128]:
```
### Visualization of general sales trends
dfMonths = dfShopItems.groupby(by=['date_block_num'] ,as_index=False).sum()
dfMonths.rename(columns={'item_cnt_day':'Total_Sales'},inplace=True)
sns.lineplot(y=dfMonths['Total_Sales'],x=dfMonths['date_block_num'])
plt.title("Evaluate Temporal Trends");
```



[130]:
```
# We see a spike at month 11 and 23. What are they
dfSales[dfSales['date_block_num'] == 11].head(1)
```

[130]:

	date	date_block_num	shop_id	item_id	item_cnt_day
1124316	04.12.2013	11	25	17769	1.0

[131]:
```
dfSales[dfSales['date_block_num'] == 23].head(1)
```

[131]:

	date	date_block_num	shop_id	item_id	item_cnt_day
2192637	24.12.2014	23	42	17279	1.0

[132]:
```
## Not a surprise that it is December
```

[133]:
```
# Now remove dfMonths to save memory
dfMonths = None
```

[93]:
```
# I want to fill in the holes, so I am making a data frame with crossproduct of␣
 ↪date_block shop id and item_id
# Then I will it with dfShopItems and fills the NaNs with 0
dfTemp1 = pd.DataFrame(data=dfShopItems['date_block_num'].
 ↪unique(),columns=["date_block_num"])
dfTemp2 = pd.DataFrame(data=dfShopItems['shop_id'].unique(),columns=["shop_id"])
dfTemp3 = pd.DataFrame(data=dfShopItems['item_id'].unique(),columns=["item_id"])
```

```
dfTemp4 = dfTemp1.merge(dfTemp2,how='cross')
dfShopItemsFull = dfTemp4.merge(dfTemp3,how='cross')
```

[94]:
```
# Clear out my temp frames so I can save memory
dfTemp1 = None
dfTemp2 = None
dfTemp3 = None
dfTemp4 = None
```

[97]:
```
dfShopItemsFull['item_cnt_month'] = 0
dfShopItemsFull.tail()
```

[97]:

	date_block_num	shop_id	item_id	item_cnt_month
44443435	33	36	12733	0
44443436	33	36	13092	0
44443437	33	36	16797	0
44443438	33	36	18060	0
44443439	33	36	15925	0

[101]:
```
# Spot checking to make sure it is expanding as expected
dfShopItemsFull[dfShopItemsFull['item_id'] == 15925]
```

[101]:

	date_block_num	shop_id	item_id	item_cnt_month
21785	0	0	15925	0
43571	0	1	15925	0
65357	0	2	15925	0
87143	0	3	15925	0
108929	0	4	15925	0
…	…	…	…	…
44356295	33	34	15925	0
44378081	33	33	15925	0
44399867	33	20	15925	0
44421653	33	11	15925	0
44443439	33	36	15925	0

[2040 rows x 4 columns]

[103]:
```
# Align the indexes
dfShopItemsFull = dfShopItemsFull.
  ↪set_index(['date_block_num','shop_id','item_id'])
dfShopItems = dfShopItems.set_index(['date_block_num','shop_id','item_id'])
```

[105]:
```
dfShopItemsFull['item_cnt_month'] = dfShopItems['item_cnt_day']
```

[106]:
```
dfShopItemsFull
# We can see just in this view that there are NaNs
```

[106]:
```
                                    item_cnt_month
date_block_num shop_id item_id
0              0       32                     6.0
                       33                     3.0
                       35                     1.0
                       43                     1.0
                       51                     2.0
…                                             …
33             36      12733                  NaN
                       13092                  NaN
                       16797                  NaN
                       18060                  NaN
                       15925                  NaN

[44443440 rows x 1 columns]
```

[110]:
```
dfShopItemsFull.fillna(0,inplace=True) # This filled the holes
# Change the index back so it is easier to deal with
# I am also going to remove the initial dataframe to save memory
dfShopItemsFull.reset_index(inplace=True)
dfShopItems = None
```

[111]:
```
# Save off data frame so I can start a new notebook with data in this format
dfShopItemsFull.to_csv("dfShopItemsFull.csv",index=False)
```

1.5 This concludes the intial EDA

I will attempt to append the other notebooks in a single PDF

- During the EDA process, I looked at the structure of the data
- I dropped some data that were outliers
- I reformatted the data into a matrix with approximately 44 million records
- This allowed me to see every combination of shop, item, and month in the training data
- During the testing phase I discovered that there are items in the test data that are not in the training data at all
- This made me realize the importance of the item_category and item CSV files
- These files have categories of items such as book, video, music, game, ticket
- The existing category on their own is not enough
- We also need to create additional categories using natural language processing
- The data is in Russian, but we don't need to speak Russian to use many NLP processes
  - We can use a Russian stop words dictionary and then general statistical methods

sales_predictions_UnsupervisedMatrixMultiplication

June 10, 2022

1 Sales Predictions using Time Series Data

- https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data

1.1 Overview of Problem

"You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge." *(src: competition page)*

1.2 Imports

[45]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import NMF
```

1.3 Load Data

- The data provided from the Kaggle competition was edited and saved
- This notebook will load the updated file and continue from there

[2]:
```
# Load the data
```

[3]:
```
dfSales = pd.read_csv("dfShopItemsFull.csv")
```

[4]:
```
# Quick Stats on the data
dfSales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44443440 entries, 0 to 44443439
Data columns (total 4 columns):
 #   Column          Dtype
---  ------          -----
 0   date_block_num  int64
 1   shop_id         int64
 2   item_id         int64
```

```
 3   item_cnt_month  float64
dtypes: float64(1), int64(3)
memory usage: 1.3 GB
```

[5]:
```
dfSales.isnull().sum()
```

[5]:
```
date_block_num    0
shop_id           0
item_id           0
item_cnt_month    0
dtype: int64
```

[6]:
```
dfSales.head()
```

[6]:

	date_block_num	shop_id	item_id	item_cnt_month
0	0	0	32	6.0
1	0	0	33	3.0
2	0	0	35	1.0
3	0	0	43	1.0
4	0	0	51	2.0

1.3.1 Matrix Factorization

- The idea in the matrix multiplication model is to use the matrix we created in the EDA notebook and factor it
- This will help us find latent factors that we can use to predict missing data

[8]:
```
dfTrain = dfSales.copy() # Make a copy that we will mess up
month_size = dfTrain[dfTrain['date_block_num'].astype(int)==1].shape[0]

trainy = dfTrain.iloc[month_size:,:]['item_cnt_month'].reset_index().
 ↪drop(['index'],axis=1)
```

[10]:
```
dfTrain['next_month'] = trainy
```

[13]:
```
dfTrain = dfTrain.fillna(0)
```

[14]:
```
# If I dont have enough memory, I will just change dfSales
dfTest = dfSales[dfSales['date_block_num'] == 33]
```

[8]:
```
# Zero out the values for the last month and see what happens
#dfTrain.loc[dfSales['date_block_num'] == 33,'item_cnt_month'] = 0
```

[15]:
```
dfTest
```

[15]:

	date_block_num	shop_id	item_id	item_cnt_month
43136280	33	0	32	0.0
43136281	33	0	33	0.0

```
43136282              33        0       35              0.0
43136283              33        0       43              0.0
43136284              33        0       51              0.0
…                   …       …       …               …
44443435              33       36    12733              0.0
44443436              33       36    13092              0.0
44443437              33       36    16797              0.0
44443438              33       36    18060              0.0
44443439              33       36    15925              0.0

[1307160 rows x 4 columns]
```

[16]:
```
# There are negative values for some months.
# How should we handle this
dfTrain[dfTrain['item_cnt_month'] < 0].describe()
```

[16]:

	date_block_num	shop_id	item_id	item_cnt_month	next_month
count	912.000000	912.000000	912.000000	912.000000	912.000000
mean	14.121711	28.736842	9752.044956	-1.081140	0.383772
std	9.364640	17.095870	6235.720134	0.853736	0.955474
min	0.000000	2.000000	31.000000	-22.000000	-2.000000
25%	6.000000	12.000000	4351.750000	-1.000000	0.000000
50%	13.000000	27.000000	8106.500000	-1.000000	0.000000
75%	22.000000	44.000000	14503.250000	-1.000000	0.000000
max	33.000000	59.000000	22164.000000	-1.000000	9.000000

[21]:
```
# I assume it will be negative if there are returns.
# We could just right shift by the min, but I think it is ok to just 0 these out
dfTrain.loc[dfTrain['item_cnt_month'] < 0, 'item_cnt_month'] = 0
dfTrain.loc[dfTrain['next_month'] < 0, 'next_month'] = 0
```

[23]:
```
model_NMF = NMF(n_components=4,max_iter=700,init='nndsvda') # Basic model␣
 ↪before tweaking
nmf_fit = model_NMF.fit_transform(dfTrain)
```

[24]:
```
nmf_fit
```

[24]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.82200147e-03],
       [1.11424001e-05, 9.13996588e-08, 0.00000000e+00, 4.95702248e-03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 5.27406353e-03],
       …,
       [7.41809603e-03, 8.99911858e-06, 0.00000000e+00, 2.52070295e+00],
       [8.52856697e-03, 7.12171543e-06, 0.00000000e+00, 2.70946389e+00],
       [6.65140509e-03, 1.02953146e-05, 0.00000000e+00, 2.39037870e+00]])
```

[25]:
```
C = model_NMF.components_
R_estimated = np.dot(nmf_fit, C)
```

[30]:
```
dfTrain.shape
```

[30]:
```
(44443440, 5)
```

[38]:
```
y_hat = R_estimated[:,4].astype(int)
```

[39]:
```
y_hat
```

[39]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[44]:
```
sum(y_hat)
# Just to spot check that it isnt all 0
```

[44]:
```
1091696
```

[42]:
```
y_hat.shape
```

[42]:
```
(44443440,)
```

[43]:
```
trainy.shape
```

[43]:
```
(43136280, 1)
```

[47]:
```
# Checking against training data
# Need to submit to Kaggle competition to get test data results
mean_squared_error(trainy, y_hat[:trainy.shape[0]])
```

[47]:
```
0.945051033607905
```

1.3.2 Format Data to Post to Kaggle

[80]:
```
# Using original sales matrix cause matrix mult may have messed up some data in␣
↪dfTrain
dfSales['yhat'] = y_hat
```

[88]:
```
dfSales['next_month'] = dfTrain['next_month']
```

[93]:
```
dfPredict34 = dfSales.iloc[-month_size:]
```

[94]:
```
dfPredict34['yhat'].sum()
```

[94]:
```
18575
```

[107]:
```
dfTest = pd.read_csv("../input/future-sales/test.csv")
```

[108]:
```
# I am going to merge with test so I want to make shop_id and item_id the␣
 ↪indices
dfPredict34.set_index(['shop_id','item_id'],inplace=True)
dfTest.set_index(['shop_id','item_id'],inplace=True)
```

[109]:
```
dfTest['item_cnt_month']=dfPredict34['next_month']
```

[110]:
```
dfTest
```

[110]:

		ID	item_cnt_month
shop_id	item_id		
5	5037	0	0.0
	5320	1	NaN
	5233	2	0.0
	5232	3	0.0
	5268	4	NaN
…		…	…
45	18454	214195	0.0
	16188	214196	0.0
	15757	214197	0.0
	19648	214198	0.0
	969	214199	0.0

[214200 rows x 2 columns]

[111]:
```
# Put the indices back to normal
dfTest.reset_index(inplace=True)
dfPredict34.reset_index(inplace=True)
```

[115]:
```
# Fix some missing items
# We can improve on how we impute - FUTURE WORK
# Manual inspection shows that several missing items are similar to the next␣
 ↪item id over
# This is not always true but will use it for initial impute
# In many cases, it is the same game but on different platform
# A better impute would check the text string then compare with statistical␣
 ↪trends of the platform
# Is PS4 or Xbox more popular?

i = 0
for index,row in dfTest[dfTest['item_cnt_month'].isnull()].iterrows():
    item_id = row['item_id'].astype(int)
    # Try add one
    query_impute =␣
 ↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id +1 ) &␣
 ↪(dfPredict34['shop_id']==row['shop_id'])]
    while query_impute.shape[0] == 0: # Try  remove 1 until we have a match
```

```
        item_id -= 1
        query_impute =␣
↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id - 1 ) &␣
↪(dfPredict34['shop_id']==row['shop_id'])]
    # if it is a series then look at it and see what is up
    #print(type(query_impute))
    #if not isinstance(query_impute,np.float64):
    #    print("error",query_impute)
    #    break
    dfTest.loc[index,'item_cnt_month'] = float(query_impute)
```

[117]:
```
# Save CSV
dfTest[['ID','item_cnt_month']].to_csv("sample_submission_MM.csv",index=False)
```

1.4 Analysis and Results

- This method worked well using the training set
- We were able to obtain a good RMSE value when we evaluated predicted results against the training data
- The test data has an RMSE of 1.26
- This is a fair but not great RMSE
- We can improve this by adding some features based on clustering items and shops
- We discuss this more in the LSTM notebook

sales_predictions_Supervised_LSTM

June 10, 2022

1 Sales Predictions using Time Series Data

- https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales/data

1.1 Overview of Problem

"You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge." *(src: competition page)*

1.2 Imports

[1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Flatten
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

1.3 Load Data

- The data provided from the Kaggle competition was edited and saved
- This notebook will load the updated file and continue from there

[2]:
```
# Load the data
```

[2]:
```
dfSales = pd.read_csv("dfShopItemsFull.csv")
```

1.4 Quick Stats on the data

dfSales.info()

There are not any null values, but the instructions mention that there are holes in data for stores and items within some months. I am going to transform the data into a dataframe with item and store as the key and the sums for each month in the columns.

[4]:
```
dfSales.head()
```

[4]:

	date_block_num	shop_id	item_id	item_cnt_month
0	0	0	32	6.0
1	0	0	33	3.0
2	0	0	35	1.0
3	0	0	43	1.0
4	0	0	51	2.0

Much of the text of the data is in Russian. This is not really relevant because we do not need to look at item descriptions and category names. We can work with just IDs. I don't see any need to load the item and shops files unless I want to include item descriptions in my report.

1.5 Supervised Learning

1.5.1 LSTM Artificial Neural Network with Linear Activation

- The Long Short Term Memory Neural Network is a good option for time sequence data
- To use this model we needed to align a lookback so that the model is trained using the data for the next time period as the result from the current
- In a single variable time series problem, we can just look at the next record
- In our problem we have a combination of items and stores so we need to have a lookback offset of items * stores
- Some models will look forward by more than one timeslot, but due to computational and time restrictions we looked ahead by a single month
- Similarly, we restricted our model to 2 LSTM layers and 1 dense layer
- Results would have been improved with a deeper model but time constraints necessitated a simple model

Scale the Data

- The data must be scaled to use an LSTM Neural Network

[5]:
```
#training_labels = dfSales['item_cnt_month']
```

[9]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
dfScaled = pd.DataFrame(scaler.
 ↪fit_transform(dfSales[['date_block_num','shop_id','item_id','item_cnt_month']]),
                                          columns=␣
 ↪['date_block_num','shop_id','item_id','item_cnt_month'])
```

[10]:
```
dfScaled
```

[10]:

	date_block_num	shop_id	item_id	item_cnt_month
0	0.0	0.000000	0.001443	0.022152
1	0.0	0.000000	0.001489	0.019778
2	0.0	0.000000	0.001579	0.018196
3	0.0	0.000000	0.001940	0.018196
4	0.0	0.000000	0.002301	0.018987
…	…	…	…	…
44443435	1.0	0.610169	0.574361	0.017405
44443436	1.0	0.610169	0.590554	0.017405
44443437	1.0	0.610169	0.757680	0.017405
44443438	1.0	0.610169	0.814651	0.017405
44443439	1.0	0.610169	0.718345	0.017405

[44443440 rows x 4 columns]

[11]:
```
# The orginal dataframe takes 1.3 GB of memory
# I am going to remove the original dataset due to memory constraints
del dfSales
```

Test and Train Data Split

[ ]:
```
# Break the dataset into test and training
# traning with everything except the last month and test with that
```

[8]:
```
last_month_size = dfScaled[dfScaled['date_block_num'].astype(int)==1].shape[0]
```

[46]:
```
#X_test = dfScaled.iloc[-last_month_size:]
```

[48]:
```
#X_train = dfScaled.iloc[:-last_month_size]
```

[50]:
```
#y_train = training_labels[:-last_month_size]
```

[51]:
```
#y_test = training_labels[-last_month_size:]
```

Model

- We tested a few different model configurations
- Some of them either took too long to train or resulted in too high loss
- For the purposes of brevity, we are only showing one model in this notebook

[12]:
```
last_month_size = dfScaled[dfScaled['date_block_num'].astype(int)==1].shape[0]
```

[13]:
```
trainx = np.array(dfScaled.iloc[:-last_month_size,:].reset_index().
 ↪drop(['index'],axis=1))
trainy = dfScaled.iloc[last_month_size:,:]['item_cnt_month'].reset_index().
 ↪drop(['index'],axis=1)
```

[14]:
```
trainy
```

[14]:
```
           item_cnt_month
0                0.025316
1                0.019778
2                0.028481
3                0.017405
4                0.019778
…                   …
43136275         0.017405
43136276         0.017405
43136277         0.017405
43136278         0.017405
43136279         0.017405

[43136280 rows x 1 columns]
```

[15]:
```
trainx
```

[15]:
```
array([[0.        , 0.        , 0.00144346, 0.0221519 ],
       [0.        , 0.        , 0.00148857, 0.01977848],
       [0.        , 0.        , 0.00157878, 0.0181962 ],
       …,
       [0.96969697, 0.61016949, 0.75767964, 0.01740506],
       [0.96969697, 0.61016949, 0.81465109, 0.01740506],
       [0.96969697, 0.61016949, 0.71834544, 0.01740506]])
```

[16]:
```
trainx = np.reshape(trainx, (trainx.shape[0], 1, 4))
#testx = np.reshape(testx, (testx.shape[0], 1, 4))
```

[17]:
```
trainx.shape
```

[17]:
```
(43136280, 1, 4)
```

[18]:
```
# create and fit the LSTM network

model = Sequential()
model.add(LSTM(256, return_sequences = True, input_shape = (trainx.shape[1],␣
 ↪4)))
model.add(LSTM(128,input_shape = (trainx.shape[1], 2)))
#model.add(Flatten())
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
```

```
2022-06-08 14:01:36.441899: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
```

best performance.

[19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 256)            267264
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               197120
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 464,513
Trainable params: 464,513
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
call_backs = [EarlyStopping(monitor='loss', patience=3, verbose=1)]
history = model.fit(trainx, trainy, epochs = 20, batch_size = 60, verbose =␣
 ↪True, shuffle = False,  callbacks = call_backs)
model.save_weights('LSTMBasic1.h5')
# This will run a day. Hopefully it comes up with something useful
```

```
2022-06-08 14:02:00.366077: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Epoch 1/20
170849/718938 [======>…] - ETA: 1:05:10 - loss: 7.9538e-07
```

[22]:
```
# This finished but I lost connection and didn't see the verbose output
history.epoch
# Look up how to get the loss for each epoch
```

[22]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
# Predict using training data to see how accurate it is for training
y_hat = model.predict(trainx)
```

[29]:
```
y_hat
```

[29]:
```
array([[0.04450752],
       [0.03698179],
       [0.03183561],
       …,
       [0.01730757],
```

```
       [0.01730722],
       [0.01730787]], dtype=float32)
```

[31]:
```
# I messed up but I dont want to restart the kernel. I should have scaled the item_cnt_month on its own
dfYhat = dfScaled.iloc[:-last_month_size,:].reset_index().drop(['index'],axis=1)
```

[35]:
```
dfYhat['item_cnt_month'] =y_hat
```

[36]:
```
dfYhat
```

[36]:

	date_block_num	shop_id	item_id	item_cnt_month
0	0.000000	0.000000	0.001443	0.044508
1	0.000000	0.000000	0.001489	0.036982
2	0.000000	0.000000	0.001579	0.031836
3	0.000000	0.000000	0.001940	0.031828
4	0.000000	0.000000	0.002301	0.034396
…	…	…	…	…
43136275	0.969697	0.610169	0.574361	0.017309
43136276	0.969697	0.610169	0.590554	0.017309
43136277	0.969697	0.610169	0.757680	0.017308
43136278	0.969697	0.610169	0.814651	0.017307
43136279	0.969697	0.610169	0.718345	0.017308

[43136280 rows x 4 columns]

[38]:
```
dfScaled.iloc[:-last_month_size,:]
```

[38]:

	date_block_num	shop_id	item_id	item_cnt_month
0	0.000000	0.000000	0.001443	0.022152
1	0.000000	0.000000	0.001489	0.019778
2	0.000000	0.000000	0.001579	0.018196
3	0.000000	0.000000	0.001940	0.018196
4	0.000000	0.000000	0.002301	0.018987
…	…	…	…	…
43136275	0.969697	0.610169	0.574361	0.017405
43136276	0.969697	0.610169	0.590554	0.017405
43136277	0.969697	0.610169	0.757680	0.017405
43136278	0.969697	0.610169	0.814651	0.017405
43136279	0.969697	0.610169	0.718345	0.017405

[43136280 rows x 4 columns]

[43]:
```
# We need to unscale the data so we can see it in the original scale
yhat_unscaled = scaler.inverse_transform(dfYhat)[:, 3]
```

[45]:
```
yhat_unscaled.shape
```

[45]:
```
(43136280,)
```

[47]:
```
trainy
```

[47]:
```
           item_cnt_month
0                0.025316
1                0.019778
2                0.028481
3                0.017405
4                0.019778
…                   …
43136275         0.017405
43136276         0.017405
43136277         0.017405
43136278         0.017405
43136279         0.017405

[43136280 rows x 1 columns]
```

[48]:
```
# Calculate RMSE of unscaled data
# The result is fair, but not greet
# We need to see what else good for the domain
mean_squared_error(trainy, yhat_unscaled)
```

[48]:
```
1.257088408350787
```

[51]:
```
# Calculate RMSE of scaled data
# This seems good for an RMSE on the training data
mean_squared_error(dfScaled.iloc[:-last_month_size,3], y_hat)
```

[51]:
```
8.372294129050908e-07
```

[52]:
```
dfScaled.iloc[-last_month_size:,:]
```

[52]:
```
           date_block_num   shop_id   item_id  item_cnt_month
43136280             1.0  0.000000  0.001443        0.017405
43136281             1.0  0.000000  0.001489        0.017405
43136282             1.0  0.000000  0.001579        0.017405
43136283             1.0  0.000000  0.001940        0.017405
43136284             1.0  0.000000  0.002301        0.017405
…                    …         …         …               …
44443435             1.0  0.610169  0.574361        0.017405
44443436             1.0  0.610169  0.590554        0.017405
44443437             1.0  0.610169  0.757680        0.017405
44443438             1.0  0.610169  0.814651        0.017405
44443439             1.0  0.610169  0.718345        0.017405
```

```
[1307160 rows x 4 columns]
```

[54]:
```
# Predict using the last month and submit to Kaggle Competition
testx = np.array(dfScaled.iloc[-last_month_size:,:].reset_index().
 ↪drop(['index'],axis=1))
testx = np.reshape(testx, (testx.shape[0], 1, 4))
predict34 = model.predict(testx)
```

[55]:
```
predict34
```

[55]:
```
array([[0.01742227],
       [0.01742227],
       [0.01742228],
       …,
       [0.01732376],
       [0.01732335],
       [0.01732411]], dtype=float32)
```

[56]:
```
dfYhatTest = dfScaled.iloc[-last_month_size:,:].reset_index().
 ↪drop(['index'],axis=1)
```

[57]:
```
dfYhatTest['item_cnt_month'] =predict34
```

[86]:
```
predict34_unscaled = scaler.inverse_transform(dfYhatTest)
```

[87]:
```
predict34_unscaled
```

[87]:
```
array([[ 3.30000000e+01,  0.00000000e+00,  3.20000000e+01,
         2.17446089e-02],
       [ 3.30000000e+01,  0.00000000e+00,  3.30000000e+01,
         2.17516720e-02],
       [ 3.30000000e+01,  0.00000000e+00,  3.50000000e+01,
         2.17634439e-02],
       …,
       [ 3.30000000e+01,  3.60000000e+01,  1.67970000e+04,
        -1.02769315e-01],
       [ 3.30000000e+01,  3.60000000e+01,  1.80600000e+04,
        -1.03284925e-01],
       [ 3.30000000e+01,  3.60000000e+01,  1.59250000e+04,
        -1.02326691e-01]])
```

[125]:
```
dfPredict34 = pd.DataFrame(data=predict34_unscaled,␣
 ↪columns=['date_block_num','shop_id','item_id','item_cnt_month'])
```

[126]:
```
dfPredict34['date_block_num'] = np.around(dfPredict34['date_block_num'])
dfPredict34['shop_id'] = np.around(dfPredict34['shop_id'])
dfPredict34['item_id'] = np.around(dfPredict34['item_id'])
```

```
dfPredict34 = dfPredict34.astype({'date_block_num':int,'shop_id':int,'item_id':
 ↪int})
```

[127]:
```
# Check for duplicates
dfPredict34[dfPredict34.duplicated(subset=['shop_id','item_id'])]
# This might be a rounding issue
# Looks good now that I rounded before casting to int
# I was getting an error that I could merge dataframes on a non-unique multi␣
 ↪index.
# The code in the above block fixed that issue
```

[127]:
```
Empty DataFrame
Columns: [date_block_num, shop_id, item_id, item_cnt_month]
Index: []
```

[134]:
```
# I am going to merge with test so I want to make shop_id and item_id the␣
 ↪indices
dfPredict34.set_index(['shop_id','item_id'],inplace=True)
```

[120]:
```
dfTest = pd.read_csv("../input/future-sales/test.csv")
```

[121]:
```
dfTest[dfTest.duplicated(subset=['shop_id','item_id'])]
# This df looks good on duplicated
```

[121]:
```
Empty DataFrame
Columns: [ID, shop_id, item_id]
Index: []
```

[129]:
```
dfTest.set_index(['shop_id','item_id'],inplace=True)
```

[130]:
```
dfTest
```

[130]:
```
                     ID
shop_id item_id
5       5037          0
        5320          1
        5233          2
        5232          3
        5268          4
…                     …
45      18454    214195
        16188    214196
        15757    214197
        19648    214198
        969      214199
```

```
[214200 rows x 1 columns]
```

[135]:
```
dfTest['item_cnt_month'] = dfPredict34['item_cnt_month']
```

[136]:
```
dfPredict34
```

[136]:
```
                 date_block_num  item_cnt_month
shop_id item_id
0       32                   33        0.021745
        33                   33        0.021752
        35                   33        0.021763
        43                   33        0.021813
        51                   33        0.021858
…                           …              …
36      12733                33       -0.100158
        13092                33       -0.100438
        16797                33       -0.102769
        18060                33       -0.103285
        15925                33       -0.102327

[1307160 rows x 2 columns]
```

[141]:
```
dfTest[dfTest['item_cnt_month'].isnull()]
# It looks like there are new iteams that are not in training
# and I will need to impute values

# This leads back to an issue that was missed during exploratory data analysis␣
 ↪(EDA)
# I created a dataframe with every item and shop listed for each month IF the␣
 ↪item was in the tranining data
# I did not use the items.csv for anything
# The test data had items that were not in the training data at all
# I will discuss this more in the analysis and results section
```

[141]:
```
                     ID  item_cnt_month
shop_id item_id
5       5320          1             NaN
        5268          4             NaN
        5826         45             NaN
        3731         54             NaN
        3538         64             NaN
…                   …              …
45      15033    214130             NaN
        7572     214150             NaN
        9030     214154             NaN
        1867     214161             NaN
        12470    214173             NaN
```

```
[15414 rows x 2 columns]
```

[142]:
```
dfTest.reset_index(inplace=True)
dfPredict34.reset_index(inplace=True)
```

[153]:
```
missing_items = dfTest[dfTest['item_cnt_month'].isnull()]['item_id'].unique()
```

[157]:
```
dfPredict34[dfPredict34['item_id'].isin(missing_items)]
```

[157]:
```
Empty DataFrame
Columns: [shop_id, item_id, date_block_num, item_cnt_month]
Index: []
```

[158]:
```
# Manual inspection shows that several missing items are similar to the next␣
 ↪item id over
# This is not always true but will use it for initial impute
# In many cases, it is the same game but on different platform
# A better impute would check the text string then compare with statistical␣
 ↪trends of the platform
# Is PS4 or Xbox more popular?
```

[158]:
```
367
```

[185]:
```
i = 0
for index,row in dfTest[dfTest['item_cnt_month'].isnull()].iterrows():
    item_id = row['item_id'].astype(int)
    # Try add one
    query_impute =␣
 ↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id +1 ) &␣
 ↪(dfPredict34['shop_id']==row['shop_id'])]
    if query_impute.shape[0] == 0: # Try  remove 1
        query_impute =␣
 ↪dfPredict34['item_cnt_month'][(dfPredict34['item_id']==item_id - 1 ) &␣
 ↪(dfPredict34['shop_id']==row['shop_id'])]
    dfTest.loc[index,'item_cnt_month'] = float(query_impute)
```

[186]:
```
# Check for NaNs now
dfTest[dfTest['item_cnt_month'].isnull()]
```

[186]:
```
Empty DataFrame
Columns: [shop_id, item_id, ID, item_cnt_month]
Index: []
```

[187]:
```
max(dfTest['item_cnt_month'])
```

[187]:
```
30.731201887130737
```

[62]:
```
min(dfTest['item_cnt_month'])
```

[62]:
```
-0.338518440723418
```

[188]:
```
dfTest.shape
```

[188]:
```
(214200, 4)
```

[189]:
```
dfTest
```

[189]:

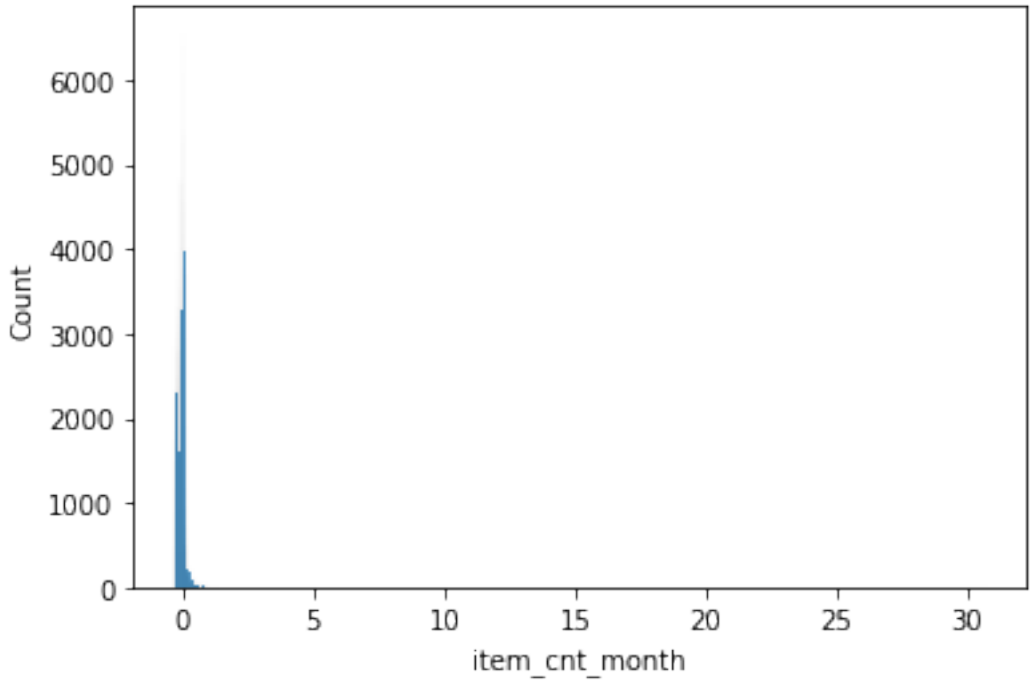
	shop_id	item_id	ID	item_cnt_month
0	5	5037	0	0.039118
1	5	5320	1	0.033375
2	5	5233	2	0.145055
3	5	5232	3	0.039855
4	5	5268	4	0.033375
…	…	…	…	…
214195	45	18454	214195	-0.071913
214196	45	16188	214196	-0.178715
214197	45	15757	214197	-0.177975
214198	45	19648	214198	-0.183713
214199	45	969	214199	-0.145012

[214200 rows x 4 columns]

[193]:
```
# Save CSV
dfTest[['ID','item_cnt_month']].to_csv("sample_submission.csv",index=False)
```

[190]:
```
sns.histplot(dfTest['item_cnt_month'])
```

[190]:
```
<AxesSubplot:xlabel='item_cnt_month', ylabel='Count'>
```



1.6 Analysis and Results

- The Kaggle Score for test data is 1.18078
  - I assume that is RMSE on test data which is similar to what I had on train
  - This puts me in the mid place on the leaderboard
- What went well
  - The general idea of the LSTM model showed promise for this problem
  - The scaled RMSE on the training data was very low
  - The training time over this very large model using a notebook with 30 GB of RAM and 8 cores was approximately 1 day
- What can be improved
  - A deeper network would improve performance
  - Using more epochs could improve performance. We limited our model to 20 epochs and used an early stop to cut that off if loss didn't improve in 3 epochs
  - The biggest issue is with feature value clustering and imputing values
  - We could improve the model by using unsupervised techniques to cluster items and shops
  - The items include categories that have some value but we need additional clustering
  - The item clusters are things such as Games - Xbox, Accessories - Xbox, Games - PS4 and various book, music, and video categories
  - These could help with imputing missing values in the test and training data
  - An issue with this type of data is that many of these items come out on a certain date, have a spike, then fade
  - To properly trend items we need to track trends on a few axis
  - Some of this could include natural language proceessing.
  - For example, if we have data on the sales for the release month of 2 different Call of

Duty games and we have new Call of Duty game in the test data, we can assume it will have similar results
  - In another example, if the release date of a game on Xbox was in November but was in December for PS4, we could use the Xbox data, scaled by platform popularity to estimate the value for the PS4 version.

1.7 References

- https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm
- https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- https://stackoverflow.com/questions/41233635/meaning-of-inter-op-parallelism-threads-and-intra-op-parallelism-threads