Capstone Project Technical Notebook - Knitting Pattern Recommender

1. Business Understanding

Ravelry.com is a database-driven website where users can browse and download knitting and crochet patterns, track their progress on a given project, and review the patterns.

It currently has a "your pattern highlights" recommender system. Compared to the front-and-centre recommendations that Netflix or Amazon make to their users, it's tucked away at the bottom of the patterns search page, displaying only thumbnail images of the recommended pattterns.

The recommendations generated appear to be based on clicks and/or favourites, rather than a more comprehensive examination of what projects users actually work on and rate positively, having experienced making them.

The aim of this project is to provide more tailored recommendations for knitting patterns to users of Ravelry.com, based on patterns they have worked on and rated already.

2. Data Understanding

The data has all been obtained through the Ravelry.com API. Since the website does not have an app, it makes all of its content available through APIs, and at present there are 41 apps which make use of some or all of the websites functionality.

The modelling features for a collaborative filtering recommender system are users, items and ratings. the model finds similarities between users based on their ratings of items, and uses these similarities to predict ratings for items for users who have not already rated them.

In [1]:

```
# import libraries

import surprise
from surprise.prediction_algorithms import *
import pandas as pd
import numpy as np
import datetime as dt
import requests
import json
import math
import random
```

In [2]:

```
# block pandas warnings

import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
# open credentials

with open('.secrets/creds.json') as f:
    creds = json.load(f)
```

In [4]:

```
# import data and drop null ratings

input_df = pd.read_csv('Data/saved_100000_calls.csv')
```

EDA:

In [14]:

```
print('Initially:', 100000, 'users sampled.')
print(len(pd.unique(input_df['user'])), 'of those had tracked', len(input_df), 'projects based on knitting patterns.')

input_df_non_NA = input_df.dropna(subset = ['rating'])

print(len(pd.unique(input_df_non_NA['user'])),
      'unique users have given',
      len(input_df_non_NA),
      'non-NA ratings to',
      len(pd.unique(input_df_non_NA['pattern_id'])),
      'unique knitting patterns (items).')
```

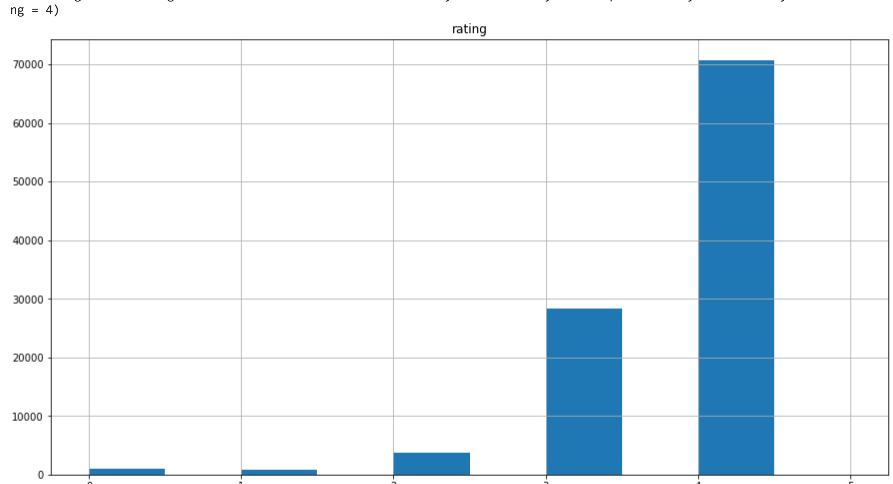
```
Initially: 100000 users sampled.
8531 of those had tracked 156206 projects based on knitting patterns.
4140 unique users have given 104710 non-NA ratings to 44459 unique knitting patterns (items).
```

In [15]:

```
input_df.dropna(subset = ['rating']).hist('rating', figsize = (15,8))

print('The histogram of ratings shows that users are far',
      ' more likely to favourably rate a pattern they have already chosen to work on, with',
      len(input_df_non_NA[input_df_non_NA['rating']>=3]),
      'of the',
      len(input_df_non_NA),
      'ratings a 3, or higher. (maximum rating = 4)')
```

```
The histogram of ratings shows that users are far  more likely to favourably rate a pattern they have already chosen to work on, with 99183 of the 104710 ratings a 3, or higher. (maximum rati
ng = 4)
```



In [16]:

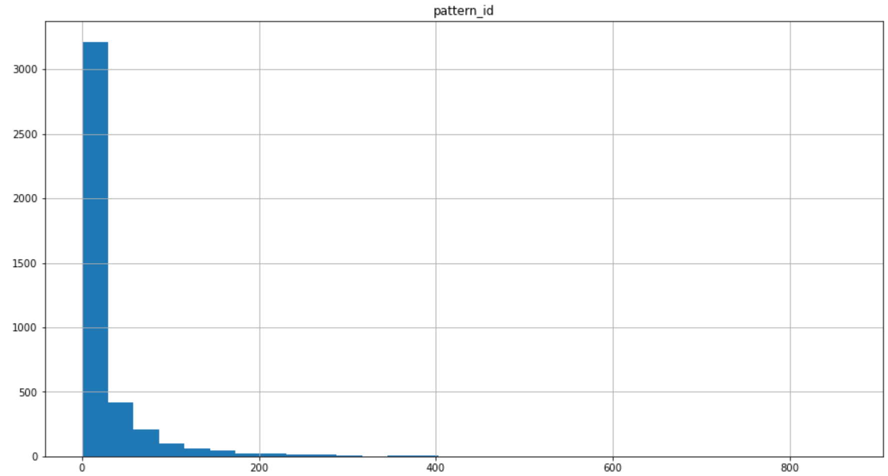
```
input_df_non_NA.groupby('user').count().hist('pattern_id',figsize = (15,8), bins = 30)

x = len(input_df_non_NA.groupby('user').count()[input_df_non_NA.groupby('user').count()['pattern_id'] <= 100])
y = len(pd.unique(input_df_non_NA['user']))
```

```
print(x, 'of the', y, 'rating users (', round(100*x/y,2),'%), tracked and rated 100 projects or fewer.')
z = len(input_df_non_NA.groupby('user').count()[input_df_non_NA.groupby('user').count()['pattern_id'] <= 10])

print(round(100*z/y,2), '% of all rating users tracked and rated 10 projects or fewer.')
```

```
3893 of the 4140 rating users ( 94.03 %), tracked and rated 100 projects or fewer.
60.7 % of all rating users tracked and rated 10 projects or fewer.
```



Note: Should I randomly sample input_df for each run? To get more variation in recommendations and speed up run time?

3. Data Preparation

The model was tested on the data with missing ratings removed, and then with missing ratings replaced with pattern averages. Replacing missing values with pattern averages negatively impacted the RMSE for SVD, so proceeded with the missing values dropped.

Also, one series of the dataframe contained lists which converted to strings after reading from the CSV. These were converted back to lists by slicing and splitting.

Lastly, the sk surprise package requires a specific version of the data to run its models, generated by passing the dataframe containing only user, item and rating (in that order) to the `load_from_df` and then `train_test_split` (for evaluation) or `build_full_trainset` (for final model) methods.

In [17]:
```
df_drop_nans = input_df[['user', 'pattern_id', 'rating']].dropna(subset = ['rating'])
```

In [18]:
```
df_drop_nans
```

Out[18]:

	user	pattern_id	rating
3	blumenkammer	572428	4.0
4	blumenkammer	879718	4.0
5	blumenkammer	911422	4.0
6	blumenkammer	766989	4.0
7	blumenkammer	743863	4.0
...	...	...	...
156194	MaryEtta	37411	0.0
156195	MaryEtta	169182	4.0
156196	MaryEtta	204902	4.0
156197	MaryEtta	476289	3.0
156198	MaryEtta	219239	3.0

104710 rows × 3 columns

In [20]:
```
len(list(df_drop_nans['pattern_id'].unique()))
```

Out[20]: 44459

In [37]:
```
# replace Nan ratings with pattern average. note - this negatively impacts RMSE, so the dataframe with dropped Nans is used
# for modelling

df_replace_nans = input_df[['user', 'pattern_id', 'rating', 'average_rating']]
rating_replace_nans = df_replace_nans['rating'].fillna(df_replace_nans['average_rating'])
df_replace_nans['rating'] = rating_replace_nans
df_replace_nans.drop(columns = 'average_rating', inplace = True)
```

In [38]:
```
# a list of users in the original data

users_list = list(df_drop_nans['user'].unique())
```

In [39]:
```
# a dataframe with a list of applicable categories (usually only one) for each pattern

# to-do: vectorize this for loop.

df_pattern_ids_and_categories = input_df[['pattern_id', 'categories']]
df_pattern_ids_and_categories = df_pattern_ids_and_categories.drop_duplicates(subset=['pattern_id'])
df_pattern_ids_and_categories['cat_list'] = ''
for pattern in list(df_pattern_ids_and_categories.index):
    df_pattern_ids_and_categories['cat_list'][pattern] = [category[1:-1] for category in df_pattern_ids_and_categories['categories'][pattern][1:-1].split(', ')]
```

In [40]:
```
# transform data for surprise

from surprise import Reader, Dataset

reader = Reader()

data_drop = Dataset.load_from_df(df_drop_nans, reader)
data_replace = Dataset.load_from_df(df_replace_nans, reader)

# train test split for model evaluation

from surprise.model_selection import train_test_split
```

```
drop_trainset, drop_testset = train_test_split(data_drop, test_size=0.25)
replace_trainset, replace_testset = train_test_split(data_replace, test_size=0.25)
```

4. Modelling

Many of the KNN, Matrix Factorization, Slope One, and Co-Clustering modelling methods within the sk surprise package were used, and evaluated based on their RMSE on predicted ratings, using a 25% train test split.

SVD - dropped NAs

In [41]:
```
from surprise import SVD, accuracy

SVD_1_drop = SVD(n_factors = 40, n_epochs = 45, lr_all = 0.002, reg_all = 0.2)
SVD_1_drop.fit(drop_trainset)

accuracy.rmse(SVD_1_drop.test(drop_testset))
```

```
RMSE: 0.6197
```
Out[41]:
```
0.6196842774387858
```

SVD - replaced NAs

In [42]:
```
SVD_1_replace = SVD(n_factors = 40, n_epochs = 45, lr_all = 0.002, reg_all = 0.2)
SVD_1_replace.fit(replace_trainset)

accuracy.rmse(SVD_1_replace.test(replace_testset))
```

```
RMSE: 0.8755
```
Out[42]:
```
0.8755475438211975
```

GridSearch on SVD - dropped NAs

In [ ]:
```
from surprise.model_selection import GridSearchCV

param_grid = {'n_factors':[5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              'n_epochs': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              'lr_all': [0.002, 0.003, 0.004, 0.005],
              'reg_all': [0.2, 0.3, 0.4, 0.5, 0.6]}

gs_model = GridSearchCV(SVD,
                        param_grid=param_grid,
                        n_jobs = -1,
                        joblib_verbose=5)

gs_model.fit(data_drop)

gs_model.best_params
```

In [43]:
```
# from GridSearch

GS_SVD = SVD(n_factors = 5, n_epochs = 40, lr_all = 0.002, reg_all = 0.2)
GS_SVD.fit(drop_trainset)

predictions = GS_SVD.test(drop_testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.6194
```
Out[43]:
```
0.6194194010544147
```

Single Variable Decomposition - SVD++

In [44]:
```
SVDppmodel = SVDpp(n_factors = 15, n_epochs = 30, lr_all = 0.003, reg_all = 0.2)

SVDppmodel.fit(drop_trainset)
predictions = SVDppmodel.test(drop_testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.6195
```
Out[44]:
```
0.6195019248684028
```

GridSearch on SVD++ - dropped NAs

In [ ]:
```
from surprise.model_selection import GridSearchCV

param_grid = {'n_factors':[5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              'n_epochs': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
              'lr_all': [0.002, 0.003, 0.004, 0.005],
              'reg_all': [0.2, 0.3, 0.4, 0.5, 0.6]}

gs_model = GridSearchCV(SVDpp,
                        param_grid=param_grid,
                        n_jobs = -1,
                        joblib_verbose=5)

gs_model.fit(data_drop)

gs_model.best_params
```

Other Surprise Models

In [45]:
```
from surprise.prediction_algorithms import knns
from surprise.similarities import cosine, msd, pearson
```

To-do: Try these 7 on the data_replace?

KNN Basic - cosine similarity

In [46]:
```
sim_cos = {'name':'cosine', 'user_based':True}

basic = knns.KNNBasic(min_k = 8, sim_options=sim_cos)
basic.fit(drop_trainset)
predictions = basic.test(drop_testset)
print(accuracy.rmse(predictions))
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 0.6738
0.6738285720096729
```

KNN Basic - Pearson similarity

In [47]:
```
sim_pearson = {'name':'pearson', 'user_based':True}

basic = knns.KNNBasic(min_k = 8, sim_options=sim_pearson)
basic.fit(drop_trainset)
predictions = basic.test(drop_testset)
print(accuracy.rmse(predictions))
```

```
Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
RMSE: 0.6737
0.6737225061045954
```

KNN with Means - Pearson similarity

In [48]:
```
sim_pearson = {'name':'pearson', 'user_based':True}

basic = knns.KNNWithMeans(min_k = 8, sim_options=sim_pearson)
basic.fit(drop_trainset)
predictions = basic.test(drop_testset)
print(accuracy.rmse(predictions))
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.6445
0.6444969457504647
```

KNN Baseline - Pearson similarity

In [49]:
```
sim_pearson = {'name':'pearson', 'user_based':True}

knn_baseline = knns.KNNBaseline(sim_options=sim_pearson)
knn_baseline.fit(drop_trainset)
predictions = knn_baseline.test(drop_testset)
print(accuracy.rmse(predictions))
```

```
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.6523
0.6523390669332535
```

Slope One

In [50]:
```
Slope_One = SlopeOne()

Slope_One.fit(drop_trainset)
predictions = Slope_One.test(drop_testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.7243
```

Out[50]:
```
0.7242588785217182
```

Co-Clustering

In [51]:
```
cocluster = CoClustering()

cocluster.fit(drop_trainset)
predictions = cocluster.test(drop_testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.7097
```

Out[51]:
```
0.7096841286360683
```

5. Evaluation

The models were tested on the ratings only for projects marked "finished", and then on ratings for projects of all statuses: "finished", "in-progress", "hibernating", and "frogged" [1]. The SVD model achieves the lowest RMSE in either case. While the RMSE is lower where only ratings for completed projects are input, this can lead to imbalanced data, as users are more likely to rate higeher projects which they have finished.

[1] "Frogged" refers to a project which was started and then un-knit, or ripped out. The word refers to the sound a frog makes: "ribbit, ribbit", or "Rip it, rip it".

In [52]:
```
# To do: maybe a voting classifier here?
```

In [53]:
```
# best model: grid searched parameters on SVD

best_model = SVD(n_factors = 15, n_epochs = 30, lr_all = 0.003, reg_all = 0.2)
```

In [54]:
```
# fit on entire dataset
from surprise.dataset import DatasetAutoFolds

trainset = DatasetAutoFolds.build_full_trainset(data_drop)

best_model.fit(trainset)
```

Out[54]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x25a594b2430>
```

6. Generate Predictions

The model is not yet deployed to a user interface. The functions below generate ratings predictions for items that users have not yet interacted with, either by tracking a project, adding a pattern to their queue, or favouriting a pattern.

Utilising only predicted ratings resulted in almost all users being recommended the same patterns: those that were highly rated in all cases. This did not achieve the type of tailored recommendations anticipated.

The categories of the user's projects: i.e. sweater, soft-toy, ankle-socks are obtained using the API and only those items matching their most frequently knit categories are returned from the function.

In [79]:
```
def get_user_projects(user):

    # a data frame of projects tracked by a given user

    try:

        url ='https://api.ravelry.com/projects/' + user + '/list.json?sort=completed_'
        response = requests.get(url, auth=(creds['id'], creds['key']))
        projects = []
        try:
            for project in response.json()['projects']:
                if project['craft_name'] == 'Knitting':
                    if project['pattern_id'] != None:
                        pattern_url ='https://api.ravelry.com/patterns.json?ids=' + str(int(project['pattern_id']))
                        pattern_response = requests.get(pattern_url, auth=(creds['id'], creds['key']))
                        project_tuple = (user, project['completed'], project['rating'], project['status_name'],
                                         project['pattern_id'],
                                         pattern_response.json()['patterns'][str(int(project['pattern_id']))]['rating_average'],
                                         pattern_response.json()['patterns'][str(int(project['pattern_id']))]['rating_count'],
                                         [attribute['permalink'] for attribute in pattern_response.json()['patterns'][str(int(project['pattern_id']))]['pattern_attributes']],
                                         [category['permalink'] for category in pattern_response.json()['patterns'][str(int(project['pattern_id']))]['pattern_categories']])
                        projects.append(project_tuple)
            df = pd.DataFrame(projects, columns = ['user', 'completed', 'rating', 'status', 'pattern_id', 'average_rating', 'rating_count', 'attributes', 'categories'])

        except ValueError:
            pass

    except ValueError:
        df = pd.DataFrame[{}]
    return df
```

In [67]:
```
def get_user_projects_not_finished(user):

    # a list of ongoing, frogged, or hibernated projects for a given user - only if user is already in modelling data

    users_projects_not_completed = requests.get('https://api.ravelry.com/projects/' + user + '/list.json',
                                               auth=(creds['id'], creds['key']))

    df = pd.DataFrame(users_projects_not_completed.json()['projects'])
    users_projects_not_completed = list(set(df[df['status_name'] != 'Finished']['pattern_id'].dropna()))

    return users_projects_not_completed
```

In [68]:
```
def get_user_queue(user):
    # a list of projects in a user's queue

    users_queue = requests.get('https://api.ravelry.com/people/' + user + '/queue/list.json?page_size=100',
                                               auth=(creds['id'], creds['key']))

    users_queue = list(set(pd.DataFrame(users_queue.json()['queued_projects'])['pattern_id'].dropna()))

    return users_queue
```

In [69]:
```
# a list of a patterns favourited by a given user

def get_user_favorites(user):

    users_favourites = requests.get('https://api.ravelry.com/people/' + user + '/favorites/list.json?page_size=100',
                                               auth=(creds['id'], creds['key']))

    df = pd.DataFrame(users_favourites.json()['favorites'])
    users_favourites = list(pd.DataFrame(list(df[df['type'] == 'pattern']['favorited']))['id'])

    return users_favourites
```

In [89]:
```
# returns patterns predicted to earn a rating of 3 or more for a given user

def top_rated(user):

    # if the user is already in the data, no need to refit model

    if user in users_list:

        # make a list of patterns in modelling data, remove any the user has previously interacted with, generate
        # predicted ratings for those patterns, output any greater than 3 to df

        patterns_list = list(input_df['pattern_id'].unique())

        predictions = []

        users_patterns = list(input_df[input_df['user'] == user]['pattern_id'])
        users_favourites = get_user_favorites(user)
        users_queue = get_user_queue(user)
        users_projects_not_completed = get_user_projects_not_finished(user)

        previously_interacted = users_patterns + users_favourites + users_queue + users_projects_not_completed

        remaining_patterns = [x for x in patterns_list if x not in previously_interacted]


        for pattern in remaining_patterns:
            x = best_model.predict(user, pattern)
            predictions.append(x)

        predictions_df = pd.DataFrame({"user": [prediction.uid for prediction in predictions],
                                       "item": [prediction.iid for prediction in predictions],
                                       "estimated" :[prediction.est for prediction in predictions]})

        predictions_df = predictions_df[predictions_df['estimated'] > 3]
        predictions_df = predictions_df.sort_values('estimated', ascending = False)

        return predictions_df

    elif user not in users_list:

        # get user data to match modelling data, transform to match, and refit model with that user included.

        try:

            new_user_ratings = get_user_projects(user)
            new_user_input_df = input_df.append(new_user_ratings).reset_index().drop(columns = 'index')

            df_drop_nans_new_user = new_user_input_df[['user', 'pattern_id', 'rating']].dropna(subset = ['rating'])

            reader = Reader()
            data_drop_new_user = Dataset.load_from_df(df_drop_nans_new_user, reader)
            trainset_new_user = DatasetAutoFolds.build_full_trainset(data_drop_new_user)

            best_model.fit(trainset_new_user)

            # make a list of patterns in modelling data, remove any the user has previously interacted with, generate
            # predicted ratings for those patterns, output any greater than 3 to df

            patterns_list = list(new_user_input_df['pattern_id'].unique())
            predictions = []

            users_patterns = list(new_user_input_df[new_user_input_df['user'] == user]['pattern_id'])
            users_favourites = get_user_favorites(user)
            users_queue = get_user_queue(user)
            users_projects_not_completed = list(set(new_user_ratings[new_user_ratings['status'] != 'Finished']['pattern_id'].dropna()))

            previously_interacted = users_patterns + users_favourites + users_queue + users_projects_not_completed

            remaining_patterns = [pattern for pattern in patterns_list if pattern not in previously_interacted]

            for pattern in remaining_patterns:
                x = best_model.predict(user, pattern)
                predictions.append(x)

            predictions_df = pd.DataFrame({"user": [prediction.uid for prediction in predictions],
                                           "item": [prediction.iid for prediction in predictions],
                                           "estimated": [prediction.est for prediction in predictions]})

            predictions_df = predictions_df[predictions_df['estimated'] > 3]
            predictions_df = predictions_df.sort_values('estimated', ascending = False)

            return predictions_df

        except:

            patterns_list = list(input_df['pattern_id'].unique())
            df_non_user = []
```

```
        random_sample_patterns = random.sample(patterns_list, 8)

        for x in random_sample_patterns:
            rating = input('How do you rate pattern ' + str(x) + '?')
            df_non_user.append({'user': 'new_user', 'pattern_id': x, 'rating': rating})

        df_non_user = pd.DataFrame(df_non_user)
        df_drop_nans.append(df_non_user).reset_index().drop(columns = 'index')

        reader = Reader()
        data_drop_non_user = Dataset.load_from_df(df_drop_nans, reader)
        trainset_non_user = DatasetAutoFolds.build_full_trainset(data_drop_non_user)

        best_model.fit(trainset_non_user)
        # generate predicted ratings for those patterns, output any greater than 3 to df

        previously_interacted = random_sample_patterns
        remaining_patterns = [pattern for pattern in patterns_list if pattern not in previously_interacted]

        predictions = []

        for pattern in remaining_patterns:
            x = best_model.predict('new_user', pattern)
            predictions.append(x)

        predictions_df = pd.DataFrame({"user": [prediction.uid for prediction in predictions],
                                       "item": [prediction.iid for prediction in predictions],
                                       "estimated": [prediction.est for prediction in predictions]})

        predictions_df = predictions_df[predictions_df['estimated'] > 3]
        predictions_df = predictions_df.sort_values('estimated', ascending = False)

        return predictions_df
```

In [90]:

```
# return a list of the users most frequently knitted types of patterns (i.e. scarves, toys, cardigans...)

def user_fave_categories(user):

    try:

        user_projects = get_user_projects(user)

        user_projects['cat'] = ''
        for project in range(0,len(user_projects)):
            user_projects['cat'][project] = user_projects['categories'][project].sort()
            for category in range(0,len(user_projects['categories'][project])):
                user_projects['cat'][project] = user_projects['categories'][project][category]

        df_count_categories = user_projects.groupby('cat').count().sort_values('user', ascending = False)
        df_count_categories = df_count_categories.reset_index()[['cat', 'user']]
        if len(df_count_categories) <= 5:
            favorite_categories = list(df_count_categories['cat'])
        elif len(df_count_categories) <=20:
            favorite_categories = list(df_count_categories.head(5)['cat'])
        elif len(df_count_categories) > 20:
            favorite_categories = list(df_count_categories.head(math.ceil(len(df_count_categories)/5))['cat'])

    except:
        favorite_categories = []

    return favorite_categories
```

In [102…

```
def get_recommendations(user):

    if len(user_fave_categories(user)) != 0:

        fave_categories = user_fave_categories(user)

        # merge df of user recommendations with input df containing item categories

        recs = top_rated(user)
        recs['pattern_id'] = recs['item']
        result = pd.merge(df_pattern_ids_and_categories, recs, how="inner", on=["pattern_id"])


        # drop any recommendations not corresponding to users top categories

        result['favourites_list'] = ''
        for rec in list(result.index):
            if len(list(set(result['cat_list'][rec]).intersection(set(fave_categories)))) != 0:
                result['favourites_list'][rec] = 1
            else:
                result['favourites_list'][rec] = 0

        result = result[result['favourites_list'] != 0]
        result = result.sort_values('estimated', ascending = False).head(15)

        recommendations = []

        # get pattern name and generate url

        for pattern in list(result['item']):

            pattern_url ='https://api.ravelry.com/patterns.json?ids=' + str(pattern)
            pattern_response = requests.get(pattern_url, auth=(creds['id'], creds['key']))
            recommendations.append('ravelry.com/patterns/library/' + str(pattern_response.json()['patterns'][str(pattern)]['permalink']))

        return recommendations

    elif len(user_fave_categories(user)) == 0:

        recs = top_rated(user)
        recs['pattern_id'] = recs['item']
        result = pd.merge(df_pattern_ids_and_categories, recs, how="inner", on=["pattern_id"])

        # drop any recommendations not corresponding to users top categories

        result = result.sort_values('estimated', ascending = False).head(15)

        recommendations = []

        # get pattern name and generate url

        for pattern in list(result['item']):

            pattern_url ='https://api.ravelry.com/patterns.json?ids=' + str(pattern)
            pattern_response = requests.get(pattern_url, auth=(creds['id'], creds['key']))
            recommendations.append('ravelry.com/patterns/library/' + str(pattern_response.json()['patterns'][str(pattern)]['permalink']))
```

```
        return recommendations
```

In [104…

```
get_recommendations('alpal')
```

```
How do you rate pattern 158410?4
How do you rate pattern 458426?2
How do you rate pattern 426148?1
How do you rate pattern 59066?3
How do you rate pattern 28060?4
How do you rate pattern 1941?1
How do you rate pattern 57660?1
How do you rate pattern 925886?2
```

Out[104…

```
['ravelry.com/patterns/library/pussyhat-4-gauges-in-the-round',
 'ravelry.com/patterns/library/botinhas-de-bebe',
 'ravelry.com/patterns/library/1898-hat',
 'ravelry.com/patterns/library/tiong-bahru',
 'ravelry.com/patterns/library/clapo-ktus',
 'ravelry.com/patterns/library/nubbins-dishcloth',
 'ravelry.com/patterns/library/getting-warmer',
 'ravelry.com/patterns/library/198-37-clean--colourful',
 'ravelry.com/patterns/library/the-almost-lost-washcloth',
 'ravelry.com/patterns/library/lady-eleanor-entrelac-stole',
 'ravelry.com/patterns/library/vintage-bubble-bag',
 'ravelry.com/patterns/library/bunny-in-a-dotty-dress',
 'ravelry.com/patterns/library/hootin-owlie-hat',
 'ravelry.com/patterns/library/lake-reed',
 'ravelry.com/patterns/library/basic-sock-pattern-in-8-sizes']
```

In [105…

```
get_recommendations('clare240')
```

Out[105…

```
['ravelry.com/patterns/library/pussyhat-4-gauges-in-the-round',
 'ravelry.com/patterns/library/getting-warmer',
 'ravelry.com/patterns/library/tiong-bahru',
 'ravelry.com/patterns/library/clapo-ktus',
 'ravelry.com/patterns/library/lady-eleanor-entrelac-stole',
 'ravelry.com/patterns/library/aeolian-shawl',
 'ravelry.com/patterns/library/lake-reed',
 'ravelry.com/patterns/library/blooming-stitch-shawl',
 'ravelry.com/patterns/library/hootin-owlie-hat',
 'ravelry.com/patterns/library/nightshift',
 'ravelry.com/patterns/library/laminaria',
 'ravelry.com/patterns/library/tulips-a-colorful-cardigan-for-baby',
 'ravelry.com/patterns/library/ferocious-briocheous',
 'ravelry.com/patterns/library/oaklet-shawl',
 'ravelry.com/patterns/library/revontuli--huivi-northern-lights']
```

Next Steps

1. User interface using streamlit - in progress
2. Cold start recommendations for non-users or users with no existing ratings - complete
3. Expand to crocheters, weavers.
4. Layer more content based filtration: attributes (v-neck, seamless, toddler-sized) in addition to categories.
5. Keep tweaking models to improve RMSE.

In [ ]: