assignment-3

April 20, 2024

[286]:
```
import pandas as pd
import numpy as np
import pickle
from datetime import datetime
import re

from tqdm import tqdm
tqdm.pandas()

import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

data = pd.read_csv(r'data.csv')
df = pd.DataFrame(data)
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\cavit\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

[287]:
```
df.head(3)
df.tail(3)
df.shape
df.columns
```

[287]:
```
Index(['beer_ABV', 'beer_beerId', 'beer_brewerId', 'beer_name', 'beer_style',
       'review_appearance', 'review_palette', 'review_overall', 'review_taste',
       'review_profileName', 'review_aroma', 'review_text', 'review_time'],
      dtype='object')
```

[288]:
```
df.describe
```

[288]:
```
<bound method NDFrame.describe of       beer_ABV  beer_beerId  beer_brewerId
beer_name  \
0          5.0        47986          10325            Sausa Weizen
1          6.2        48213          10325                Red Moon
2          6.5        48215          10325  Black Horse Black Beer
3          5.0        47969          10325              Sausa Pils
```

4	7.7	64883	1075	Cauldron DIPA
…	…	…	…	…
1606	10.5	3635	22	La Terrible
1607	10.5	3635	22	La Terrible
1608	10.5	3635	22	La Terrible
1609	10.5	3635	22	La Terrible
1610	10.5	3635	22	La Terrible

	beer_style	review_appearance	review_palette	\
0	Hefeweizen	2.5	2.0	
1	English Strong Ale	3.0	2.5	
2	Foreign / Export Stout	3.0	2.5	
3	German Pilsener	3.5	3.0	
4	American Double / Imperial IPA	4.0	4.5	
…	…	…	…	
1606	Belgian Strong Dark Ale	3.5	3.5	
1607	Belgian Strong Dark Ale	4.0	4.0	
1608	Belgian Strong Dark Ale	3.5	4.0	
1609	Belgian Strong Dark Ale	4.0	4.0	
1610	Belgian Strong Dark Ale	4.0	3.5	

	review_overall	review_taste	review_profileName	review_aroma	\
0	1.5	1.5	stcules	1.5	
1	3.0	3.0	stcules	3.0	
2	3.0	3.0	stcules	3.0	
3	3.0	2.5	stcules	3.0	
4	4.0	4.0	johnmichaelsen	4.5	
…	…	…	…	…	
1606	4.0	4.0	bump8628	4.0	
1607	4.0	3.5	StlHopHead77	4.0	
1608	3.5	4.5	weazal	4.0	
1609	4.5	4.5	GRG1313	4.5	
1610	3.0	3.0	coldmeat23	4.0	

	review_text	review_time
0	A lot of foam. But a lot. In the smell some ba…	1234817823
1	Dark red color, light beige foam, average. In …	1235915097
2	Almost totally black. Beige foam, quite compac…	1235916604
3	Golden yellow color. White, compact foam, quit…	1234725145
4	According to the website, the style for the Ca…	1293735206
…	…	…
1606	Nice surprise to find this on draft locally, a…	1319036708
1607	A-Pours a somewhere between darkest possible b…	1318246936
1608	S: Dark and rich. Primarily raisins and malt, …	1246942776
1609	Pours very dark brown with orange tint. Big ca…	1246936211
1610	GLASS: Snifter TEMP: Cellared @ approx 45 degr…	1246541885

```
[1611 rows x 13 columns]>
```

[289]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1611 entries, 0 to 1610
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   beer_ABV            1580 non-null   float64
 1   beer_beerId         1611 non-null   int64
 2   beer_brewerId       1611 non-null   int64
 3   beer_name           1611 non-null   object
 4   beer_style          1611 non-null   object
 5   review_appearance   1611 non-null   float64
 6   review_palette      1611 non-null   float64
 7   review_overall      1611 non-null   float64
 8   review_taste        1611 non-null   float64
 9   review_profileName  1611 non-null   object
 10  review_aroma        1611 non-null   float64
 11  review_text         1611 non-null   object
 12  review_time         1611 non-null   int64
dtypes: float64(6), int64(3), object(4)
memory usage: 163.7+ KB
```

[290]:
```python
df.columns
```

[290]:
```
Index(['beer_ABV', 'beer_beerId', 'beer_brewerId', 'beer_name', 'beer_style',
       'review_appearance', 'review_palette', 'review_overall', 'review_taste',
       'review_profileName', 'review_aroma', 'review_text', 'review_time'],
      dtype='object')
```

[291]:
```python
# check unique cols in df
for col in df.columns:
    if df[col].is_unique:
        print(f'Unique Column : {col} ')
```

```
Unique Column : review_text
Unique Column : review_time
```

[292]:
```python
# reset indexes
df = df.reset_index()
```

Null values

[293]:
```python
# check null counts
df.isnull().sum()
```

[293]:
```
index                 0
beer_ABV             31
beer_beerId           0
beer_brewerId         0
beer_name             0
beer_style            0
review_appearance     0
review_palette        0
review_overall        0
review_taste          0
review_profileName    0
review_aroma          0
review_text           0
review_time           0
dtype: int64
```

[294]:
```
# drop null values
df = df.dropna()
df.isnull().sum()
```

[294]:
```
index                0
beer_ABV             0
beer_beerId          0
beer_brewerId        0
beer_name            0
beer_style           0
review_appearance    0
review_palette       0
review_overall       0
review_taste         0
review_profileName   0
review_aroma         0
review_text          0
review_time          0
dtype: int64
```

[295]:
```
df.shape
```

[295]:
```
(1580, 14)
```

Remove duplicate data

[296]:
```
data.review_profileName.head()
```

[296]:
```
0            stcules
1            stcules
2            stcules
```

```
3          stcules
4    johnmichaelsen
Name: review_profileName, dtype: object
```

[297]:
```
# sort by "review_overall" in descending order
df = df.sort_values('review_overall', ascending=False)

# keep the highest rating from each "review_profilename" and drop the rest
df = df.drop_duplicates(subset= ['review_profileName','beer_beerId'],␣
↪keep='first')
df.shape
```

[297]:
```
(1574, 14)
```

1. Rank top 3 Breweries which produce the strongest beers?

[298]:
```
# group by brewerId and calculate the average ABV for each brewery
brewery_avg_abv = df.groupby('beer_brewerId')['beer_ABV'].mean()

# sort breweries by average ABV in descending order and select the top 3
top_3_breweries = brewery_avg_abv.sort_values(ascending=False).head(3)

print("Top 3 Breweries Producing the Strongest Beers:")
print(top_3_breweries)
```

```
Top 3 Breweries Producing the Strongest Beers:
beer_brewerId
22      10.500000
694     10.100000
2724     7.643243
Name: beer_ABV, dtype: float64
```

2. Which year did beers enjoy the highest ratings?

[299]:
```
# convert review_time to datetime
df['review_time'] = pd.to_datetime(df['review_time'], unit='s')

# extract year from review_time
df['year'] = df['review_time'].dt.year

# group by year and calculate the average rating for each year
average_ratings_by_year = df.groupby('year')['review_overall'].mean()

# find the year with the highest average rating
highest_rated_year = average_ratings_by_year.idxmax()

print("Year with the highest average ratings for beers:", highest_rated_year)
```

```
Year with the highest average ratings for beers: 2012
```

3. Based on the user's ratings which factors are important among taste, aroma, appearance, and palette?

[300]:
```python
# Calculate correlation matrix
correlation_matrix = df[['review_taste', 'review_aroma', 'review_appearance',␣
↪'review_palette', 'review_overall']].corr()

# Extract correlations with review_overall
correlations_with_overall = correlation_matrix['review_overall'].
↪drop('review_overall')

# Sort correlations in descending order
sorted_correlations = correlations_with_overall.sort_values(ascending=False)

print("Correlation between each factor and overall review rating:")
print(sorted_correlations)
```

```
Correlation between each factor and overall review rating:
review_aroma         0.846082
review_taste         0.783294
review_palette       0.739443
review_appearance    0.657417
Name: review_overall, dtype: float64
```

so review_aroma has highest corellation which is important

4. If you were to recommend 3 beers to your friends based on this data which ones will you recommend? * need to edit

[301]:
```python
# assigning custom weights
weights = {'review_overall': 0.4, 'review_taste': 0.2, 'review_aroma': 0.1,␣
↪'review_appearance': 0.1, 'review_palette': 0.2}
df['weighted_rating'] = (df[list(weights.keys())] * pd.Series(weights)).
↪sum(axis=1)

# sort beers by weighted rating in descending order
recommended_beers = df.sort_values(by='weighted_rating', ascending=False).
↪head(3)

print("Recommended beers for my friends:")
#print(recommended_beers[['beer_name', 'weighted_rating', 'review_text']])
recommended_beers[['beer_name', 'weighted_rating', 'review_text']].head(3)
```

```
Recommended beers for my friends:
```

[301]:
```
                        beer_name  weighted_rating  \
1533          T.J.'s Best Bitter             5.00
```

```
433                  Caldera IPA            5.00
281   Old Growth Imperial Stout             4.95

                                          review_text
1533  Holy crap. This beer is amazing. Wow. Holy cra…
433   12 oz can poured into duvel snifter A - pours …
281   Aroma is absolutely heavenly - smoky with firm…
```

how the weights were decided:

Review Overall: represents the overall review rating given by users. Since it reflects the overall satisfaction with the beer; highest weight of 0.4 Review Taste: taste is a crucial aspect of beer enjoyment; 0.2, reflecting its importance in the overall rating Review Aroma: aroma contributes significantly to the sensory experience of drinking beer, but it may be slightly less important than taste; 0.1 Review Appearance: can influence the initial impression of a beer, overall enjoyment may be lower compared to taste and aroma; 0.1 Review Palette: mouthfeel or texture of the beer; 0.2

5. Which Beer style seems to be the favorite based on reviews written by users?, 6. How does written review compare to overall review score for the beer styles?

[302]:
```
# taking relevant columns
reviewTextData =␣
↪data[['beer_beerId','beer_name','beer_ABV','beer_style','review_overall','review_text']]

# taking higher ranked reviews only >/=4 (from the overall reviews column)
reviewTextData = reviewTextData.loc[reviewTextData['review_overall'] >= 4]

# resetting Index
reviewTextData.reset_index(drop=True,inplace=True)

reviewTextData.head()
```

[302]:
```
   beer_beerId           beer_name  beer_ABV                      beer_style  \
0        64883        Cauldron DIPA       7.7  American Double / Imperial IPA
1        52159  Caldera Ginger Beer       4.7             Herbed / Spiced Beer
2        52159  Caldera Ginger Beer       4.7             Herbed / Spiced Beer
3        52159  Caldera Ginger Beer       4.7             Herbed / Spiced Beer
4        52159  Caldera Ginger Beer       4.7             Herbed / Spiced Beer

   review_overall                                        review_text
0             4.0  According to the website, the style for the Ca…
1             4.0  I'm not sure why I picked this up… I like gi…
2             4.5  Poured from a 22oz bomber into my Drie Fontein…
3             5.0  OK, so the only reason I bought this while sho…
4             4.0  Notes from 6/24 A: Bright golden glowing beer …
```

[303]:
```
reviewTextData.review_text[0]
```

[303]: "According to the website, the style for the Caldera Cauldron changes every year. The current release is a DIPA, which frankly is the only cauldron I'm familiar with (it was an IPA/DIPA the last time I ordered a cauldron at the horsebrass several years back). In any event… at the Horse Brass yesterday. The beer pours an orange copper color with good head retention and lacing. The nose is all hoppy IPA goodness, showcasing a huge aroma of dry citrus, pine and sandlewood. The flavor profile replicates the nose pretty closely in this West Coast all the way DIPA. This DIPA is not for the faint of heart and is a bit much even for a hophead like myslf. The finish is quite dry and hoppy, and there's barely enough sweet malt to balance and hold up the avalanche of hoppy bitterness in this beer. Mouthfeel is actually fairly light, with a long, persistentely bitter finish. Drinkability is good, with the alcohol barely noticeable in this well crafted beer. Still, this beer is so hugely hoppy/bitter, it's really hard for me to imagine ordering more than a single glass. Regardless, this is a very impressive beer from the folks at Caldera."

[304]:
```python
# text preprocessing
import re

# initial text processing replacing short forms
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"it\'s", "it is", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    return phrase
```

[305]:
```python
# extracting text reviews and applying text preprocessing on it
preprocessed_reviews = []

for sentance in tqdm(reviewTextData['review_text'].values): # tqdm prints the␣
 ↪status bar
    sentance = decontracted(sentance) # deconstructiong short forms
    sentance = re.sub("\S*\d\S*", "", sentance).strip() # remove words with␣
 ↪numbers
```

```
    preprocessed_reviews.append(sentance) # form sentence again
```

```
100%|      | 835/835 [00:00<00:00, 12732.65it/s]
```

[306]:
```
preprocessed_reviews[0]
```

[306]:
```
'According to the website, the style for the Caldera Cauldron changes every
year. The current release is a DIPA, which frankly is the only cauldron I am
familiar with (it was an IPA/DIPA the last time I ordered a cauldron at the
horsebrass several years back). In any event… at the Horse Brass yesterday.
The beer pours an orange copper color with good head retention and lacing. The
nose is all hoppy IPA goodness, showcasing a huge aroma of dry citrus, pine and
sandlewood. The flavor profile replicates the nose pretty closely in this West
Coast all the way DIPA. This DIPA is not for the faint of heart and is a bit
much even for a hophead like myslf. The finish is quite dry and hoppy, and there
is barely enough sweet malt to balance and hold up the avalanche of hoppy
bitterness in this beer. Mouthfeel is actually fairly light, with a long,
persistentely bitter finish. Drinkability is good, with the alcohol barely
noticeable in this well crafted beer. Still, this beer is so hugely
hoppy/bitter, it is really hard for me to imagine ordering more than a single
glass. Regardless, this is a very impressive beer from the folks at Caldera.'
```

[307]:
```
# appending preprocessed reviews to the filtered dataframe
reviewTextData['preprocessed_review_text'] = preprocessed_reviews
```

[308]:
```
# instantiating Sentiment Analyzer
sianalyzer = SentimentIntensityAnalyzer()

# loop over the 'preprocessed_review_text' column and calculate the polarity␣
↪score for each review
reviewTextData['polarity_score2'] = reviewTextData['preprocessed_review_text'].
↪progress_apply(lambda x: sianalyzer.polarity_scores(x)['compound'])
```

```
100%|      | 835/835 [00:00<00:00, 942.40it/s]
```

[309]:
```
# grouping and calculate mean polarity score
reviewTextDataGroupped = reviewTextData.
↪groupby('beer_style')['polarity_score2'].mean()

# sort the grouped data by mean polarity score
reviewTextDataGroupped.sort_values(ascending=False)[0:5]
```

[309]:
```
beer_style
Dortmunder / Export Lager    0.9826
English Porter               0.9668
American Blonde Ale          0.9659
Märzen / Oktoberfest         0.9626
```

```
Cream Ale                    0.9587
Name: polarity_score2, dtype: float64
```

[310]:
```
# observing the top 'polarity_score2' and 'beer_beerId' associated with i
reviewTextData.loc[reviewTextData['beer_style'] == 'Dortmunder / Export Lager']
reviewTextData.loc[reviewTextData['beer_style'] == 'American Blonde Ale']
```

[310]:
```
     beer_beerId                          beer_name  beer_ABV  \
225        61427  Caldera Rose Petal (Kettle Series)       6.7
226        61427  Caldera Rose Petal (Kettle Series)       6.7
810        38275                  Alaskan Summer Ale       5.5

              beer_style  review_overall  \
225  American Blonde Ale             4.0
226  American Blonde Ale             4.0
810  American Blonde Ale             4.0

                                           review_text  \
225  A- is cloudy and light glassy goldeness S- sme…
226  It's a beautiful beer to look at. Pours crysta…
810  A: Poured a straw yellow color with a 1 finger…

                              preprocessed_review_text  polarity_score2
225  A- is cloudy and light glassy goldeness S- sme…           0.9693
226  It is a beautiful beer to look at. Pours cryst…           0.9827
810  A: Poured a straw yellow color with a  finger …           0.9457
```

5. By observing the mean compound polarity score , we can say that the beer style "Dortmunder / Export Lager" is liked most but has only one person that likes it as much, we can instead say "American Blonde Ale" is the most famous, based on combination of polarity and higher frequency

6. By observing the mean compound polarity score calculated we can get an idea how the user written review text is collaborating in calculating the overall review score

7. How to find similar beer drinkers by using written reviews only?

[311]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

[312]:
```
reviewTextData.columns
```

[312]:
```
Index(['beer_beerId', 'beer_name', 'beer_ABV', 'beer_style', 'review_overall',
       'review_text', 'preprocessed_review_text', 'polarity_score2'],
      dtype='object')
```

[313]:
```
# feature Extraction
# initialize TF-IDF vectorizer
```

```
tfidf_vectorizer = TfidfVectorizer()

# fit and transform the preprocessed text data to create TF-IDF features
tfidf_matrix = tfidf_vectorizer.
↪fit_transform(reviewTextData['preprocessed_review_text'])

# similarity calculation
# calculate cosine similarity between user reviews
cosine_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

[314]:
```
from sklearn.cluster import KMeans
```

[315]:
```
# grouping together similiar customers based on reviews
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(cosine_similarities)

# analyze cluster assignments
# assign each user to a cluster
user_clusters = {}

for user_id, cluster_id in enumerate(clusters):
    if cluster_id not in user_clusters:
        user_clusters[cluster_id] = []
    user_clusters[cluster_id].append(user_id)

# print the users in each cluster
for cluster_id, users in user_clusters.items():
    print(f"Cluster {cluster_id}: {users}")
```

```
c:\Users\cavit\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
Cluster 2: [0, 3, 12, 24, 25, 31, 32, 36, 40, 47, 50, 54, 59, 64, 65, 67, 74,
75, 76, 77, 79, 90, 93, 94, 95, 96, 102, 108, 112, 114, 116, 117, 118, 124, 127,
130, 132, 137, 142, 145, 156, 158, 171, 174, 177, 178, 182, 194, 199, 209, 210,
221, 222, 224, 230, 234, 237, 240, 247, 248, 250, 252, 257, 261, 263, 264, 265,
270, 271, 276, 277, 278, 279, 281, 283, 284, 285, 289, 290, 291, 296, 298, 299,
302, 303, 304, 305, 306, 307, 311, 317, 319, 324, 326, 328, 329, 333, 335, 337,
341, 342, 343, 344, 350, 352, 353, 354, 355, 356, 357, 358, 360, 361, 363, 365,
366, 370, 371, 375, 377, 380, 382, 385, 386, 388, 393, 399, 401, 402, 403, 407,
408, 409, 410, 430, 434, 438, 445, 449, 452, 453, 455, 458, 462, 463, 464, 465,
466, 467, 468, 470, 473, 474, 479, 487, 492, 495, 496, 497, 506, 508, 511, 512,
513, 515, 517, 520, 523, 525, 528, 531, 532, 536, 546, 549, 553, 559, 561, 569,
573, 574, 576, 579, 581, 584, 585, 587, 604, 606, 607, 610, 611, 615, 616, 617,
621, 630, 631, 632, 633, 636, 638, 639, 641, 644, 646, 648, 650, 688, 698, 700,
```

706, 707, 710, 720, 730, 735, 759, 773, 781, 789, 815, 827, 830]
Cluster 1: [1, 4, 8, 10, 11, 13, 14, 19, 22, 26, 27, 28, 29, 33, 35, 39, 41, 43, 44, 46, 48, 51, 56, 63, 71, 73, 78, 81, 83, 98, 106, 110, 113, 115, 120, 121, 122, 125, 126, 131, 139, 144, 148, 149, 153, 155, 161, 162, 163, 165, 166, 167, 168, 170, 172, 173, 175, 176, 185, 186, 187, 188, 192, 193, 195, 198, 200, 201, 203, 205, 207, 208, 211, 213, 215, 218, 219, 220, 223, 227, 228, 229, 231, 238, 245, 246, 249, 253, 254, 256, 259, 266, 274, 275, 280, 293, 297, 301, 308, 314, 318, 332, 334, 339, 347, 351, 372, 376, 378, 383, 390, 397, 415, 423, 427, 429, 436, 439, 442, 443, 450, 461, 480, 481, 486, 494, 498, 503, 516, 518, 526, 535, 538, 539, 540, 542, 544, 547, 551, 552, 554, 555, 556, 558, 560, 566, 575, 589, 590, 593, 595, 597, 599, 603, 605, 612, 613, 618, 619, 625, 628, 629, 640, 642, 643, 651, 653, 654, 655, 656, 657, 660, 665, 666, 667, 668, 671, 672, 674, 678, 680, 681, 682, 683, 684, 686, 687, 690, 692, 694, 695, 696, 697, 702, 704, 705, 709, 711, 714, 715, 718, 722, 725, 728, 731, 733, 736, 738, 739, 740, 741, 742, 745, 746, 747, 748, 750, 751, 753, 754, 755, 761, 762, 763, 765, 768, 769, 770, 774, 780, 783, 784, 788, 790, 791, 793, 795, 797, 798, 799, 800, 801, 802, 803, 804, 806, 809, 810, 811, 812, 813, 814, 817, 819, 823, 824, 826, 828, 829, 831, 832, 833, 834]
Cluster 0: [2, 5, 6, 7, 9, 15, 16, 17, 18, 20, 21, 23, 30, 34, 37, 38, 42, 45, 49, 52, 53, 55, 57, 58, 60, 61, 62, 66, 68, 69, 70, 72, 80, 82, 84, 85, 86, 87, 88, 89, 91, 92, 97, 99, 100, 101, 103, 104, 105, 107, 109, 111, 119, 123, 128, 129, 133, 134, 135, 136, 138, 140, 141, 143, 146, 147, 150, 151, 152, 154, 157, 159, 160, 164, 169, 179, 180, 181, 183, 184, 189, 190, 191, 196, 197, 202, 204, 206, 212, 214, 216, 217, 225, 226, 232, 233, 235, 236, 239, 241, 242, 243, 244, 251, 255, 258, 260, 262, 267, 268, 269, 272, 273, 282, 286, 287, 288, 292, 294, 295, 300, 309, 310, 312, 313, 315, 316, 320, 321, 322, 323, 325, 327, 330, 331, 336, 338, 340, 345, 346, 348, 349, 359, 362, 364, 367, 368, 369, 373, 374, 379, 381, 384, 387, 389, 391, 392, 394, 395, 396, 398, 400, 404, 405, 406, 411, 412, 413, 414, 416, 417, 418, 419, 420, 421, 422, 424, 425, 426, 428, 431, 432, 433, 435, 437, 440, 441, 444, 446, 447, 448, 451, 454, 456, 457, 459, 460, 469, 471, 472, 475, 476, 477, 478, 482, 483, 484, 485, 488, 489, 490, 491, 493, 499, 500, 501, 502, 504, 505, 507, 509, 510, 514, 519, 521, 522, 524, 527, 529, 530, 533, 534, 537, 541, 543, 545, 548, 550, 557, 562, 563, 564, 565, 567, 568, 570, 571, 572, 577, 578, 580, 582, 583, 586, 588, 591, 592, 594, 596, 598, 600, 601, 602, 608, 609, 614, 620, 622, 623, 624, 626, 627, 634, 635, 637, 645, 647, 649, 652, 658, 659, 661, 662, 663, 664, 669, 670, 673, 675, 676, 677, 679, 685, 689, 691, 693, 699, 701, 703, 708, 712, 713, 716, 717, 719, 721, 723, 724, 726, 727, 729, 732, 734, 737, 743, 744, 749, 752, 756, 757, 758, 760, 764, 766, 767, 771, 772, 775, 776, 777, 778, 779, 782, 785, 786, 787, 792, 794, 796, 805, 807, 808, 816, 818, 820, 821, 822, 825]