In [1]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.corpus import stopwords
import nltk
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from sklearn.decomposition import TruncatedSVD
```

In [95]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.corpus import stopwords
import nltk
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.pipeline import Pipeline
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
```

Data Collection

In [2]:
```python
review_json_path = 'yelp_academic_dataset_review.json'
business_json_path = 'yelp_academic_dataset_business.json'
checkin_json_path = 'yelp_academic_dataset_review.json'
```

Reading Business Data

In [3]:
```python
df_b = pd.read_json(business_json_path, lines=True)
```

In [22]:
```python
df_b.head(1)
```

Out[22]:

	business_id	name	city	state	stars	review_count	is_open	attributes	categories	
0	AtD6B83S4Mbmq0t7iDnUVA	Veggie House	Las Vegas	NV	4.5	1142	1	{'RestaurantsPriceRange2': '2', 'BikeParking':...	Restaurants, Specialty Food, Japanese, Sushi B...	PhN14AYrE5NgFKS

Cleaning Data

Dropping Irrelevant Columns

In [5]:
```python
df_b = df_b.drop(columns = ['postal_code', 'address', 'latitude', 'longitude', 'hours'])
```

Filtering

In [6]:
```python
df_b = df_b.loc[df_b.categories.str.contains("Restaurants", na=False)]
```

In [7]:
```python
df_b = df_b.loc[df_b['city'] == 'Las Vegas']
```

In [8]:
```python
df_b = df_b.loc[df_b.categories.str.contains("Japanese", na=False)]
```

NA Values

In [9]:
```python
df_b.isna().sum()
```

Out[9]:
```
business_id     0
name            0
city            0
state           0
stars           0
review_count    0
is_open         0
attributes      2
categories      0
dtype: int64
```

In [10]:
```python
df_b = df_b[df_b['attributes'].notna()]
```

In [11]:
```python
business_data_for_eda = df_b.copy()
```

Reading Review Data

In [12]:
```python
reviews_json = 'yelp_academic_dataset_review.json'
size = 500000
reviews = pd.read_json(reviews_json, lines=True,
                      dtype={'review_id':str,'user_id':str,
                             'business_id':str,'stars':int,
                             'useful':int,'funny':int,'cool':int,
                             'text':str},
                      chunksize=size)
```

Joining the Data

In [13]:
```python
chunk_list = []
for chunk_review in reviews:
    # Renaming column name to avoid conflict with business overall star rating
    chunk_review = chunk_review.rename(columns={'stars': 'review_stars'})
    # Inner merge with edited business file so only reviews related to the business remain
    chunk_merged = pd.merge(df_b, chunk_review, on='business_id', how='inner')
    # Show progress
    print(f"{chunk_merged.shape[0]} out of {size:,} related reviews")
    chunk_list.append(chunk_merged)
# After trimming down the review file, concatenate all relevant data back to one dataframe
df_b = pd.concat(chunk_list, ignore_index=True, join='outer', axis=0)
```

```
13793 out of 500,000 related reviews
12587 out of 500,000 related reviews
9575 out of 500,000 related reviews
7024 out of 500,000 related reviews
6014 out of 500,000 related reviews
6175 out of 500,000 related reviews
9005 out of 500,000 related reviews
10734 out of 500,000 related reviews
8803 out of 500,000 related reviews
7182 out of 500,000 related reviews
7663 out of 500,000 related reviews
9327 out of 500,000 related reviews
8578 out of 500,000 related reviews
8681 out of 500,000 related reviews
10684 out of 500,000 related reviews
12022 out of 500,000 related reviews
377 out of 500,000 related reviews
```

Descriptive Analytics and EDA

In [26]:
```python
business_data_for_eda.describe()
```

Out[26]:

	stars	review_count	is_open
count	437.000000	437.000000	437.000000
mean	3.899314	326.407323	0.581236
std	0.612503	447.293409	0.493922
min	1.000000	3.000000	0.000000
25%	3.500000	44.000000	0.000000
50%	4.000000	161.000000	1.000000
75%	4.500000	429.000000	1.000000
max	5.000000	3512.000000	1.000000

In [110]:
```python
business_data_for_eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 437 entries, 246 to 209265
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   business_id   437 non-null    object
 1   name          437 non-null    object
 2   city          437 non-null    object
 3   state         437 non-null    object
 4   stars         437 non-null    float64
 5   review_count  437 non-null    int64
 6   is_open       437 non-null    int64
 7   attributes    437 non-null    object
 8   categories    437 non-null    object
dtypes: float64(1), int64(2), object(6)
memory usage: 34.1+ KB
```

Star Ratings Distribution

In [27]:
```python
x = business_data_for_eda['stars'].value_counts()
x = x.sort_index()
```

In [163]:
```python
plt.figure(figsize=(22,5))
sns.barplot(x.index,x.values)
plt.xlabel('Review Stars')
sns.set_palette("cubehelix")
plt.savefig('stars.png')
```

Distribution of Review Count

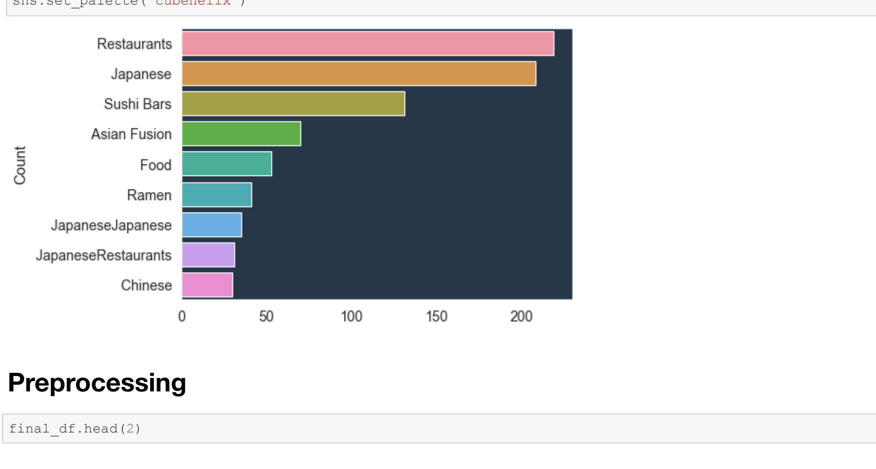
In [120]:
```python
plt.figure(figsize=(6,5))
sns.distplot(business_data_for_eda["review_count"], bins=10)
plt.show()
sns.set_palette("cubehelix")
plt.savefig('histo.png')
```


```
<Figure size 432x288 with 0 Axes>
```

Top 10 Restaurants

In [31]:
```python
rest = business_data_for_eda.sort_values('review_count', ascending=False)
rest = rest.head(10)
```

In [121]:
```python
plt.figure(figsize=(30,10))
sns.barplot(x=rest['name'], y=rest['review_count'])
plt.xlabel("Name", fontsize=15)
plt.savefig('top_rest.png')
sns.set_palette("cubehelix")
```



Top Categories

In [33]:
```python
business_cats = ''.join(business_data_for_eda['categories'].astype('str'))
cats=pd.DataFrame(business_cats.split(','),columns=['categories'])

x=cats.categories.value_counts()
x=x.sort_values(ascending=False)
x=x.iloc[0:9]
```

In [122]:
```python
plt.figure(figsize=(7,5))
sns.barplot(x.values, x.index)
plt.ylabel("Category", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.savefig('categories.png')
sns.set_palette("cubehelix")
```

Preprocessing

In [35]:
```python
final_df.head(2)
```

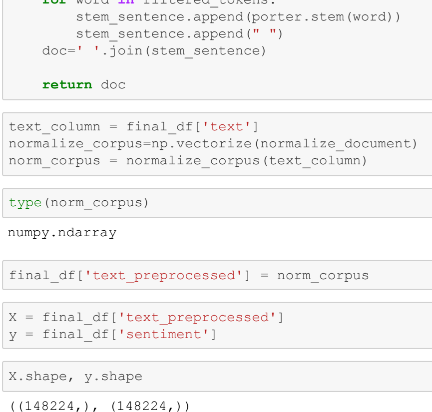
Out[35]:

	business_id	name	city	state	stars	review_count	is_open	attributes	categories	
0	AtD6B83S4Mbmq0t7iDnUVA	Veggie House	Las Vegas	NV	4.5	1142	1	{'RestaurantsPriceRange2': '2', 'BikeParking':...	Restaurants, Specialty Food, Japanese, Sushi B...	PhN14AYrE5NgFKS
1	AtD6B83S4Mbmq0t7iDnUVA	Veggie House	Las Vegas	NV	4.5	1142	1	{'RestaurantsPriceRange2': '2', 'BikeParking':...	Restaurants, Specialty Food, Japanese, Sushi B...	ea4un5C91F_sO6

Label Sentiment of Reviews

In [36]:
```python
final_df['sentiment'] = final_df['review_stars'].apply(lambda rating : +1 if rating > 3 else 0)
```

In [37]:
```python
plt.figure(figsize=(6,5))
sns.set(rc={"axes.facecolor":"#283747", "axes.grid":False,'xtick.labelsize':14,'ytick.labelsize':14})
ax = sns.countplot(x=final_df['sentiment'])
plt.savefig('sentiment_counts.png')
```



In [38]:
```python
wpt=nltk.WordPunctTokenizer()
stop_words=nltk.corpus.stopwords.words('english')
```

In [39]:
```python
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer

porter=PorterStemmer()
```

Preprocessing Pipeline

In [40]:
```python
def normalize_document (doc):
    doc=re.sub(r'[^a-zA-Z\s]', '', doc)
    doc=doc.lower()
    doc=doc.strip()
    tokens=wpt.tokenize(doc)
    filtered_tokens=[token for token in tokens if token not in stop_words]
    stem_sentence=[]
    for word in filtered_tokens:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    doc=' '.join(stem_sentence)

    return doc
```

In [41]:
```python
text_column = final_df['text']
normalize_corpus=np.vectorize(normalize_document)
norm_corpus = normalize_corpus(text_column)
```

In [42]:
```python
type(norm_corpus)
```

Out[42]:
```
numpy.ndarray
```

In [43]:
```python
final_df['text_preprocessed'] = norm_corpus
```

In [57]:
```python
X = final_df['text_preprocessed']
y = final_df['sentiment']
```

In [45]:
```python
X.shape, y.shape
```

Out[45]:
```
((148224,), (148224,))
```

In [46]:
```python
norm_corpus = X.tolist()
y = y.values
```

In [47]:
```python
type(norm_corpus), type(y)
```

Out[47]:
```
(list, numpy.ndarray)
```

Saving Normalized Data

In [48]:
```python
with open ('X.pickle','wb') as f:#wb, write-byte
    pickle.dump (X,f)
```

In [49]:
```python
with open ('y.pickle','wb') as f:#wb, write-byte
    pickle.dump (y,f)
```

In [50]:
```python
with open ('norm_corpus.pickle', 'wb') as f:
    pickle.dump(norm_corpus,f)
```

In [51]:
```python
#final_df = final_df[['text', 'sentiment']]
```

Wordclouds

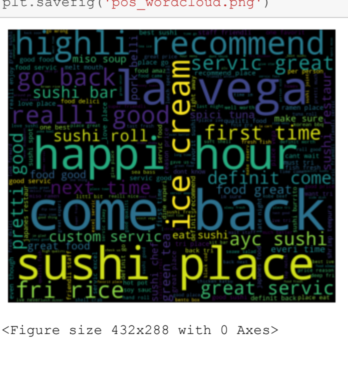
In [52]:
```python
negative = final_df.loc[final_df['review_stars'] <= 2.0]
positive = final_df.loc[final_df['review_stars'] > 2.0]
```

Negative Sentiment

In [58]:
```python
stopwords = set(STOPWORDS)
negative_rev = "_".join(review for review in negative.text_preprocessed)
```

In [62]:
```python
wordcloud_neg = WordCloud(
                          stopwords = STOPWORDS,
                          background_color = 'black',
                          width = 1200,
                          height = 1000,
                          collocation_threshold = 5
                          ).generate(negative_rev)
```

In [67]:
```python
plt.figure(figsize=(6,5))
plt.imshow(wordcloud_neg,interpolation="bilinear")
plt.axis('off')
plt.show()
plt.savefig('neg_wordcloud.png')
```



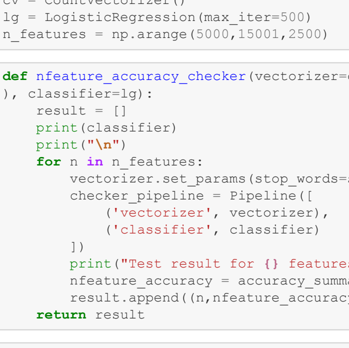

```
<Figure size 432x288 with 0 Axes>
```

Positive Sentiment

In [68]:
```python
positive_rev = "_".join(review for review in positive.text_preprocessed)
```

In [69]:
```python
wordcloud_pos = WordCloud(
                          stopwords = STOPWORDS,
                          background_color = 'black',
                          width = 1200,
                          height = 1000,
                          collocation_threshold = 5
                          ).generate(positive_rev)
```

In [70]:
```python
plt.figure(figsize=(6,5))
plt.imshow(wordcloud_pos,interpolation="bilinear")
plt.axis('off')
plt.show()
plt.savefig('pos_wordcloud.png')
```




```
<Figure size 432x288 with 0 Axes>
```

Feature Selection

In [71]:
```python
with open('norm_corpus.pickle', 'rb') as f:
    norm_corpus = pickle.load(f)
```

In [72]:
```python
train_X, valid_X, train_y, valid_y = train_test_split(norm_corpus, y, random_state = 0, test_size=0.2)
```

Pipeline

In [73]:
```python
def accuracy_summary(pipeline, train_X, train_y, valid_X, valid_y):
    sentiment_fit = pipeline.fit(train_X, train_y)
    y_pred = sentiment_fit.predict(valid_X)
    accuracy = accuracy_score(valid_y, y_pred)
    print("accuracy score: {0:.2f}%".format(accuracy*100))
    return accuracy
```

In [74]:
```python
cv = CountVectorizer()
lg = LogisticRegression(max_iter=500)
n_features = np.arange(5000,15001,2500)
```

In [75]:
```python
def nfeature_accuracy_checker(vectorizer=cv, n_features=n_features, stop_words=None, ngram_range=(1, 3
), classifier=lg):
    result = []
    print(classifier)
    print("\n")
    for n in n_features:
        vectorizer.set_params(stop_words=stop_words, max_features=n, ngram_range=ngram_range)
        checker_pipeline = Pipeline([
            ('vectorizer', vectorizer),
            ('classifier', classifier)
        ])
        print("Test result for {} features".format(n))
        nfeature_accuracy = accuracy_summary(checker_pipeline, train_X, train_y, valid_X, valid_y)
        result.append((n,nfeature_accuracy))
    return result
```

In [76]:
```python
tfidf = TfidfVectorizer()
print("Result for trigram with stop words (Tfidf)\n")
feature_result_tgt = nfeature_accuracy_checker(vectorizer=tfidf,ngram_range=(1, 3))
```

```
Result for trigram with stop words (Tfidf)

LogisticRegression(max_iter=500)


Test result for 5000 features
accuracy score: 91.99%
Test result for 7500 features
accuracy score: 92.18%
Test result for 10000 features
accuracy score: 92.22%
Test result for 12500 features
accuracy score: 92.31%
Test result for 15000 features
accuracy score: 92.30%
```

N-gram Selection

Pipeline

In [77]:
```python
train_X, valid_X, train_y, valid_y = train_test_split(norm_corpus, y, random_state = 0, test_size=0.2)
```

In [78]:
```python
def accuracy_summary(pipeline, train_X, train_y, valid_X, valid_y):
    sentiment_fit = pipeline.fit(train_X, train_y)
    y_pred = sentiment_fit.predict(valid_X)
    accuracy = accuracy_score(valid_y, y_pred)
    print("accuracy score: {0:.2f}%".format(accuracy*100))
    return accuracy
```

In [79]:
```python
cv = CountVectorizer()
lg = LogisticRegression(max_iter=500)
n_features = 12500
n_gram_range = np.arange(1,4,1)
```

In [80]:
```python
def nfeature_accuracy_checker(vectorizer=cv, n_features=n_features, stop_words=None, ngram_range=n_gram
_range, classifier=lg):
    result = []
    print(classifier)
    print("\n")
    for n in n_gram_range:
        ngram_tuple = (n, n)
        vectorizer.set_params(stop_words=stop_words, max_features=n_features, ngram_range=ngram_tuple)
        checker_pipeline = Pipeline([
            ('vectorizer', vectorizer),
            ('classifier', classifier)
        ])
        print("Test result for {} n-grams".format(n))
        nfeature_accuracy = accuracy_summary(checker_pipeline, train_X, train_y, valid_X, valid_y)
        result.append((n,nfeature_accuracy))
    return result
```

In [81]:
```python
tfidf = TfidfVectorizer()
print("Result for N-grams with stop words (Tfidf)\n")
feature_result_tgt = nfeature_accuracy_checker(vectorizer=tfidf)
```

```
Result for N-grams with stop words (Tfidf)

LogisticRegression(max_iter=500)


Test result for 1 n-grams
accuracy score: 91.41%
Test result for 2 n-grams
accuracy score: 89.66%
Test result for 3 n-grams
accuracy score: 82.76%
```

LSA and Topic Modeling

In [123]:
```python
vectorizer = TfidfVectorizer (ngram_range=(2,2),max_features=12500)
X = vectorizer.fit_transform (norm_corpus)
```

In [83]:
```python
print (X[0])
```

```
  (0, 8843)	0.18812766654629726
  (0, 2393)	0.16630467617742986
  (0, 9493)	0.16726543800662116
  (0, 4563)	0.18564291909443825
  (0, 8339)	0.14594701885535208
  (0, 3492)	0.16944363250039138
  (0, 4649)	0.16776055186072664
  (0, 7943)	0.18978249203128297
  (0, 10113)	0.1815504435017228
  (0, 3933)	0.1938428303245804
  (0, 10060)	0.17729801259287284
  (0, 4417)	0.1371868474298085
  (0, 2387)	0.1398278978925348
  (0, 7368)	0.17612041543407408
  (0, 9690)	0.18399520648254186
  (0, 2823)	0.14798044432496385
  (0, 10767)	0.20159820548591126
  (0, 7074)	0.1683299716810083
  (0, 6488)	0.18351004534279217
  (0, 12249)	0.1857744505045105
  (0, 6376)	0.18993871234375476
  (0, 5184)	0.2063570448339135
  (0, 2324)	0.18886582664644025
  (0, 11648)	0.18812766654629726
  (0, 11347)	0.1829169945236995
  (0, 5178)	0.13763135270063573
  (0, 11763)	0.17576934082219892
  (0, 6044)	0.11412466713140795
  (0, 1311)	0.15992702356566776
  (0, 10802)	0.19206732758593884
  (0, 2837)	0.18538198732048225
  (0, 7751)	0.16696107137527108
  (0, 6918)	0.15908317841778793
```

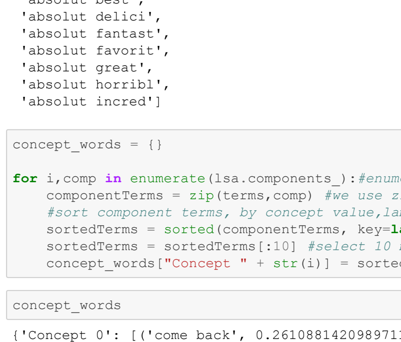
In [152]:
```python
lsa = TruncatedSVD (n_components=5, n_iter=100)#n_com nmber of concepts we want to find
lsa.fit(X)
```

Out[152]:
```
TruncatedSVD(n_components=5, n_iter=100)
```

In [153]:
```python
Sigma = lsa.singular_values_
Sigma
```

Out[153]:
```
array([25.40447251, 17.73228187, 17.35564355, 16.83238397, 15.75597973])
```

In [154]:
```python
ax = sns.barplot(x=list(range(len(Sigma))), y = Sigma)
ax.set(xlabel='Components', ylabel='Relative of Importance')
plt.savefig('lsa.png')
```



In [155]:
```python
row1 = lsa.components_[0] #the row 1
row1
```

Out[155]:
```
array([0.00245209, 0.00384008, 0.00201168, ..., 0.0036876 , 0.00173308,
       0.00252203])
```

In [156]:
```python
terms = vectorizer.get_feature_names()
terms[:20]
```

Out[156]:
```
['abl accommod',
 'abl eat',
 'abl enjoy',
 'abl find',
 'abl finish',
 'abl get',
 'abl make',
 'abl order',
 'abl seat',
 'abl see',
 'abl sit',
 'abl tri',
 'absolut amaz',
 'absolut best',
 'absolut delici',
 'absolut fantast',
 'absolut favorit',
 'absolut great',
 'absolut horribl',
 'absolut incred']
```

In [157]:
```python
concept_words = {}

for i,comp in enumerate(lsa.components_):#enumerate return index and row, a list of tuples
    componentTerms = zip(terms,comp) #we use zip to combine values and terms
    #sort component terms, by concept value,lambda x (x correpsnding tuples) and X[1] the value
    sortedTerms = sorted(componentTerms, key=lambda x:x[1], reverse=True) #decending order
    sortedTerms = sortedTerms[:10] #select 10 most imp. terms in a specific concept
    concept_words["Concept " + str(i)] = sortedTerms #all concepts mapped with list of tuples
```

In [158]:
```python
concept_words
```

Out[158]:
```
{'Concept 0': [('come back', 0.2610881420989711),
  ('sushi place', 0.17762168268430287),
  ('great servic', 0.17005236805145235),
  ('happi hour', 0.16837189195650834),
  ('la vega', 0.165744424357845),
  ('great food', 0.16431657482206977),
  ('servic great', 0.1446263466971937),
  ('food great', 0.13852485499128933),
  ('first time', 0.13774220653694222),
  ('definit come', 0.1374244166121682)],
 'Concept 1': [('great food', 0.5197522773292587),
  ('great servic', 0.49422089314580225),
  ('food great', 0.42189735672269424),
  ('servic great', 0.24267815851865887),
  ('great price', 0.07562810087200386),
  ('place great', 0.055468283821366225),
  ('food servic', 0.052253581112073164),
  ('great atmospher', 0.044495860067142824),
  ('love place', 0.0402711213803595),
  ('good food', 0.040266959475811145)],
 'Concept 2': [('come back', 0.6812702456573786),
  ('definit come', 0.4353660548966254),
  ('first time', 0.10295856844841544),
  ('would definit', 0.1027055119506235),
  ('would come', 0.06624988453422108),
  ('next time', 0.06390540529712112),
  ('back tri', 0.06322440313815943),
  ('wait come', 0.05987966144406406),
  ('back next', 0.05612477616158514),
  ('cant wait', 0.04614997985044288)],
 'Concept 3': [('happi hour', 0.879294413418335),
  ('hour menu', 0.1534758882496293),
  ('great happi', 0.0809859841538179),
  ('late night', 0.06935186017593915),
  ('hour price', 0.06833485184124125),
  ('come back', 0.06037015983973385),
  ('hour special', 0.05410433735999839),
  ('hour pm', 0.05091225708658377),
  ('night happi', 0.046558171781468295),
  ('hour great', 0.041666673197814)],
 'Concept 4': [('sushi place', 0.32657779491273337),
  ('come back', 0.31669435500563464),
  ('best sushi', 0.31402467119517430),
  ('la vega', 0.2522151043406546),
  ('definit come', 0.23275141486528148),
  ('happi hour', 0.17443080724709564),
  ('favorit sushi', 0.1430096685276292),
  ('one best', 0.08453838720199408),
  ('sushi restaur', 0.0757840931801512),
  ('sushi spot', 0.0617264076546378)]}
```

Model Implementation

BOW and TF-IDF

In [96]:
```python
vectorizer = CountVectorizer(max_features=12500,min_df=3, max_df=0.8, stop_words=stopwords.words('engli
sh'),
                             ngram_range=(2,2))
X = vectorizer.fit_transform(norm_corpus).toarray()
```

In [97]:
```python
transformer = TfidfTransformer(norm='l2', use_idf=True)
```

In [98]:
```python
X=transformer.fit_transform(X).toarray()
```

Vocabulary

In [99]:
```python
vocab=vectorizer.get_feature_names()
pd.DataFrame(X,columns=vocab).head(1)
```

Out[99]:

	abl accommod	abl eat	abl enjoy	abl find	abl finish	abl get	abl make	abl order	abl seat	abl see	...	yummi grill	yummi love	yummi servic	yummi sushi	yuzu sauc	yuzu sorbet	yuzu soy	yuzu yellowtail
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1 rows × 12500 columns

Splitting Data

In [100]:
```python
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.2, random_state=1)
```

Fitting to Model

In [101]:
```python
classifier = LogisticRegression(max_iter=100)
classifier.fit(train_X,train_y)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[101]: LogisticRegression()

In [102]:
```python
y_pred = classifier.predict(valid_X)
```

Model Evaluation

In [103]:
```python
cm = confusion_matrix(valid_y,y_pred)
cm
```

Out[103]:
```
array([[ 5190,  2151],
       [  762, 21542]])
```

In [104]:
```python
sns.heatmap(cm, annot=True, fmt='g')
```

Out[104]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff44db1ceb0>

In [105]:
```python
from sklearn.metrics import accuracy_score
accuracy_score (valid_y,y_pred)
```

Out[105]: 0.9017372238151459

In [106]:
```python
from sklearn.metrics import recall_score
recall_score (valid_y,y_pred)
```

Out[106]: 0.9658357245337159

In [107]:
```python
from sklearn.metrics import f1_score
f1_score (valid_y,y_pred)
```

Out[107]: 0.9366697828119226

In [108]:
```python
from sklearn.metrics import precision_score
precision_score (valid_y,y_pred)
```

Out[108]: 0.909213691807707

In [109]:
```python
print(classification_report(valid_y, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.71      0.78      7341
           1       0.91      0.97      0.94     22304

    accuracy                           0.90     29645
   macro avg       0.89      0.84      0.86     29645
weighted avg       0.90      0.90      0.90     29645
```

ROC Curve

In [111]:
```python
lr_probs = classifier.predict_proba(valid_X)
lr_probs = lr_probs[:, 1]
```

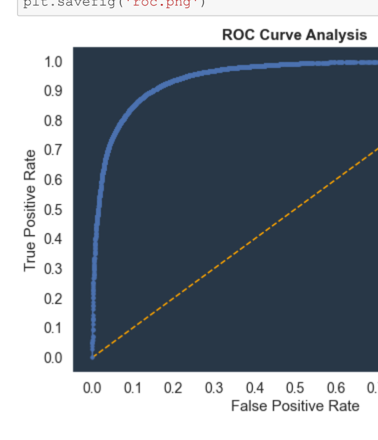
In [112]:
```python
lr_auc = roc_auc_score(valid_y, lr_probs)
```

In [113]:
```python
print('Logistic: ROC AUC=%.3f' % (lr_auc))
```

```
Logistic: ROC AUC=0.949
```

In [114]:
```python
lr_fpr, lr_tpr, _ = roc_curve(valid_y, lr_probs)
```

In [115]:
```python
fig = plt.figure(figsize=(8,6))
plt.plot([0,1], [0,1], color='orange', linestyle='--')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)
plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.savefig('roc.png')
```



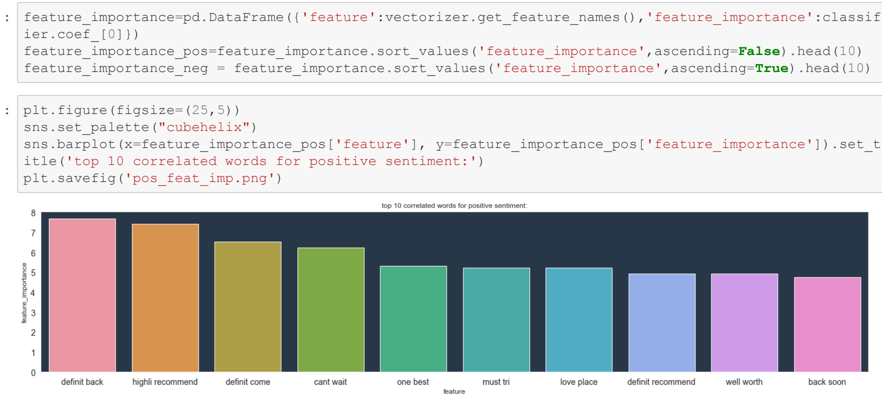
Feature Importance

In [116]:
```python
classifier.coef_
```

Out[116]:
```
array([[ 0.39481493, -0.10126172, -0.04663601, ...,  1.00785253,
         0.15580939, -2.5588048 ]])
```

In [117]:
```python
feature_importance=pd.DataFrame({'feature':vectorizer.get_feature_names(),'feature_importance':classif
ier.coef_[0]})
feature_importance_pos=feature_importance.sort_values('feature_importance',ascending=False).head(10)
feature_importance_neg = feature_importance.sort_values('feature_importance',ascending=True).head(10)
```

In [118]:
```python
plt.figure(figsize=(25,5))
sns.set_palette("cubehelix")
sns.barplot(x=feature_importance_pos['feature'], y=feature_importance_pos['feature_importance']).set_t
itle('top 10 correlated words for positive sentiment:')
plt.savefig('pos_feat_imp.png')
```



In [119]:
```python
plt.figure(figsize=(25,5))
sns.set_palette("cubehelix")
sns.barplot(x=feature_importance_neg['feature'], y=feature_importance_neg['feature_importance']).set_t
itle('top 10 correlated words for negative sentiment:')
plt.savefig('neg_feat_imp.png')
```