In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import json
import datetime
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
```

Loading and Cleaning with Pandas

In [2]:
```
def start_timer():
    global start_timer_timestamp
    start_timer_timestamp = datetime.datetime.now()

def stop_timer():
    stop_timestamp = datetime.datetime.now()
    tm_msec = (stop_timestamp - start_timer_timestamp).total_seconds() * 10(
    print(f'Time spent: {tm_msec} msec')
```

In [3]:
```
start_timer()
review_cols = ['review_id', 'user_id', 'business_id', 'stars', 'useful', 'fu
with open('dataset/review.json', 'r', encoding="utf-8") as review:
    review_data = [json.loads(line) for line in review]
    review_df = pd.DataFrame(review_data, columns=review_cols)
    print("Loaded Reviews. Total ", review_df.size, " records")
print('loaded review_df')
stop_timer()
```

```
Loaded Reviews. Total  33158279  records
loaded review_df
Time spent: 51987.651999999995 msec
```

In [4]:
```
start_timer()
with open('dataset/business.json', 'r', encoding="utf-8") as business:
    business_data = [json.loads(line) for line in business]
    business_df = pd.DataFrame(business_data)
    print("Loaded Businesses. Total ", business_df.size, " records")
print('loaded business_df')
stop_timer()
```

```
Loaded Businesses. Total  2349585  records
loaded business_df
Time spent: 5217.623 msec
```

In [5]:
```
user_cols = ["user_id", "review_count", "average_stars", "cool", "compliment
             "compliment_funny", "compliment_hot", "compliment_list",
            "compliment_more", "compliment_note", "compliment_photos",
            "compliment_plain" , "compliment_profile", "compliment_writer",
            "funny", "useful", "yelping_since"]
start_timer()
with open('dataset/user.json', 'r', encoding="utf-8") as user:
    user_data = [json.loads(line) for line in user]
    user_df = pd.DataFrame(user_data, columns=user_cols)
    print("Loaded Users. Total ", user_df.size, " records")
print('loaded user_df')
stop_timer()
```

```
Loaded Users. Total  22483878  records
loaded user_df
Time spent: 49537.604999999996 msec
```

In [6]:
```
review_df.head(3)
```

Out[6]:

	review_id	user_id	business_id	stars	useful	fu
0	VfBHSwC5Vz_pbFluy07i9Q	cjpdDjZyprfyDG3RlkVG3w	uYHaNptLzDLoV_JZ_MuzUA	5	0	
1	3zRpneRKDsOPq92tq7ybAA	bjTcT8Ty4cJZhEOEo01FGA	uYHaNptLzDLoV_JZ_MuzUA	3	0	
2	ne5WhI1jUFOcRn-b-gAzHA	AXgRULmWcME7J6Ix3I--ww	uYHaNptLzDLoV_JZ_MuzUA	3	0	

In [7]:
```
business_df.head(3)
```

Out[7]:

	address	attributes	business_id	categories	city	hour
0	691 Richmond Rd	{'RestaurantsPriceRange2': 2, 'BusinessParking...	YDf95gJZaq05wvo7hTQbbQ	[Shopping, Shopping Centers]	Richmond Heights	{'Monday' '10:00 21:00' 'Tuesday' '10:00 21..
1	2824 Milton Rd	{'GoodForMeal': {'dessert': False, 'latenight'...	mLwM-h2YhXl2NCgdS84_Bw	[Food, Soul Food, Convenience Stores, Restaura...	Charlotte	{'Monday' '10:00 22:00' 'Tuesday' '10:00 22..
2	337 Danforth Avenue	{'BusinessParking': {'garage': False, 'street'...	v2WhjAB3PIBA8J8VxG3wEg	[Food, Coffee & Tea]	Toronto	{'Monday' '10:00 19:00' 'Tuesday' '10:00 19..

In [8]:
```
user_df.head(3)
```

Out[8]:

	user_id	review_count	average_stars	cool	compliment_cool	compliment_cu
0	lsSiIjAKVl-QRxKjRErBeg	272	3.80	16856	5174	2
1	om5ZiponkpRqUNa3pVPiRg	2559	3.94	40110	1556	2
2	-lGwMGHMC_XihFJNKCJNRg	277	4.72	55	15	

In [9]:
```
print(business_df.shape)
print(business_df.dtypes)
```

```
(156639, 15)
address          object
attributes       object
business_id      object
categories       object
city             object
hours            object
is_open           int64
latitude        float64
longitude       float64
name             object
neighborhood     object
postal_code      object
review_count      int64
stars           float64
state            object
dtype: object
```

In [10]:
```
print(user_df.shape)
print(user_df.dtypes)
```

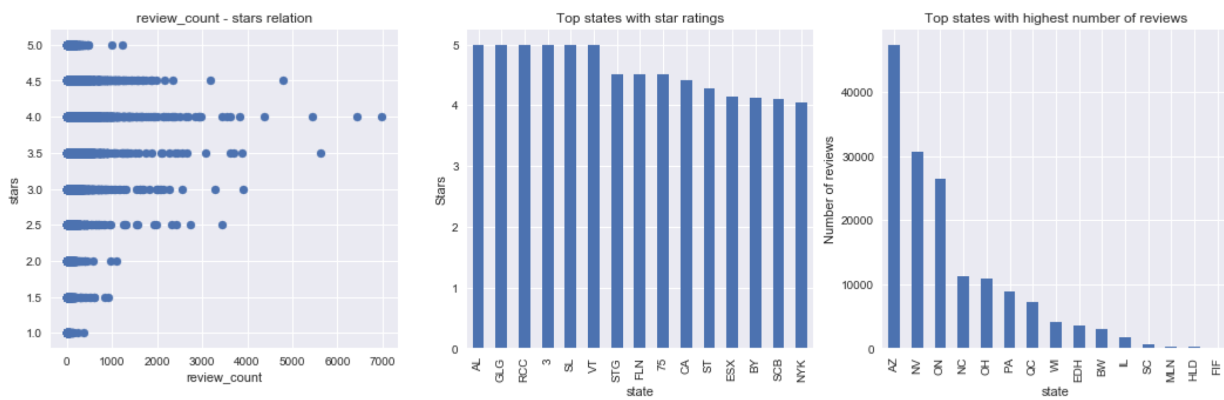
```
(1183362, 19)
user_id                 object
review_count             int64
average_stars          float64
cool                     int64
compliment_cool          int64
compliment_cute          int64
compliment_funny         int64
compliment_hot           int64
compliment_list          int64
compliment_more          int64
compliment_note          int64
compliment_photos        int64
compliment_plain         int64
compliment_profile       int64
compliment_writer        int64
fans                     int64
funny                    int64
useful                   int64
yelping_since           object
dtype: object
```

In [11]:
```
print(review_df.shape)
print(review_df.dtypes)
```

```
(4736897, 7)
review_id      object
user_id        object
business_id    object
stars           int64
useful          int64
funny           int64
cool            int64
dtype: object
```

In [12]:
```
fig = plt.figure(figsize = (15,5))
fig.clf()
fig.subplots_adjust(hspace=.3)
ax0 = fig.add_subplot(1, 3, 1)
ax1 = fig.add_subplot(1, 3, 2)
ax2 = fig.add_subplot(1, 3, 3)

business_df.groupby('stars').size().plot(kind='bar', ax = ax0)
ax0.set_title('Businesses Stars Count')
ax0.set_ylabel('Number of Businesses')

business_review_count = business_df.groupby('review_count').size()
bins=[0, 5, 10, 20, 40, 80,160, 320, 640, 1280, 2560]
ax1.hist(business_review_count, bins=bins, edgecolor="k")
ax1.set_title('Businesses Reviews Count')
ax1.set_xlabel('Number of Businesses')
ax1.set_ylabel('Number of Reviews Got')
ax1.set_xscale('log')

business_review_count = user_df.groupby('review_count').size()
bins=[0, 5, 10, 20, 40, 80,160, 320, 640, 1280, 2560]
ax2.hist(business_review_count, bins=bins, edgecolor="k")
ax2.set_title('User Reviews Count')
ax2.set_xlabel('Number of Businesses')
ax2.set_ylabel('Number of Reviews Give')
ax2.set_xscale('log')

fig.savefig('plot1.png')
```

In [13]:
```
igure(figsize = (18,5))

s_adjust(hspace=.3)
dd_subplot(1, 3, 1)
dd_subplot(1, 3, 2)
dd_subplot(1, 3, 3)

(business_df['review_count'], business_df['stars'])
le('review_count - stars relation')
bel('review_count')
bel('stars')

[['state', 'stars']].groupby('state')['stars'].agg('mean').sort_values(ascend
le('Top states with star ratings')
bel('Stars')

[['state', 'stars']].groupby('state')['stars'].count().sort_values(ascending
le('Top states with highest number of reviews')
bel('Number of reviews')

('plot2.png')
```



Data Selection/Cleaning

In [14]:
```
# First of all let's filter out closed businesses
open_business_df = business_df[business_df['is_open'] == 1]
print("After removing businesses that are closed we left with ", open_busine

# Next, filter out all none restaurant businesses, because we only care abou
restaurant_df = open_business_df[open_business_df['categories'].apply(lambda
print("Open restaurant business records: ", restaurant_df.size)
```

```
After removing businesses that are closed we left with  1983930  records
Open restaurant business records:  579855
```

In [15]:
```
# We need to process hours column to factor out time
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
def get_hours(hours_df):
    hours = []

    for s in hours_df:
        open_list = []
        close_list = []

        for d in days:
            opn = 0.0
            cls = 0.0
            if d in s:
                hourz = s[d].split('-')
                hrs1 = hourz[0].split(':')
                hrs2 = hourz[1].split(':')

                opn = float(hrs1[0]) + float(hrs1[1])/60
                cls = float(hrs2[0]) + float(hrs2[1])/60

                # handle overnight hours
                if (opn > cls):
                    cls += 24

            open_list.append(opn)
            close_list.append(cls)
        hours.append((open_list, close_list))
    return hours

restaurant_hours = get_hours(restaurant_df['hours'])
```

In [16]:
```
print(restaurant_df.columns)
```

```
Index(['address', 'attributes', 'business_id', 'categories', 'city', 'hou
rs', 'is_open', 'latitude', 'longitude', 'name', 'neighborhood', 'postal_
code', 'review_count', 'stars', 'state'], dtype='object')
```

In [17]:
```
# Append `business_` and `review_` prefix to all columns in restaurants and
# to distinguish columns after merge
restaurant_df.columns = ['business_' + str(col) for col in restaurant_df.col
review_df.columns = ['review_' + str(col) for col in review_df.columns]
# rename *_id columns back
restaurant_df.rename(columns={"business_business_id": "business_id"}, inplac
review_df.rename(columns={"review_business_id": "business_id",
                          "review_review_id": "review_id",
                          "review_user_id": "user_id"}, inplace=True)
```

In [18]:
```
# We need to process hours column to factor out time
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
open_close = ['_open', '_close']

def get_hours(df):
    hours = []
    for index, row in df.iterrows():
        record = {'business_id': row['business_id']}
        s = row['business_hours']
        for d in days:
            opn = 0.0
            cls = 0.0
            if d in s:
                hourz = s[d].split('-')
                hrs1 = hourz[0].split(':')
                hrs2 = hourz[1].split(':')

                opn = float(hrs1[0]) + float(hrs1[1])/60
                cls = float(hrs2[0]) + float(hrs2[1])/60

                # handle overnight hours
                if (opn > cls):
                    cls += 24

            record[str(d) + '_open'] = opn
            record[str(d) + '_close'] = cls

        hours.append(record)
    return hours

restaurant_hours = get_hours(restaurant_df[['business_id', 'business_hours']
```

In [19]:
```
restaurant_hours_df = pd.DataFrame(restaurant_hours)
restaurant_df_merged = restaurant_df.merge(restaurant_hours_df, on=['busines
restaurant_df_merged.head(3)
```

Out[19]:

	business_address	business_attributes	business_id	business_categories	bu
0	9616 E Independence Blvd	{'Alcohol': 'full_bar', 'HasTV': True, 'NoiseL...	SDMRxmcKPNt1AHPBKqO64Q	[Burgers, Bars, Restaurants, Sports Bars, Nigh...	
1	190 E Dallas Rd	{'RestaurantsAttire': 'casual', 'Alcohol': 'no...	iFEiMJoEqyB9O8OUNSdLzA	[Chinese, Restaurants]	
2	4759 Liberty Ave	{'RestaurantsTableService': True, 'GoodForMeal...	HmI9nhgOkrXlUr6KZGZZew	[Sandwiches, Restaurants, Italian, Diners, Bre...	

In [20]:
```
# Next let's take a look at all categories that has 'Restaurant'
categories = set()
restaurant_df_merged['business_categories'].apply(lambda r: categories.updat
# number of categories
len(categories)
```

Out[20]: 635

In [21]:
```
# one hot encoding for categories
def process_categories(df):
    records = []
    for index, row in df.iterrows():
        record = {'business_id': row['business_id']}
        current_cats = row['business_categories']
        for c in current_cats:
            record[c] = 1
        records.append(record)
    return records

b_cats = process_categories(restaurant_df_merged)
```

In [22]:
```
cats_df = pd.DataFrame(b_cats).fillna(0)
```

In [23]:
```
restaurant_df_merged = restaurant_df_merged.merge(cats_df, on=['business_id'
restaurant_df_merged.head(3)
```

Out[23]:

	business_address	business_attributes	business_id	business_categories	bu
0	9616 E Independence Blvd	{'Alcohol': 'full_bar', 'HasTV': True, 'NoiseL...	SDMRxmcKPNt1AHPBKqO64Q	[Burgers, Bars, Restaurants, Sports Bars, Nigh...	
1	190 E Dallas Rd	{'RestaurantsAttire': 'casual', 'Alcohol': 'no...	iFEiMJoEqyB9O8OUNSdLzA	[Chinese, Restaurants]	
2	4759 Liberty Ave	{'RestaurantsTableService': True, 'GoodForMeal...	HmI9nhgOkrXlUr6KZGZZew	[Sandwiches, Restaurants, Italian, Diners, Bre...	

3 rows × 664 columns

In [24]:
```
restaurant_df_merged.isnull().values.any()
```

Out[24]: False

In [25]:
```
# one hot encoding for restaurant attributes
def process_attributes(df):
    records = []
    for index, row in df.iterrows():
        attrs = row['business_attributes']
        attrs['business_id'] = row['business_id']
        records.append(attrs)
    return records

b_attrs = process_attributes(restaurant_df_merged)
```

In [26]:
```
pd.DataFrame(b_attrs).head(5)
```

Out[26]:

	AcceptsInsurance	AgesAllowed	Alcohol	Ambience	BYOB	BYOBCorkage	BestNights	BikeParl
0	NaN	NaN	full_bar	{'romantic': False, 'intimate': False, 'classy...	NaN	NaN	NaN	
1	NaN	NaN	none	NaN	NaN	NaN	NaN	
2	NaN	NaN	none	{'romantic': False, 'intimate': False, 'classy...	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	full_bar	{'romantic': False, 'intimate': False, 'classy...	NaN	NaN	NaN	

In [27]:
```
restaurant_df_merged = restaurant_df_merged.drop(['business_attributes', 'bu
```

In [28]:
```
restaurant_df_merged.head()
```

Out[28]:

	business_address	business_id	business_city	business_is_open	business_latitude
0	9616 E Independence Blvd	SDMRxmcKPNt1AHPBKqO64Q	Matthews	1	35.13519
1	190 E Dallas Rd	iFEiMJoEqyB9O8OUNSdLzA	Stanley	1	35.35508
2	4759 Liberty Ave	HmI9nhgOkrXlUr6KZGZZew	Pittsburgh	1	40.46135
3	7070 Saint Barbara Boulevard	qnpvw-uQyRn9nlClWFK9aA	Mississauga	1	43.63923
4	4502 East Towne Blvd	TXiEgINSZ75d3EtvLvkc4Q	Madison	1	43.12803

5 rows × 661 columns

In [29]:
```
yelp_reviews = pd.merge(pd.merge(restaurant_df_merged, review_df, on='busine
                 user_df, on='user_id', how='left')
yelp_reviews.head(3)
print("Merged data frame contains ", yelp_reviews.columns.size)
```

```
Merged data frame contains  685
```

In [30]:
```
# Most of the resturants are found in Phoniex, AZ
yelp_reviews_az = yelp_reviews[yelp_reviews.business_city == 'Phoenix']
```

In [31]:
```
# parse the yelping since date to timestamps
yelp_since_sec = pd.to_datetime(yelp_reviews_az['yelping_since']).astype(int
len(yelp_since_sec)
```

Out[31]: 259950

In [32]:
```
# normalize the yelping_since column
earlist = yelp_since_sec.min()
yelp_reviews_az['yelp_since'] = yelp_since_sec / earlist
```

In [33]:
```
yelp_reviews_az.head()
```

Out[33]:

	business_address	business_id	business_city	business_is_open	business_latitude
752	2641 N 44th St, Ste 100	01xXe2m_z048W5gcBFpoJA	Phoenix	1	33.47804
753	2641 N 44th St, Ste 100	01xXe2m_z048W5gcBFpoJA	Phoenix	1	33.47804
754	2641 N 44th St, Ste 100	01xXe2m_z048W5gcBFpoJA	Phoenix	1	33.47804
755	2641 N 44th St, Ste 100	01xXe2m_z048W5gcBFpoJA	Phoenix	1	33.47804
756	2641 N 44th St, Ste 100	01xXe2m_z048W5gcBFpoJA	Phoenix	1	33.47804

5 rows × 686 columns

In [34]:
```
# Split the training and testing dataset
mask = np.random.rand(len(yelp_reviews_az)) < 0.75
train_df = yelp_reviews_az[mask]
test_df = yelp_reviews_az[~mask]
```

In [35]:
```
# initialize train and test set
def init_data():
    global X_train, y_train, X_test, y_test;
    X_train = train_df[['business_stars', 'average_stars', 'business_review_
                          'African', 'American (Traditional)' ,
                          'Vietnamese', 'Vegetarian',
                          'Chinese' , 'Mexican', 'Indian',
                          'Japanese', 'German',
                          'Greek', 'yelp_since']]
    y_train = train_df['review_stars']
    X_test = test_df[['business_stars', 'average_stars', 'business_review_co
                          'African', 'American (Traditional)' ,
                          'Vietnamese', 'Vegetarian',
                          'Chinese' , 'Mexican', 'Indian',
                          'Japanese', 'German',
                          'Greek', 'yelp_since']]
    y_test = test_df['review_stars']
```

Baseline Model

The baseline linear regression model is created by OLS api in statsmodels, which minimize the sum of the squares of the differences between the observed responses. The resulting baseline rating can be expressed by a simple linear combination of single-user bias across all restaurants and specific restaurant bias for all users. The R2_score is really low (<0.36) here, which indicating the model is not working well to recommend user restaurants. However, according to the reference for Netflix Prize, modeling these biases turned out to be fairly important.

For example, if we already know Tom tends to rate 0.5 lower than average, and Otto Pizza is a pretty awesome restaurant that should be rated 0.7 starts higher than we normally expect. Suppose Tom and Jerry share the similar taste for food, and Jerry gave Otto Pizza with rating 4.0 stars, we could simply predict Tom's rating for Otto Pizza would be 4.0 - 0.5 + 0.7 = 4.2

In [36]:
```
import statsmodels.api as sm
from statsmodels.api import OLS
from sklearn.metrics import r2_score
```

In [37]:
```
# Split the training and testing dataset
mask = np.random.rand(len(yelp_reviews)) < 0.75
train_df = yelp_reviews[mask]
test_df = yelp_reviews[~mask]
```

In [38]:
```
# The baseline rating is the mean over all user-business ratings
baseline_rating = np.mean(review_df['review_stars'])

x_train = train_df[['business_stars', 'average_stars']]
x_train['const'] = baseline_rating
y_train = train_df['review_stars']

x_test = test_df[['business_stars', 'average_stars']]
x_test['const'] = baseline_rating
y_test = test_df['review_stars']
```

In [39]:
```
model = sm.OLS(y_train, x_train)
regr = model.fit()

y_trian_hat = regr.predict(x_train)
print("R2 score for train data", r2_score(y_train, y_trian_hat))

y_test_hat = regr.predict(x_test)
print("R2 score for test data", r2_score(y_test, y_test_hat))
```

```
R2 score for train data 0.35752009596
R2 score for test data 0.359754820839
```

In [40]:
```
# To predict the rating for the user that already give this business review.
user_test = train_df [train_df.user_id == 'M0cI78odeq_GKqLzk8sIrw']
x_user_test = user_test[[ 'business_stars', 'average_stars']]
x_user_test['const'] = baseline_rating
y_user_test = user_test['review_stars']
```

In [41]:
```
y_user_test_hat = regr.predict(x_user_test)
```

In [42]:
```
y_user_test_hat.head()
```

Out[42]:
```
0         2.287769
286887    3.721762
350264    3.363264
559444    3.721762
651125    3.721762
dtype: float64
```

To recommend for the user "M0cI78odeq_GKqLzk8sIrw", we predict the ratings for all business and then give top recommendations based on rank of ratings.

In [43]:
```
one_user_test = user_df[user_df.user_id == 'M0cI78odeq_GKqLzk8sIrw']
```

In [44]:
```
# To form a x_train dataframe, with this user data and ALL different busines
one_user_test = pd.DataFrame(np.tile(one_user_test.values, len(business_df.i
                    columns=one_user_test.columns)
business_df = business_df.rename(columns={'stars': 'business_stars', 'name'
one_user_all_business_test = one_user_test.join(business_df)
```

In [45]:
```
# the x_train dataframe of the user
x_one_user_all_business_test = one_user_all_business_test[['business_stars',
x_one_user_all_business_test['const'] = baseline_rating
```

In [46]:
```
y_user_test_hat = regr.predict(x_one_user_all_business_test)
predictions_df = one_user_all_business_test
predictions_df['pred_rating'] = y_user_test_hat
predictions_df[['business_id', 'business_name', 'pred_rating']].sort_values(
```

Out[46]:

	business_id	business_name	pred_rating
96523	WP--xg2QZ9W9_Jw-2Jd1SA	Arizona Biltmore Dentistry	4.43876
74681	7V4Cc-xK-fDPYYG2WkPN3w	Gentle Dental Associates & Spa	4.43876
103900	0dYiTboT1N7Sqji85Pt8wA	305 Kustoms	4.43876
97320	KxlkLRpb9gSTL1S6UnFXzw	Harbourside Fish and Chips	4.43876
97318	qlugM5IFpL1sgp27KiFTqg	MacTEK Consulting & Repairs	4.43876

Additional baseline predictors could be included.

- Yelping_since field (square root); users become harsher critic over time
- Review_count in business.json
- Review_count in user.json

However, the R2 score of 0.3498 is not improved compare to the most basic basline

In [47]:
```
# Split the training and testing dataset
mask = np.random.rand(len(yelp_reviews_az)) < 0.75
train_df_az = yelp_reviews_az[mask]
test_df_az = yelp_reviews_az[~mask]
```

In [48]:
```
# The baseline rating is the mean over all user-business ratings
baseline_rating = np.mean(review_df['review_stars'])

x_train = train_df_az[['business_stars', 'average_stars', 'yelp_since' , 're
x_train['const'] = baseline_rating
y_train = train_df_az['review_stars']

x_test = test_df_az[['business_stars', 'average_stars', 'yelp_since' , 'revi
x_test['const'] = baseline_rating
y_test = test_df_az['review_stars']
```

In [49]:
```
model = sm.OLS(y_train, x_train)
regr2 = model.fit()

y_trian_hat = regr2.predict(x_train)
print("R2 score for train data", r2_score(y_train, y_trian_hat))

y_test_hat = regr2.predict(x_test)
print("R2 score for test data", r2_score(y_test, y_test_hat))
```

```
R2 score for train data 0.348162899525
R2 score for test data 0.350567872004
```

Regularized Regression

Type *Markdown* and LaTeX: α^2

In [50]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [51]:
```
# Split the training and testing dataset
mask = np.random.rand(len(yelp_reviews_az)) < 0.75
train_df = yelp_reviews_az[mask]
test_df = yelp_reviews_az[~mask]
```

In [52]:
```
init_data()
X_train.columns
```

Out[52]:
```
Index(['business_stars', 'average_stars', 'business_review_count', 'revie
w_count', 'African', 'American (Traditional)', 'Vietnamese', 'Vegetaria
n', 'Chinese', 'Mexican', 'Indian', 'Japanese', 'German', 'Greek', 'yelp_
since'], dtype='object')
```

With many predictors after merge, we expect number of them to be not singificant and possibly contribute to overfitting. The best way to check for which predictors are significant is to apply Regularization. Instead of fitting a linear regression model on all predictors, we will shrink or regularize, the coefficient estimates to make sure that the model does not "overfit" the training set. We had learned 2 models that are good at regularization: Ridge regression Lasso regression We need to choose shrikage parameter λ from the set {0.00001,...,100000}. We be doing it by computing R^2 score for each alpha to identify which model perfomed the best and with which alpha.

In [53]:
```
# result dictionary
r2_dict = {'alpha': [], 'ridge':[], 'lasso':[]}

#List of Lambda (lol!) values
lol = [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 1e-2, 1e-1, 1.0, 10.0, 100.0, 1000.0, 1

# Find R^2 scores for each model (Linear, Lasso, Ridge) while
# varying alpha value for Lasso and Ridge models.
for alpha in lol:
    r2_dict['alpha'].append(alpha)
    lasso = Lasso(alpha=alpha, fit_intercept=True)
    lasso.fit(X_train, y_train)

    lasso_preds = lasso.predict(X_test)
    r2_dict['lasso'].append(r2_score(y_test, lasso_preds))

    ridge = Ridge(alpha=alpha, fit_intercept=True)
    ridge.fit(X_train, y_train)
    ridge_preds = ridge.predict(X_test)
    r2_dict['ridge'].append(r2_score(y_test, ridge_preds))

# build data frame and inspect data.
r2_df = pd.DataFrame(r2_dict)
r2_df.head()
```

Out[53]:

	alpha	lasso	ridge
0	0.00001	0.345777	0.345778
1	0.00005	0.345778	0.345778
2	0.00010	0.345780	0.345778
3	0.00050	0.345754	0.345778
4	0.00100	0.345718	0.345778

Next, we will use principal components analysis (PCA) to fit the model. Normalizing the predictors helps in finding the proper principal components. We will be looking for number of components contributing contributing to 90% of the variance in the predictors. Next we will build PCA model using best number of principal predictors and fit the normalized model.

In [54]:
```
# normalize
# X_train_norm = (X_train - X_train.mean()) / (X_train.max() - X_train.min(
# X_test_norm = (X_test - X_test.mean()) / (X_test.max() - X_test.min())

pca = PCA().fit(X_train)
print('Explained variance ratio:', pca.explained_variance_ratio_)
# look for number of principal components contributing to 90%
# of the variance in the predictors
for i in range(X_train.shape[0]):
    if pca.explained_variance_ratio_[0:i+1].sum() > 0.9:
        n_comp = i+1
        break
n_comp
```

```
Explained variance ratio: [  7.10929379e-01   2.89064697e-01   2.54445913
e-06   1.24911023e-06
   7.19263720e-07   5.82536482e-07   2.28877236e-07   1.95434190e-07
   1.74336726e-07   8.70068453e-08   6.98030975e-08   4.91841874e-08
   2.02155161e-08   3.23232048e-09   1.31906328e-42]
```

Out[54]: 2

In [55]:
```
pca = PCA(n_comp).fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

pca_model = LogisticRegression().fit(X_train_pca, y_train)
y_train_pca = pca_model.predict(X_train_pca)
y_test_pca = pca_model.predict(X_test_pca)

R2_train_pca = pca_model.score(X_train_pca, y_train)
R2_test_pca = pca_model.score(X_test_pca, y_test)

print("Accuracy rate on TRAIN data: {:1.2f}%".format(R2_train_pca*100))
print("Accuracy rate on TEST data: {:1.2f}%".format(R2_test_pca*100))
```

```
Accuracy rate on TRAIN data: 44.33%
Accuracy rate on TEST data: 44.13%
```

Matrix Factorization

Matrix Factorization aims to capture the latent factors. We will use Singular Value Decomposition (SVD) to create the matrices. Using SVD, we can create following three matrices:

- How much a user liked various features of the restaurant?
- Which features were offered by each restaurant?
- A weight matrix that relates the two matrices above

In [56]:
```
from scipy.sparse.linalg import svds
```

In [57]:
```
review_df_copy = review_df.sample(100)
```

In [58]:
```
R_df = review_df_copy.pivot(index = 'user_id', columns ='business_id', value
R = R_df.as_matrix()
user_ratings_mean = np.mean(R, axis = 1)
R_demeaned = R - user_ratings_mean.reshape(-1, 1)
```

In [59]:
```
U, sigma, Vt = svds(R_demeaned, k = 50)
sigma = np.diag(sigma)
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mea
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = R_df.columns,
preds_df.head()
```

Out[59]:

business_id	-F5mm0-YeCI7viSiOwVAAw	0Fs4Z2nKGzZAEM1bDIDoiQ	2NsEac9xCBI05bo5l4yI7G
user_id			
-GfVotKwVsob_0NLwyv6OA	0.027901	0.012886	0.027901
15HQIKBadXrteo7E4DbMHw	-0.059251	-0.014266	-0.556184
2XYdguaaZ7dgi6fAIddujg	-0.306811	-0.014266	0.094947
2ad9Ug6RNpE3AhCRpus4mg	0.047111	0.014593	0.047111
2e5V6M4GNufEnbGJpVdCjw	-0.647306	-0.014266	0.849015

In [60]:
```
sorted_user_predictions = preds_df.iloc[0].sort_values(ascending=False)
raw_rec = pd.DataFrame(sorted_user_predictions).reset_index()
preds_df.head()
```

Out[60]:

business_id	-F5mm0-YeCI7viSiOwVAAw	0Fs4Z2nKGzZAEM1bDIDoiQ	2NsEac9xCBI05bo5l4yI7G
user_id			
-GfVotKwVsob_0NLwyv6OA	0.027901	0.012886	0.027901
15HQIKBadXrteo7E4DbMHw	-0.059251	-0.014266	-0.556184
2XYdguaaZ7dgi6fAIddujg	-0.306811	-0.014266	0.094947
2ad9Ug6RNpE3AhCRpus4mg	0.047111	0.014593	0.047111
2e5V6M4GNufEnbGJpVdCjw	-0.647306	-0.014266	0.849015

In [61]:
```
def recommend_business(predictions_df, user_id, business_df, original_rating
    sorted_user_predictions = predictions_df.loc[user_id].sort_values(ascend
    recommendations = (business_df.merge(pd.DataFrame(sorted_user_predictior
               left_on = 'business_id',
               right_on = 'business_id').
         rename(columns = {user_id: 'Predictions'}).
        sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations])
    return recommendations
```

```
In [63]: # Make 10 recommendations for user '-GfVotKwVsob_0NLwyv6OA'
         predictions = recommend_business(preds_df, '-GfVotKwVsob_0NLwyv6OA', busines
         predictions
```

Out[63]:

	address	attributes	business_id	categories	city	
33130	836 W St. Clair Ave	{'Alcohol': 'full_bar', 'HasTV': True, 'NoiseL...	QjZFYd5hme7EHegpuJngMQ	[Restaurants, Tapas/Small Plates, Wine Bars, N...	Cleveland	{'I '-
106721	3146 E Camelback Rd	{'Alcohol': 'full_bar', 'HasTV': True, 'NoiseL...	NJ0RzuWd5xDqfJejYQZ65g	[Restaurants, Bars, Nightlife, Sushi Bars, Ame...	Phoenix	{'I '-
68292	4178 Koval Ln	{'RestaurantsTableService': True, 'GoodForMeal...	dn51F67VLgPuqy_8SFk9oA	[American (Traditional), Restaurants]	Las Vegas	{'I '-
6947	9425 Leslie St	{'GoodForMeal': {'dessert': False, 'latenight'...	BzIEV-DIrnTbAB2EtIDI4g	[Barbeque, Restaurants]	Richmond Hill	
24983	505 Highway 7 E, Unit 238, Building B	{'RestaurantsTableService': True, 'GoodForMeal...	VGZ_MJ7P3vSQMYyQf-mNjw	[Chinese, Restaurants, Food, Dim Sum]	Markham	{'I

Ensemble

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement

In [64]:
```
%matplotlib inline
import datetime
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import json
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from itertools import product
from collections import OrderedDict

from IPython.display import *
```

In [65]:
```
init_data()
```

In [66]:
```
X_train.shape
```

Out[66]: (194881, 15)

In [67]:
```
def ensemble(Xtrain, ytrain, param_dict):
#     n_estimators is the number of trees in the forest.
# The number of features to consider when looking for the best split.
# Here max_features is a percentage and int(max_features * n_features) featu
    start_timer()
    results = {}
    estimators= {}
    for n, f in product(*param_dict.values()):
        params = (n, f)
        print(f'Creating RandomForestRegressor using {n} estimators and {f}%
        est = RandomForestRegressor(oob_score=True,
                                    n_estimators=n, max_features=f, n_jobs=-
        est.fit(Xtrain, ytrain)
        results[params] = est.oob_score_
        print('OOB score is:', est.oob_score_)
        estimators[params] = est
    print(results)
    outparams = max(results, key = results.get)
    stop_timer()
    print(outparams)

    # get the regressor corresponding to the outparams
    rf1 = estimators[outparams]
    return (rf1,results)
```

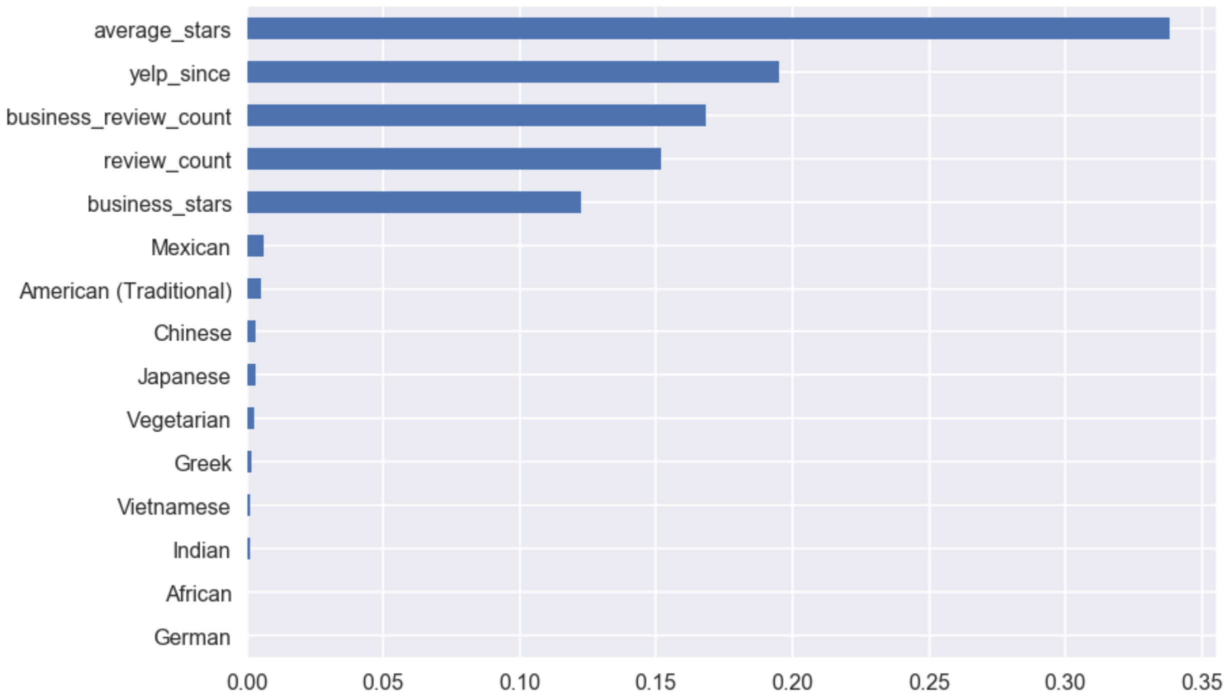
In [68]:

```
param_dict = OrderedDict(
        n_estimators = [50, 100, 200],
        max_features = [0.2, 0.4]
    )
rf1 = ensemble(X_train, y_train, param_dict)
```

```
Creating RandomForestRegressor using 50 estimators and 0.2% max features:

OOB score is: 0.291515800251
Creating RandomForestRegressor using 50 estimators and 0.4% max features:

OOB score is: 0.291189448212
Creating RandomForestRegressor using 100 estimators and 0.2% max feature
s:
OOB score is: 0.311966218049
Creating RandomForestRegressor using 100 estimators and 0.4% max feature
s:
OOB score is: 0.309594158171
Creating RandomForestRegressor using 200 estimators and 0.2% max feature
s:
OOB score is: 0.319990230992
Creating RandomForestRegressor using 200 estimators and 0.4% max feature
s:
OOB score is: 0.31841837413
{(50, 0.2): 0.2915158002506627, (50, 0.4): 0.29118944821180537, (100, 0.
2): 0.31196621804922364, (100, 0.4): 0.30959415817116798, (200, 0.2): 0.3
1999023099200796, (200, 0.4): 0.31841837413037344}
Time spent: 68917.66 msec
(200, 0.2)
```

In [69]:
```
# Compute the R squared on training and test datasets
print('Train R-squared using RandomForestRegression is:', rf1[0].score(X_tra
print('Test R-squared using RandomForestRegression is:', rf1[0].score(X_test

# Show Top-25 most important features
features = list(X_train.columns)
sns.set_context("poster")
pd.Series(rf1[0].feature_importances_, index=features).nlargest(n=25).sort_v
```

```
Train R-squared using RandomForestRegression is: 0.906737849494
Test R-squared using RandomForestRegression is: 0.320851843524
```



In [70]:
```
# Predict the rating using the random forest regressor:
rf1[0].predict(X_test)
```

Out[70]:
```
array([ 3.995,  2.725,  1.595, ...,  3.245,  4.985,  2.925])
```

Summary

Data:

3 of the 6 yelp datasets that we are interested are consist of records: Reviews - 42,632,073 Businesses - 2,349,585 Users - 2,6033,964 Running EDA on entire data set seemed to be a problem due to somewhat large datasets. On our fastest laptop - we could wait over 15 mins for one plot to finish. Hence, we cleaned business dataset by filtering out all businesses that are closed. While they can provide some interest to us, we believe that size of the population isn't a problem given the initial data size (42.6 m).

In addition, we had filtered out businesses that don't have Restaurant in their categories, since based on our project goal we are focusing just on restaurant businesses. Hence, we ended up with 579,855 records which is 1.36% from our initial input for bussiness.

After that we have build a data frame by joining reviews with business through business_id (left outer join) result was then joined with user data frame by user_id (also left outer join). Before we joined 3 data frames, we renamed columns to be prefixed with original dataframe name. This was done to avoid confusion betwen review rating and business rating. After merge we ended up with 44 columns. Not all of those columns will be used in training data set.

For the next step, we built 4 models (baseline, regularized regression using Ridge and Lasso, matrix factorization and Random Forrest) to predict star rating for a given restaurant.

BEST MODEL

Here is the comparison of train and test R2 for all the models. The best model is PCA with number of components = 2. These two components were able to explain the 90% of variance. For Ensemble, the training R2 is quite high but the test R2 is low. This clearly indicates overfitting.

STRENGTHS AND SHORTCOMINGS

The biggest strength of PCA is its simplicity. PCA does not require much memory and processing power. Using PCA we were able to explain 90% of variance using just two components.Therefore, the resulting model was simple and fast. The downside of PCA is that the results cannot be explained using a flow chart. We don't know which features are captured in these two components.

FURTHER IMPROVEMENTS

Test R2 for RandomForestRegressor were quite disappointing when compared with PCA. It would be good to try AdaBoost and compare it with the PCA results. We can use natural language processing and sentiment analysis on the "text" field of the review.json. We can generate positivity scores for words either globally or per-category. We can create a social networks using the friends field for the reviewer and make predictions based on the social network.