# Goethe University

# Data Science and Marketing Analysis, Yelp Dataset

# A - Preparation

## Read Libraries

In [ ]:
```
import pandas as pd
import json
from datetime import datetime
import seaborn as sns
from collections import Counter
import random
import matplotlib.pyplot as plt
import geopy
import numpy as np
import re
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")
import gmaps
```

# 1 Check-in

- The Reason starting with Check-in is to filter our dataset to only 1 year

## 1.1- Read & Normalize File

In [ ]:
```
# To read & Normalize File because it was in Json format
check = pd.read_csv('check_az_open.csv') #Original File
check = pd.json_normalize(check.j.apply(json.loads))
check
```

Out[ ]:

| | date | business_id |
|---|---|---|
| **0** | 2010-03-23 05:49:25, 2010-08-27 05:38:45, 2010... | -01XupAWZEXbdNbxNg5mEg |
| **1** | 2010-07-28 21:07:28, 2010-07-31 21:16:54, 2010... | -092wE7j5HZOogMLAh40zA |
| **2** | 2010-09-04 16:52:19, 2010-09-11 16:26:25, 2010... | 4RSJzjaxYJ9CRTfrk1sHZg |
| **3** | 2010-03-15 06:48:51, 2010-10-15 20:56:00, 2011... | 4RV97YE8VEw05tu0WO425g |
| **4** | 2010-12-05 23:20:18, 2011-01-19 19:16:49, 2011... | -0Sgh0QlUKVsWosCWJzGqQ |
| **...** | ... | ... |
| **7603** | 2010-09-24 22:43:39, 2010-09-26 00:30:52, 2010... | zwmps5SXn30g-f5wqg_r9A |
| **7604** | 2012-09-24 19:27:35, 2012-12-10 17:20:54, 2012... | zy6Hzao6lTmuJMZo4Bemlg |
| **7605** | 2016-12-22 02:40:01, 2016-12-22 03:48:05, 2016... | zyPGYeXF4XKCqNN1pjFWhg |
| **7606** | 2011-02-05 20:11:04, 2011-05-02 16:31:11, 2011... | zzsOLFhgUw8gnjLTVVItFA |
| **7607** | 2012-09-22 00:26:15, 2012-09-23 20:12:00, 2012... | zzwicjPC9g246MK2M1ZFBA |

7608 rows × 2 columns

## 1.2- Create Every Unique Day As A List

In [ ]:

```
# To analyze Check-in Numbers Daily, we should get check-in numbers in tabu

#(~10 Minutes to Run)
datelist = []
for i in range(len(check)):
    list_string2 = pd.Series(check['date'][i]).str.cat(sep=',')
    list_split2 = list_string2.split(',') #211050
    list_split_all2 = [x.strip(' ') for x in list_split2] #NoChange
    datetime_list = []
    for i in range(len(list_split_all2)):
        datetime_list.append(datetime.strptime(list_split_all2[i], '%Y-%m-%
    date_list= []
    for i in range(len(datetime_list)):
        date_list.append(datetime.date(datetime_list[i]))
    datelist.append(date_list)

unique_datelist1 = []
for i in datelist:
    unique_datelist1.append(list(set(i)))

unique_datelist2 = []
for i in range(len(unique_datelist1)): #7230
    for k in range(len(unique_datelist1[i])):
        unique_datelist2.append(unique_datelist1[i][k])
#
unique_datelist2 = list(set(unique_datelist2))
unique_datelist2.sort()
for i in range(len(unique_datelist2)):
    check[str(unique_datelist2[i])] = ""

"""
Date count for each business to columns
"""
for i in range(len(datelist)):
    for date in datelist[i]:
            trh = date.strftime("%Y-%m-%d")
            count = datelist[i].count(date)
            check.loc[i,trh] = int(count)

for i in range(len(datelist)):
    for date in datelist[i]:
            trh = date.strftime("%Y-%m-%d")
            count = datelist[i].count(date)
            check.loc[i,trh] = int(count)

check.drop(columns='date',inplace=True)
#Save This File as check_1_by_dates.csv
#check.to_csv('check_1_by_dates.csv',index=False)

check.head()
```

| | business_id | 2010-01-16 | 2010-01-17 | 2010-01-18 | 2010-01-19 | 2010-01-20 | 2010-01-21 | 2010-01-22 | 2010-01-23 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -01XupAWZEXbdNbxNg5mEg | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | -092wE7j5HZOogMLAh40zA | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | 4RSJzjaxYJ9CRTfrk1sHZg | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | 4RV97YE8VEw05tu0WO425g | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | -0Sgh0QlUKVsWosCWJzGqQ | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

5 rows × 3227 columns

## 1.3- Transpose for Daily #s by Business

In [ ]:

```
#Transposing to Have Business ID's in Columns and Dates in Rows
#Later we will transpose again to get Weekly&Monthly Numbers

# Create check_t
check= pd.read_csv('check_1_by_dates.csv')
check.drop(columns='Unnamed: 0',inplace=True)
check_t = check.T
check_t.reset_index(inplace=True)
check_t.columns = check_t.loc[0,:] #Change column names
check_t.drop(0,inplace=True) # Drop first row
check_t.reset_index(drop=True,inplace=True)
check_t.rename(columns={'business_id':'Date'},inplace=True)
check_t = check_t[:-1] #Drop Sum Row, Last
check_t.sort_values('Date',inplace=True)
check_t['Date'] =  pd.to_datetime(check_t['Date'])

# Filter only for 2015
start_date = pd.to_datetime('01/01/2015')
end_date = pd.to_datetime('01/01/2016')
check_t_2015 = check_t.loc[(check_t['Date'] >= start_date) & (check_t['Date
check_t_2015.head()
```

Out[ ]:

| | Date | -01XupAWZEXbdNbxNg5mEg | -092wE7j5HZOogMLAh40zA | 4RSJzjaxYJ9CRTfrk' |
|---|---|---|---|---|
| **1811** | 2015-01-01 | NaN | NaN | |
| **1812** | 2015-01-02 | NaN | NaN | |
| **1813** | 2015-01-03 | NaN | NaN | |
| **1814** | 2015-01-04 | NaN | NaN | |
| **1815** | 2015-01-05 | 1.0 | NaN | |

5 rows × 7609 columns

## 1.4- Weekly Check-ins

In [ ]:

```
# To Create Weekly and Monthly Checkin Numbers as Sum

#Create Weekly Checkin Numbers as Sum
ch_t_2015_weekly= check_t_2015.groupby(pd.Grouper(key='Date',freq='W'),as_i
ch_2015_Weekly =ch_t_2015_weekly.T
ch_2015_Weekly = pd.DataFrame(ch_2015_Weekly)
ch_2015_Weekly.columns += 1 # Add 1 to each week number, (to fix)
ch_2015_Weekly.reset_index(inplace=True)
ch_2015_Weekly['sum'] = np.sum(ch_2015_Weekly.iloc[:,1:],axis=1)
ch_2015_Weekly.rename(columns={0:'business_id'},inplace=True)

#Save This file as check_2_2015_Weekly.csv
#ch_2015_Weekly.to_csv('check_2_2015_Weekly.csv')
ch_2015_Weekly
```

Out[ ]:

| | business_id | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 45 | 46 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -01XupAWZEXbdNbxNg5mEg | 0.0 | 2.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0. |
| **1** | -092wE7j5HZOogMLAh40zA | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0. |
| **2** | 4RSJzjaxYJ9CRTfrk1sHZg | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 1. |
| **3** | 4RV97YE8VEw05tu0WO425g | 0.0 | 0.0 | 2.0 | 1.0 | 0.0 | 0.0 | 2.0 | 1.0 | 0.0 | ... | 1.0 | 0.0 | 1. |
| **4** | -0Sgh0QlUKVsWosCWJzGqQ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0. |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **7603** | zwmps5SXn30g-f5wqg_r9A | 0.0 | 2.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 2.0 | 1.0 | ... | 1.0 | 1.0 | 1. |
| **7604** | zy6Hzao6lTmuJMZo4BemIg | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0. |
| **7605** | zyPGYeXF4XKCqNN1pjFWhg | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0. |
| **7606** | zzsOLFhgUw8gnjLTVVItFA | 0.0 | 0.0 | 1.0 | 2.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 1. |
| **7607** | zzwicjPC9g246MK2M1ZFBA | 1.0 | 1.0 | 1.0 | 5.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | ... | 0.0 | 0.0 | 0. |

7608 rows × 55 columns

## 1.5 Monthly Check-ins

In [ ]:

```
# Create Monthly Checkin Numbers as Sum
ch_2015_monthly_1 = check_t_2015.groupby(pd.Grouper(key='Date',freq='M'),as
ch_2015_Monthly = ch_2015_monthly_1.T
ch_2015_Monthly.columns += 1 # Add 1 to each month number, (to fix)
ch_2015_Monthly['sum'] = np.sum(ch_2015_Monthly,axis=1)
ch_2015_Monthly.reset_index(inplace=True)
ch_2015_Monthly.rename(columns={0:'business_id'},inplace=True)

#Save This file as check_3_2015_Monthly.csv
#ch_2015_Monthly.to_csv('check_3_2015_Monthly.csv')

ch_2015_Monthly
```

Out[ ]:

| | business_id | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -01XupAWZEXbdNbxNg5mEg | 3.0 | 1.0 | 3.0 | 2.0 | 6.0 | 3.0 | 1.0 | 2.0 | 3.0 | 3.0 | 0.0 | 2.0 | 2 |
| **1** | -092wE7j5HZOogMLAh40zA | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | |
| **2** | 4RSJzjaxYJ9CRTfrk1sHZg | 5.0 | 2.0 | 7.0 | 1.0 | 3.0 | 1.0 | 4.0 | 3.0 | 3.0 | 3.0 | 2.0 | 0.0 | 3 |
| **3** | 4RV97YE8VEw05tu0WO425g | 3.0 | 3.0 | 4.0 | 3.0 | 4.0 | 2.0 | 2.0 | 4.0 | 1.0 | 6.0 | 2.0 | 0.0 | 3 |
| **4** | -0Sgh0QlUKVsWosCWJzGqQ | 0.0 | 3.0 | 0.0 | 0.0 | 1.0 | 0.0 | 2.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **7603** | zwmps5SXn30g-f5wqg_r9A | 5.0 | 5.0 | 9.0 | 3.0 | 5.0 | 3.0 | 5.0 | 4.0 | 2.0 | 4.0 | 4.0 | 3.0 | 5 |
| **7604** | zy6Hzao6lTmuJMZo4BemIg | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | |
| **7605** | zyPGYeXF4XKCqNN1pjFWhg | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **7606** | zzsOLFhgUw8gnjLTVVItFA | 3.0 | 2.0 | 1.0 | 2.0 | 0.0 | 5.0 | 5.0 | 2.0 | 0.0 | 2.0 | 1.0 | 7.0 | 3 |
| **7607** | zzwicjPC9g246MK2M1ZFBA | 8.0 | 2.0 | 2.0 | 0.0 | 3.0 | 1.0 | 1.0 | 1.0 | 4.0 | 2.0 | 0.0 | 1.0 | 2 |

7608 rows × 14 columns

## 1.6- Filtering Businesses At Least 1 Monthly Check-in

In [ ]:

```
# To Filter out Business that does not have at least #1 Check-in in each mo
ch_2015_Monthly_NZ = ch_2015_Monthly.loc[(ch_2015_Monthly[1] != 0)
& (ch_2015_Monthly[2] != 0)
& (ch_2015_Monthly[3] != 0)
& (ch_2015_Monthly[4] != 0)
& (ch_2015_Monthly[5] != 0)
& (ch_2015_Monthly[6] != 0)
& (ch_2015_Monthly[7] != 0)
& (ch_2015_Monthly[8] != 0)
& (ch_2015_Monthly[9] != 0)
& (ch_2015_Monthly[10] != 0)
& (ch_2015_Monthly[11] != 0)
& (ch_2015_Monthly[12] != 0)
]
ch_2015_Monthly_NZ.reset_index(drop=True,inplace=True)
ch_2015_Monthly_NZ.rename(columns={0:'business_id'},inplace=True)

#Save This file as check_4_2015_Monthly_NZ.csv
#ch_2015_Monthly_NZ.to_csv('check_4_2015_Monthly_NZ.csv')

#2
#Create and Save business_id's in check_4_2015_Monthly_NZ.csv
#Also Add Business Numbers from 0, reason is having business_ID's in numbe

b_id_NZ_df = ch_2015_Monthly_NZ['business_id'].to_frame()
b_id_NZ_df['b_num'] = b_id_NZ_df.index

#Save This file as b_id_NZ_df.csv
#b_id_NZ_df.to_csv('b_id_NZ_df.csv')
ch_2015_Monthly_NZ
```

| | business_id | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0tgMGl7D9B10YjSN2ujLA | 20.0 | 4.0 | 9.0 | 9.0 | 5.0 | 3.0 | 5.0 | 6.0 | 13.0 | 6.0 |
| **1** | -1UMR00eXtwaeh59pEiDjA | 8.0 | 20.0 | 21.0 | 9.0 | 15.0 | 12.0 | 14.0 | 8.0 | 13.0 | 16.0 |
| **2** | -1VaIJza42Hjev6ukacCNg | 6.0 | 3.0 | 6.0 | 6.0 | 6.0 | 4.0 | 9.0 | 7.0 | 7.0 | 8.0 |
| **3** | -faYl3ejjjm7ciR_j1ke7A | 4.0 | 8.0 | 4.0 | 5.0 | 2.0 | 9.0 | 6.0 | 2.0 | 5.0 | 6.0 |
| **4** | -4TMQnQJW1yd6NqGRDvAeA | 13.0 | 21.0 | 24.0 | 14.0 | 19.0 | 23.0 | 3.0 | 10.0 | 9.0 | 21.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2256** | zqNgwQjj0_XAll-neGikIw | 12.0 | 14.0 | 21.0 | 11.0 | 28.0 | 7.0 | 17.0 | 16.0 | 19.0 | 11.0 |
| **2257** | zrMGL9riavdJR4olXaACOQ | 17.0 | 9.0 | 8.0 | 2.0 | 6.0 | 6.0 | 3.0 | 3.0 | 3.0 | 7.0 |
| **2258** | ztP466jMUMtqLwwHqXbk9w | 15.0 | 8.0 | 16.0 | 4.0 | 20.0 | 14.0 | 15.0 | 15.0 | 10.0 | 12.0 |
| **2259** | zwNLJ2VglfEvGu7DDZjJ4g | 3.0 | 14.0 | 7.0 | 5.0 | 9.0 | 12.0 | 3.0 | 4.0 | 6.0 | 8.0 |
| **2260** | zwmps5SXn30g-f5wqg_r9A | 5.0 | 5.0 | 9.0 | 3.0 | 5.0 | 3.0 | 5.0 | 4.0 | 2.0 | 4.0 |

2261 rows × 14 columns

## 1.7- Total # of CH for Each Day

In [ ]:

```
# To see # of CH for Every Day in the Dataset

check_dates = pd.DataFrame(check.apply(lambda x: pd.to_numeric(x,errors='co
check_dates.reset_index(inplace=True)
check_dates.rename(columns={'index':'date',0:'count'},inplace=True)
check_dates.drop(check_dates.index[[[0,1,len(check_dates)-1]]],inplace=True
check_dates.reset_index(drop=True,inplace=True)
check_dates['date'] = pd.to_datetime(check_dates.date)
check_dates['WeekDay']= check_dates.date.dt.day_name()
#Save This file as check_5_dates.csv
#check_dates.to_csv('check_5_dates.csv')
check_dates.sample(5)
```

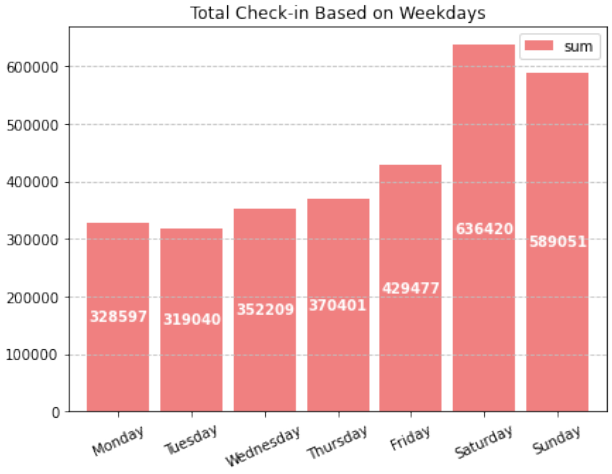
| | date | count | WeekDay |
|---|---|---|---|
| **1813** | 2015-01-04 | 1536.0 | Sunday |
| **2173** | 2015-12-30 | 1198.0 | Wednesday |
| **2816** | 2017-10-03 | 647.0 | Tuesday |
| **2289** | 2016-04-24 | 1809.0 | Sunday |
| **2172** | 2015-12-29 | 1043.0 | Tuesday |

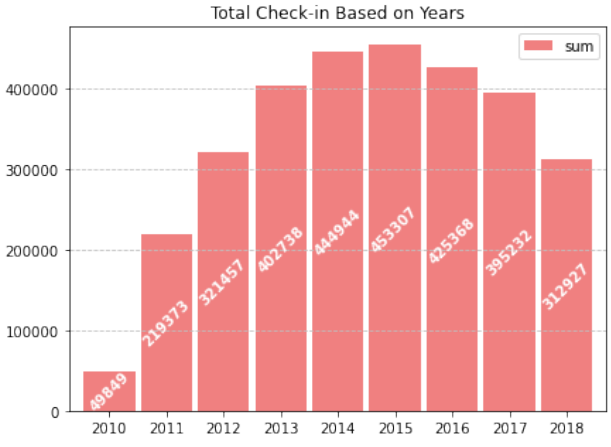
## 1.8- Create Check dates to Plot

In [ ]:
```
# Create Check Dates by Month
ch_bymonth = check_dates.groupby(pd.Grouper(key='date',freq='M')).sum() #Da
ch_bymonth.reset_index(inplace=True)
ch_bymonth.rename(columns={'count':'sum'},inplace=True)
ch_bymonth['month']=pd.to_datetime(ch_bymonth['date']).dt.strftime('%b-%Y')

# Create Check Dates by Year
ch_byyear = check_dates.groupby(pd.Grouper(key='date',freq='Y')).sum() #Dat
ch_byyear.reset_index(inplace=True)
ch_byyear.rename(columns={'count':'sum'},inplace=True)
ch_byyear['year']=pd.to_datetime(ch_byyear['date']).dt.strftime('%Y')

#Check dates by Weekday
ch_byweekday = check_dates.groupby('WeekDay').agg({       #Dates Group By W
'count': ['sum','mean','min','max']
})
ch_byweekday.reset_index(inplace=True)
#Create days of week as a list
days = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sund

#Make ch_byweekday a single level column
ch_byweekday.columns = ['WeekDay','sum','mean','min','max']

ch_byweekday.set_index('WeekDay',inplace=True)
ch_byweekday['WeekDay'] = days
ch_byweekday = ch_byweekday.reindex(days)
ch_byweekday.drop('WeekDay',axis=1,inplace=True)
ch_byweekday.reset_index(inplace=True)
```

## 1.9- Plot Checkin By WeekDay and Year

In [ ]:
```
fig, axs = plt.subplots(1,2, figsize=(15,5))

#Check-in Based on Weekdays
ax0 = ch_byweekday.plot(kind='bar',x='WeekDay',y='sum',ax=axs[0],color='lig
for container in ax0.containers:
    ax0.bar_label(container,label_type='center',color='white',weight='bold'
ax0.set_title('Total Check-in Based on Weekdays')
#ax1 = ch_bymonth.plot(kind='bar',x='month',y='sum',ax=axs[1,[0,1]],color=
ax0.set(xlabel='')
ax2 = ch_byyear.plot(kind='bar',x='year',y='sum',ax=axs[1],color='lightcora
for container in ax2.containers:
    ax2.bar_label(container,label_type='center',color='white',weight='bold'
ax2.set_title('Total Check-in Based on Years')
ax2.set(xlabel='')
ax_list = [ax0,ax2]
for ax in ax_list:
    ax.yaxis.grid(color='silver',which='major',linestyle='--')
```




## 1.10- Create Check-in Frequency

In [ ]:

```
ch_freq = check_t_2015.T

ch_freq.reset_index(inplace=True)

for i in range(1,len(ch_freq.columns)):
    ch_freq.iloc[0,i] = ch_freq.iloc[0,i].date()

ch_freq.columns = ch_freq.iloc[0,:]

ch_freq.drop(0,axis=0,inplace=True)
ch_freq.rename(columns={'Date':'business_id'},inplace=True)
ch_freq.replace(ch_freq.iloc[0,1],0,inplace=True)
ch_freq['ch_freq'] = (np.count_nonzero(ch_freq,axis=1) / (len(ch_freq.colur

ch_freq = ch_freq[['business_id','ch_freq']]
#Save ch_freq as ch_freq.csv
#ch_freq.to_csv('check_6_freq.csv')

#Create ch_freq 2015, also with b_num
ch_freq_2015 = ch_freq.merge(b_id_NZ_df)
#Save ch_freq_2015 as check_7_freq_2015.csv
#ch_freq_2015.to_csv('check_7_freq_2015.csv')
ch_freq_2015
```

Out[ ]:

| | business_id | ch_freq | b_num |
|---|---|---|---|
| **0** | -0tgMGl7D9B10YjSN2ujLA | 0.183562 | 0 |
| **1** | -1UMR00eXtwaeh59pEiDjA | 0.372603 | 1 |
| **2** | -1VaIJza42Hjev6ukacCNg | 0.200000 | 2 |
| **3** | -faYl3ejjjm7ciR_j1ke7A | 0.145205 | 3 |
| **4** | -4TMQnQJW1yd6NqGRDvAeA | 0.279452 | 4 |
| **...** | ... | ... | ... |
| **2255** | zqNgwQjj0_XAll-neGikIw | 0.328767 | 2255 |
| **2256** | zrMGL9riavdJR4olXaACOQ | 0.150685 | 2256 |
| **2257** | ztP466jMUMtqLwwHqXbk9w | 0.358904 | 2257 |
| **2258** | zwNLJ2VglfEvGu7DDZjJ4g | 0.194521 | 2258 |
| **2259** | zwmps5SXn30g-f5wqg_r9A | 0.131507 | 2259 |

2260 rows × 3 columns

# 2 Business

## Read & Normalize

In [ ]:

```
#1.1 Read Original File & Normalize
bus = pd.read_csv('bus_az_open.csv')
bus = pd.json_normalize(bus.j.apply(json.loads))
bus
```

Out[ ]:

| | city | name | stars | state | address | is_open | latitude | longitude | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Phoenix | Taco Bell | 3.0 | AZ | 2450 E Indian School Rd | 1 | 33.495194 | -112.028588 | R |
| **1** | Phoenix | Nee House Chinese Restaurant | 3.5 | AZ | 13843 N Tatum Blvd, Ste 15 | 1 | 33.613020 | -111.977036 | I |
| **2** | Phoenix | Vals Getaway DES Cafeteria | 3.0 | AZ | 1789 W Jefferson | 1 | 33.446806 | -112.097129 | R I |
| **3** | Scottsdale | Salt Cellar | 3.5 | AZ | 550 N Hayden Rd | 1 | 33.454286 | -111.909405 | I |
| **4** | Phoenix | Sushi Mocorito | 3.0 | AZ | 3415 West Glendale Ave, Bldg 25A | 1 | 33.537164 | -112.132926 | R |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **7675** | Scottsdale | Cherry On Top | 4.0 | AZ | 32421 N Scottsdale Rd, Ste 115 | 1 | 33.779794 | -111.923232 | I R |
| **7676** | Phoenix | Subway | 2.0 | AZ | 4747 E Elliot Rd, Ste 33 | 1 | 33.347499 | -111.981221 | R S |
| **7677** | Peoria | Razzleberriez Frozen Yogurt | 4.0 | AZ | 25225 North Lake Pleasant Pkwy | 1 | 33.713475 | -112.269849 | I |
| **7678** | Gilbert | Pomo Pizzeria Gilbert | 4.0 | AZ | 366 N Gilbert Rd, Ste 106 | 1 | 33.356542 | -111.790019 | R It |
| **7679** | Mesa | Smashburger | 3.0 | AZ | 2837 N Power Rd, Ste 102 | 1 | 33.467537 | -111.682818 | R |

7680 rows × 60 columns

## 1.1 Add Chain Data

In [ ]:
```
#Whether Restaurant is A Chain or Not
name_value_count = bus.name.value_counts().to_frame().reset_index().rename(
name_value_count.sample(5)
bus = pd.merge(bus, name_value_count, on='name')
for i in range(len(bus)):
    if bus.loc[i,'count'] > 1:
        bus.loc[i,'is_chain'] = 1
    else:
        bus.loc[i,'is_chain'] = 0
bus.drop(columns='count',inplace=True)
bus.loc[:,['name','is_chain']].sample(5)
```

Out[ ]:

| | name | is_chain |
|---|---|---|
| **151** | Filiberto's Mexican Food | 1.0 |
| **1109** | Colados Coffee & Crepes | 0.0 |
| **1029** | Famous Dave's Bar-B-Que | 1.0 |
| **874** | Pomo Pizzeria - Phoenix | 0.0 |
| **1597** | Sacks Sandwiches - Phoenix | 0.0 |

## 1.2 Filter only for 2015

In [ ]:
```
#Filter only for 2015
b_id_NZ_df = pd.read_csv('b_id_NZ_df.csv')
b_id_NZ_df.drop(['Unnamed: 0'], axis=1, inplace=True)
bus = bus.merge(b_id_NZ_df, on='business_id')

#Save This File as bus_1_2015.csv
#bus.to_csv('bus_1_2015.csv', index=False)
bus
```

Out[ ]:

| | city | name | stars | state | address | is_open | latitude | longitude |
|---|---|---|---|---|---|---|---|---|
| **0** | Phoenix | Nee House Chinese Restaurant | 3.5 | AZ | 13843 N Tatum Blvd, Ste 15 | 1 | 33.613020 | -111.977036 |
| **1** | Scottsdale | Salt Cellar | 3.5 | AZ | 550 N Hayden Rd | 1 | 33.454286 | -111.909405 |
| **2** | Scottsdale | Yummy Yummy Chinese Restaurant | 3.0 | AZ | 2765 N Scottsdale Rd, Ste 105 | 1 | 33.478754 | -111.925484 |
| **3** | Phoenix | Oregano's Pizza Bistro | 3.5 | AZ | 1008 E Camelback Rd | 1 | 33.509510 | -112.060238 |
| **4** | Phoenix | La Flor De Calabaza | 3.0 | AZ | 705 N 1st St, Ste 110 | 1 | 33.455949 | -112.072089 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **2255** | Goodyear | Ada's Fish Fry | 4.0 | AZ | 14960 W Indian School Rd, Ste 380 | 1 | 33.492340 | -112.380230 |
| **2256** | Scottsdale | Jersey Mike's Subs | 3.0 | AZ | 15807 E Frank Lloyd Wright | 1 | 33.629769 | -111.887512 |
| **2257** | Avondale | Pita Kitchen - Avondale | 4.5 | AZ | 9915 W McDowell Rd, Ste 104 | 1 | 33.464143 | -112.273057 |
| **2258** | Peoria | Sushi Yuki | 4.0 | AZ | 8345 W Thunderbird Rd | 1 | 33.608420 | -112.238536 |
| **2259** | Mesa | Smashburger | 3.0 | AZ | 2837 N Power Rd, Ste 102 | 1 | 33.467537 | -111.682818 |

2260 rows × 61 columns

## 1.2 Distance to Center

In [ ]:
```
#1.2 Distance to City Center
def haversine_vectorize(lon1, lat1, lon2, lat2):

    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    newlon = lon2 - lon1
    newlat = lat2 - lat1

    haver_formula = np.sin(newlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * r

    dist = 2 * np.arcsin(np.sqrt(haver_formula ))
    km = 6367 * dist #6367 for distance in KM for miles use 3958
    return km

bus['haversine_dist'] = haversine_vectorize(bus['longitude'],bus['latitude'
bus.loc[:,['business_id','haversine_dist']].sample(5)
```

Out[ ]:

| | business_id | haversine_dist |
|---|---|---|
| **352** | RZdrNayqoUjovdoJ4GWe-Q | 23.602938 |
| **1468** | Jd2NUZeVeyWi1tTNKXMt1Q | 24.987059 |
| **1098** | 4nLqlBBt6jcvphkIQxz0gQ | 26.367048 |
| **284** | JIETwXSEGIHMNyxZVEnNKQ | 14.881568 |
| **2245** | _JZ7hXqOZ_MngjPWFgER0w | 1.669024 |

## 1.3 Add Income & Population

In [ ]:
```
# Income and Population Data for 2015
#Source for income is http://www.usa.com/rank/arizona-state--per-capita-inc
income = pd.read_csv('Income_Az.csv')
income.reset_index(drop=True,inplace=True)
income['zip'] = income['zip'].astype(int, errors = 'raise')
bus.rename(columns={'postal_code':'zip'},inplace=True)
bus['zip'] = bus['zip'].astype(int, errors = 'ignore')
bus = pd.merge(bus, income ,on= 'zip',how='left')

#No Population & income info for 5 restaurants, so drop them
bus.drop(bus[bus.population.isnull()].index,inplace=True)
bus.reset_index(drop=True,inplace=True)

#Since we dropped 5 businesses, also drop them from business_id_NZ_df, and
b_id_NZ_df_2 = bus.loc[:,['business_id','b_num']]
b_id_NZ_df_2.reset_index(drop=True,inplace=True)
b_id_NZ_df_2.to_csv('b_id_NZ_df_2.csv')

bus.loc[:,['business_id','zip','pc_income']].sample(5)
```

Out[ ]:

| | business_id | zip | pc_income |
|---|---|---|---|
| **1753** | k6pWX0v60LIRS2guzkpxqQ | 85256 | 14446.0 |
| **2051** | 5pMNyj77S1P3rrT69y1pkQ | 85251 | 40167.0 |
| **514** | JwXNyXdJmk_YnXokLhpI8w | 85042 | 19688.0 |
| **1256** | LLWOrt8Vo5kyA9qeo-Y76g | 85204 | 19082.0 |
| **635** | EAs61Wm1O6tLjCs8t2eP-Q | 85338 | 24097.0 |

## 1.6 Attributes Fix for Parking & Ambiance & GoodForMeal

In [ ]:

```
def Att_Fix(data,attribute_name):
    data[attribute_name] = data[attribute_name].astype('str')
    data[attribute_name] = data[attribute_name].replace({'\'': '"'},regex=T
    att_list = []
    value_list = []
    df = pd.DataFrame()
    for i in range(len(data)):
        df.loc[i,'business_id'] = data.loc[i,'business_id']
        pos_colon = [m.start() for m in re.finditer(':',data.loc[i,attribut
        pos_dq = [m.start() for m in re.finditer('"',data.loc[i,attribute_r
        pos_comma = [m.start() for m in re.finditer(',',data.loc[i,attribut
        pos_comma.append(len(data.loc[i,attribute_name])-1)
        for j in range(len(pos_colon)):
            j1 = j*2
            j2 = (j*2 + 1)
            att = data.loc[i,attribute_name][pos_dq[j1]+1:pos_dq[j2]]
            att_list.append(att)

            value = data.loc[i,attribute_name][pos_colon[j]+2:pos_comma[j]]
            value_list.append(value)
            pos1 = attribute_name.find('.')
            attribute_name_2 = attribute_name[pos1+1:]
            att_name = str(attribute_name_2) + '_' + att
            df.loc[i,att_name] = value

        df = df.fillna(0)
        df.replace({False: 0, True: 1,'False':0, 'True':1, 'None':0},inplac

    return df

parking_df = Att_Fix(bus,'attributes.BusinessParking')
bus = pd.merge(bus,parking_df, on='business_id')
ambiance_df = Att_Fix(bus,'attributes.Ambience')
bus = pd.merge(bus,ambiance_df, on='business_id')
meal_df = Att_Fix(bus,'GoodForMeal')
bus = pd.merge(bus,meal_df, on='business_id')
```

In [ ]:

```
#Check BusinessParking
bus.iloc[:,64:69].sample(5)
```

| | population | BusinessParking_garage | BusinessParking_street | BusinessParking_validat |
|---|---|---|---|---|
| **1813** | 62767.0 | 0 | 0 | |
| **1680** | 33432.0 | 0 | 0 | |
| **1282** | 19260.0 | 0 | 0 | |
| **2032** | 37550.0 | 0 | 0 | |
| **811** | 57507.0 | 0 | 0 | |

In [ ]:
```
#Check Ambiance
bus.iloc[:,70:77].sample(5)
```

| | Ambience_romantic | Ambience_intimate | Ambience_classy | Ambience_hipster | Ambier |
|---|---|---|---|---|---|
| **1578** | 0 | 0 | 0 | 0 | |
| **816** | 0 | 0 | 0 | 0 | |
| **1428** | 0 | 0 | 0 | 1 | |
| **378** | 0 | 0 | 0 | 0 | |
| **139** | 0 | 0 | 0 | 0 | |

In [ ]:
```
#Check GoodForMeal
bus.iloc[:,-6:].sample(5)
```

Out[ ]:

| | GoodForMeal_dessert | GoodForMeal_latenight | GoodForMeal_lunch | GoodForMeal_dinn |
|---|---|---|---|---|
| **949** | 0 | 1 | 1 | |
| **1193** | 0 | 0 | 0 | |
| **252** | 0 | 0 | 0 | |
| **1067** | 0 | 0 | 1 | |
| **1356** | 0 | 1 | 1 | |

## 1.7 Wifi & Alcohol Fix

In [ ]:

```
#Fix Wifi Column and Preview
bus['attributes.WiFi'].replace({
"u'free'": 1,
"u'no'": 0,
"'free'":1,
"'no'":0,
"u'paid'":1,
"'paid'":1,
'None':0},inplace=True)

bus['attributes.Alcohol'].replace({
"'beer_and_wine'": 1,
"u'no'": 0,
"'full_bar'":1,
"'no'":0,
"u'full_bar'":1,
"u'beer_and_wine'":1,
'None':0,
"'none'":0,
"u'none'":0},inplace=True)

bus.loc[:,['business_id','attributes.WiFi']].sample(5)
```

Out[ ]:

| | business_id | WiFi | Alcohol |
|---|---|---|---|
| **92** | D6xj8xnc-R7Y2y18isc0-A | 1.0 | 0.0 |
| **865** | uU8up3hGwW9qnzQD1HZsMQ | 1.0 | 0.0 |
| **422** | 2c9Vptks_vowLgVUMnCgjw | 1.0 | 1.0 |
| **1203** | hDJZh2TD9f-TVEVun75IJg | 0.0 | 1.0 |
| **1480** | Rg43a31sV7gNLZMjy-ZdCg | 1.0 | 1.0 |

## 1.8 Categories List to Columns

In [ ]:
```
#Business Categories
list_string = bus['categories'].str.cat(sep=',')
list_split = list_string.split(',') #211050
list_split_all = [x.strip(' ') for x in list_split] # Take Spaces
count = Counter(list_split_all)
unique_list = list(count.keys())
most_frequent = count.most_common(20) #Change i to get i different categori
bus_cat_unique = pd.DataFrame(most_frequent,columns=['Categories','count'])
bus_cat_unique = bus_cat_unique[bus_cat_unique['Categories'] != 'Restaurant
bus_cat_unique.reset_index(inplace=True,drop='index')
bus_cat_unique.sort_values('count',inplace=True,ascending=False)
bus_cat_unique.drop(index=bus_cat_unique.index[0],
        axis=0,
        inplace=True)
bus_cat_unique.reset_index(inplace=True,drop=True)

#Categories List to Columns
cat_list = bus_cat_unique.iloc[:,0].to_list()

for i in range(len(bus)):
    for j in cat_list:
        name = str('c_') + j
        if j in bus.iloc[i,8]:
            bus.loc[i,name] = 1
        else:
            bus.loc[i,name] = 0
```

## 1.9 Fix City Names

In [ ]:
```
#Make All Start with Capital Letter
for i in range(len(bus)):
    bus.loc[i,'city'] = bus.loc[i,'city'].title()
```

## 1.10 Additional Changes for Attributes

In [ ]:
```
#Additional Changes for Attributes
bus.drop(columns=['','Ambience','BusinessParking','RestaurantsAttire'],inpl

bus.rename(columns={'_breakfast':'GFM_breakfast',
 '_brunch':'GFM_brunch',
 '_dessert':'GFM_dessert',
 '_dinner':'GFM_dinner',
 '_latenight':'GFM_latenight',
 '_lunch':'GFM_lunch'},inplace=True)

#bus.to_csv('bus_3_2015.csv',index=False)
```

# 3 Photo

## 3.1 Photo Compliment

In [ ]:
```
photo_comp = pd.read_csv('photo_az_compliment.csv')
photo_comp.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
photo_comp.sample(5)
```

Out[ ]:

| | business_id | 1_count | 2_count | 3_count | 4_count | 5_count | 6_count |
|---|---|---|---|---|---|---|---|
| **6319** | W7Dt3b6H_pMIHfxn49Pkzg | 0 | 0 | 0 | 0 | 0 | 0 |
| **5504** | sL7RKR9srM8vd0HAMApgpg | 0 | 0 | 0 | 0 | 0 | 0 |
| **936** | 7_YOEQXlrbGpF9t9TPG_nQ | 0 | 0 | 0 | 0 | 0 | 0 |
| **1640** | bSbAq9ylam6RInhbm2aZKA | 0 | 0 | 0 | 0 | 0 | 0 |
| **6986** | yXj1P-pc6CGDFSta-45L5Q | 0 | 0 | 0 | 0 | 0 | 0 |

## 3.2 Photo Table

In [ ]:
```
# Read, filter to 2015 and Preview Photo Table
photo = pd.read_csv('photo_az_open.csv')
photo.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
#Filter it for 2015, and save as photo_2015.csv
photo_2015_NZ = photo.merge(b_id_NZ_df_2, on='business_id')
photo_2015_NZ.to_csv('photo_2015_NZ.csv',index=False)
photo_2015_NZ.sample(5)
```

Out[ ]:

| | business_id | photo_count | inside_count | outside_count | food_count | dr |
|---|---|---|---|---|---|---|
| **550** | dUffgo9Lh_Vk9TLuFR5ywg | 10 | 1 | 1 | 8 | |
| **846** | Gl01qPpjC9DKN_Py3lEnpA | 6 | 1 | 0 | 5 | |
| **1381** | XcOy_IMkPO9jGBox6mHT8w | 3 | 0 | 0 | 3 | |
| **101** | m1Y47fK3vWCJXV3TdCn_Xw | 28 | 5 | 0 | 17 | |
| **165** | NAPuzJcanOp91dE4GXPOow | 23 | 9 | 0 | 9 | |

# 4 Review & Users

## 4.1 Review - Read and Filter for 2015 & NZ Businesses

In [ ]:

```
review = pd.read_csv('review_az_open.csv')
review.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
#After Read it, first for 2015, and then Filter for Monthly NZ businesses,

start_date = pd.to_datetime('01/01/2015')
end_date = pd.to_datetime('01/01/2016')
review['date'] = pd.to_datetime(review['date']).dt.normalize()
review_2015 = review.loc[(review['date'] >= start_date) & (review['date'] <
review_2015.reset_index(drop=True,inplace=True)

b_id_NZ_df_2 = pd.read_csv('b_id_NZ_df_2.csv')
b_id_NZ_df_2.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
review_2015_NZ = pd.merge(review_2015, b_id_NZ_df_2 ,on= 'business_id')
#Save This as review_2015_NZ.csv
#review_2015_NZ.to_csv('review_2015_NZ.csv',index=False)
```

In [ ]:

```
#Preview All Reviews (before Filter)
review
```

Out[ ]:

| | date | business_id | cool | useful | stars | user_id |
|---|---|---|---|---|---|---|
| **0** | 2018-03-06 | Ap17p-tA4lpzy1s35DfCfg | 0 | 0 | 5.0 | MUq3DlU0YeDHb1yG8tvfHA |
| **1** | 2017-09-11 | OBnqMdlsS8vWPrzPtnlb-w | 0 | 0 | 5.0 | 4ylg0SlpEuyOJF_c8b06cw |
| **2** | 2015-05-02 | QYlhMNwxXejDoCr-F1BxDg | 0 | 0 | 4.0 | VHalHQZYevvanHSzMHMVcQ |
| **3** | 2016-08-28 | aXiXvy5cBMeb6m2j7WFuzg | 0 | 1 | 5.0 | TMaBWSv75DEuXyZlGJj6cg |
| **4** | 2015-02-25 | 0Ryxx6uSpt4TOKYVGBekLQ | 0 | 0 | 1.0 | Tmw8c42MbUC968uKahphAw |
| **...** | ... | ... | ... | ... | ... | .. |
| **1013461** | 2017-06-13 | fnPefa87H-3CJnDungeetQ | 0 | 0 | 1.0 | jrme91vu5xKwY1U2-ljlqA |

| | | | | | | |
|---|---|---|---|---|---|---|
| **1013462** | 2017-02-24 | wu5nk3pY1e_ou_Le_U52Sw | 0 | 0 | 5.0 | Wv0-jQ_XHF6LX0mfXqTCjw |
| **1013463** | 2017-08-02 | mZK8IBkMFzOX2UmA7_BylA | 1 | 1 | 5.0 | NEgAr1eamnNR-mDI9aGXNQ |
| **1013464** | 2017-07-22 | fVJszMsossyPFYEgw1dQ0Q | 0 | 1 | 1.0 | zY_G0lOSD-CfKWB377JU5w |
| **1013465** | 2016-04-28 | IkLAUX7lZIWqE9_YC66SRw | 2 | 1 | 4.0 | CLt1Y9JoMW41YUBRLYfRLQ |

1013466 rows × 8 columns

In [ ]:

```
#Preview Reviews for 2015 NZ businesses, almost 10% of All Reviews
review_2015_NZ.sample(10)
```

Out[ ]:

| | date | business_id | cool | useful | stars | user_id | f |
|---|---|---|---|---|---|---|---|
| **31316** | 2015-08-26 | llQfDCKPXBW1-hSb_qUC-g | 0 | 0 | 5.0 | SrWLnzA9m72HIjFPV9xW0w | |
| **61980** | 2015-08-23 | pQSn1vvdiZ3oumWjE1XUKQ | 0 | 0 | 5.0 | RGwsJFX8cLIngZM-0sUnog | |
| **53394** | 2015-06-23 | momjERcb6G6ienrfPuNArw | 0 | 1 | 3.0 | 8YbgE8nxu5nHkG_YJ4UuwQ | |
| **96560** | 2015-08-14 | uIE9H0cz2AvxorEqy5cZ0w | 2 | 1 | 3.0 | yVux4cP2BIuHDUv7RXpXPw | |
| **25970** | 2015-05-18 | xZcy6M1gi478O0lJMcA72g | 0 | 0 | 5.0 | l_9tGgUPR8WXOPfjJY3Jmw | |
| **29740** | 2015-05-23 | LHyj66waFlDcw9UToo27JQ | 2 | 0 | 5.0 | 8HEYCY23N2EgVtAld0Q_YA | |
| **34950** | 2015-06-11 | pSQFynH1VxkfSmehRXlZWw | 1 | 2 | 2.0 | eXLlai_1ntJVTBBaeanmPg | |
| **85746** | 2015-11-30 | -vCLrTTgw6pBufdarW8ynA | 0 | 0 | 3.0 | 0M-UeT6SmKmUqa31kSZQWA | |
| **91187** | 2015-06-01 | bVPTxKhjj4qyTF7_43nY-g | 0 | 1 | 1.0 | EUqbkOCHKgEoyT-q89AtBg | |
| **101813** | 2015-02-20 | 6pQ1DpZo6zygNc3qTch-iA | 1 | 0 | 5.0 | -QNiYuDcMOX6epQGZ7Z0Mg | |

### 4.1.2 Create b_id_NZ_df_3

In [ ]:
```
#Number of Unique Businesses dropped to 1793, so save them as b_id_NZ_df_3

#Add Unique values of business_id in review_nz_2015 to a new dataframe
b_id_nz_3_df = pd.DataFrame(list(review_2015_NZ['business_id'].unique()))
b_id_nz_3_df.rename(columns={0:'business_id'},inplace=True)

b_id_nz_3_df = b_id_nz_3_df.merge(review_2015_NZ[['business_id','b_num']],o
#Drop Duplicates
b_id_nz_3_df.drop_duplicates(inplace=True)
b_id_nz_3_df.reset_index(drop=True,inplace=True)
#b_id_nz_3_df.to_csv('b_id_NZ_df_3.csv',index=False)
```

## 4.2 Read Users

In [ ]:
```
users = pd.read_csv('users_az_open.csv')
users.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
users
```

Out[ ]:

| | user_id | name | review_count | yelping_since | friends_count |
|---|---|---|---|---|---|
| **0** | l6BmjZMeQD3rDxWUbiAiow | Rashmi | 95 | 2013-10-08 | 98 |
| **1** | 4XChL029mKr5hydo79Ljxg | Jenna | 33 | 2013-02-21 | 1151 |
| **2** | bc8C_eETBWL0olvFSJJd0w | David | 16 | 2013-10-04 | 14 |
| **3** | dD0gZpBctWGdWo9WlGuhlA | Angela | 17 | 2014-05-22 | 524 |
| **4** | MM4RJAeH6yuaN8oZDSt0RA | Nancy | 361 | 2013-10-23 | 230 |
| **...** | ... | ... | ... | ... | ... |
| **1637133** | oC4BjbB4I-wx5KBAO1k2Gg | Jiaxin | 2 | 2016-02-24 | 0 |
| **1637134** | LKd_-BT2GMWp8_1ho18kOg | Wes | 8 | 2017-02-25 | 0 |
| **1637135** | ib_elCBhGOamS0AALkh4Og | Lee | 4 | 2009-08-19 | 0 |
| **1637136** | WV27LoSFDG_YmiDKTLKKLA | Melissa | 1 | 2018-10-15 | 0 |
| **1637137** | agVqH9xDhNw8n7ve66H1Uw | Erin | 1 | 2016-03-03 | 0 |

1637138 rows × 11 columns

## 4.3 Add Gender for Users

In [ ]:
```
# Get gender for All & Show Value Count
import gender_guesser.detector as gender
d = gender.Detector()
users.loc[:,'gender'] = users.loc[:,'name'].map(lambda x: d.get_gender(x))
users.gender.value_counts()
```

Out[ ]:
```
female           639576
male             556370
unknown          261299
mostly_female     87294
mostly_male       68210
andy              24389
Name: gender, dtype: int64
```

### Fix Genders to Categorize

In [ ]:
```
users.replace({'female':'F','male':'M','mostly_female':'F','mostly_male':'M
```

## 4.4 Filter Users for 2015

In [ ]:
```
#Since our analysis on 2015, only works with users who starts yelping befor
end_date = pd.to_datetime('01/01/2015')
users['yelping_since'] = pd.to_datetime(users['yelping_since']).dt.normaliz
users_2015 = users.loc[(users['yelping_since'] < end_date)]
#users_2015.to_csv('users_2015.csv')
users_2015
```

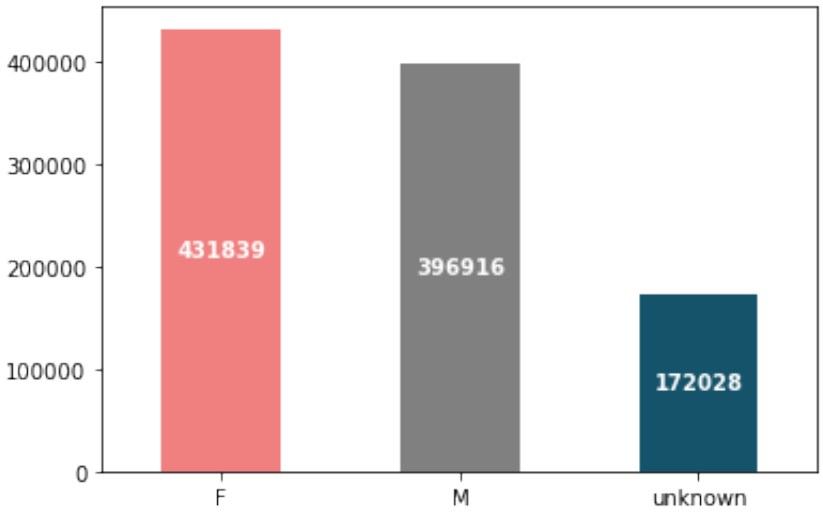
Out[ ]:

| | user_id | name | review_count | yelping_since | friends_count |
|---|---|---|---|---|---|
| **0** | l6BmjZMeQD3rDxWUbiAiow | Rashmi | 95 | 2013-10-08 | 98 |
| **1** | 4XChL029mKr5hydo79Ljxg | Jenna | 33 | 2013-02-21 | 1151 |
| **2** | bc8C_eETBWL0olvFSJJd0w | David | 16 | 2013-10-04 | 14 |
| **3** | dD0gZpBctWGdWo9WlGuhlA | Angela | 17 | 2014-05-22 | 524 |
| **4** | MM4RJAeH6yuaN8oZDSt0RA | Nancy | 361 | 2013-10-23 | 230 |
| **...** | ... | ... | ... | ... | ... |
| **1637127** | jFdjTXrUIBRZYDYlJXeWlg | K | 3 | 2014-08-27 | 0 |
| **1637128** | aEs0rWXz86cHpKeqLEJH5g | Sarah | 31 | 2012-04-05 | 0 |
| **1637129** | 5AKN9P61NGv6WAceYsJq7w | Tony | 1 | 2013-09-20 | 0 |
| **1637132** | G2HQPlmwW6kosWptBEbqrw | Michelle | 2 | 2013-10-09 | 0 |
| **1637135** | ib_elCBhGOamS0AALkh4Og | Lee | 4 | 2009-08-19 | 0 |

1000783 rows × 12 columns

### Gender Graph

In [ ]:
```
ax0 = users_2015.groupby('gender').size().plot(kind='bar',rot=0,color=['lig
for container in ax0.containers:
    ax0.bar_label(container,label_type='center',color='white',weight='bold'
```




### 4.4.1 Add User's Number of Rev in 2015

In [ ]:

```
u_rev_count_2015 = review_2015_NZ.groupby('user_id').size().to_frame().rese
users_2015 = users_2015.merge(u_rev_count_2015,on='user_id',how='left')
```

### 4.4.2 Create NZ User List

In [ ]:

```
#Users that have at least 1 fans, funny, cool, useful, and review
users_2015_NZ = users_2015[(users_2015['fans'] != 0)
& (users_2015['funny_count'] != 0)
& (users_2015['cool_count'] != 0)
& (users_2015['useful_count'] != 0)
& (users_2015['friends_count'] != 0)
& (users_2015['review_count'] != 0)]
users_2015_NZ.reset_index(drop=True,inplace=True)
users_2015_NZ
```

Out[ ]:

| | user_id | name | review_count | yelping_since | friends_count |
|---|---|---|---|---|---|
| **0** | l6BmjZMeQD3rDxWUbiAiow | Rashmi | 95 | 2013-10-08 | 98 |
| **1** | 4XChL029mKr5hydo79Ljxg | Jenna | 33 | 2013-02-21 | 1151 |
| **2** | dD0gZpBctWGdWo9WlGuhlA | Angela | 17 | 2014-05-22 | 524 |
| **3** | MM4RJAeH6yuaN8oZDSt0RA | Nancy | 361 | 2013-10-23 | 230 |
| **4** | 0rK89TS8xqy1wI4nYI1wfw | Marilyn | 214 | 2011-06-23 | 5449 |
| **...** | ... | ... | ... | ... | ... |
| **228682** | rlmg0UUK7Nf3un69DFhw6g | Stephanie | 80 | 2008-05-19 | 1 |
| **228683** | 5uPJ6try4np37ICIvb_Glg | Judy | 52 | 2010-03-08 | 18 |
| **228684** | KhzTmXUCXIm4sLE4gxZg0g | Paul | 45 | 2013-09-28 | 1 |
| **228685** | mSaD4g37jCy4-95pB6NWKA | Suzy | 74 | 2010-03-15 | 1 |
| **228686** | 51VmTQD_CMkNjAHkR2RRyQ | Robert | 19 | 2013-01-13 | 3 |

228687 rows × 13 columns

## 4.5 Merge User and Review Files, for 2015

In [ ]:

```
review_user_2015 = pd.merge(review_2015_NZ, users_2015, on='user_id')
review_user_2015.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
review_user_2015
```

### 4.5.1 Create b_id_NZ_df_4

In [ ]:

```
#Number of Unique Businesses dropped to 1790, so save them as b_id_NZ_df_4.

#Add Unique values of business_id in review_nz_2015 to a new dataframe
b_id_nz_4_df = pd.DataFrame(list(review_user_2015['business_id'].unique()))
b_id_nz_4_df.rename(columns={0:'business_id'},inplace=True)

b_id_nz_4_df = b_id_nz_4_df.merge(review_user_2015[['business_id','b_num']]
#Drop Duplicates
b_id_nz_4_df.drop_duplicates(inplace=True)
b_id_nz_4_df.reset_index(drop=True,inplace=True)
#b_id_nz_4_df.to_csv('b_id_NZ_df_4.csv',index=False)
```

## 4.6 Sentiment Analysis on Reviews

In [ ]:

```
from nltk.corpus import stopwords
from sklearn.feature_extraction import _stop_words
#from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
my_stop_words = set(stopwords.words('english') + list(_stop_words.ENGLISH_S

from wordcloud import WordCloud
# concatenate all the reviews into one single string
full_text = ' '.join(review_user_2015.loc[100001:,'text'])
#cloud_no_stopword = WordCloud(background_color='white', stopwords=my_stop_

# Important to select data to use in analysis
data_to_use = review_user_2015.loc[:,'text']
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk import FreqDist
lower_full_text = full_text.lower()
word_tokens = word_tokenize(lower_full_text)
tokens = list()
for word in word_tokens:
    if word.isalpha() and word not in my_stop_words:
        tokens.append(word)
token_dist = FreqDist(tokens)
dist = pd.DataFrame(token_dist.most_common(20),columns=['Word', 'Frequency'

from nltk.stem import PorterStemmer
porter = PorterStemmer()
stemmed_tokens =[porter.stem(word) for word in tokens]
stemmed_token_dist = FreqDist(stemmed_tokens)
stemmed_dist = pd.DataFrame(stemmed_token_dist.most_common(20),columns=['Wo

from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(stop_words=my_stop_words, ngram_range=(2,2))
bigrams = vect.fit_transform(data_to_use)
bigram_df = pd.DataFrame(bigrams.toarray(), columns=vect.get_feature_names(
bigram_frequency = pd.DataFrame(bigram_df.sum(axis=0)).reset_index()
bigram_frequency.columns = ['bigram', 'frequency']
bigram_frequency = bigram_frequency.sort_values(by='frequency', ascending=F

# Load SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# Instantiate new SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
# Generate sentiment scores
sentiment_scores = data_to_use.apply(sid.polarity_scores)
#review_2015_NZ = review_2015_NZ.set_index('date')
review_user_2015.loc[:,'sentiment_score'] = sentiment_scores.apply(lambda x
#monthly_sentiment = sentiment.resample('M').mean()
```

In [ ]:
```
#Create New Dataframe review_user_2015_sent, with sentiment scores
review_user_2015_sent = review_user_2015.copy()

#Add rev_id for each review, and save DataFrame XX
review_user_2015_sent.reset_index(drop=True,inplace=True)
review_user_2015_sent['rev_id'] = review_user_2015_sent.index
#review_user_2015_sent.to_csv('review_user_2015_sent.csv',index=False)

#Also save rev_id and sentiment score only DataFrame
review_user_2015_sent[['rev_id','sentiment_score']].to_csv('review_sentsco
```

## 4.7 Rev_Users AVG

In [ ]:
```
# Further Fixes on Review_User File

#1 Add How Many month that users have been yelping
review_user_2015_sent['yelping_since'] = pd.to_datetime(review_user_2015_se
review_user_2015_sent['date'] = pd.to_datetime(review_user_2015_sent['date'

review_user_2015_sent['yelping_month'] = review_user_2015_sent['date'].dt.t
    review_user_2015_sent['yelping_since'].dt.to_period('M').astype(int)

#2 Add Whether Users is Elite or not in 2015
review_user_2015_sent['elite'] = review_user_2015_sent['elite'].astype(str)

for i in range(len(review_user_2015_sent)):
    if '2015' in review_user_2015_sent.loc[i,'elite']:
        review_user_2015_sent.loc[i,'is_elite_2015'] = 1
    else:
        review_user_2015_sent.loc[i,'is_elite_2015'] = 0

#3 Replace Naming for Genders
review_user_2015_sent.replace({'female':'F','male':'M','mostly_female':'F',

#4 Add True Star, which is difference of star on review and avg. star of th
review_user_2015_sent['true_star'] = review_user_2015_sent['stars'] - revie


#5 Get Mean Values for each business in each features, with avg. user value
rev_us_2015_avg = review_user_2015_sent.groupby('business_id').agg({'text'
'cool':'mean',
'useful':'mean',
'stars':'mean',
'funny':'mean',
'review_count':'mean',
'friends_count':'mean',
'fans':'mean',
'true_star':'mean',
'avg_stars':'mean',
'sentiment_score':'mean',
'is_elite_2015':'mean',
'yelping_month':'mean',
'u_review_count_2015':'mean'})
rev_us_2015_avg.reset_index(inplace=True)
```

```
rev_us_2015_avg.rename(columns={'text':'rev_count',
'cool':'r_avg_cool',
'useful':'r_avg_useful',
'stars':'r_avg_stars',
'funny':'r_avg_funny',
'review_count':'u_avg_rev_count',
'friends_count':'u_avg_friend',
'fans':'u_avg_fans',
'true_star':'u_avg_true_star',
'avg_stars':'u_avg_avg_star',
'sentiment_score':'avg_sentiment_score',
'is_elite_2015':'u_avg_is_elite_2015',
'yelping_month':'u_avg_yelping_month'},inplace=True)

#6 Get Gender Count And Ratio
rev_us_gend_count_m = pd.DataFrame(review_user_2015_sent.groupby('business_
rev_us_gend_count_f = pd.DataFrame(review_user_2015_sent.groupby('business_
rev_us_gend_count_m.reset_index(inplace=True)
rev_us_gend_count_f.reset_index(inplace=True)
rev_us_gend_count_m.rename(columns={'gender':'M_count'},inplace=True)
rev_us_gend_count_f.rename(columns={'gender':'F_count'},inplace=True)
rev_us_gend_count = pd.merge(rev_us_gend_count_f,rev_us_gend_count_m,on='bu
rev_us_gend_count['M/F Ratio'] = rev_us_gend_count['M_count'] / rev_us_genc
rev_us_gend_count['M/F Ratio'] = rev_us_gend_count['M/F Ratio'].astype(floa
rev_us_gend_count = rev_us_gend_count.replace({np.inf:'0'})

#7 Merge With Rev-User File
rev_us_2015_avg = pd.merge(rev_us_2015_avg,rev_us_gend_count,on='business_i
rev_us_2015_avg.drop(columns=['F_count','M_count'],inplace=True)

#8 Add Check Frequencies
rev_us_2015_avg = pd.merge(rev_us_2015_avg,ch_freq_2015,on='business_id')
rev_us_2015_avg['M/F Ratio'] = rev_us_2015_avg['M/F Ratio'].astype(float)

#Save Avg File as review_user_2015_avg.csv
rev_us_2015_avg.to_csv('review_user_2015_avg.csv',index=False)

rev_us_2015_avg
```

Out[ ]:

| | business_id | rev_count | r_avg_cool | r_avg_useful | r_avg_stars | r_avg_ |
|---|---|---|---|---|---|---|
| **0** | -1VaIJza42Hjev6ukacCNg | 39 | 0.102564 | 1.410256 | 4.051282 | 0.2 |
| **1** | -4TMQnQJW1yd6NqGRDvAeA | 70 | 0.728571 | 1.314286 | 3.942857 | 0.7 |
| **2** | -9eNGMp8XiygI8t8QFuFWw | 34 | 0.147059 | 0.735294 | 3.294118 | 0.1 |
| **3** | -9nai28tnoylwViuJVrYEQ | 91 | 0.879121 | 1.428571 | 3.813187 | 0.6 |
| **4** | -Bdw-5H5C4AYSMGnAvmnzw | 112 | 0.517857 | 1.446429 | 4.071429 | 0.6 |
| **...** | ... | ... | ... | ... | ... | |
| **1785** | zr93wrNyXzc-HW4IcK4iRQ | 22 | 0.909091 | 1.818182 | 4.454545 | 0.9 |
| **1786** | zrDi4gEaUi64lAMfJU51dw | 156 | 1.141026 | 1.884615 | 4.076923 | 0.9 |
| **1787** | zrMGL9riavdJR4olXaACOQ | 4 | 0.750000 | 0.250000 | 3.250000 | 0.2 |
| **1788** | ztP466jMUMtqLwwHqXbk9w | 73 | 0.095890 | 0.356164 | 4.246575 | 0.0 |
| **1789** | zwNLJ2VglfEvGu7DDZjJ4g | 30 | 0.033333 | 0.233333 | 3.100000 | 0.1 |

1790 rows × 18 columns

## 4.9 Daily Review Numbers

In [ ]:

```
review_daily = review_user_2015_sent[['date','business_id']]
review_daily.date = review_daily.apply(lambda x: datetime.date(x['date']),a
review_daily = review_daily.groupby(['business_id','date']).size().unstack(
#Save Review_Daily as review_daily.csv
review_daily.to_csv('review_daily.csv',index=False)

review_daily.head()
```

Out[ ]:

| date | business_id | 2015-01-01 | 2015-01-02 | 2015-01-03 | 2015-01-04 | 2015-01-05 | 2015-01-06 | 2015-01-07 | 2015 01 0 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -1VaIJza42Hjev6ukacCNg | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 1. |
| **1** | -4TMQnQJW1yd6NqGRDvAeA | NaN | NaN | 1.0 | NaN | 1.0 | NaN | NaN | NaI |
| **2** | -9eNGMp8XiygI8t8QFuFWw | NaN | NaN | 1.0 | NaN | NaN | NaN | NaN | NaI |
| **3** | -9nai28tnoylwViuJVrYEQ | NaN | NaN | NaN | 1.0 | 1.0 | NaN | NaN | 3. |
| **4** | -Bdw-5H5C4AYSMGnAvmnzw | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaI |

5 rows × 366 columns

# 5 Tips

In [ ]:
```
tip = pd.read_csv('tip_az_open2.csv') #Previous one doesn't have dates
tip.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
tip
```

Out[ ]:

| | business_id | date | tip | compliment_count | stars |
|---|---|---|---|---|---|
| **0** | 5KheTjYPu1HcQzQFtm4_vw | 2011-12-26 | Good chips and salsa. Loud at times. Good serv... | 0 | 3.5 |
| **1** | 5KheTjYPu1HcQzQFtm4_vw | 2010-09-26 | Queso Fundido is the best. | 0 | 3.5 |
| **2** | 5KheTjYPu1HcQzQFtm4_vw | 2013-12-13 | FYI the green chilis are SPICY!!! I don't mind... | 0 | 3.5 |
| **3** | 5KheTjYPu1HcQzQFtm4_vw | 2012-03-29 | This therapist is cheap. 99cent margaritas eve... | 0 | 3.5 |
| **4** | 5KheTjYPu1HcQzQFtm4_vw | 2012-08-09 | Really great refried beans. | 0 | 3.5 |
| **...** | ... | ... | ... | ... | ... |
| **182102** | 471fUA9qM7oQtVmGdf_Y9g | 2018-01-03 | Great location, seating for 16 and delivery as... | 0 | 2.5 |
| **182103** | X1r8ugVdi4r6siqkNmIkkg | 2018-05-09 | Best | 0 | 5.0 |
| **182104** | AQUa3o3wbv9mt8h9dCRUUQ | 2018-09-15 | Delicious and great for fighting local allergies! | 0 | 5.0 |
| **182105** | J2yHV9KNwBHD7bnGMANCpw | 2016-08-23 | Our family went here and really enjoyed our me... | 0 | 4.5 |
| **182106** | gm8-9LPN5nqcxKQrQdZYNQ | 2015-06-11 | This is a great show. Good for all ages. | 0 | 5.0 |

182107 rows × 5 columns

In [ ]:
```
#Filter it for 2015 NZ, and preview. %43 Decrease

start_date = pd.to_datetime('01/01/2015')
end_date = pd.to_datetime('01/01/2016')
tip['date'] = pd.to_datetime(review['date']).dt.normalize()
tip_2015 = tip.loc[(tip['date'] >= start_date) & (tip['date'] < end_date)]
tip_2015.reset_index(drop=True,inplace=True)
tip_2015_NZ = tip_2015.merge(b_id_nz_4_df, on='business_id', how='inner')

tip_2015_NZ
```

Out[ ]:

| | business_id | date | tip | compliment_count | stars | b_nur |
|---|---|---|---|---|---|---|
| **0** | JwXNyXdJmk_YnXokLhpI8w | 2015-01-01 | Half expected Hawaiian shirts in here :p | 0 | 3.0 | 76 |
| **1** | JwXNyXdJmk_YnXokLhpI8w | 2015-05-24 | The katsu is our favorite! | 0 | 3.0 | 76 |
| **2** | JwXNyXdJmk_YnXokLhpI8w | 2015-11-12 | Fire chicken is great! | 0 | 3.0 | 76 |
| **3** | sQ6DZZLLJRfwUr0eDWKDDg | 2015-01-01 | Yum | 0 | 4.0 | 198 |
| **4** | sQ6DZZLLJRfwUr0eDWKDDg | 2015-02-16 | Kung Pow Chicken & Orange Chicken\nTasty!!! | 0 | 4.0 | 198 |
| **...** | ... | ... | ... | ... | ... | . |
| **14626** | b3zsV44_SBz5JLwyfiHsRA | 2015-12-09 | Meeting good friends here | 0 | 3.5 | 138 |
| **14627** | p8H_WsH1SltPmo0-c67gFQ | 2015-12-20 | Get the Haystack Onion Rings! | 0 | 3.0 | 187 |
| **14628** | TKp9PDs8jFbshfg6xOcM6A | 2015-12-21 | bloomin onion! | 0 | 3.0 | 107 |
| **14629** | 0pazeZax1Nt9cn0U_8aSUQ | 2015-12-29 | Great authentic Thai food. Pad Thai chicken... | 0 | 3.5 | 8 |
| **14630** | lUcSe9mjppbAa8510xj0Tg | 2015-12-30 | Try the Chic Fil A Sauce! | 0 | 3.5 | 172 |

14631 rows × 6 columns

## 5.1 Sentiment Analysis on Tips

In [ ]:

```
# Cleaning
tip_2015_NZ['tip'] = [i.replace("&amp;amp;", '').replace("\'",'') for i in
from nltk.corpus import stopwords
from sklearn.feature_extraction import _stop_words
#from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
my_stop_words = set(stopwords.words('english') + list(_stop_words.ENGLISH_S

from wordcloud import WordCloud
# concatenate all the reviews into one single string
full_text = ' '.join(tip_2015_NZ.loc[:,'tip'])
cloud_no_stopword = WordCloud(background_color='white', stopwords=my_stop_w
```

In [ ]:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk import FreqDist
lower_full_text = full_text.lower()
word_tokens = word_tokenize(lower_full_text)
tokens = list()
for word in word_tokens:
    if word.isalpha() and word not in my_stop_words:
        tokens.append(word)
token_dist = FreqDist(tokens)
dist = pd.DataFrame(token_dist.most_common(20),columns=['Word', 'Frequency'

from nltk.stem import PorterStemmer
porter = PorterStemmer()
stemmed_tokens =[porter.stem(word) for word in tokens]
stemmed_token_dist = FreqDist(stemmed_tokens)
stemmed_dist = pd.DataFrame(stemmed_token_dist.most_common(20),columns=['Wo

from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(stop_words=my_stop_words, ngram_range=(2,2))
bigrams = vect.fit_transform(tip_2015_NZ.loc[:,'tip'])
bigram_df = pd.DataFrame(bigrams.toarray(), columns=vect.get_feature_names(
bigram_frequency = pd.DataFrame(bigram_df.sum(axis=0)).reset_index()
bigram_frequency.columns = ['bigram', 'frequency']
bigram_frequency = bigram_frequency.sort_values(by='frequency', ascending=F
"""
from nltk.tokenize import sent_tokenize
df_good = review_2015_NZ.loc[:5000][review_2015_NZ['stars'] >= 4]
good_reviews = ' '.join(df_good.text)
# split the long string into sentences
sentences_good = sent_tokenize(good_reviews)
good_token_clean = list()
# get tokens for each sentence
import re
for sentence in sentences_good:
    eng_word = re.findall(r'[A-Za-z\-]+', sentence)
    good_token_clean.append([i.lower() for i in eng_word if i.lower() not i

from gensim.models import Word2Vec
model_ted = Word2Vec(sentences=good_token_clean, size=500, window=10, min_c
model_ted.predict_output_word(['good'], topn=10)
"""
# Load SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# Instantiate new SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
# Generate sentiment scores
sentiment_scores = tip_2015_NZ.loc[:,'tip'].apply(sid.polarity_scores)
#review_2015_NZ = review_2015_NZ.set_index('date')
tip_2015_NZ.loc[:,'sentiment_score'] = sentiment_scores.apply(lambda x: x['
#monthly_sentiment = sentiment.resample('M').mean()
```

```
[nltk_data] Downloading package punkt to /Users/nuritas/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/nuritas/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [ ]:

```
#Add tip_id for each review, and save DataFrame
tip_2015_NZ['tip_id'] = tip_2015_NZ.index
tip_2015_NZ.to_csv('tip_2015_NZ.csv',index=False)

#Also save tip_id and sentiment score only DataFrame
tip_2015_NZ[['tip_id','sentiment_score']].to_csv('tip_sentscore_tipid.csv',
```

## 5.2 Daily Tip Numbers

In [ ]:

```
tip_daily = tip_2015_NZ[['date','business_id']]
tip_daily.date = tip_daily.apply(lambda x: datetime.date(x['date']),axis=1)
tip_daily = tip_daily.groupby(['business_id','date']).size().unstack().rese
#Save Review_Daily as review_daily.csv
tip_daily.to_csv('tip_daily.csv',index=False)

tip_daily.head()
```

Out[ ]:

| date | business_id | 2015-01-01 | 2015-01-02 | 2015-01-03 | 2015-01-04 | 2015-01-05 | 2015-01-06 | 2015-01-07 | 2015 01 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -4TMQnQJW1yd6NqGRDvAeA | NaN | 1.0 | NaN | NaN | NaN | NaN | NaN | NaI |
| 1 | -9eNGMp8XiygI8t8QFuFWw | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaI |
| 2 | -9nai28tnoylwViuJVrYEQ | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaI |
| 3 | -Bdw-5H5C4AYSMGnAvmnzw | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaI |
| 4 | -Dnh48f029YNugtMKkkI-Q | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaI |

5 rows × 366 columns

# B - Descriptive Analysis

## Read Files

In [ ]:

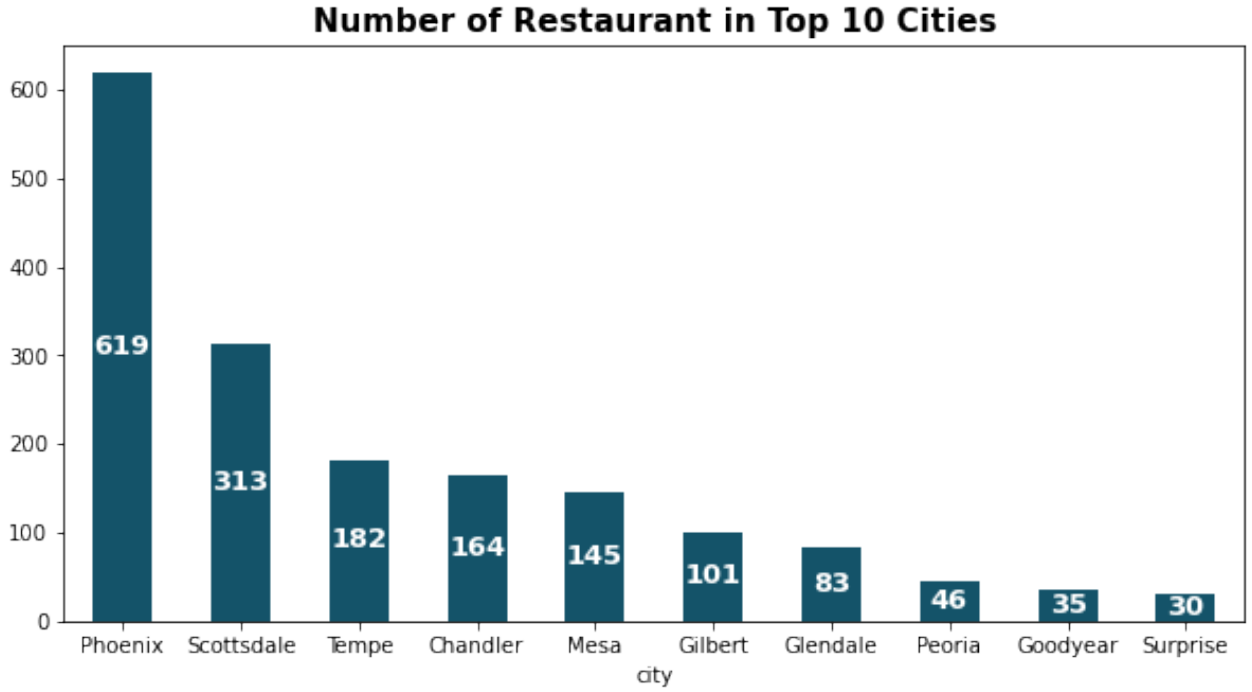
```
bus = pd.read_csv('bus_3_2015.csv')
bus.drop(columns='Unnamed: 0',inplace=True, errors='ignore')

b_id_NZ_df_4 = pd.read_csv('b_id_NZ_df_4.csv')
b_id_NZ_df_4.drop(columns='Unnamed: 0',inplace=True, errors='ignore')

#Even if b_id_NZ_df_5 will be created after the weather data drop,
```

```
#it will be used in Descriptive Analysis
b_id_NZ_df_5 = pd.read_csv('b_id_NZ_df_5.csv')
b_id_NZ_df_5.drop(columns='Unnamed: 0',inplace=True, errors='ignore')

bus = bus.merge(b_id_NZ_df_5, on=['business_id','b_num'], how='inner')
bus.reset_index(inplace=True,drop=True)
bus.columns = bus.columns.map(lambda x: x.removeprefix("attributes."))
bus.columns = bus.columns.map(lambda x: x.removeprefix("GoodForMeal"))

#check
check_dates = pd.read_csv('check_5_dates.csv')
check_dates.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
check_dates.date = pd.to_datetime(check_dates.date)
check_dates = check_dates.loc[check_dates.date.dt.year == 2015]

#check_t_2015 = pd.read_csv('check_t_2015.csv')
#check_t_2015.drop(columns='Unnamed: 0',inplace=True, errors='ignore')

check_weekly = pd.read_csv('check_2_2015_Weekly.csv')
check_weekly.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
check_weekly = check_weekly.merge(b_id_NZ_df_4, on='business_id', how='inne

check_monthly = pd.read_csv('check_3_2015_Monthly.csv')
check_monthly.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
check_monthly = check_monthly.merge(b_id_NZ_df_4, on='business_id', how='ir

#photo
photo = pd.read_csv('photo_2015_NZ.csv')
photo.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
photo = photo.merge(b_id_NZ_df_4, on=['business_id','b_num'], how='inner')
photo.reset_index(drop=True, inplace=True)

photo_comp = pd.read_csv('photo_comp_2015_NZ.csv')
photo_comp.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
photo_comp = photo_comp.merge(b_id_NZ_df_4, on=['business_id','b_num'], how
photo_comp.reset_index(drop=True, inplace=True)

#review-user
review_user = pd.read_csv('review_user_2015_sent.csv')
review_user.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
review_user = review_user.merge(b_id_NZ_df_4, on=['business_id','b_num'], h
review_user.reset_index(drop=True, inplace=True)

review_avg = pd.read_csv('review_user_2015_avg.csv')
review_avg.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
review_avg = review_avg.merge(b_id_NZ_df_4, on=['business_id','b_num'], how
review_avg.reset_index(drop=True, inplace=True)

#tips
tips = pd.read_csv('tip_2015_NZ.csv')
tips.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
tips = tips.merge(b_id_NZ_df_4, on=['business_id','b_num'], how='inner')
tips.reset_index(drop=True, inplace=True)

ch_freq = pd.read_csv('check_7_freq_2015.csv')
ch_freq.drop(columns='Unnamed: 0',inplace=True, errors='ignore')
```

## Number of Businesses in Each City

In [ ]:
```
for i in range(len(bus)):
    bus.loc[i,'city'] = bus.loc[i,'city'].title()
bus.city.replace({'Scottdale':'Scottsdale'},inplace=True)

ax0 = bus.groupby('city').size().sort_values(ascending=False)[:10].plot(kir
for container in ax0.containers:
    ax0.bar_label(container, label_type='center', fontsize=13,color='white'
ax0.set_title('Number of Restaurant in Top 10 Cities', fontsize=15, fontwei
#plt.savefig('Number of Restaurant in Top 10 Cities.png',dpi=300,bbox_inche
```

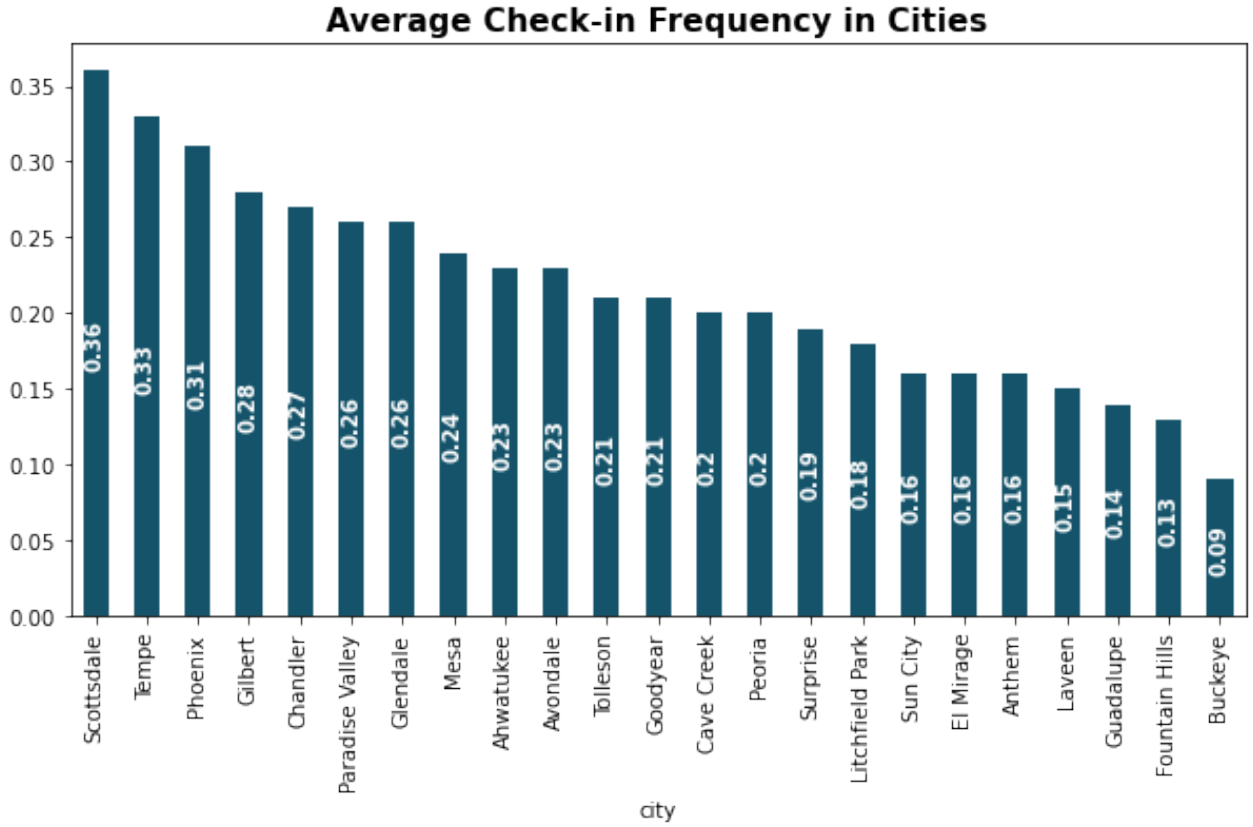
Out[ ]:
```
Text(0.5, 1.0, 'Number of Restaurant in Top 10 Cities')
```




## Ch_freq for each Cities

In [ ]:
```
ax_c_c = round(bus.groupby('city')['ch_freq'].mean().sort_values(ascending=
for container in ax_c_c.containers:
    ax_c_c.bar_label(container, label_type='center', fontsize=10,color='whi
ax_c_c.set_title('Average Check-in Frequency in Cities', fontsize=15, fontw
#plt.savefig('Average Check-in Frequency in Cities.png',dpi=300,bbox_inches
```

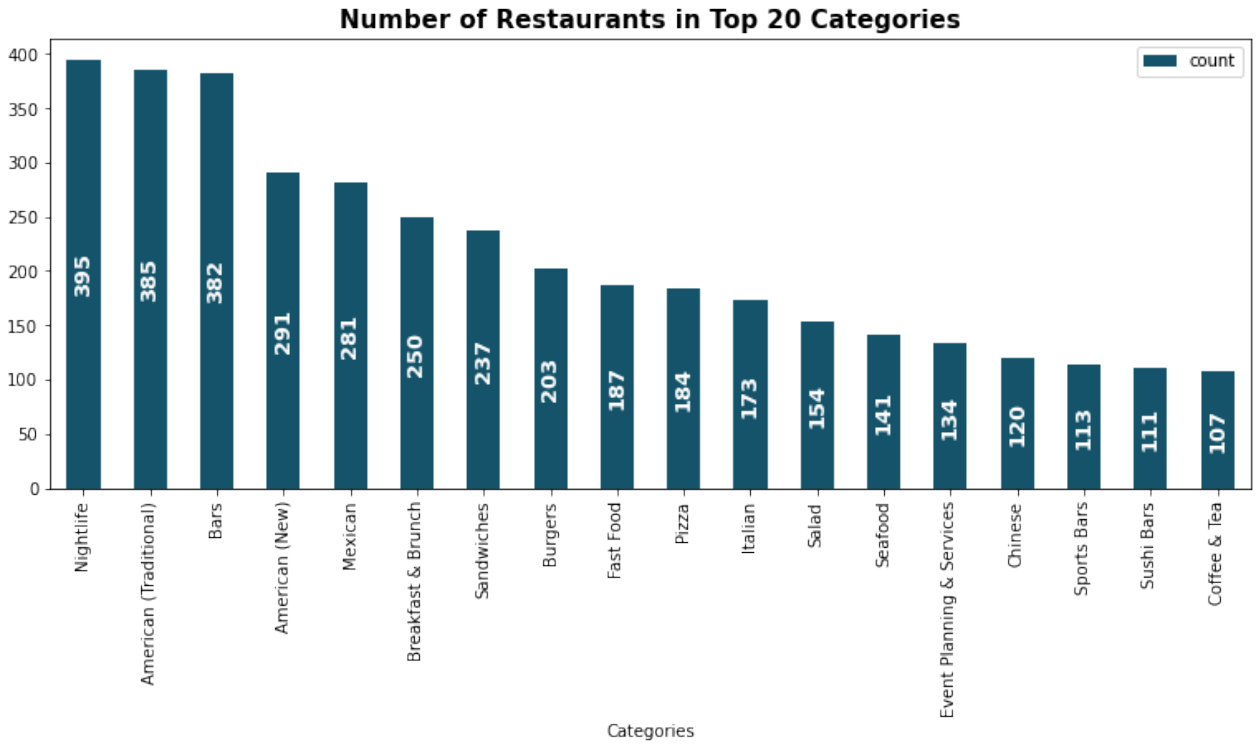
Out[ ]:
```
Text(0.5, 1.0, 'Average Check-in Frequency in Cities')
```




## Business Categories Plot

In [ ]:
```
#Business Categories Plot
list_string = bus['categories'].str.cat(sep=',')
list_split = list_string.split(',') #211050
list_split_all = [x.strip(' ') for x in list_split] # Take Spaces
count = Counter(list_split_all)
unique_list = list(count.keys())
most_frequent = count.most_common(20) #Change i to get i different categori
bus_cat_unique = pd.DataFrame(most_frequent,columns=['Categories','count'])
bus_cat_unique = bus_cat_unique[bus_cat_unique['Categories'] != 'Restaurant
bus_cat_unique.reset_index(inplace=True,drop='index')
bus_cat_unique.sort_values('count',inplace=True,ascending=False)
bus_cat_unique.drop(index=bus_cat_unique.index[0],
        axis=0,
        inplace=True)
bus_cat_unique.reset_index(inplace=True,drop=True)

#Plot Categories
ax0 = bus_cat_unique.plot(kind='bar', y='count',x='Categories',figsize=(13,
for container in ax0.containers:
        ax0.bar_label(container, label_type='center',rotation=90,color='whi
ax0.set_title('Number of Restaurants in Top 20 Categories', fontsize=15, fo
#plt.savefig('Number of Restaurants in Top 20 Categories.png',dpi=300,bbox_
```

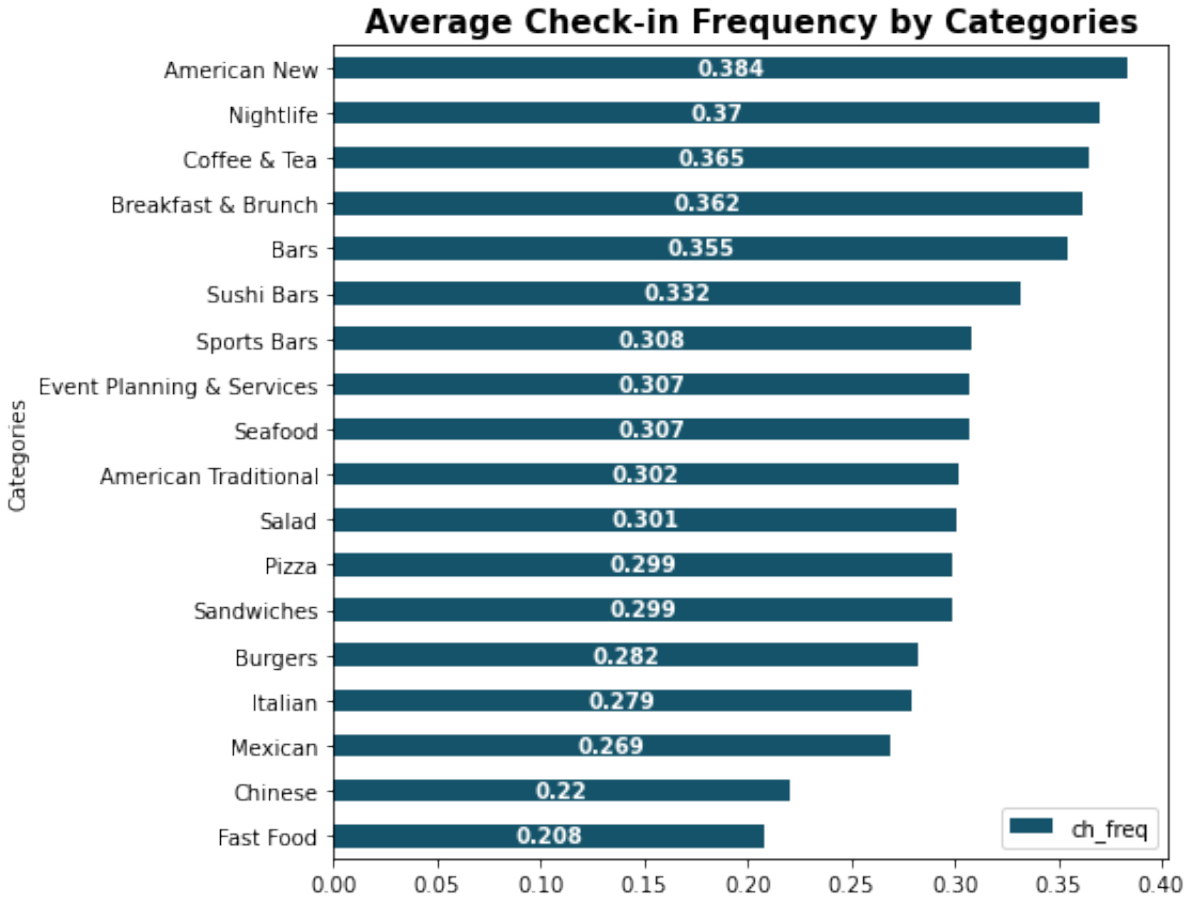
Out[ ]: Text(0.5, 1.0, 'Number of Restaurants in Top 20 Categories')




In [ ]:
```
bus.categories = bus.categories.str.replace(r"(","")
bus.categories = bus.categories.str.replace(r")","")
bus_cat_unique.Categories = bus_cat_unique.Categories.str.replace(r"(","")
bus_cat_unique.Categories = bus_cat_unique.Categories.str.replace(r")","")
cat_ch_df = pd.DataFrame(columns=['Categories','ch_freq'])
for i in range(len(bus_cat_unique)):
    cat_ch_df.loc[len(cat_ch_df),:] = [bus_cat_unique.iloc[i,0],bus[bus['ca
cat_ch_df.ch_freq = cat_ch_df.ch_freq.astype(float)
cat_ch_df.ch_freq = round(cat_ch_df.ch_freq,3)
```

## Check-in Freq by Categories

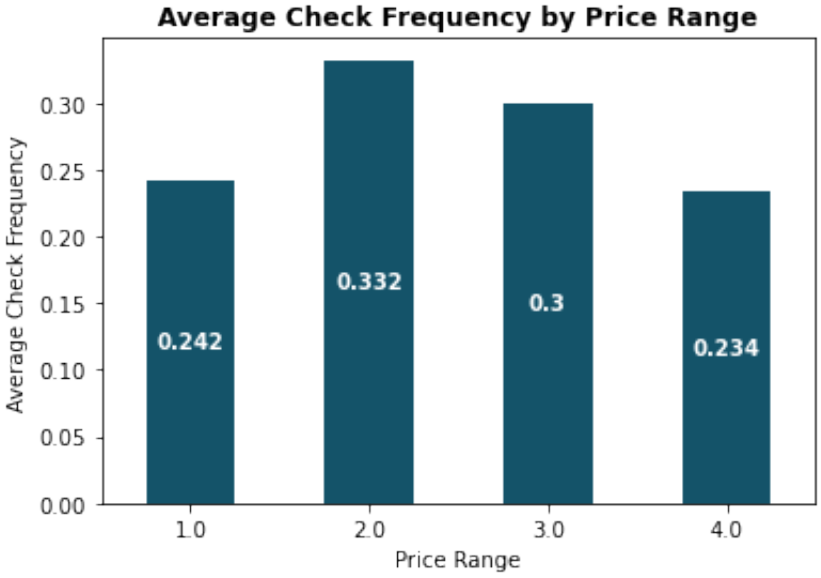
In [ ]:
```
ax_cat = cat_ch_df.sort_values('ch_freq',ascending=True).plot(kind='barh',y
for container in ax_cat.containers:
        ax_cat.bar_label(container, label_type='center',color='white',weigh
ax_cat.set_title('Average Check-in Frequency by Categories', fontsize=15, f
#plt.savefig('Average Check-in Frequency by Categories.png',dpi=300,bbox_in
```




## Check-in by Price Range

In [ ]:

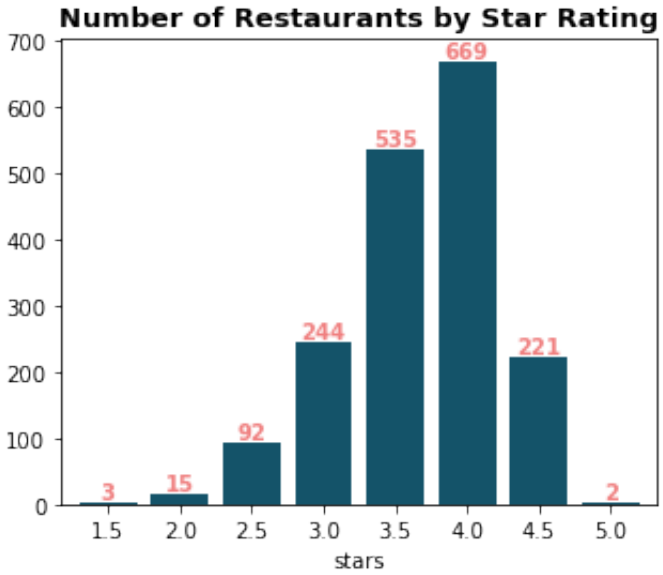
```
ax_c_pr = round(bus.groupby('RestaurantsPriceRange2')['ch_freq'].mean(),3)
for container in ax_c_pr.containers:
    ax_c_pr.bar_label(container, label_type='center',color='white',weight='
ax_c_pr.set_xlabel('Price Range')
ax_c_pr.set_ylabel('Average Check Frequency')
ax_c_pr.set_title('Average Check Frequency by Price Range',weight='bold')
plt.savefig('Average Check Frequency by Price Range.png',dpi=300,bbox_inche
```



## Stars Plot

In [ ]:

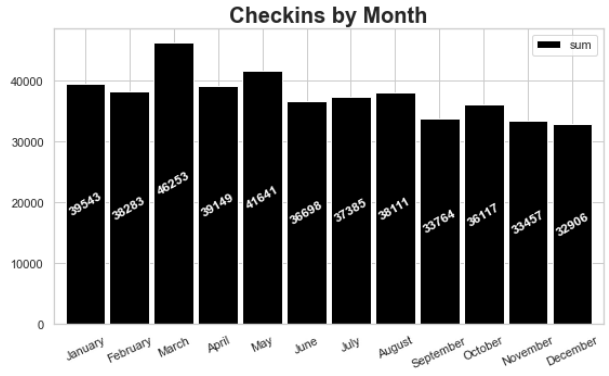
```
ax1 = bus.groupby('stars').size().plot(kind='bar',figsize=(5,4),color='#145
for container in ax1.containers:
        ax1.bar_label(container, label_type='edge',color='lightcoral',weigh
ax1.set_title('Number of Restaurants by Star Rating', fontsize=13, fontweig
#plt.savefig('Number of Restaurants by Star Rating.png',dpi=300,bbox_inches
```

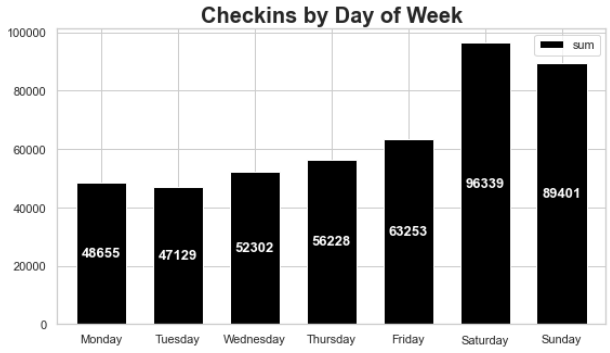



## Check-ins by Day & Month

In [ ]:

```
fig, axs = plt.subplots(1,2, figsize=(20,5))
ch_byweekday = check_dates.groupby('WeekDay').sum()
ch_byweekday.reset_index(inplace=True)
#Create days of week as a list
days = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sund

#Make ch_byweekday a single level column
ch_byweekday.columns = ['WeekDay','sum']

ch_byweekday.set_index('WeekDay',inplace=True)
ch_byweekday['WeekDay'] = days
ch_byweekday = ch_byweekday.reindex(days)
ch_byweekday.drop('WeekDay',axis=1,inplace=True)
ch_byweekday.reset_index(inplace=True)

ax2 = ch_byweekday.plot(kind='bar',color='black',x='WeekDay',y='sum',rot=0,
for container in ax2.containers:
        ax2.bar_label(container, label_type='center',rotation=0,color='whit
ax2.set_title('Checkins by Day of Week', fontsize=20, fontweight='bold')
ax2.set(xlabel='')
#Create a list of the months
months = ['January','February','March','April','May','June','July','August'
ch_bymonth = check_dates.groupby(pd.Grouper(key='date', freq='M')).sum().re
ch_bymonth['date'] = months
ch_bymonth.rename(columns={'date':'Month','count':'sum'},inplace=True)
ax3 = ch_bymonth.plot(kind='bar',color='black',x='Month',y='sum',rot=25,wid
for container in ax3.containers:
        ax3.bar_label(container, label_type='center',rotation=30,color='whi
ax3.set_title('Checkins by Month', fontsize=20, fontweight='bold')
ax3.set(xlabel='')
plt.savefig('Checkins by Day of Week and Month_black.png',dpi=300,bbox_inch
```




## WordClouds for Tips & Reviews

In [ ]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction import _stop_words
#from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
my_stop_words = set(stopwords.words('english') + list(_stop_words.ENGLISH_S

from wordcloud import WordCloud
# concatenate all the reviews into one single string
full_text = ' '.join(tips.loc[:,'tip'])
cloud_no_stopword = WordCloud(background_color='white', stopwords=my_stop_w


full_text_rew = ' '.join(review_user.loc[:,'text'])
cloud_no_stopword2 = WordCloud(background_color='white', stopwords=my_stop_


fig, ax1 = plt.subplots(1,2, figsize=(20,5))
#Plot cloud_no_stopword and cloud_no_stopword2 in subplots
ax1[0].imshow(cloud_no_stopword, interpolation='bilinear')
ax1[0].axis('off')
ax1[0].set_title('Word Cloud of Tips', fontsize=20, fontweight='bold')
ax1[1].imshow(cloud_no_stopword2, interpolation='bilinear')
ax1[1].axis('off')
ax1[1].set_title('Word Cloud of Reviews', fontsize=20, fontweight='bold')
#plt.savefig('Word Cloud of Tips and Reviews.png',dpi=300,bbox_inches='tigl
```

Out[ ]:
```
Text(0.5, 1.0, 'Word Cloud of Reviews')
```

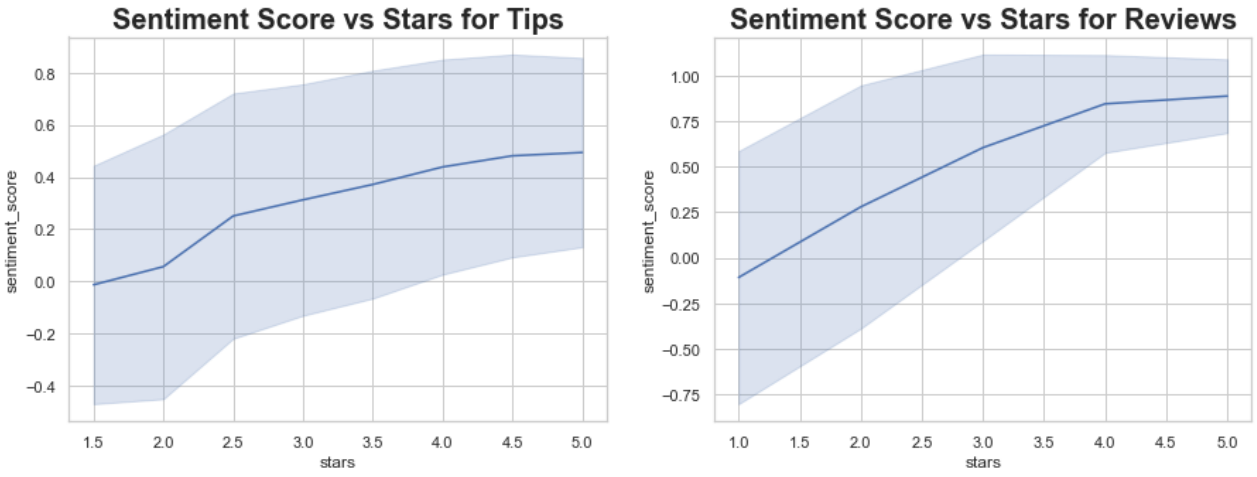

In [ ]:
```
from os import path
from PIL import Image
def grey_color_func(word, font_size, position, orientation, random_state=No
                    **kwargs):
    return "hsl(0, 0%%, %d%%)" % random.randint(60, 100)

d = path.dirname(__file__) if "__file__" in locals() else os.getcwd()
mask = np.array(Image.open(path.join(d, "yelp_logo_cmyk_2.png")))
full_text_2 = full_text + full_text_rew
wc = WordCloud(max_words=1000, mask=mask, stopwords=my_stop_words).generate
# store default colored image
#wc2 = WordCloud(max_words=1000, mask=mask, stopwords=my_stop_words).genera
```

In [ ]:
```
plt.figure(figsize=(10, 10),dpi=300)
plt.imshow(wc.recolor(color_func=grey_color_func, random_state=3),interpola
#wc.to_file("a_new_hope.png")
plt.axis("off")
plt.savefig('Word Cloud of for Tips and Review_black.png',dpi=300,bbox_inch
plt.show()
```


## Sentiment Analysis Graphs for Tips & Reviews

In [ ]:
```
# Create lineplot for sentiment score and stars in tip_2015_NZ with varianc
sns.set(style='whitegrid')
fig, ax3 = plt.subplots(1,2, figsize=(15,5))
ax3_1 = sns.lineplot(x='stars', y='sentiment_score', data=tips ,err_style='
ax3_1.set_title('Sentiment Score vs Stars for Tips', fontsize=20, fontweigh

# Create lineplot for sentiment score and stars in review_user_2015_sent wi
ax3_2 = sns.lineplot(x='stars', y='sentiment_score', data=review_user,err_s
ax3_2.set_title('Sentiment Score vs Stars for Reviews', fontsize=20, fontwe
#plt.savefig('Sentiment Score vs Stars for Tips and Reviews.png',dpi=300,bl
```

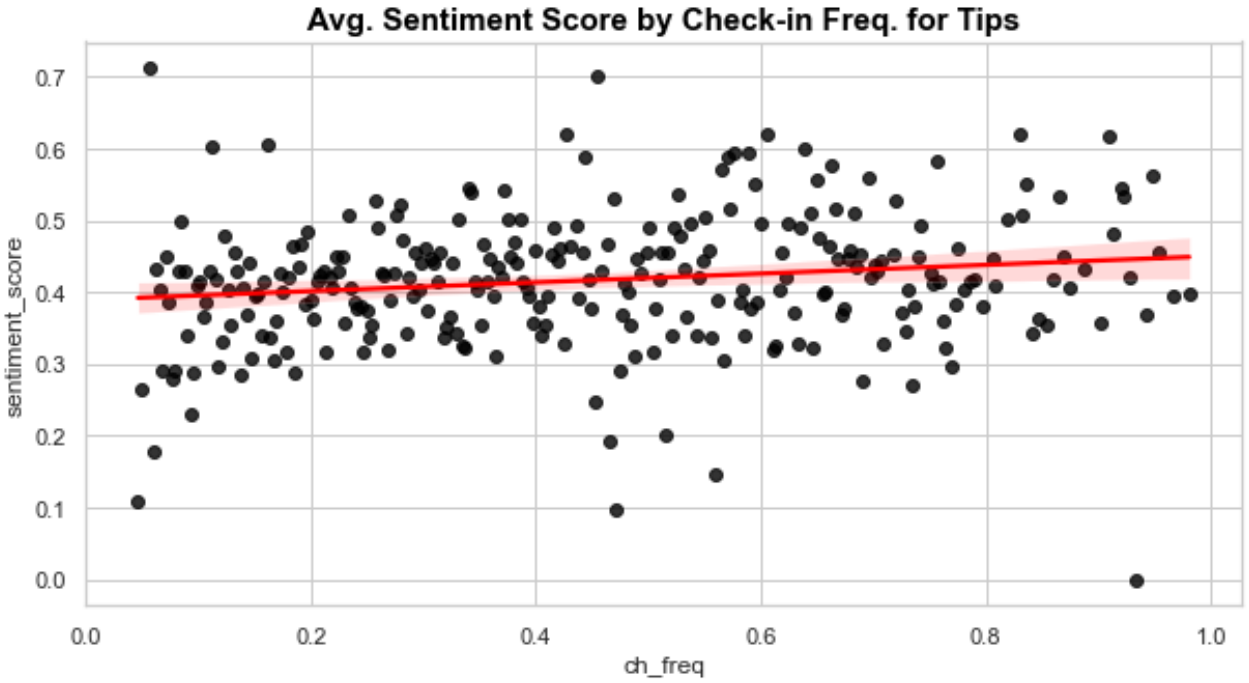
Out[ ]:
```
Text(0.5, 1.0, 'Sentiment Score vs Stars for Reviews')
```

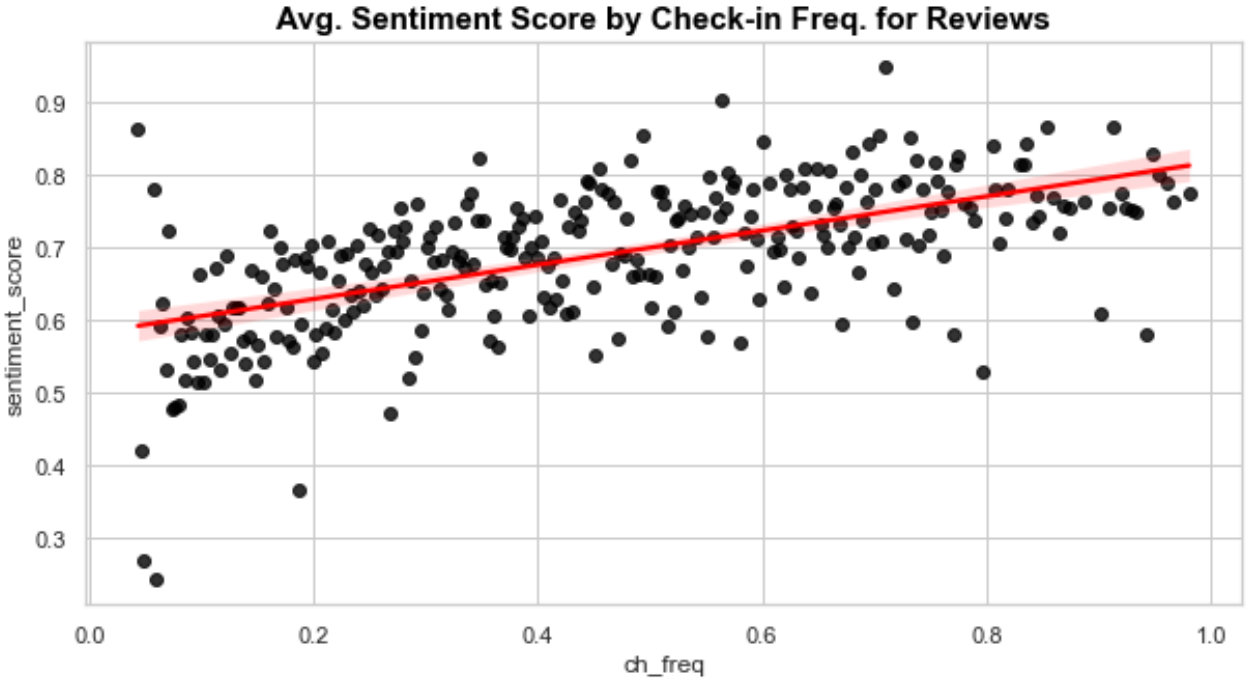



## AVG Sentiment by Check-in

In [ ]:

```
tips2 = tips.merge(ch_freq)
rew2 = review_user.merge(ch_freq)

#rew2.drop('rev_id',axis=1,inplace=True)
#rew2.drop_duplicates(inplace=True)
#rew2.reset_index(inplace=True,drop=True)

rew_2_df = rew2.groupby('ch_freq')['sentiment_score'].mean().reset_index()
tips_2_df = tips2.groupby('ch_freq')['sentiment_score'].mean().reset_index(
# Create lineplot for sentiment score and stars in tip_2015_NZ with varianc
plt.figure(figsize=(10,5))
sns.set(style='whitegrid')
ax3_1 = sns.regplot(x='ch_freq', y='sentiment_score',data=tips_2_df,scatter
ax3_1.set_title('Avg. Sentiment Score by Check-in Freq. for Tips', fontsize
#plt.show()
plt.savefig('Avg. Sentiment Score by Check-in for Tips.png',dpi=300,bbox_in
# Create lineplot for sentiment score and stars in review_user_2015_sent wi
plt.figure(figsize=(10,5))
ax3_2 = sns.regplot(x='ch_freq', y='sentiment_score',data=rew_2_df,scatter_
ax3_2.set_title('Avg. Sentiment Score by Check-in Freq. for Reviews', fonts
#plt.show()
plt.savefig('Avg. Sentiment Score by Check-in for Reviews.png',dpi=300,bbox
```



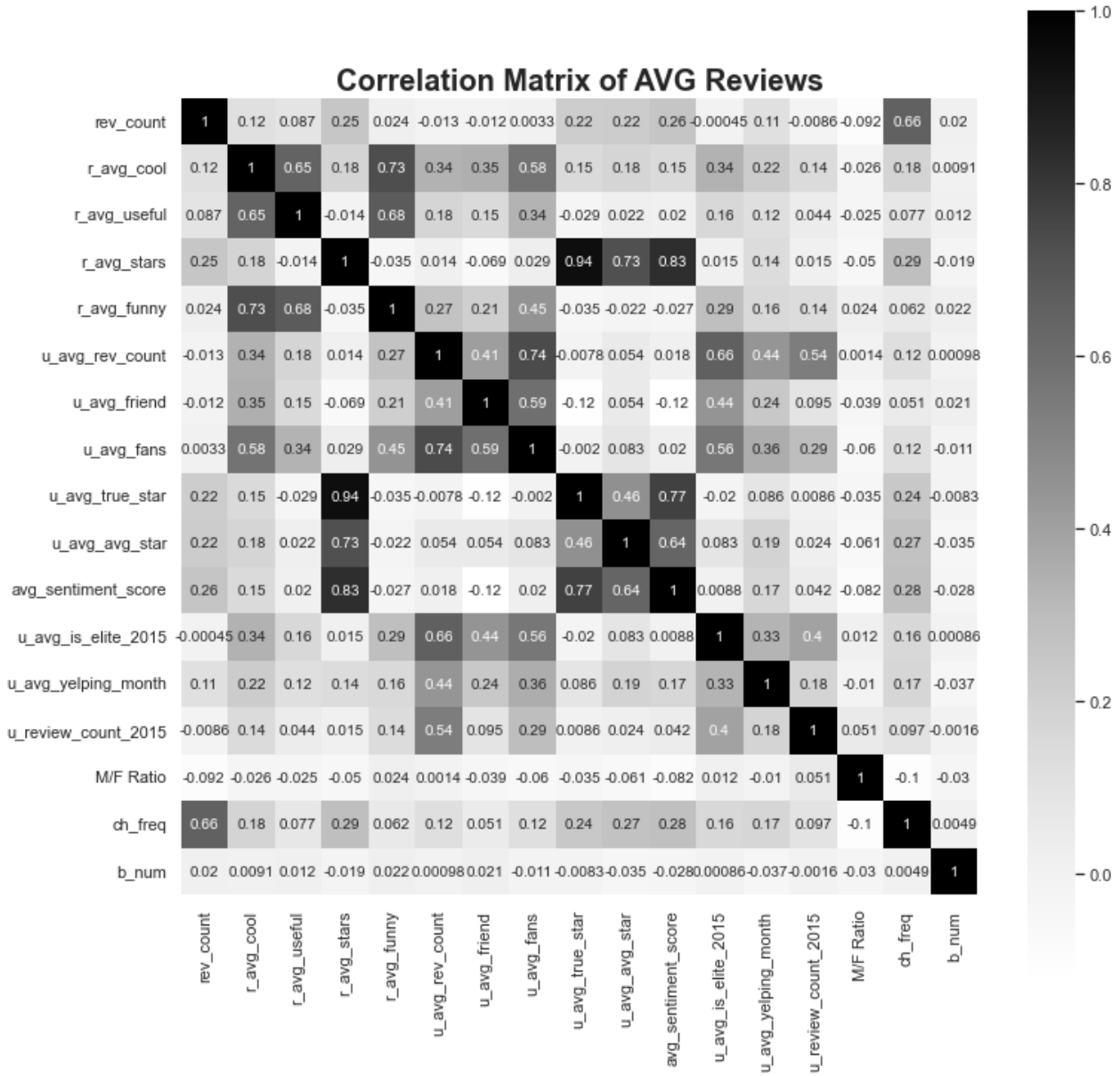




## Rev_AVG

In [ ]:

```
review_avg = review_avg.merge(b_id_NZ_df_5, on=['business_id','b_num'], hov
plt.figure(figsize=(12,12))
ax_avg = sns.heatmap(review_avg.corr(), annot=True, cmap='Greys',square=Tru
ax_avg.set_title('Correlation Matrix of AVG Reviews', fontsize=20, fontweig
#plt.savefig('Correlation Matrix of AVG Reviews.png',dpi=300,bbox_inches='t
```

Out[ ]: Text(0.5, 1.0, 'Correlation Matrix of AVG Reviews')



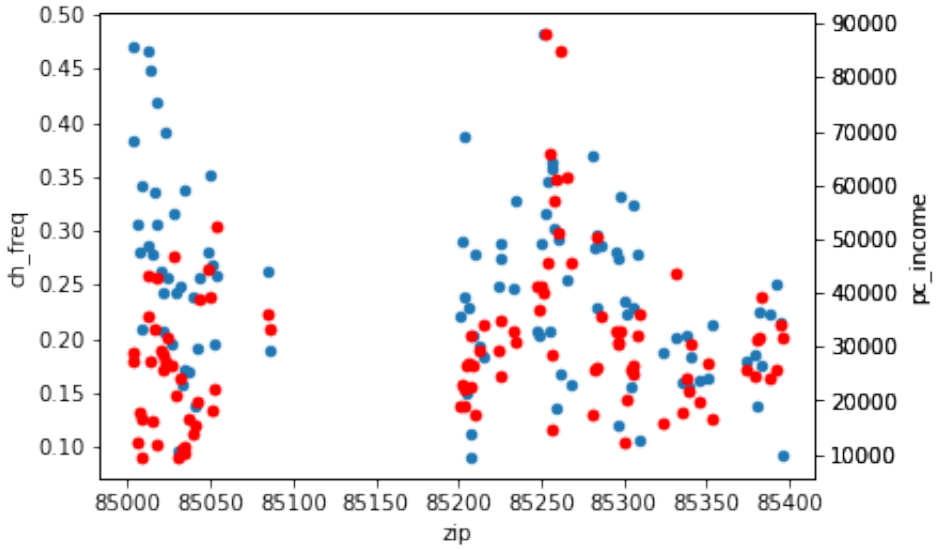

## Rev_Avg to Latex

In [ ]:
```
print(round(review_avg.iloc[:,:-2].describe().T.iloc[:,[0,1,2,3,7]],3).to_l
```

# Income and Check-freq by ZipCode

In [ ]:
```
ax0 = bus[['zip','pc_income','ch_freq']].groupby('zip').mean().reset_index(
ax1 = ax0.twinx()
bus[['zip','pc_income','ch_freq']].groupby('zip').mean().reset_index().plot
y='pc_income',ax=ax1,color='r',)
ax1 = ax0.twinx()
bus[['zip','pc_income','ch_freq']].groupby('zip').mean().reset_index().plot
y='pc_income',ax=ax1,color='r',)
```

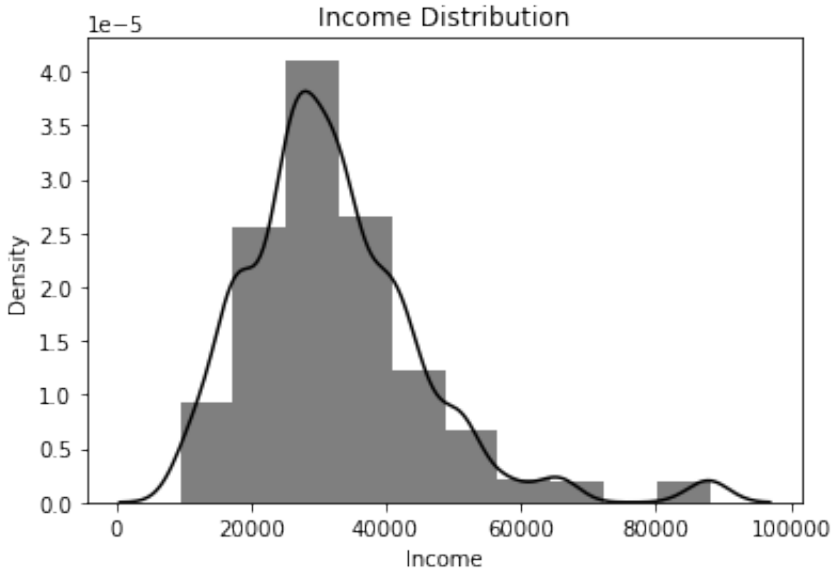
```
<AxesSubplot:xlabel='zip', ylabel='pc_income'>
```




## Income Distribution Plot

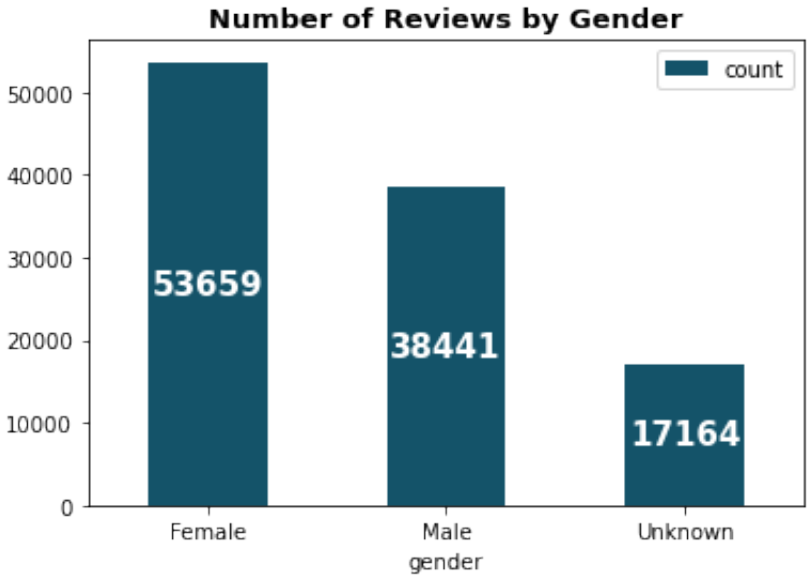
In [ ]:

```
#ax_h = plt.hist(bus.pc_income)
#Add Title and Labels
#Add Density Plot
ax_d = sns.distplot(bus.pc_income, bins=10, kde=True, color='black', hist_k
ax_d.set_xlabel('Income')
ax_d.set_ylabel('Density')
ax_d.set_title('Income Distribution')
plt.savefig('Income_Dist.png',dpi=300,bbox_inches='tight')
```




## Reviews by Gender

In [ ]:
```
review_user.gender.replace(np.NaN,'Unknown',inplace=True)
ax_g = review_user.groupby('gender').size().to_frame().reset_index().rename
for container in ax_g.containers:
    ax_g.bar_label(container,color='white',label_type='center',size=15,weig
ax_g.set_xticklabels(['Female','Male','Unknown'])
ax_g.set_title('Number of Reviews by Gender',size=13,weight='bold')
plt.savefig('Number of Reviews by Gender',dpi=300,bbox_inches='tight')
```



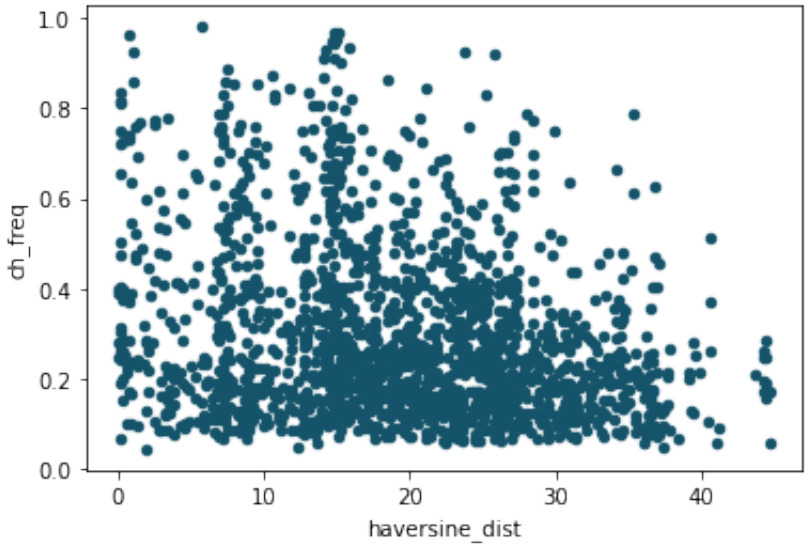

## HeatMap of Resturants Location

In [ ]:
```
locations = bus[['latitude', 'longitude']]
gmaps.configure(api_key='AIzaSyBgSwgEYAEQRn7UC6dy47lDa2bqueXztHU')

figure_layout = {
    'width': '500px',
    'height': '400px',}
figx = gmaps.figure()
figx.add_layer(gmaps.heatmap_layer(locations))
figx
#plt.savefig('Heatmap of Business Locations.png',dpi=300,bbox_inches='tight
```

## Income & Dist & Check Plots

In [ ]:
```
#Dist and Check Freq
bus.plot(kind='scatter',x='haversine_dist',y='ch_freq',color='#145369')
```

Out[ ]: <AxesSubplot:xlabel='haversine_dist', ylabel='ch_freq'>




# Attributes Check

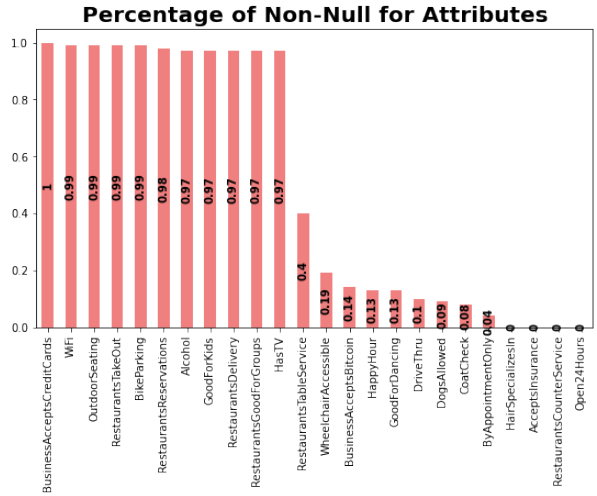
## Null Count and Plot Attributes

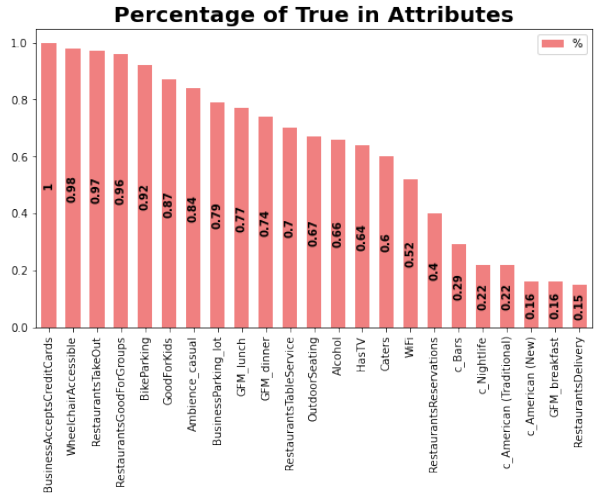
In [ ]:

```
# Attributes List
att_list1 = ['WiFi', 'HasTV',
       'Alcohol', 'GoodForKids', 'OutdoorSeating', 'RestaurantsTakeOut',
       'RestaurantsDelivery',
       'RestaurantsReservations', 'RestaurantsGoodForGroups',
       'BusinessAcceptsCreditCards', 'Caters', 'BikeParking',
       'RestaurantsTableService',  'DriveThru', 'WheelchairAccessible',
       'BusinessAcceptsBitcoin',   'CoatCheck',
       'HappyHour', 'BestNights', 'DogsAllowed', 'GoodForDancing',
        'ByAppointmentOnly', 'DietaryRestrictions',
       'AgesAllowed', 'HairSpecializesIn', 'AcceptsInsurance',
       'RestaurantsCounterService', 'Open24Hours',
        'BusinessParking_garage',
       'BusinessParking_street', 'BusinessParking_validated',
       'BusinessParking_lot', 'BusinessParking_valet', 'Ambience_romantic',
       'Ambience_intimate', 'Ambience_classy', 'Ambience_hipster',
       'Ambience_divey', 'Ambience_touristy', 'Ambience_trendy',
       'Ambience_upscale', 'Ambience_casual', 'GFM_dessert', 'GFM_latenight
       'GFM_lunch', 'GFM_dinner', 'GFM_brunch', 'GFM_breakfast', 'c_Nightli
       'c_American (Traditional)', 'c_Bars', 'c_American (New)', 'c_Mexican
       'c_Breakfast & Brunch', 'c_Sandwiches', 'c_Burgers', 'c_Fast Food',
       'c_Pizza', 'c_Italian', 'c_Salad', 'c_Seafood',
       'c_Event Planning & Services', 'c_Chinese', 'c_Sports Bars',
       'c_Sushi Bars', 'c_Coffee & Tea']

att_list2 = ['WiFi', 'HasTV',
       'Alcohol', 'GoodForKids', 'OutdoorSeating', 'RestaurantsTakeOut',
       'RestaurantsDelivery',
       'RestaurantsReservations', 'RestaurantsGoodForGroups',
       'BusinessAcceptsCreditCards', 'BikeParking',
       'RestaurantsTableService',  'DriveThru', 'WheelchairAccessible',
```

```
        'BusinessAcceptsBitcoin',  'CoatCheck',
        'HappyHour', 'DogsAllowed', 'GoodForDancing',
         'ByAppointmentOnly',
        'HairSpecializesIn', 'AcceptsInsurance',
        'RestaurantsCounterService', 'Open24Hours']

# Null Counter
bus.replace({'1':1, '0':0}, inplace=True)
b_null_count = pd.isnull(bus[att_list2])

b_null_count
# If isnull = False, value is 1, otherwise 0
b_null_count.replace({False: 1, True: 0},inplace=True)
b_null_count = b_null_count.append((b_null_count.sum()/len(b_null_count)).r

fig, axat = plt.subplots(1,2, figsize=(20,5))
ax_b = round(b_null_count.iloc[-1,:],2).sort_values(ascending=False).plot(k
rot=90,color=['lightcoral'],ax=axat[0])
for container in ax_b.containers:
    ax_b.bar_label(container,color='black',label_type='center',weight='bold
ax_b.set_title('Percentage of Non-Null for Attributes', fontsize=20, fontwe


# Number of True

attributes_check = bus[att_list1]
attributes_check.replace({False:0,True:1,'False':0,'True':1},inplace=True)

#ax_at = attributes_check.sum(axis=0).sort_values(ascending=False)[0:20].pl
#rot=90,color=['lightcoral'],ax=axat[1])
#for container in ax_at.containers:
#    ax_at.bar_label(container,color='black',label_type='center',weight='bo
#ax_at.set_title('Number of True for Attributes', fontsize=20, fontweight=



#Attribute True %

b_null_df = b_null_count.iloc[-1,:].sort_values(ascending=False).to_frame()
b_null_df = b_null_df[b_null_df.Data_Percentage > 0.2]
att_list = b_null_df.attribute.tolist()

att_list.extend(attributes_check.sum(axis=0).sort_values(ascending=False)[0
att_list = set(att_list)



att_count = pd.DataFrame()
for x,y in enumerate(att_list):
    new_df = pd.DataFrame(bus.loc[:,y].value_counts())
    att_count = pd.concat([att_count,new_df],axis=1)
att_count.reset_index(inplace=True)
att_count.loc[3,:] = ""
for i in range(1,24):
    att_count.iloc[3,i] = att_count.iloc[1,i] / np.sum(att_count.iloc[:2,i]
```

```
att_df = att_count.iloc[3,1:].sort_values(ascending=False).reset_index().re
att_df['%'] = att_df['%'].astype(float)
att_df['%'] = round(att_df['%'],2)

ax_att = att_df.plot(kind='bar',x='attribute',y='%',color=['lightcoral'],wi
for container in ax_att.containers:
    ax_att.bar_label(container,color='black',label_type='center',weight='bo
ax_att.set_title('Percentage of True in Attributes', fontsize=20, fontweigh
plt.savefig('Per. of Non-Null & True Att.',dpi=300,bbox_inches='tight')
#plt.savefig('Attribute_True_%.png',dpi=300,bbox_inches='tight')

#Select first 20 Attributes
selected_att = att_df.iloc[:20,0].tolist()
#Add From Percentage of Non-Null
selected_att.extend(att_list)
#Add From Number of True
selected_att.extend(attributes_check.sum(axis=0).sort_values(ascending=Fals
selected_att.append('RestaurantsPriceRange2')
selected_att.append('is_chain')
selected_att = set(selected_att)
```

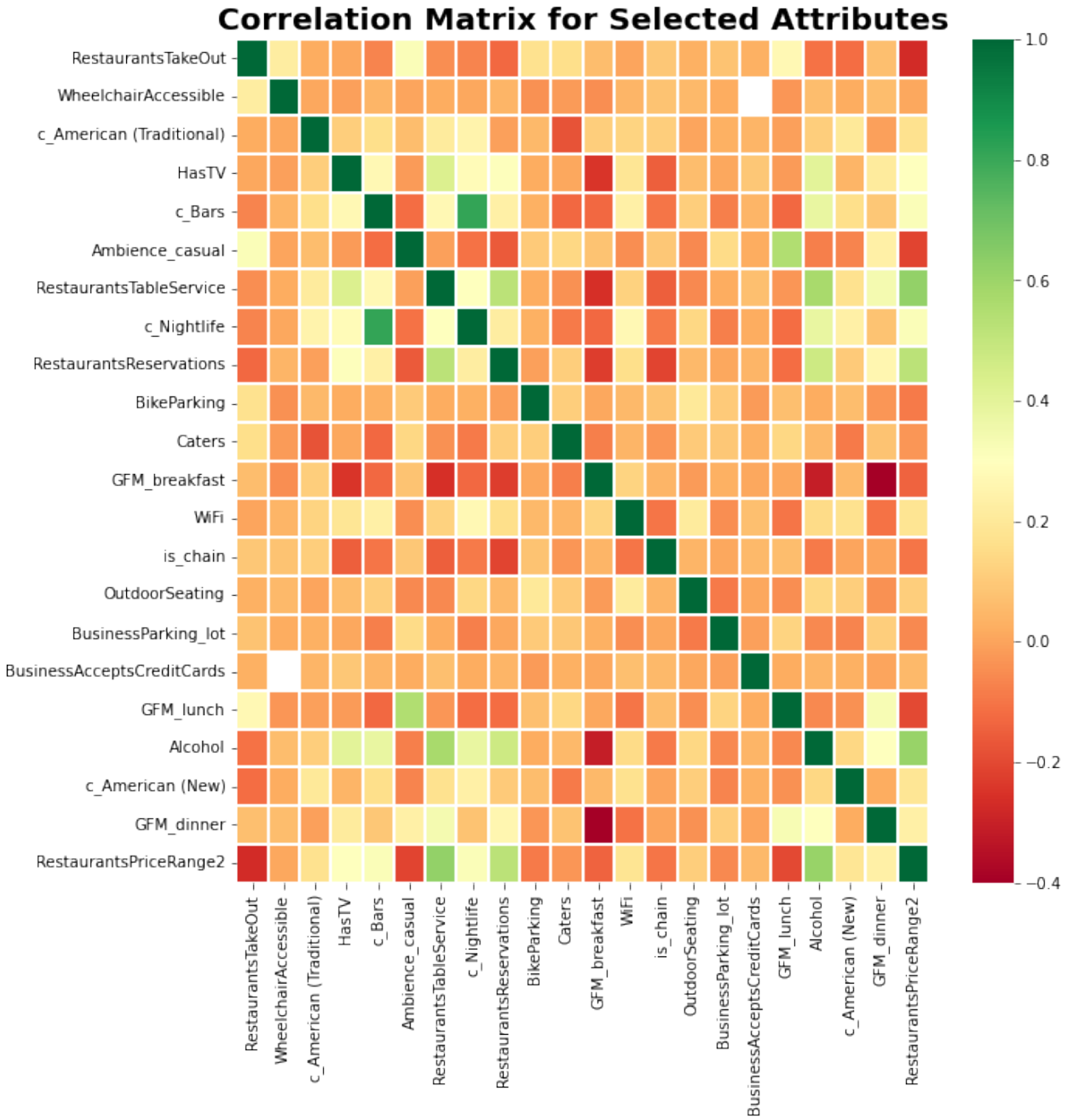



## Show Selected Attributes

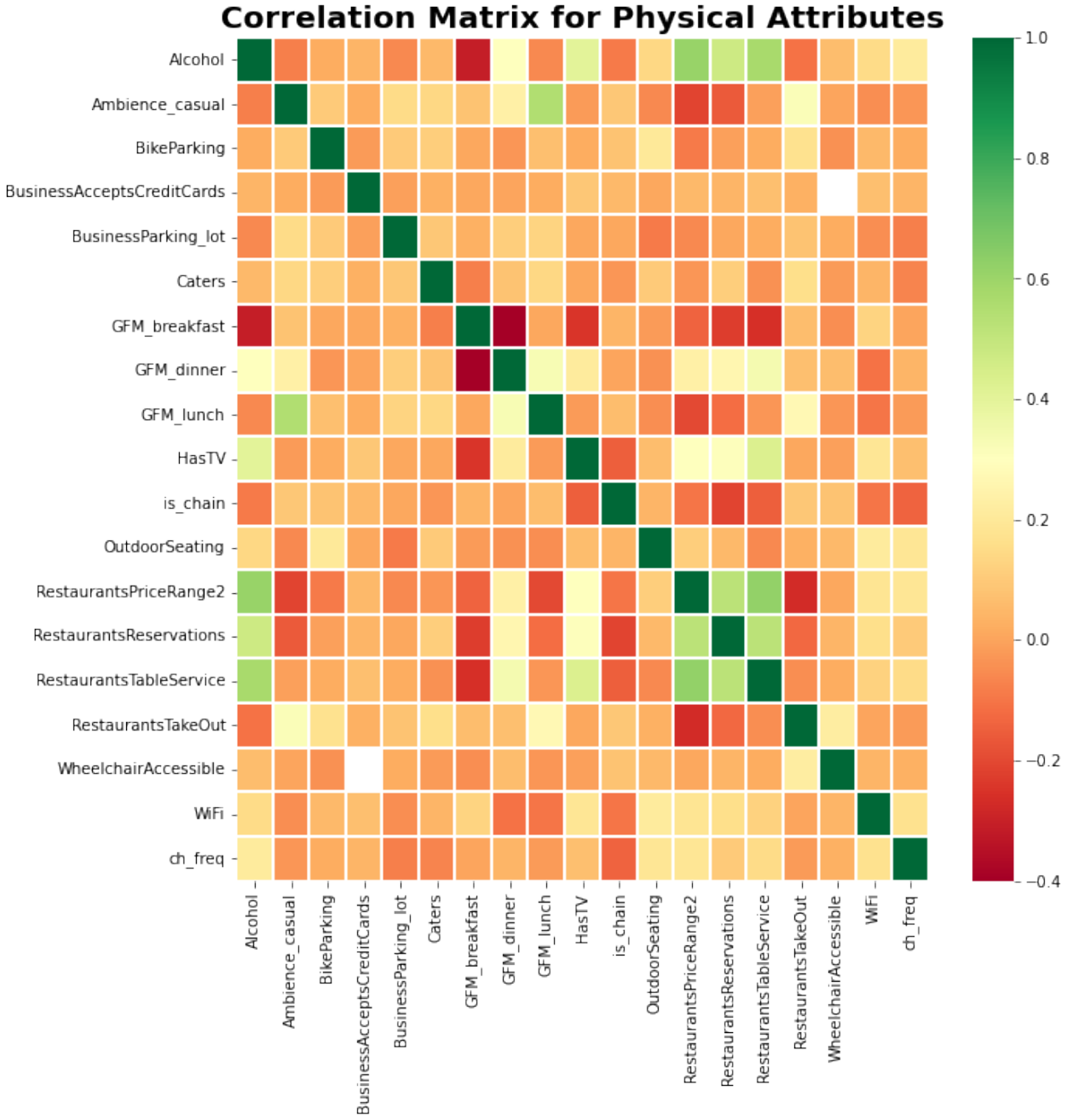
In [ ]:

```
plt.figure(figsize=(10,10))
sns.heatmap(round(bus[selected_att].corr(),2),cmap='RdYlGn',linewidths=0.2,
plt.title('Correlation Matrix for Selected Attributes', fontsize=20, fontwe
plt.show()
```


Correlation Matrix for Selected Attributes
RestaurantsTakeOut
WheelchairAccessible
c_American (Traditional)
HasTV
c_Bars
Ambience_casual
RestaurantsTableService
c_Nightlife
RestaurantsReservations
BikeParking
Caters
GFM_breakfast
WiFi
is_chain
OutdoorSeating
BusinessParking_lot
BusinessAcceptsCreditCards
GFM_lunch
Alcohol
c_American (New)
GFM_dinner
RestaurantsPriceRange2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

In [ ]:

```
selected_phy_att = ['Alcohol',
 'Ambience_casual',
 'BikeParking',
 'BusinessAcceptsCreditCards',
 'BusinessParking_lot',
 'Caters',
 'GFM_breakfast',
 'GFM_dinner',
 'GFM_lunch',
 'GoodForKids',
 'HasTV',
 'is_chain',
 'OutdoorSeating',
 'RestaurantsDelivery',
 'RestaurantsGoodForGroups',
 'RestaurantsPriceRange2',
 'RestaurantsReservations',
 'RestaurantsTableService',
 'RestaurantsTakeOut',
 'WheelchairAccessible',
 'WiFi',
 'ch_freq']

plt.figure(figsize=(10,10))
sns.heatmap(round(bus[selected_phy_att].corr(),2),cmap='RdYlGn',linewidths=
plt.title('Correlation Matrix for Physical Attributes', fontsize=20, fontwe
plt.savefig('Correlation Matrix for Physical Attributes.png', dpi=300, bbox
plt.show()
```




## Latex for Physical Attributes

In [ ]:

```
print(round(bus[selected_phy_att].describe(),3).T.iloc[:,[0,1,2,3,7]].to_la
```

```
\begin{tabular}{lrrrrr}
\toprule
{} &   count &   mean &    std &    min &    max \\ \hline
\midrule
Alcohol                   &  1736.0 &  0.662 &  0.473 &  0.000 &  1.000 \\
\hline
Ambience\_casual           &  1781.0 &  0.836 &  0.370 &  0.000 &  1.000 \
\ \hline
BikeParking               &  1772.0 &  0.920 &  0.272 &  0.000 &  1.000 \\
\hline
BusinessAcceptsCreditCards &  1780.0 &  0.996 &  0.067 &  0.000 &  1.000 \\
\hline
BusinessParking\_lot       &  1781.0 &  0.787 &  0.410 &  0.000 &  1.000 \
\ \hline
Caters                    &  1749.0 &  0.600 &  0.490 &  0.000 &  1.000 \\
\hline
GFM\_breakfast             &  1781.0 &  0.158 &  0.365 &  0.000 &  1.000 \
\ \hline
GFM\_dinner                &  1781.0 &  0.736 &  0.441 &  0.000 &  1.000 \
\ \hline
GFM\_lunch                 &  1781.0 &  0.765 &  0.424 &  0.000 &  1.000 \
\ \hline
HasTV                     &  1728.0 &  0.643 &  0.479 &  0.000 &  1.000 \\
\hline
is\_chain                  &  1781.0 &  0.385 &  0.487 &  0.000 &  1.000 \
\ \hline
OutdoorSeating            &  1756.0 &  0.666 &  0.472 &  0.000 &  1.000 \\
\hline
RestaurantsPriceRange2    &  1780.0 &  1.657 &  0.562 &  1.000 &  4.000 \\
\hline
RestaurantsReservations   &  1739.0 &  0.397 &  0.489 &  0.000 &  1.000 \\
\hline
RestaurantsTableService   &   704.0 &  0.699 &  0.459 &  0.000 &  1.000 \\
\hline
RestaurantsTakeOut        &  1760.0 &  0.966 &  0.182 &  0.000 &  1.000 \\
\hline
WheelchairAccessible      &   330.0 &  0.979 &  0.144 &  0.000 &  1.000 \\
\hline
WiFi                      &  1755.0 &  0.519 &  0.500 &  0.000 &  1.000 \\
\hline
ch\_freq                   &  1781.0 &  0.296 &  0.190 &  0.044 &  0.981 \
\ \hline
\bottomrule
\end{tabular}
```

# Create Numerical Analysis DataFrame

In [ ]:
```
desc_attributes = [ 'stars', 'is_chain', 'b_num', 'haversine_dist', 'pc_inc
desc_attributes.extend(selected_att)

bus_desc = bus[desc_attributes]
bus_desc.replace({False:0,True:1},inplace=True)

bus_desc = bus_desc.merge(review_avg,on='b_num')
bus_desc = round(bus_desc,3)
bus_desc
```

Out[ ]:

| | stars | is_chain | b_num | haversine_dist | pc_income | population | RestaurantsPriceRang |
|---|---|---|---|---|---|---|---|
| **0** | 2.5 | 1.0 | 2173 | 28.731 | 41054.0 | 30253.0 | |
| **1** | 3.0 | 1.0 | 118 | 16.422 | 25744.0 | 51930.0 | |
| **2** | 4.0 | 1.0 | 1855 | 26.245 | 26350.0 | 36244.0 | |
| **3** | 3.0 | 1.0 | 568 | 8.896 | 19688.0 | 41677.0 | |
| **4** | 2.5 | 1.0 | 2233 | 38.343 | 25628.0 | 46507.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **1785** | 3.5 | 0.0 | 1971 | 36.176 | 34176.0 | 15594.0 | |
| **1786** | 3.5 | 0.0 | 50 | 12.765 | 17556.0 | 57507.0 | |
| **1787** | 4.0 | 0.0 | 1332 | 28.689 | 34095.0 | 25000.0 | |
| **1788** | 4.5 | 0.0 | 777 | 18.422 | 25789.0 | 37172.0 | |
| **1789** | 4.0 | 0.0 | 823 | 23.343 | 31298.0 | 26100.0 | |

1790 rows × 47 columns

# C - Prediction

## Fix Files before Prediction

### Weather Data

In [ ]:

```
# Get Weather Data

# Import Meteostat library and dependencies
from datetime import datetime
from meteostat import Point, Daily

# Set time period
start = datetime(2015, 2, 2)
end = datetime(2015, 4, 1)
weather_data = pd.DataFrame()

for i in range(len(bus)):
    loc = Point(bus.iloc[i,6],bus.iloc[i,7])
    # Get Daily data
    w_data = Daily(loc, start, end)
    w_data = w_data.fetch()
    w_data.reset_index(inplace=True)
    w_data['b_num'] = bus.loc[i,'b_num']
    weather_data = weather_data.append(w_data)
    weather_data.reset_index(drop=True,inplace=True)

weather_data.rename(columns={'time':'date'},inplace=True)
weather_data_full = weather_data.copy()
#weather_data_full.to_csv('weather_data_full.csv')
```

In [ ]:

```
weather_data_full = pd.read_csv('weather_data_full.csv')
weather_data_full.drop(['Unnamed: 0'],axis=1,inplace=True)
"""
weather_data_full.date  = weather_data_full.date.apply(lambda x: pd.to_date
weather_data_full.date =  weather_data_full.date.apply(lambda x: datetime.c
"""
weather_data_full.date = pd.to_datetime(weather_data_full.date)
weather_data_full.sample(10)
```

Out[ ]:

| | date | tavg | tmin | tmax | prcp | snow | wdir | wspd | wpgt | pres | tsun | b_num | s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **170081** | 2015-04-04 | 23.9 | 16.7 | 31.7 | 0.0 | NaN | NaN | 9.0 | NaN | 1010.4 | NaN | 956 | |
| **391712** | 2015-08-28 | 32.1 | 27.2 | 42.8 | 0.0 | NaN | NaN | 7.2 | NaN | 1008.6 | NaN | 308 | |
| **8508** | 2015-01-29 | 18.4 | 14.4 | 21.7 | 6.1 | NaN | NaN | 4.7 | NaN | 1016.2 | NaN | 1572 | |
| **582675** | 2015-07-08 | 36.9 | 30.6 | 42.8 | 0.0 | NaN | NaN | 17.3 | NaN | 1004.2 | NaN | 75 | |
| **9910** | 2015-01-23 | 12.6 | 6.1 | 17.8 | 0.0 | NaN | NaN | 7.6 | NaN | 1023.7 | NaN | 2006 | |
| **620617** | 2015-09-13 | 32.7 | 23.9 | 38.3 | 1.6 | NaN | NaN | 7.7 | NaN | 1007.2 | NaN | 28 | |
| **151959** | 2015-03-10 | 19.5 | 9.2 | 28.9 | 0.0 | NaN | NaN | 7.1 | NaN | 1012.5 | NaN | 1923 | |
| **427160** | 2015-07-13 | 34.9 | 30.0 | 40.6 | 0.0 | NaN | NaN | 7.3 | NaN | 1008.6 | NaN | 108 | |
| **430542** | 2015-10-09 | 28.6 | 23.3 | 35.0 | 0.0 | NaN | 103.0 | 12.2 | NaN | 1013.5 | NaN | 1934 | |
| **502686** | 2015-12-04 | 15.7 | 8.3 | 22.8 | 0.0 | NaN | NaN | 6.8 | NaN | 1017.0 | NaN | 756 | |

In [ ]:

```
weather_data_full.iloc[:,[1,4]].describe()
```

Out[ ]:

| | tavg | prcp |
|---|---|---|
| **count** | 649899.000000 | 644572.000000 |
| **mean** | 24.151001 | 0.476732 |
| **std** | 8.608273 | 2.071452 |
| **min** | 2.000000 | 0.000000 |
| **25%** | 17.100000 | 0.000000 |
| **50%** | 24.200000 | 0.000000 |
| **75%** | 32.300000 | 0.000000 |
| **max** | 39.700000 | 23.600000 |

## Drop Businesses without Weather Data

In [ ]:
```
b_num_no_weather = np.setdiff1d(bus['b_num'].unique(), weather_data_full['b
b_num_no_weather
#No Weather Data for 9 Businesses, Drop Them
#bus.drop(bus[bus['b_num'].isin(b_num_no_weather)].index,inplace=True)
#bus.reset_index(drop=True,inplace=True)
```

Out[ ]:
```
[174, 395, 510, 1159, 1506, 1556, 1561, 1569, 2150]
```

### Create b_id_NZ_df_5

In [ ]:
```
#Number of Unique Businesses dropped to 1790, so save them as b_id_NZ_df_4.

#Add Unique values of business_id in review_nz_2015 to a new dataframe
b_id_nz_5_df = pd.DataFrame(list(bus['b_num'].unique()))
b_id_nz_5_df.rename(columns={0:'b_num'},inplace=True)

b_id_nz_5_df = b_id_nz_5_df.merge(bus[['business_id','b_num']],on='b_num',h
#Drop Duplicates
b_id_nz_5_df.drop_duplicates(inplace=True)
b_id_nz_5_df.reset_index(drop=True,inplace=True)
#b_id_nz_5_df.to_csv('b_id_NZ_df_5.csv',index=False)
```

## Bus Pred

In [ ]:
```
bus_pred = bus_desc.copy()
bus_pred.drop('business_id',inplace=True,axis=1)
bus_pred.replace({np.NaN:0,'None':0},inplace=True)
bus_pred = bus_pred.merge(b_id_nz_5_df,on='b_num',how='right')
bus_pred
```

Out[ ]:

| | stars | is_chain | b_num | haversine_dist | pc_income | population | RestaurantsPriceRang |
|---|---|---|---|---|---|---|---|
| **0** | 2.5 | 1.0 | 2173 | 28.731 | 41054.0 | 30253.0 | |
| **1** | 3.0 | 1.0 | 118 | 16.422 | 25744.0 | 51930.0 | |
| **2** | 4.0 | 1.0 | 1855 | 26.245 | 26350.0 | 36244.0 | |
| **3** | 3.0 | 1.0 | 568 | 8.896 | 19688.0 | 41677.0 | |
| **4** | 2.5 | 1.0 | 2233 | 38.343 | 25628.0 | 46507.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **1785** | 3.5 | 0.0 | 1971 | 36.176 | 34176.0 | 15594.0 | |
| **1786** | 3.5 | 0.0 | 50 | 12.765 | 17556.0 | 57507.0 | |
| **1787** | 4.0 | 0.0 | 1332 | 28.689 | 34095.0 | 25000.0 | |
| **1788** | 4.5 | 0.0 | 777 | 18.422 | 25789.0 | 37172.0 | |
| **1789** | 4.0 | 0.0 | 823 | 23.343 | 31298.0 | 26100.0 | |

1790 rows × 47 columns

# Frac Part

- This part used on sample analysis

## Create 30 Random Date, 25% of Businesses

In [ ]:

```
#Create a datelist

datelist_random = pd.date_range(start=np.datetime64('2015-01-01'), end= np.
random_datelist = datelist_random
random_datelist.sort()
for i in range(len(random_datelist)):
    random_datelist[i] = datetime.date(random_datelist[i])

#Get 25% of Businesses
bus_frac = bus_pred.sample(frac=1)
bus_frac= bus_frac.sort_values('b_num')
bus_frac.reset_index(drop=True,inplace=True)

date_df = pd.DataFrame(random_datelist)
date_df.rename(columns={0:'date'},inplace=True)
date_df = date_df.iloc[np.repeat(np.arange(len(date_df)), len(bus_frac))]
date_df['date'] = pd.to_datetime(date_df['date']).dt.normalize()
date_df.sort_values('date',ascending=True).reset_index(drop=True,inplace=Tr

#Crete Data File
data_col2 = ['date','b_num']
data = pd.DataFrame(columns=data_col2)

data['b_num'] = bus_frac['b_num']

data1 = data.copy()
for i in range(len(random_datelist)-1):
    data = data.append(data1)

data.reset_index(drop=True, inplace=True)
date_df.reset_index(drop=True,inplace=True)
data['date'] = date_df['date']
data['date'] = pd.to_datetime(data['date']).dt.normalize()

data = data.sort_values(['date','b_num'])
data.reset_index(drop=True,inplace=True)
data = pd.merge(data,bus_frac,on='b_num',how='left')

#Add Weather
data = pd.merge(data,weather_data_full[['b_num','date','tavg','prcp']], on=
data.reset_index(drop=True,inplace=True)
```

# Review

In [ ]:

```
review_daily = pd.read_csv('review_daily.csv')

review_daily_col = review_daily.columns.tolist()
for i in range(1, len(review_daily.columns)):
    review_daily_col[i] = pd.to_datetime(review_daily_col[i])
    review_daily_col[i] = datetime.date(review_daily_col[i])
review_daily.columns = review_daily_col

review_daily = review_daily.merge(b_id_nz_5_df,on='business_id',how='right'

review_daily.sort_values(['b_num'],inplace=True,ignore_index=True)

#Frac
review_daily_frac = review_daily[review_daily.b_num.isin(bus_frac.b_num.uni
review_daily_frac.reset_index(drop=True,inplace=True)

first_line = np.arange(len(data))
# 0 to 13350
second_line = np.repeat(np.arange(0,len(bus_frac)), len(random_datelist))
# 0 to 445, each 30 times
third_line = np.tile(random_datelist, len(bus_frac))
data['review'] = ""
for x,y,z in zip(first_line,second_line,third_line):
    data.iloc[x,-1] = review_daily_frac.loc[y,z]

data.review.replace(np.NaN,0,inplace=True)

for i in range(len(data)):
    if data.loc[i,'review'] > 0:
        data.loc[i,'review'] = 1
```

# Tips

In [ ]:

```
tip_daily = pd.read_csv('tip_daily.csv')

tip_daily_col = tip_daily.columns.tolist()
for i in range(1, len(tip_daily.columns)):
    tip_daily_col[i] = pd.to_datetime(tip_daily_col[i])
    tip_daily_col[i] = datetime.date(tip_daily_col[i])
tip_daily.columns = tip_daily_col

tip_daily = tip_daily.merge(b_id_nz_5_df,on='business_id',how='right')

tip_daily.sort_values(['b_num'],inplace=True,ignore_index=True)

#Frac
tip_daily_frac = tip_daily[tip_daily.b_num.isin(bus_frac.b_num.unique().tol
tip_daily_frac.reset_index(drop=True,inplace=True)

first_line = np.arange(len(data))
# 0 to 13350
second_line = np.repeat(np.arange(0,len(bus_frac)), len(random_datelist))
# 0 to 445, each 30 times
third_line = np.tile(random_datelist, len(bus_frac))
data['tips'] = ""
for x,y,z in zip(first_line,second_line,third_line):
    data.iloc[x,-1] = tip_daily_frac.loc[y,z]


data.tips.replace(np.NaN,0,inplace=True)

for i in range(len(data)):
    if data.loc[i,'tips'] > 0:
        data.loc[i,'tips'] = 1
```

## Check-in

In [ ]:

```
check_t_2015 = pd.read_csv('check_t_2015.csv')
check_tt = check_t_2015.T
check_tt.iloc[0,:] = pd.to_datetime(check_tt.iloc[0,:])
for i in range(0,len(check_tt.columns)):
    check_tt.iloc[0,i] = check_tt.iloc[0,i].date()
check_tt.columns += 1

check_tt.reset_index(inplace=True)
check_tt.rename(columns={'index':'business_id'},inplace=True)

check_tt.columns = check_tt.iloc[0,:]
check_tt = check_tt[1:]
check_tt.rename(columns={'Date':'business_id'},inplace=True)

check_tt = check_tt.merge(b_id_nz_5_df,on='business_id',how='right')
check_tt.drop('business_id',inplace=True,axis=1)
check_tt.sort_values(['b_num'],inplace=True,ignore_index=True)

check_tt_frac = check_tt[check_tt.b_num.isin(bus_frac.b_num.unique().tolist
check_tt_frac.reset_index(drop=True,inplace=True)


###

first_line = np.arange(len(data))
# 0 to 13350
second_line = np.repeat(np.arange(0,len(bus_frac)), len(random_datelist))
# 0 to 445, each 30 times
third_line = np.tile(random_datelist, len(bus_frac))
data['check'] = ""
for x,y,z in zip(first_line,second_line,third_line):
    data.iloc[x,-1] = check_tt_frac.loc[y,z]

data.check.replace(np.NaN,0,inplace=True)

for i in range(len(data)):
    if data.loc[i,'check'] > 0:
        data.loc[i,'check'] = 1
```

# Full Part

## Create Full Data

In [ ]:
```
#Create a datelist

random_datelist = pd.date_range(start=np.datetime64('2015-01-01'), end= np.
random_datelist.sort()
for i in range(len(random_datelist)):
    random_datelist[i] = datetime.date(random_datelist[i])

date_df = pd.DataFrame(random_datelist)
date_df.rename(columns={0:'date'},inplace=True)
date_df = date_df.iloc[np.repeat(np.arange(len(date_df)), len(bus_frac))]
date_df['date'] = pd.to_datetime(date_df['date']).dt.normalize()
date_df.sort_values('date',ascending=True)
date_df.reset_index(drop=True,inplace=True)


bus_frac = bus_pred
bus_frac= bus_frac.sort_values('b_num')
bus_frac.reset_index(drop=True,inplace=True)

#Crete Data File


data_1 = pd.DataFrame(np.tile(bus_frac['b_num'], len(random_datelist)))
data_1.rename(columns={0:'b_num'},inplace=True)

data = pd.DataFrame(columns=['date','b_num'])
data['b_num'] = data_1['b_num']
data['date'] = date_df['date']

data['date'] = pd.to_datetime(data['date']).dt.normalize()

data = data.sort_values(['date','b_num'])
data.reset_index(drop=True,inplace=True)
data = pd.merge(data,bus_frac,on='b_num',how='left')

#Add Weather
data = pd.merge(data,weather_data_full[['b_num','date','tavg','prcp']], on=
data.reset_index(drop=True,inplace=True)
```

## Add Daily Review, Tips, Checkin

In [ ]:
```
review_daily = pd.read_csv('review_daily.csv')

review_daily_col = review_daily.columns.tolist()
for i in range(1, len(review_daily.columns)):
    review_daily_col[i] = pd.to_datetime(review_daily_col[i])
    review_daily_col[i] = datetime.date(review_daily_col[i])
review_daily.columns = review_daily_col

review_daily = review_daily.merge(b_id_nz_5_df,on='business_id',how='right'

review_daily.sort_values(['b_num'],inplace=True,ignore_index=True)

review_daily.drop(['business_id','b_num'],inplace=True,axis=1)
```

```
review_list = []
for i in range(len(review_daily)):
    review_list.extend(review_daily.iloc[i,:])
data['review'] = review_list


#Tips
tip_daily = pd.read_csv('tip_daily.csv')

tip_daily_col = tip_daily.columns.tolist()
for i in range(1, len(tip_daily.columns)):
    tip_daily_col[i] = pd.to_datetime(tip_daily_col[i])
    tip_daily_col[i] = datetime.date(tip_daily_col[i])
tip_daily.columns = tip_daily_col

tip_daily = tip_daily.merge(b_id_nz_5_df,on='business_id',how='right')

tip_daily.sort_values(['b_num'],inplace=True,ignore_index=True)

tip_daily.drop(['business_id','b_num'],inplace=True,axis=1)
tip_list = []
for i in range(len(tip_daily)):
    tip_list.extend(tip_daily.iloc[i,:])
data['tips'] = tip_list


#Check

check_t_2015 = pd.read_csv('check_t_2015.csv')
check_tt = check_t_2015.T
check_tt.iloc[0,:] = pd.to_datetime(check_tt.iloc[0,:])
for i in range(0,len(check_tt.columns)):
    check_tt.iloc[0,i] = check_tt.iloc[0,i].date()
check_tt.columns += 1

check_tt.reset_index(inplace=True)
check_tt.rename(columns={'index':'business_id'},inplace=True)

check_tt.columns = check_tt.iloc[0,:]
check_tt = check_tt[1:]
check_tt.rename(columns={'Date':'business_id'},inplace=True)

check_tt = check_tt.merge(b_id_nz_5_df,on='business_id',how='right')
check_tt.drop('business_id',inplace=True,axis=1)
check_tt.sort_values(['b_num'],inplace=True,ignore_index=True)

check_tt.drop('b_num',inplace=True,axis=1)
check_list = []
for i in range(len(check_tt)):
    check_list.extend(check_tt.iloc[i,:])
data['check'] = check_list



data.review.replace(np.NaN,0,inplace=True)
data.tips.replace(np.NaN,0,inplace=True)
```

```
data.check.replace(np.NaN,0,inplace=True)

data.review = data.review.astype(bool).astype(int)
data.tips = data.tips.astype(bool).astype(int)
data.check = data.check.astype(bool).astype(int)
```

## Drop Data Without Prcp

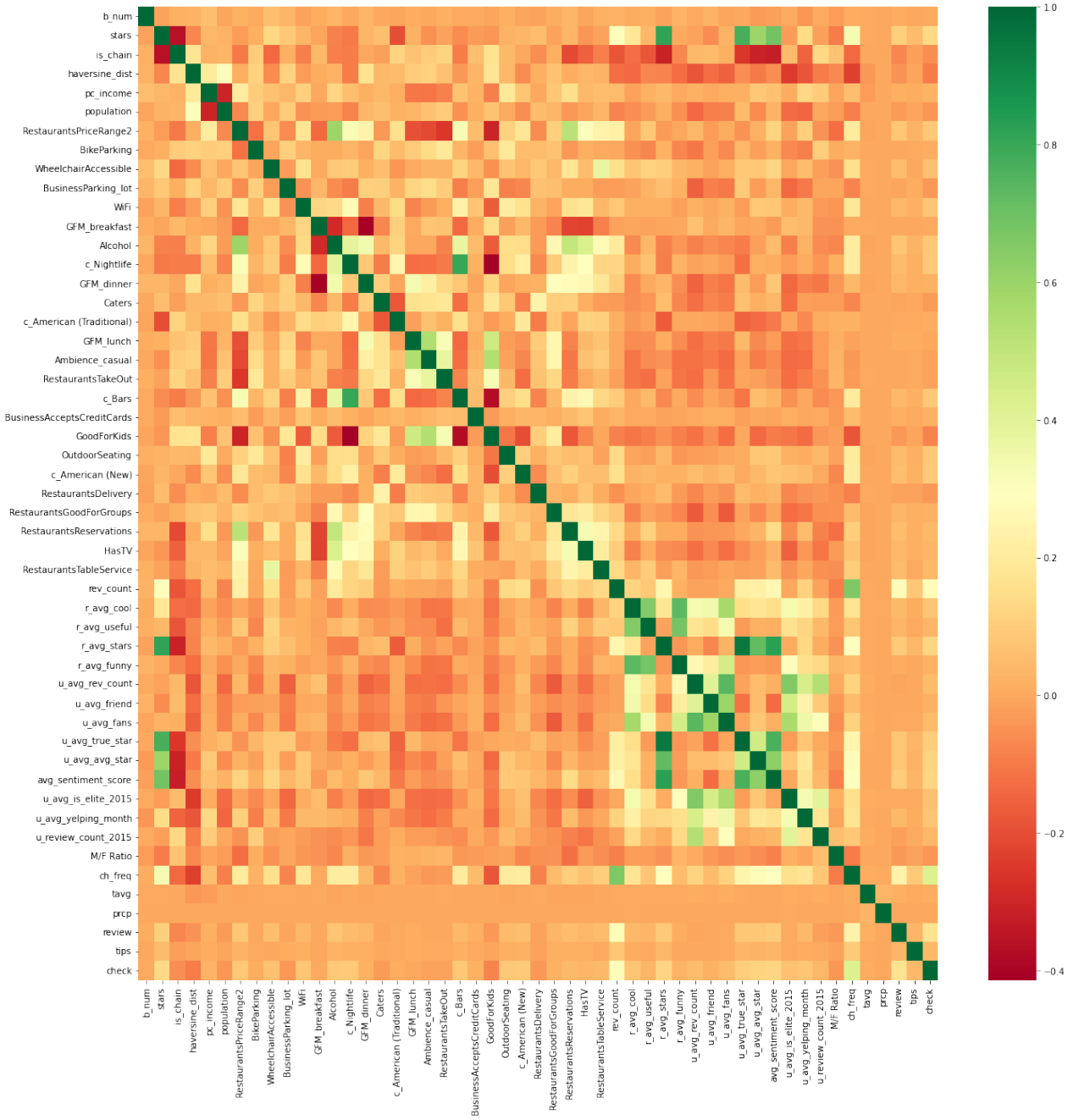
In [ ]:
```
# List of Business w/out Prcp
drop_weather = data.loc[:,'b_num'][data['prcp'].isnull()].unique()
#62
data = data.drop(data[data['b_num'].isin(drop_weather)].index)
data.reset_index(drop=True,inplace=True)
len(data)
```

Out[ ]:
```
627435
```

## Corr Table for Whole Data

In [ ]:
```
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),cmap='RdYlGn')
```

Out[ ]:
```
<AxesSubplot:>
```

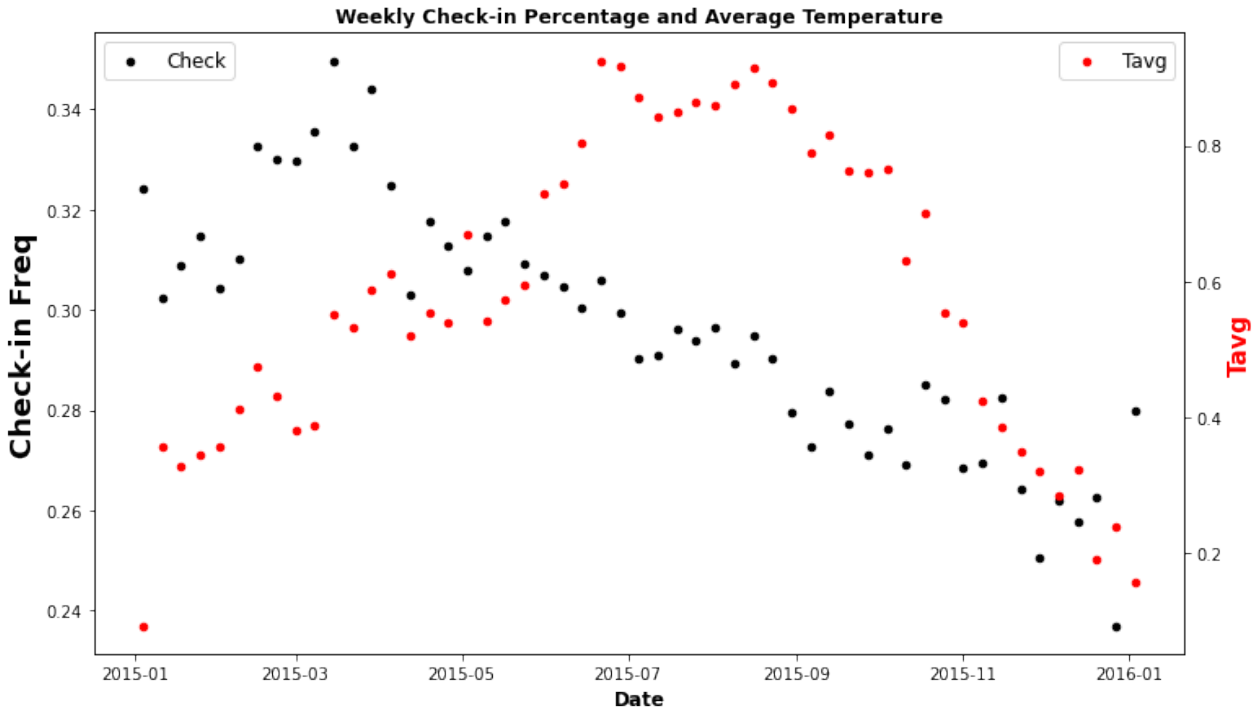



In [ ]:

```
data.date = pd.to_datetime(data.date)
```

In [ ]:

```
#ax0 = data.iloc[:,[0,-1,-5]].groupby(data.date).mean().reset_index().plot(
ax0 = data.set_index('date').iloc[:,[-1,-5]].groupby(pd.Grouper(freq='W')).
    kind='scatter',y='check',x='date',color='black',figsize=(12,7))
ax0.set_ylabel('Check-in Freq',color='black',fontsize=18,fontweight='bold')
ax0.set_xlabel('Date',fontsize=12,fontweight='bold')
ax0.set_title('Weekly Check-in Percentage and Average Temperature',fontsize
ax0.legend(['Check'],loc='upper left',fontsize=12,frameon=True)

ax1 = ax0.twinx()
ax1 = data.set_index('date').iloc[:,[0,-1,-5]].groupby(pd.Grouper(freq='W')
    kind='scatter',y='tavg',x='date',ax=ax1,color='red')
ax1.set_ylabel('Tavg',color='red',fontsize=15,fontweight='bold')
ax1.legend(['Tavg'],loc='upper right',fontsize=12,frameon=True)

plt.savefig('Weekly_Checkin&Temp.png',dpi=300,bbox_inches='tight')
```




## Scaling

In [ ]:

```
# Do Min Max Scaling for Necessary Columns in data
data.drop('business_id',axis=1,inplace=True)
data.drop('ch_freq',axis=1,inplace=True)
```

In [ ]:
```
#Min Max Scaling
scaled_columns = ['stars', 'RestaurantsPriceRange2','haversine_dist',
'pc_income', 'population', 'rev_count','r_avg_cool','r_avg_useful','r_avg_s
'r_avg_funny','u_avg_rev_count','u_avg_friend','u_avg_fans','u_avg_true_sta
'u_review_count_2015','M/F Ratio','tavg','prcp']

#Min Max Scale
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
for i in scaled_columns:
    data.loc[:,i] = scaler.fit_transform(data.loc[:,i].values.reshape(-1,1)
```

# D - ML

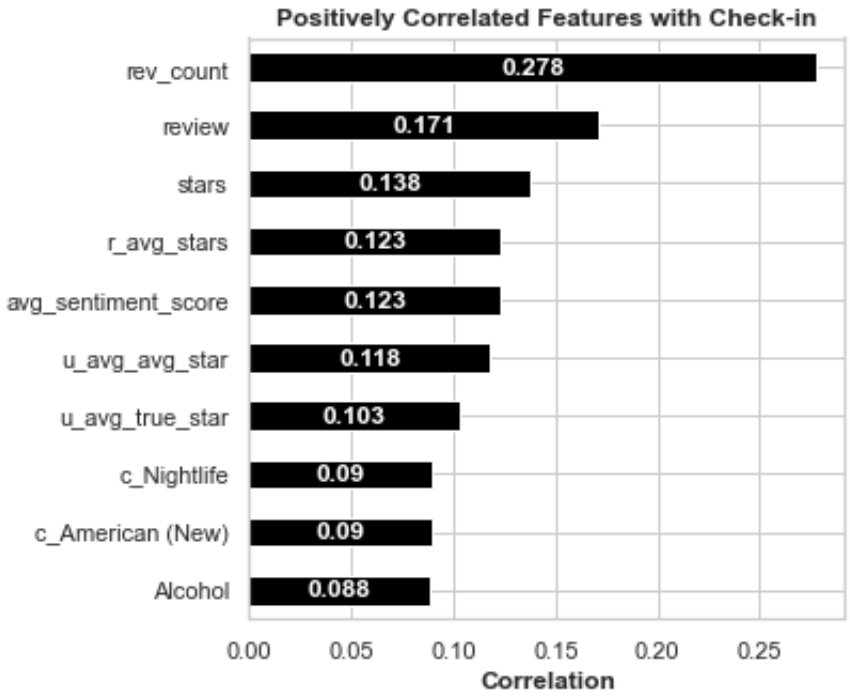
## Read File & Libraries

In [ ]:
```
data = pd.read_csv('data_to_predict.csv')
from sklearn.linear_model import LogisticRegression
from xgboost import XGBRFClassifier
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
import time
from sklearn.ensemble import RandomForestRegressor, StackingClassifier
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import HalvingGridSearchCV
```

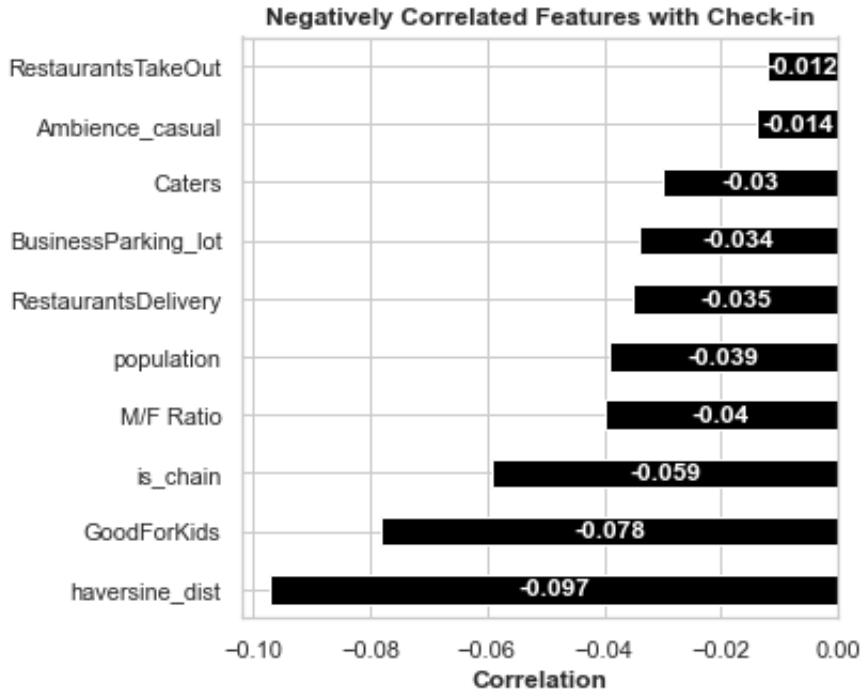
## Correlation HeatMap for All Data

In [ ]:
```
corr= data.corr()
corr = round(corr,3)
```

In [ ]:
```
ax_corr = corr['check'].sort_values(ascending=True)[39:49].plot(kind='barh'
ax_corr.set_xlabel('Correlation',fontsize=12,fontweight='bold')
ax_corr.set_title('Positively Correlated Features with Check-in',fontsize=1
for container in ax_corr.containers:
    ax_corr.bar_label(container, fontsize=12, fontweight='bold', color='whi
plt.savefig('Positive_Correlation_Checkin.png',dpi=300,bbox_inches='tight')
```




In [ ]:
```
ax_corr2 = corr['check'].sort_values(ascending=True)[:10].plot(kind='barh',
ax_corr2.set_xlabel('Correlation',fontsize=12,fontweight='bold')
ax_corr2.set_title('Negatively Correlated Features with Check-in',fontsize=
for container in ax_corr2.containers:
    ax_corr2.bar_label(container, fontsize=12, fontweight='bold', color='wh
plt.savefig('Negative_Correlation_Checkin.png',dpi=300,bbox_inches='tight')
```




## Split Data

In [ ]:
```
data_backup = data.copy()
len(data_backup)
```

Out[ ]: 627435

In [ ]:
```
#Split Train Test Dataset

X = data.drop('check',axis=1)
X = X.iloc[:,2:]
y = data['check']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, r
```

## Results DF

In [ ]:
```
#results_df = pd.DataFrame(columns=['Model','param','Accuracy','R2','MSE',

actual = ['Actual',"","","","","","",
np.array(y_test.reset_index()['check']),
"",""]
results_df.loc[len(results_df),:] = actual
```

# Ml Models

## 1.KNN

In [ ]:
```
#Do K-nearest Neighbors for Check Column in data
knn_results = []
for i in [10,20,50,100]:
#Perform Knn
    knn = KNeighborsClassifier(n_neighbors=i)
    start_time = time.time()
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    proba = knn.predict_proba(X_test)
    end_time = time.time()

    # Evaluate the model
    #print('-------',i,'-------')
    #print('KNN Accuracy: ', accuracy_score(y_test, y_pred))
    #print('KNN ROC AUC: ', roc_auc_score(y_test, y_pred))
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    results_df.loc[len(results_df),:] = [f'KNN_{20}',
    i,
    round(accuracy_score(y_test, y_pred),4),
    round(r2_score(y_test, y_pred),4),
    round(mean_squared_error(y_test, y_pred),4),
    round(roc_auc_score(y_test, y_pred),4),
    round(end_time-start_time,4),
    y_pred,
    proba[:,1],
    'KNN']
```

## 2.Linear Regression

In [ ]:
```
#Perform Linear Regression on dataframe data
lr_results = []
#Perform Linear Regression on dataframe data, column check
lr = LinearRegression()
start_time = time.time()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
end_time = time.time()
results_df.loc[len(results_df),:] = ['Linear Regression',
"",
round(r2_score(y_test, y_pred),4),
round(mean_squared_error(y_test, y_pred),4),
round(roc_auc_score(y_test, y_pred),4),
round(end_time-start_time,4),
y_pred,
'LR']
#Show the Results of Linear Regression
print('Linear Regression Mean Squared Error: ', mean_squared_error(y_test,
print('Linear Regression R2 Score: ', r2_score(y_test, y_pred))
```

```
Linear Regression Mean Squared Error:  0.18773407400624464
Linear Regression R2 Score:  0.1074343030528302
```

## 3.Naive Bayes

In [ ]:
```
#Perform Naive Bayes on dataframe data, column check
#var_smoothing, default 1e-09 try until 5
nb = GaussianNB(var_smoothing=1)
start_time = time.time()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
proba = np.around(nb.predict_proba(X_test)[:,1],4)
end_time = time.time()

#Show the Results of Naive Bayes

#Import scikit-learn metrics module for accuracy calculation

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
results_df.loc[len(results_df),:] = ['Naive Bayes',
1,
round(metrics.accuracy_score(y_test, y_pred),4),
round(r2_score(y_test, y_pred),4),
round(mean_squared_error(y_test, y_pred),4),
round(roc_auc_score(y_test, y_pred),4),
round(end_time-start_time,4),
y_pred,
proba,
'NB']
```

```
Accuracy: 0.7094224148922607
```

## 4.SVM

In [ ]:
```
#Perform SVM on dataframe data, column check

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
#Show the Results of SVM
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
results_df.loc[len(results_df),:] = ['SVM',
round(metrics.accuracy_score(y_test, y_pred),4),
round(r2_score(y_test, y_pred),4),
round(mean_squared_error(y_test, y_pred),4),
round(roc_auc_score(y_test, y_pred),4),
round(end_time-start_time,4),
y_pred,
'SVM']
```

## 5.Neural Network

In [ ]:

```
#Perform ANN on dataframe data, column check
for solv in ['adam','lbfgs']:
    for layer in [100,200,500]:
        ann = MLPClassifier(solver=solv, alpha=1e-5, hidden_layer_sizes=(la
        start_time = time.time()
        ann.fit(X_train, y_train)
        y_pred = ann.predict(X_test)
        proba = np.around(ann.predict_proba(X_test)[:,1],4)
        end_time = time.time()
        #Show the Results of ANN
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
        #print("Confusion Matrix:",confusion_matrix(y_test, y_pred))
        #print("Classification Report:",classification_report(y_test, y_pre
        print("ROC AUC:",roc_auc_score(y_test, y_pred))
        results_df.loc[len(results_df),:] = ['ANN',
        str(solv)+'_'+str(layer),
        round(metrics.accuracy_score(y_test, y_pred),4),
        round(r2_score(y_test, y_pred),4),
        round(mean_squared_error(y_test, y_pred),4),
        round(roc_auc_score(y_test, y_pred),4),
        round(end_time-start_time,4),
        y_pred,
        proba,
        'ANN']
```

```
Accuracy: 0.7487696034680607
ROC AUC: 0.6119582587097405
Accuracy: 0.7500956266734667
ROC AUC: 0.6318501596413348
Accuracy: 0.7046028305495347
ROC AUC: 0.5
Accuracy: 0.7449190360831315
ROC AUC: 0.6213834931054699
Accuracy: 0.7496876195333418
ROC AUC: 0.6286778826543211
Accuracy: 0.7046028305495347
ROC AUC: 0.5
```

## 6.Random Forest

In [ ]:

```
#Perform Random Forest on dataframe data, column check
#n_estimator 100,200,500, max_depth 0,20
for est in [100,200,500]:
    for depth in [5,20]:
            rf = RandomForestClassifier(n_estimators=est, random_state=42,r
            start_time = time.time()
            rf.fit(X_train, y_train)
            y_pred = rf.predict(X_test)
            proba = np.around(rf.predict_proba(X_test)[:,1],4)
            end_time = time.time()
            #Evaluate the Results of Random Forest Classifier
            print("Accuracy:",metrics.accuracy_score(y_test, y_pred))


            """
            results_df.loc[len(results_df),:] = ['RF',
            str(est)+'_'+str(depth),
            round(metrics.accuracy_score(y_test, y_pred),4),
            round(r2_score(y_test, y_pred),4),
            round(mean_squared_error(y_test, y_pred),4),
            round(roc_auc_score(y_test, y_pred),4),
            round(end_time-start_time,4),
            y_pred,
            proba,
            'RF']"""
```

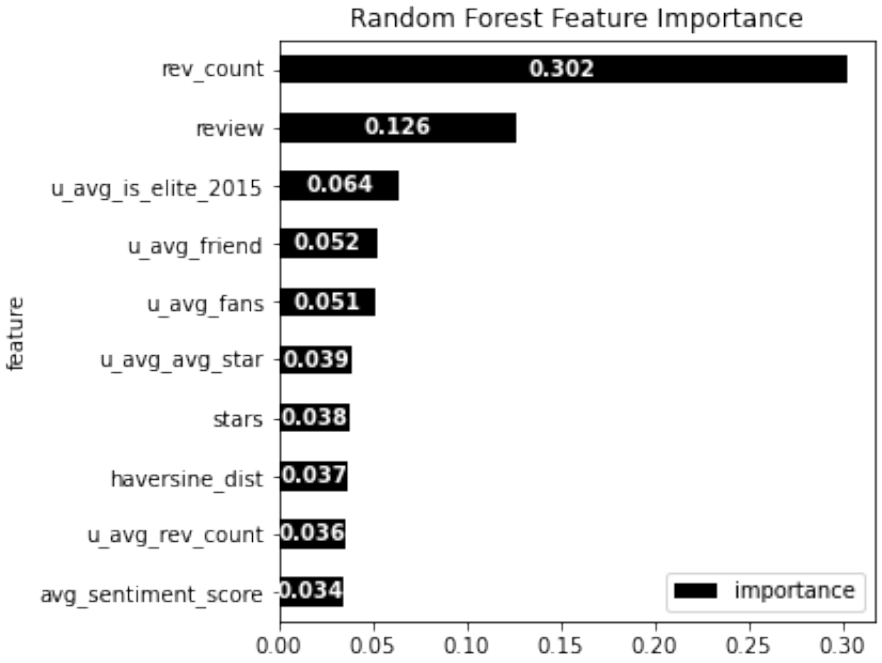
In [ ]:

```
rf = RandomForestClassifier(n_estimators=200, random_state=42,max_depth=5)
start_time = time.time()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
proba = np.around(rf.predict_proba(X_test)[:,1],4)
end_time = time.time()
#Evaluate the Results of Random Forest Classifier
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
rf_features = round(pd.DataFrame(X_train.columns,rf.feature_importances_).r
            0:'feature'}).sort_values('importance',ascending=False)[:10],3)
            ascending=True)
```

```
Accuracy: 0.7252071911258448
```

In [ ]:

```
ax0 = rf_features.plot(kind='barh',x='feature',y='importance',color='black'
for container in ax0.containers:
    ax0.bar_label(container, label_type='center',fontsize=10, color='white'
ax0.set_title('Random Forest Feature Importance')
plt.savefig('RF_Feature_Importance_black.png',dpi=300,bbox_inches='tight')
```




## 7.XG Boost

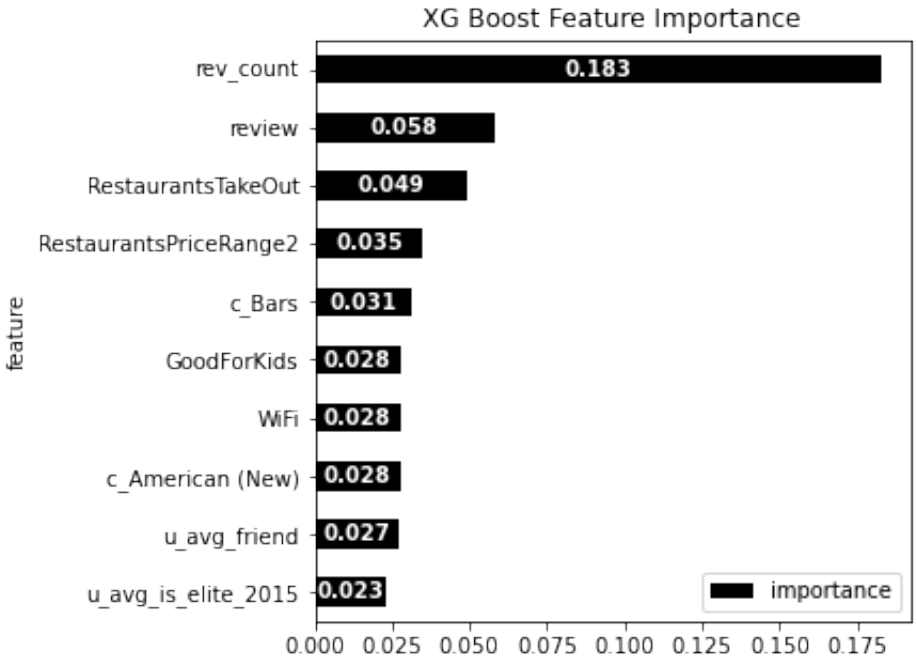
In [ ]:

```
#Perform XG Boost on dataframe data, column check

#max_depth, def6, try 3,10, eta def 0.3 try 0.1 0.3 0.5
for depth in [3,6,10]:
    for etax in [0.1,0.3,0.5]:
        xgb = XGBClassifier(eta=etax,max_depth=depth)
        start_time = time.time()
        xgb.fit(X_train, y_train)
        y_pred = xgb.predict(X_test)
        proba = np.around(xgb.predict_proba(X_test)[:,1],4)
        end_time = time.time()
        #Evaluate the Results of XG Boost Classifier
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
        print("ROC AUC:",metrics.roc_auc_score(y_test, y_pred))
        """
        ax_gb = pd.DataFrame(X_train.columns,xgb.feature_importances_).rese
        0:'feature'}).sort_values('importance',ascending=False)[:20].sort_v
        ascending=True).plot(kind='barh',x='feature',y='importance')
        ax_gb.title.set_text('XG Boost Feature Importance')
        plt.show()
        """
        results_df.loc[len(results_df),:] = ['XG Boost',
        str(depth)+'_'+str(etax),
        round(metrics.accuracy_score(y_test, y_pred),4),
        round(r2_score(y_test, y_pred),4),
        round(mean_squared_error(y_test, y_pred),4),
        round(metrics.roc_auc_score(y_test, y_pred),4),
        round(end_time-start_time,4),
        y_pred,
        proba,
        'XGB']
```

In [ ]:
```
xgb = XGBClassifier(eta=0.3,max_depth=6)
start_time = time.time()
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
proba = np.around(xgb.predict_proba(X_test)[:,1],4)
end_time = time.time()
#Evaluate the Results of XG Boost Classifier
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("ROC AUC:",metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.7496876195333418
ROC AUC: 0.6346808556679006
```

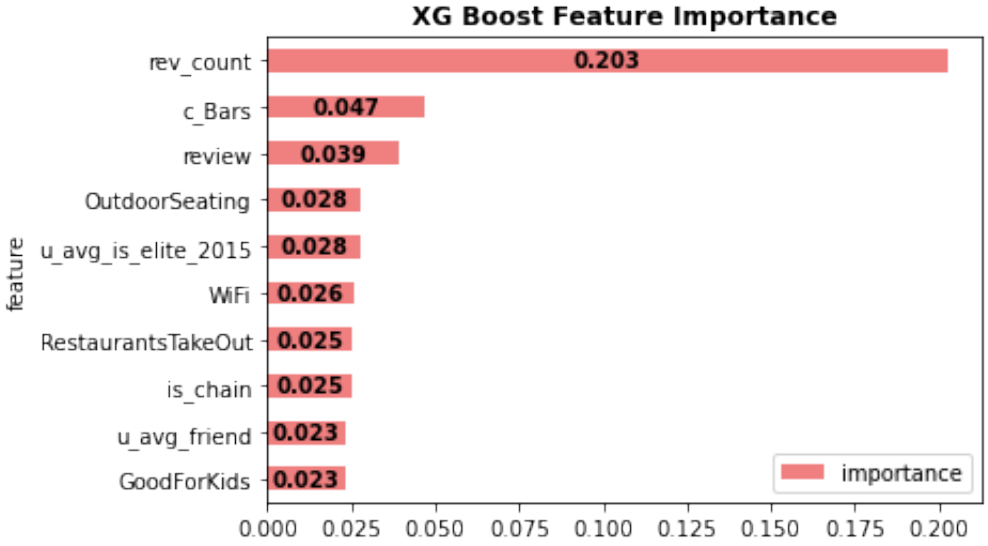
In [ ]:
```
ax_gb = round(pd.DataFrame(X_train.columns,xgb.feature_importances_).reset_
0:'feature'}).sort_values('importance',ascending=False)[:10],3).sort_values
ascending=True).plot(kind='barh',x='feature',y='importance',figsize=(5,5),c
for container in ax_gb.containers:
    ax_gb.bar_label(container, label_type='center',fontsize=10, color='whit
ax_gb.title.set_text('XG Boost Feature Importance')
#Change background color to white
ax_gb.set_facecolor('white')
plt.savefig('XG Boost Feature Importance_black.png',dpi=300,bbox_inches='ti
```

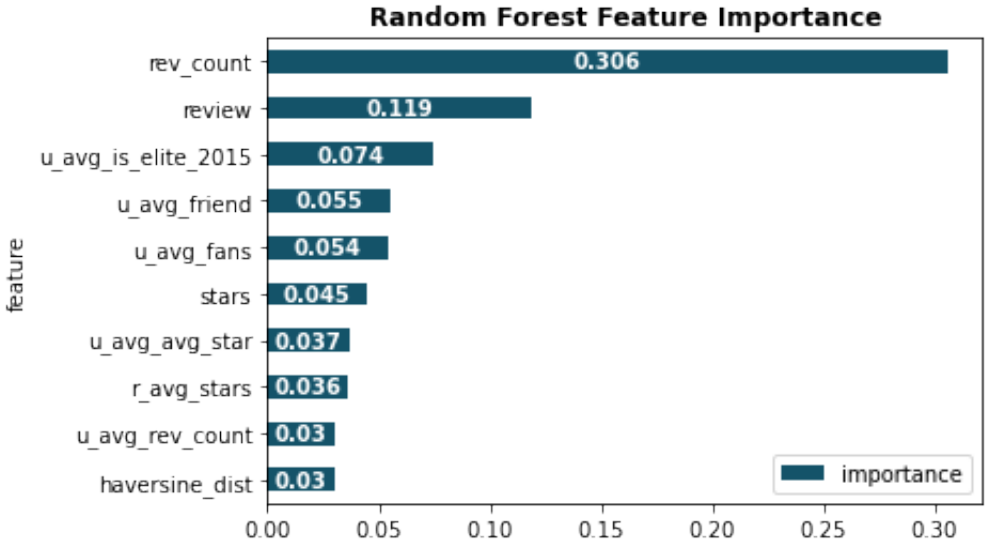



## Feature Imp. Plot for RF & XG Boost

In [ ]:

```
ax_gb = round(pd.DataFrame(X_train.columns,xgb.feature_importances_).reset_
0:'feature'}).sort_values('importance',ascending=False)[:10],3).sort_values
ascending=True).plot(kind='barh',x='feature',y='importance',color='lightcor
for container in ax_gb.containers:
    ax_gb.bar_label(container, label_type='center',fontsize=10, color='blac
ax_gb.set_title('XG Boost Feature Importance',fontsize=12,fontweight='bold'
plt.savefig('XG Boost Feature Importance.png',dpi=300,bbox_inches='tight')

#Increase the white space between the subplots
ax0 = rf_features.plot(kind='barh',x='feature',y='importance',color='#14536
for container in ax0.containers:
    ax0.bar_label(container, label_type='center',fontsize=10, color='white'
ax0.set_title('Random Forest Feature Importance',fontsize=12,fontweight='bo
plt.savefig('Random Forest Feature Importance.png',dpi=300,bbox_inches='tig
plt.show()
```




## 7.1 XGBRF

In [ ]:

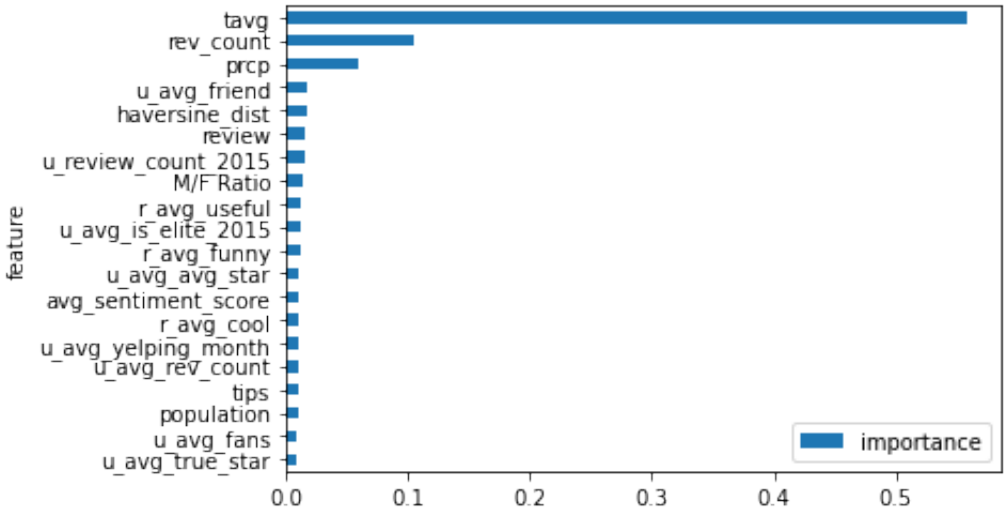
```
#Perform XGBoost Random Forest Classifier on dataframe data, column check
#nestimators, 20,50,100, max depth def 6 try 3,6,10

for est in [20,50,100]:
    for depth in [3,6,10]:
        start_time = time.time()
        xgbrf = XGBRFClassifier(n_estimators=est, max_depth=depth)
        xgbrf.fit(X_train, y_train)
        y_pred = xgbrf.predict(X_test)
        proba = np.around(xgbrf.predict_proba(X_test)[:,1],4)
        end_time = time.time()
        #Evaluate the Results of XG Boost Random Forest Classifier
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
        print("ROC AUC:",metrics.roc_auc_score(y_test, y_pred))
        pd.DataFrame(X_train.columns,xgbrf.feature_importances_).reset_inde
        0:'feature'}).sort_values('importance',ascending=False)[:20].sort_v
        ascending=True).plot(kind='barh',x='feature',y='importance')
        plt.show()
        results_df.loc[len(results_df),:] = ['XGBRF',
        str(est)+'_'+str(depth),
        round(metrics.accuracy_score(y_test, y_pred),4),
        round(r2_score(y_test, y_pred),4),
        round(mean_squared_error(y_test, y_pred),4),
        round(metrics.roc_auc_score(y_test, y_pred),4),
        round(end_time-start_time,4),
        y_pred,
        proba,
        'XGBRF']
```

## 8.Decision Tree

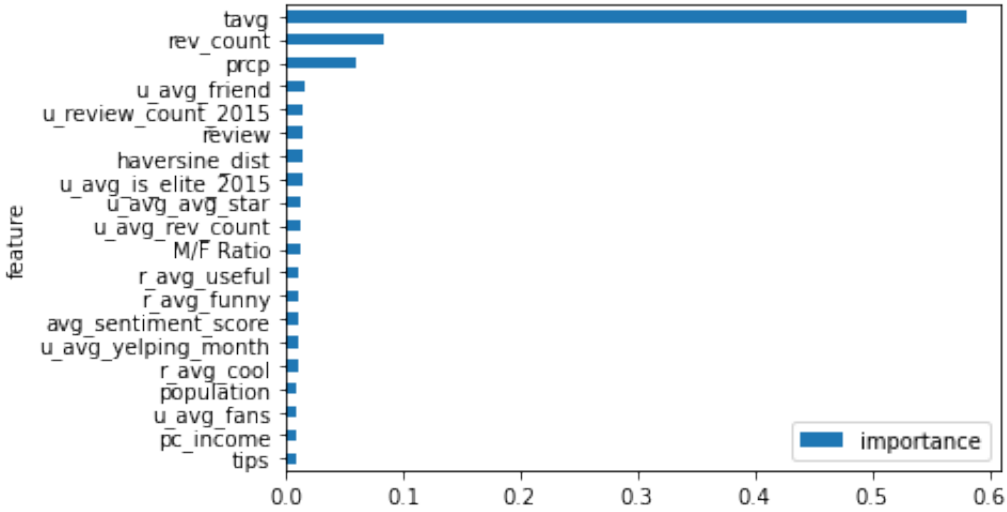
In [ ]:

```
#Perform Tree on dataframe data, column check
#criterion def gini, try entropy, log_loss
for crit in ['gini','entropy']:
    tree = DecisionTreeClassifier(criterion=crit)
    start_time = time.time()
    tree.fit(X_train, y_train)
    y_pred = tree.predict(X_test)
    proba = np.around(tree.predict_proba(X_test)[:,1],4)
    end_time = time.time()
    #Evaluate the Results of Decision Tree Classifier
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
    print("ROC AUC:",metrics.roc_auc_score(y_test, y_pred))
    pd.DataFrame(X_train.columns,tree.feature_importances_).reset_index().r
    0:'feature'}).sort_values('importance',ascending=False)[:20].sort_value
    ascending=True).plot(kind='barh',x='feature',y='importance')
    plt.show()
    results_df.loc[len(results_df),:] = ['Decision Tree',
    str(crit),
    round(metrics.accuracy_score(y_test, y_pred),4),
    round(r2_score(y_test, y_pred),4),
    round(mean_squared_error(y_test, y_pred),4),
    round(metrics.roc_auc_score(y_test, y_pred),4),
    round(end_time-start_time,4),
    y_pred,
    proba,
    'DecisionT']
```

```
Accuracy: 0.6566874920311105
ROC AUC: 0.5864470297169025
```




```
Accuracy: 0.6594670406732118
ROC AUC: 0.588269047799123
```




# 9.Logistic Regression

In [ ]:

```
#penalty default l2, try l1, elasticnet, max_iter def 100 try 100,250,500
for solv in ['newton-cg', 'lbfgs','saga']:
    for iter in [100,250,500]:
        start_time = time.time()
        logreg = LogisticRegression(solver=solv,max_iter=iter,)
        logreg.fit(X_train, y_train)
        y_pred = logreg.predict(X_test)
        proba = np.around(logreg.predict_proba(X_test)[:,1],4)
        end_time = time.time()
        #Evaluate the Results of Logistic Regression
        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
        print("ROC AUC:",metrics.roc_auc_score(y_test, y_pred))
        """
        pd.DataFrame(X_train.columns,logreg.coef_).reset_index().rename(col
        0:'feature'}).sort_values('importance',ascending=False)[:20].sort_v
        ascending=True).plot(kind='barh',x='feature',y='importance')
        plt.show()
        """
        results_df.loc[len(results_df),:] = ['Logistic Regression',
        str(solv)+'_'+str(iter),
        round(metrics.accuracy_score(y_test, y_pred),4),
        round(r2_score(y_test, y_pred),4),
        round(mean_squared_error(y_test, y_pred),4),
        round(metrics.roc_auc_score(y_test, y_pred),4),
        round(end_time-start_time,4),
        y_pred,
        proba,
        'LogReg']
```

```
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
Accuracy: 0.7319393089379064
ROC AUC: 0.5786326879754723
Accuracy: 0.7319393089379064
ROC AUC: 0.5784822837783689
Accuracy: 0.7318883080453908
ROC AUC: 0.5784711599100698
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
Accuracy: 0.7318883080453908
ROC AUC: 0.5784460925438858
```

## 10.Stacked

In [ ]:

```
base_models = [
    ('ANN',MLPClassifier()),
    ('XGB', XGBClassifier()),
    ('RF', RandomForestClassifier()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('XGBRF', XGBRFClassifier()),
    ('NB',GaussianNB()),
    ('LogReg',LogisticRegression()),
    ]
stacked = StackingClassifier(
    estimators = base_models,
    final_estimator = XGBClassifier(),
    cv = 5)

start_time = time.time()
stacked.fit(X_train, y_train)
stacked_prediction = stacked.predict(X_test)
#proba = np.around(stacked.prediction_proba(X_test)[:,1],4)
end_time = time.time()
stacked_r2 = stacked.score(X_test, y_test)
stacked_rmse = mean_squared_error(y_test, stacked_prediction, squared = Fal
print("-------Stacked Ensemble-------")
print("Coefficient of determination: {}".format(stacked_r2))
print("Root Mean Squared Error: {}".format(stacked_rmse))
print("Computation Time: {}".format(end_time - start_time))
print("Accuracy:",metrics.accuracy_score(y_test, stacked_prediction))
print("ROC AUC:",metrics.roc_auc_score(y_test, stacked_prediction))
print("---------------------------------")

results_df.loc[len(results_df),:] = ['Stacked',
"",
round(metrics.accuracy_score(y_test, stacked_prediction),4),
round(r2_score(y_test, stacked_prediction),4),
round(mean_squared_error(y_test, stacked_prediction),4),
round(metrics.roc_auc_score(y_test, stacked_prediction),4),
round(end_time-start_time,4),
y_pred,
"",
'Stacked']
```

```
-------Stacked Ensemble-------
Coefficient of determination: 0.7425984954736707
Root Mean Squared Error: 0.5073475184982471
Computation Time: 550.0522971153259
Accuracy: 0.7425984954736707
ROC AUC: 0.623185920002
---------------------------------
```

## Parameter DF

In [ ]:
```
#Create a Dataframe for Parameters
param_df = pd.DataFrame(columns=['Model','1st Parameter','2nd Parameter'])
#Add all models to Model Column
param_df['Model'] = results_df_best['Model']
param_df.reset_index(drop=True, inplace=True)
param_df.iloc[0,1] = 'Var Smoothing: Default = 1e-09,\n Tested = [1e-09:1]'
print(param_df.to_latex(index=False))
```

```
\begin{tabular}{lll}
\toprule
              Model &                                          1st Parameter &
2nd Parameter \\
\midrule
        Naive Bayes & Var Smoothing: Default = 1e-09,\textbackslash n Teste
d = [1e-... &            NaN \\
                ANN &                                                    NaN &
NaN \\
                 RF &                                                    NaN &
NaN \\
           XG Boost &                                                    NaN &
NaN \\
              XGBRF &                                                    NaN &
NaN \\
      Decision Tree &                                                    NaN &
NaN \\
Logistic Regression &                                                    NaN &
NaN \\
            KNN\_50 &                                                     NaN &
NaN \\
            Stacked &                                                    NaN &
NaN \\
\bottomrule
\end{tabular}
```

# Results DF

In [ ]:
```
results_df = pd.read_pickle('results_df_2.pkl')
```

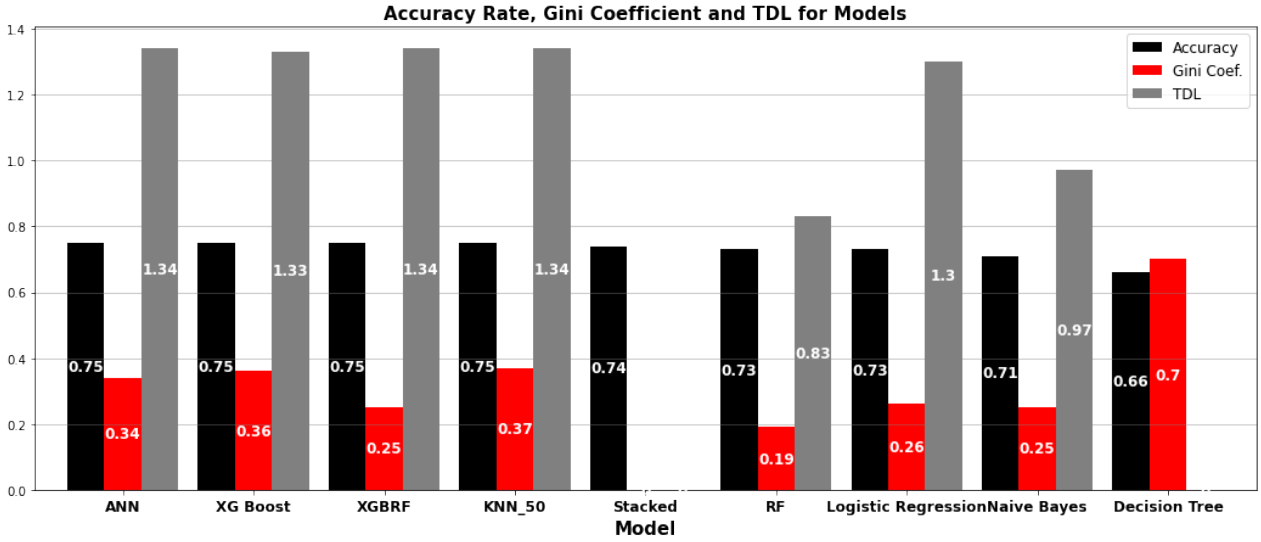
In [ ]:
```
results_df_best = results_df.iloc[[0,5,9,17,30,32,36,44,46],:]
for i in list([2,3,4,5,6,10,11]):
    results_df_best.iloc[:,i] = pd.to_numeric(results_df_best.iloc[:,i],err
results_df_best.iloc[8,1] = 'Default'
results_df_best = round(results_df_best,2)
```

In [ ]:
```
results_df_best.replace('KNN_20','KNN_50',inplace=True)
results_df_best
```

Out[ ]:

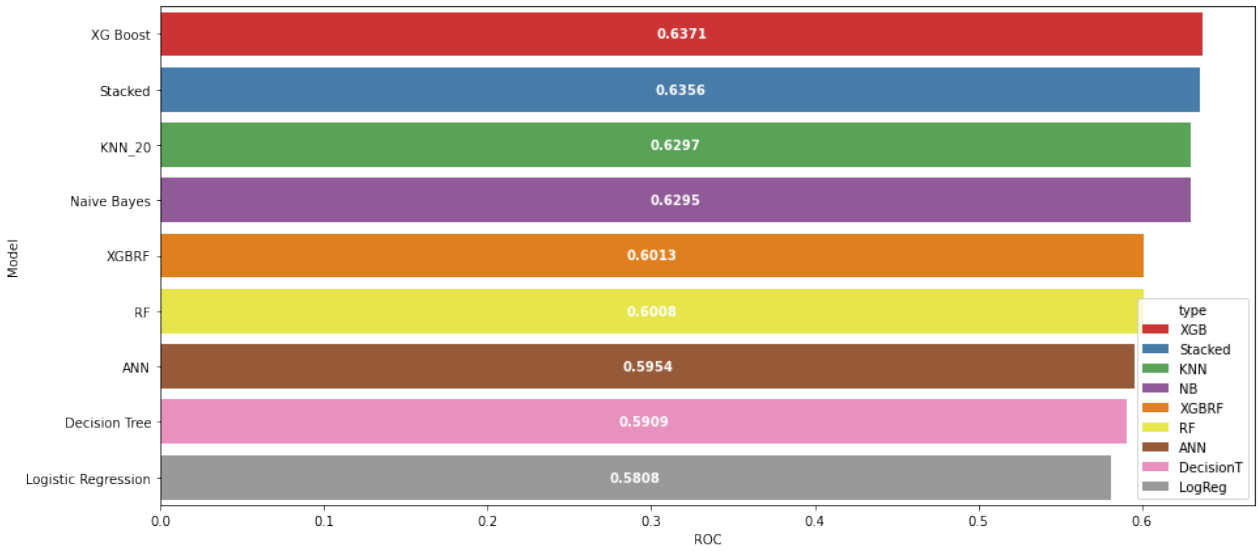
| | Model | param | Accuracy | R2 | MSE | AUC | Time | all_results | Predict_Proba |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Naive Bayes | 1 | 0.71 | -0.39 | 0.29 | 0.54 | 0.13 | [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | [0.3598, 0.2451, 0.242, 0.1736, 0.4524, 0.3278... |
| **5** | ANN | lbfgs_200 | 0.75 | -0.22 | 0.25 | 0.63 | 7396.03 | [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ... | [0.1254, 0.148, 0.1843, 0.1646, 0.206, 0.7113,... |
| **9** | RF | 200_5 | 0.73 | -0.31 | 0.27 | 0.56 | 8.77 | [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... | [0.2665, 0.1986, 0.319, 0.2073, 0.2404, 0.5313... |
| **17** | XG Boost | 6_0.3 | 0.75 | -0.20 | 0.25 | 0.63 | 7.34 | [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, ... | [0.2625, 0.1573, 0.16, 0.1288, 0.1815, 0.6168,... |
| **30** | XGBRF | 100_10 | 0.75 | -0.21 | 0.25 | 0.63 | 14.57 | [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ... | [0.2366, 0.2092, 0.2788, 0.1955, 0.2, 0.7244, ... |
| **32** | Decision Tree | entropy | 0.66 | -0.63 | 0.34 | 0.59 | 1.29 | [1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, ... | [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, ... |
| **36** | Logistic Regression | lbfgs_100 | 0.73 | -0.30 | 0.27 | 0.58 | 1.17 | [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ... | [0.2158, 0.1812, 0.2873, 0.1734, 0.2602, 0.583... |
| **44** | KNN_50 | 50 | 0.75 | NaN | NaN | 0.62 | 18.23 | [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, ... | [0.54, 0.28, 0.64, 0.68, 0.28, 0.04, 0.76, 0.1... |
| **46** | Stacked | Default | 0.74 | -0.22 | 0.26 | 0.62 | 550.05 | [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, ... | |

In [ ]:
```
ax0 = results_df_best.iloc[:,:].sort_values('Accuracy',ascending=False).plo
    x='Model',y=['Accuracy','Gini Coef.','TDL'],kind='bar',figsize=(18,7),r
    color=['black','red','grey'],width=0.85)
for container in ax0.containers:
    ax0.bar_label(container, fontsize=12,label_type='center',color='white',
ax0.set_title('Accuracy Rate, Gini Coefficient and TDL for Models',fontsize
ax0.set_xticklabels(ax0.get_xticklabels(), fontsize=12, weight='bold')
#Add Horizontal Grid
ax0.yaxis.grid(True, linestyle='-', which='major', color='grey',alpha=0.5)
ax0.set_xlabel('Model',fontsize=15,weight='bold')
plt.legend(loc='upper right',  ncol=1, fontsize=12)
plt.savefig('Gini Coef., Acc, TDL for Models_2.png',dpi=300,bbox_inches='ti
```




## Plot ROC Graph

In [ ]:
```
results_df.sort_values('ROC',ascending=False,inplace=True)
plt.figure(figsize=(15,7))
ax1 = sns.barplot(x='ROC',y='Model',data=results_df,palette='Set1',hue='typ
for container in ax1.containers:
    ax1.bar_label(container, label=container.get_label(), label_type='cente
```




## Latex

In [ ]:
```
print(results_df.iloc[:,:-2].sort_values('Accuracy',ascending=False).to_lat
```

## Gini and TDL

In [ ]:
```
results_df = pd.read_pickle('results_df_2.pkl')
results_df_copy = results_df.copy()

#results_df.reset_index(drop=True,inplace=True)
#results_df = results_df.iloc[:-1,:]
```

### Gini Func

In [ ]:
```
def gini(array):
    """Calculate the Gini coefficient of a numpy array."""
    # based on bottom eq: http://www.statsdirect.com/help/content/image/sta
    # from: http://www.statsdirect.com/help/default.htm#nonparametric_metho
    array = array.flatten() #all values are treated equally, arrays must be
    if np.amin(array) < 0:
        array -= np.amin(array) #values cannot be negative
    array += 0.0000001 #values cannot be 0
    array = np.sort(array) #values must be sorted
    index = np.arange(1,array.shape[0]+1) #index per array element
    n = array.shape[0]#number of array elements
    return ((np.sum((2 * index - n  - 1) * array)) / (n * np.sum(array))) #

results_df['Gini Coef.'] = ""
for i in range(0,len(results_df)-1):
    results_df.iloc[i,10] = round(gini(results_df.iloc[i,8]),4)
```

### TDL

In [ ]:

```
results_df_2 = pd.DataFrame(columns=['Model','Proba','DecileGroup'])

results_df_best = results_df.iloc[[0,5,9,17,30,32,36,44,46],:]

#To Create TDL
results_tdl = pd.DataFrame()
for i in range(0,len(results_df)):
    print(i)
    try:
        if results_df.loc[i,'Model'] != 'Decision Tree':
            results_df_2 = pd.DataFrame(columns=['Model','param','Proba','P
            results_df_2.iloc[:,2] = results_df.iloc[i,8]
            results_df_2.iloc[:,0] = results_df.iloc[i,0]
            results_df_2.iloc[:,1] = results_df.iloc[i,1]
            results_df_2.iloc[:,3] = results_df.iloc[i,7]
            results_df_2.iloc[:,4] = pd.qcut(results_df_2.iloc[:,2],10,labe
            results_tdl = results_tdl.append(results_df_2)
    except:
        print('Error')
        pass

results_tdl_2 = results_tdl.groupby(['Model','param','DecileGroup'])['Resul

results_tdl_3 = results_tdl_2.iloc[:,[0,1,11]].merge(results_df[['Model','p

results_tdl_3['TDL'] = results_tdl_3.iloc[:,2] / results_tdl_3.iloc[:,3]
#Merge with the results_df
results_df_2 = results_df.merge(results_tdl_3[['Model','param','TDL']],on=[
results_df_2
```

In [ ]:

```
results_df_best.rename(columns={'param':'Parameter'},inplace=True)
```

In [ ]:

```
print(results_df_best.iloc[:,[0,1,2,3,4,5,10,11]].to_latex(index=False).rep
```

```
\begin{tabular}{llrrrrrr}
\toprule
              Model &     param &  Accuracy &    R2 &  MSE &  AUC &  Gini C
oef. &  TDL \\ \hline
\midrule
        Naive Bayes &         1 &      0.71 & -0.39 & 0.29 & 0.54 &
0.25 & 0.97 \\ \hline
                ANN & lbfgs\_200 &      0.75 & -0.22 & 0.25 & 0.63 &
0.34 & 1.34 \\ \hline
                 RF &     200\_5 &      0.73 & -0.31 & 0.27 & 0.56 &
0.19 & 0.83 \\ \hline
           XG Boost &     6\_0.3 &      0.75 & -0.20 & 0.25 & 0.63 &
0.36 & 1.33 \\ \hline
              XGBRF &    100\_10 &      0.75 & -0.21 & 0.25 & 0.63 &
0.25 & 1.34 \\ \hline
      Decision Tree &   entropy &      0.66 & -0.63 & 0.34 & 0.59 &
0.70 &  NaN \\ \hline
Logistic Regression & lbfgs\_100 &      0.73 & -0.30 & 0.27 & 0.58 &
0.26 & 1.30 \\ \hline
             KNN\_50 &        50 &      0.75 & -0.20 & 0.25 & 0.62 &
0.37 & 1.34 \\ \hline
            Stacked &   Default &      0.74 & -0.22 & 0.26 & 0.62 &
NaN &  NaN \\ \hline
\bottomrule
\end{tabular}
```