Assignment 01 – Generative and Non-Generative Methods

Group 2 - Eric Lim - ml1859

Group 2 - Klass van Kempen - kjv13

Group 2 - Jude Moukarzel - jjm385

Initialization

Load the dataset. (0.5 x 2)

In [ ]:
```
import urllib.request
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00468/online_shopp
filename = 'datasets/online_shoppers_intention.csv'
urllib.request.urlretrieve(url, filename)
```

Out[ ]:
```
('datasets/online_shoppers_intention.csv',
 <http.client.HTTPMessage at 0x20a088ef040>)
```

Show first 6 data points using head(). (0.5 x 2)

In [ ]:
```
import pandas as pd
filename = 'datasets/online_shoppers_intention.csv'
OSI = pd.read_csv(filename, header=0)
OSI["Weekend"] = OSI["Weekend"].astype(int)
OSI["Revenue"] = OSI["Revenue"].astype(int)
OSI.insert(loc=16, column="VisitorTypeNumeric", value=pd.factorize(OSI['VisitorType
OSI = OSI.drop('VisitorType', axis=1)

print(OSI.head(6))
```

```
   Administrative  Administrative_Duration  Informational  \
0               0                      0.0              0   
1               0                      0.0              0   
2               0                      0.0              0   
3               0                      0.0              0   
4               0                      0.0              0   
5               0                      0.0              0   

   Informational_Duration  ProductRelated  ProductRelated_Duration  \
0                     0.0               1                 0.000000   
1                     0.0               2                64.000000   
2                     0.0               1                 0.000000   
3                     0.0               2                 2.666667   
4                     0.0              10               627.500000   
5                     0.0              19               154.216667   

   BounceRates  ExitRates  PageValues  SpecialDay Month  OperatingSystems  \
0     0.200000   0.200000         0.0         0.0   Feb                 1   
1     0.000000   0.100000         0.0         0.0   Feb                 2   
2     0.200000   0.200000         0.0         0.0   Feb                 4   
3     0.050000   0.140000         0.0         0.0   Feb                 3   
4     0.020000   0.050000         0.0         0.0   Feb                 3   
5     0.015789   0.024561         0.0         0.0   Feb                 2   

   Browser  Region  TrafficType  VisitorTypeNumeric  Weekend  Revenue  
0        1       1            1                   1        0        0  
1        2       1            2                   1        0        0  
2        1       9            3                   1        0        0  
3        2       2            4                   1        0        0  
4        3       1            4                   1        1        0  
5        2       1            3                   1        0        0  
```

Describe the Dataframe by using describe. (0.5 x 2)

In [ ]:
```
print(OSI.describe())
```

```
       Administrative  Administrative_Duration  Informational  \
count    12330.000000             12330.000000   12330.000000
mean         2.315166                80.818611       0.503569
std          3.321784               176.779107       1.270156
min          0.000000                 0.000000       0.000000
25%          0.000000                 0.000000       0.000000
50%          1.000000                 7.500000       0.000000
75%          4.000000                93.256250       0.000000
max         27.000000              3398.750000      24.000000

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
count            12330.000000    12330.000000             12330.000000
mean                34.472398       31.731468              1194.746220
std                140.749294       44.475503              1913.669288
min                  0.000000        0.000000                 0.000000
25%                  0.000000        7.000000               184.137500
50%                  0.000000       18.000000               598.936905
75%                  0.000000       38.000000              1464.157214
max               2549.375000      705.000000             63973.522230

        BounceRates     ExitRates    PageValues    SpecialDay  \
count  12330.000000  12330.000000  12330.000000  12330.000000
mean       0.022191      0.043073      5.889258      0.061427
std        0.048488      0.048597     18.568437      0.198917
min        0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.014286      0.000000      0.000000
50%        0.003112      0.025156      0.000000      0.000000
75%        0.016813      0.050000      0.000000      0.000000
max        0.200000      0.200000    361.763742      1.000000

       OperatingSystems       Browser        Region   TrafficType  \
count      12330.000000  12330.000000  12330.000000  12330.000000
mean           2.124006      2.357097      3.147364      4.069586
std            0.911325      1.717277      2.401591      4.025169
min            1.000000      1.000000      1.000000      1.000000
25%            2.000000      2.000000      1.000000      2.000000
50%            2.000000      2.000000      3.000000      2.000000
75%            3.000000      2.000000      4.000000      4.000000
max            8.000000     13.000000      9.000000     20.000000

       VisitorTypeNumeric       Weekend       Revenue
count        12330.000000  12330.000000  12330.000000
mean             1.151176      0.232603      0.154745
std              0.376989      0.422509      0.361676
min              1.000000      0.000000      0.000000
25%              1.000000      0.000000      0.000000
50%              1.000000      0.000000      0.000000
75%              1.000000      0.000000      0.000000
max              3.000000      1.000000      1.000000
```

In [ ]:

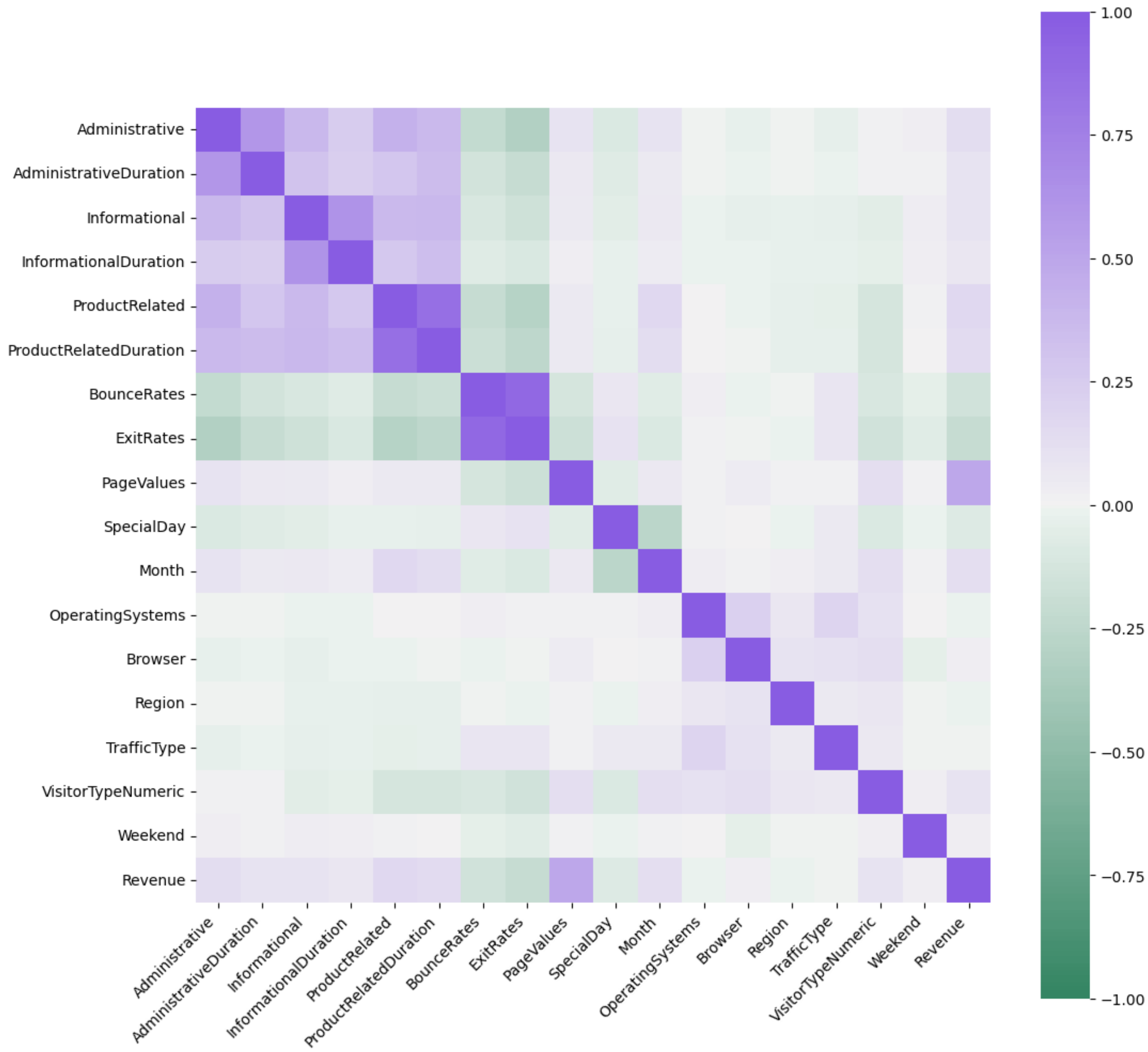
```
new_col_names = {
    'Administrative_Duration' : 'AdministrativeDuration',
    'Informational_Duration' : 'InformationalDuration',
    'ProductRelated_Duration' : 'ProductRelatedDuration'
}
```

```
OSI.rename(columns=new_col_names, inplace=True)
```

Show correlation heat plot of the entire dataset using matplotlib and sns, choose any color pallet (except blue) you like (experiment). (0.5 x 2)

In [ ]:
```
month_numeric_encoding = {
    "Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4,
    "May": 5, "Jun": 6, "Jul": 7, "Aug": 8,
    "Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12
}

OSI["Month"] = OSI["Month"].map(month_numeric_encoding)
```

In [ ]:
```
from matplotlib import rcParams
import seaborn as sns

rcParams['figure.figsize'] = 12,12
rcParams['figure.dpi'] = 100
corr = OSI.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(150, 275, s=80, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



Show the distribution plots of each variable using hist function from matplotlib. Also, experiment with visual aspects of the image (not a lot, but an excellent visual will always leave a better impression. You can change color, thickness, font, font size, font color, etc.). Explain the plot distributions as much as you can. For example, you can describe the attributes of the distributions like *"From the distribution plot of variable x we can see that the mean is xx with std dev of yy and the variable seems to be skewed towards left."* (0.5 x 2)

In [ ]:

```
import matplotlib.pyplot as plt

OSI[['Month', 'OperatingSystems', 'Browser', 'Region', 'TrafficType', 'VisitorTypeN
```

```
plt.figure(figsize=(12, 12),dpi=150)
OSI.hist(color='green', rwidth=0.8)
plt.suptitle("Online Shopping Feature Distributions\n", size=20)
plt.tight_layout()
```

```
<Figure size 1800x1800 with 0 Axes>
```

Code Snippets used in Interpretation

```
In [ ]: month_vc = OSI['Month'].value_counts()
        month_perc = (month_vc / len(OSI)) * 100
        print(month_perc)
```

```
Month
5.0     27.283049
11.0    24.314680
3.0     15.466342
12.0    14.006488
10.0     4.452555
9.0      3.633414
8.0      3.511760
7.0      3.503650
2.0      1.492295
Name: count, dtype: float64
```

In [ ]:
```
os_vc = OSI['OperatingSystems'].value_counts()
os_perc = (os_vc / len(OSI)) * 100
print(os_perc)
```

```
OperatingSystems
2    53.536091
1    20.965126
3    20.721817
4     3.876723
8     0.640714
6     0.154096
7     0.056772
5     0.048662
Name: count, dtype: float64
```

In [ ]:
```
browser_vc = OSI['Browser'].value_counts()
browser_perc = (browser_vc / len(OSI)) * 100
print(browser_perc)
```

```
Browser
2     64.566099
1     19.967559
4      5.969181
5      3.787510
6      1.411192
10     1.321979
8      1.094891
3      0.851582
13     0.494728
7      0.397405
12     0.081103
11     0.048662
9      0.008110
Name: count, dtype: float64
```

In [ ]:
```
reg_vc = OSI['Region'].value_counts()
reg_perc = (reg_vc / len(OSI)) * 100
print(reg_perc)
```

```
Region
1    38.767234
3    19.489051
4     9.586375
2     9.213301
6     6.528792
7     6.171938
9     4.144363
8     3.519870
5     2.579075
Name: count, dtype: float64
```

In [ ]:
```python
tt_vc = OSI['TrafficType'].value_counts()
tt_perc = (tt_vc / len(OSI)) * 100
print(tt_perc)
```

```
TrafficType
2     31.735604
1     19.878345
3     16.642336
4      8.669911
13     5.985401
10     3.649635
6      3.600973
8      2.781833
5      2.108678
11     2.003244
20     1.605839
9      0.340633
7      0.324412
15     0.308191
19     0.137875
14     0.105434
18     0.081103
16     0.024331
12     0.008110
17     0.008110
Name: count, dtype: float64
```

In [ ]:
```python
vtn_vc = OSI['VisitorTypeNumeric'].value_counts()
vtn_perc = (vtn_vc / len(OSI)) * 100
print(vtn_perc)
```

```
VisitorTypeNumeric
1    85.571776
2    13.738848
3     0.689376
Name: count, dtype: float64
```

In [ ]:
```python
weekend_vc = OSI['Weekend'].value_counts()
weekend_perc = (weekend_vc / len(OSI)) * 100
print(weekend_perc)
```

```
Weekend
0    76.739659
1    23.260341
Name: count, dtype: float64
```

```
In [ ]: rev_vc = OSI['Revenue'].value_counts()
        rev_perc = (rev_vc / len(OSI)) * 100
        print(rev_perc)
```

```
Revenue
0    84.525547
1    15.474453
Name: count, dtype: float64
```

NOTE: referenced Kaggle for details on variables. Also, I couldn't find documentation on what some of the categorical variables' encodings, so the description is fairly limited.

The figure above shows the distributions for each feature variable in the Online Shoppers Purchasing Intention dataset.

Continuous Features

Features that represent the number of different types of pages visited by a user (i.e. Administrative, Informational, and ProductRelated) are right-skewed, suggesting that the median and IQR may be more accurate measures of central tendencies for user behavior. These features, respectively, have medians of 1, 0, and 0 and IQRs of 4, 0, 31.

Features that represent the amount of time spent on the aforementioned three pages (i.e. AdministrativeDuration, InformationalDuration, and ProductRelatedDuration) are right-skewed. These features, respectively, have medians of 7.5, 0, and 598.94 and IQRs of 93.26, 0, and 1280.02.

Features that are metrics of "Google Analytics" (i.e. BounceRates, ExitRates, and PageValues) are right skewed. These features, respectively, have medians of 0.003, 0.025, and 0 and IQRs of 0.017, 0.036, and 0.

The SpecialDay feature represents the closeness of browsing date to some special day or holiday, and, unsurprisingly, the feature is right-skewed. SpecialDay has a median of 0 and an IQR of 1.

Categorical Features

The most frequent months for users partaking in online shopping sessions are May (27%), November (24%), March (15%), and December (14%). May is a bit strange, since the only public holiday in May is Memorial Day, which isn't necessarily a holiday associated with shopping when compared to holidays in November, March, and December, such as Thanksgiving, Mother's day, and Christmas.

The most frequent OS for users partaking in online shopping sessions is encoded 2 (54%), followed by encodings 1 and 3 (both 21%).

The most frequent Browser for users partaking in online shopping sessions is encoded 2 (65%), followed by encodings 1 (20%).

The most frequent Region for users partaking in online shopping sessions is encoded 1 (39%), followed by encodings 3 (20%).

The most frequent TrafficType for users partaking in online shopping sessions is encoded 2 (32%), followed by encodings 1 (20%) and 3 (17%).

Among the users partaking in online shopping sessions, about 86% are returning users and 14% are new visitors.

The online shopping sessions mostly occur on weekdays (77%), and most sessions (85%) do not occur in a completed purchase.

Load the dataset. (0.5 x 2)

In [ ]:
```
import pandas as pd
filename = 'datasets/Bike-Sharing-Hour.csv'
BSH = pd.read_csv(filename, header=0)
```

Show first 6 data points using head(). (0.5 x 2)

In [ ]:
```
print(BSH.head(6))
```

```
   instant      dteday  season  yr  mnth  hr  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1   0        0        6           0
1        2  2011-01-01       1   0     1   1        0        6           0
2        3  2011-01-01       1   0     1   2        0        6           0
3        4  2011-01-01       1   0     1   3        0        6           0
4        5  2011-01-01       1   0     1   4        0        6           0
5        6  2011-01-01       1   0     1   5        0        6           0

   weathersit  temp   atemp   hum  windspeed  casual  registered  cnt
0           1  0.24  0.2879  0.81     0.0000       3          13   16
1           1  0.22  0.2727  0.80     0.0000       8          32   40
2           1  0.22  0.2727  0.80     0.0000       5          27   32
3           1  0.24  0.2879  0.75     0.0000       3          10   13
4           1  0.24  0.2879  0.75     0.0000       0           1    1
5           2  0.24  0.2576  0.75     0.0896       0           1    1
```

Describe the Dataframe by using describe. (0.5 x 2)

In [ ]:
```
print(BSH.describe())
```

```
            instant        season            yr          mnth            hr  \
count  17379.0000  17379.000000  17379.000000  17379.000000  17379.000000
mean    8690.0000      2.501640      0.502561      6.537775     11.546752
std     5017.0295      1.106918      0.500008      3.438776      6.914405
min        1.0000      1.000000      0.000000      1.000000      0.000000
25%     4345.5000      2.000000      0.000000      4.000000      6.000000
50%     8690.0000      3.000000      1.000000      7.000000     12.000000
75%    13034.5000      3.000000      1.000000     10.000000     18.000000
max    17379.0000      4.000000      1.000000     12.000000     23.000000

            holiday       weekday    workingday    weathersit          temp  \
count  17379.000000  17379.000000  17379.000000  17379.000000  17379.000000
mean       0.028770      3.003683      0.682721      1.425283      0.496987
std        0.167165      2.005771      0.465431      0.639357      0.192556
min        0.000000      0.000000      0.000000      1.000000      0.020000
25%        0.000000      1.000000      0.000000      1.000000      0.340000
50%        0.000000      3.000000      1.000000      1.000000      0.500000
75%        0.000000      5.000000      1.000000      2.000000      0.660000
max        1.000000      6.000000      1.000000      4.000000      1.000000

              atemp           hum     windspeed        casual    registered  \
count  17379.000000  17379.000000  17379.000000  17379.000000  17379.000000
mean       0.475775      0.627229      0.190098     35.676218    153.786869
std        0.171850      0.192930      0.122340     49.305030    151.357286
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.333300      0.480000      0.104500      4.000000     34.000000
50%        0.484800      0.630000      0.194000     17.000000    115.000000
75%        0.621200      0.780000      0.253700     48.000000    220.000000
max        1.000000      1.000000      0.850700    367.000000    886.000000

                cnt
count  17379.000000
mean     189.463088
std      181.387599
min        1.000000
25%       40.000000
50%      142.000000
75%      281.000000
max      977.000000
```

Show correlation heat plot of the entire dataset using matplotlib and sns, choose any color pallet (except blue) you like (experiment). (0.5 x 2)

In [ ]:

```
BSH.drop(columns=['dteday'], inplace=True)
```

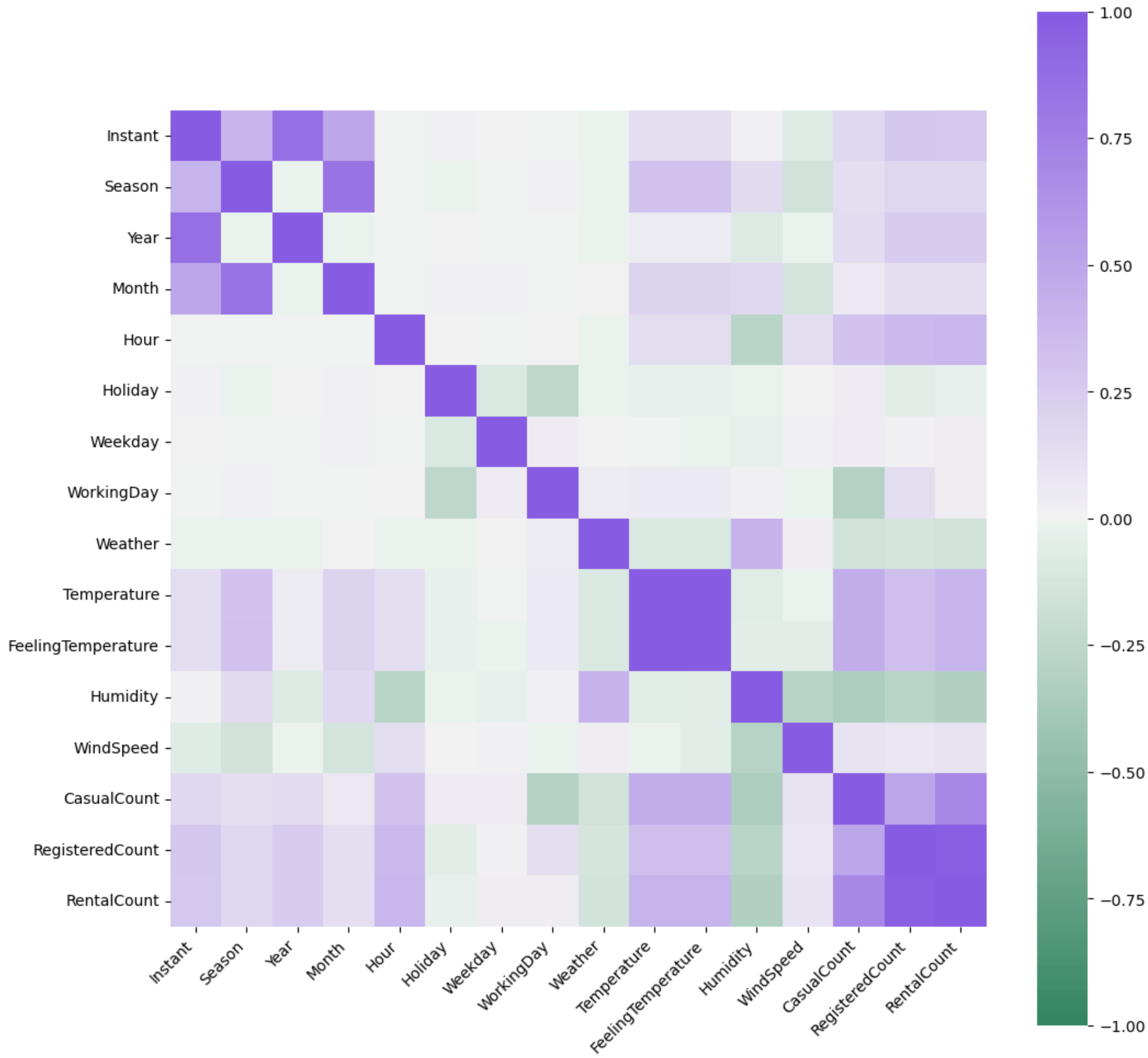
In [ ]:

```
BSH.rename(columns={
    'instant' : 'Instant',
    'season' : 'Season',
    'yr' : 'Year',
    'mnth' : 'Month',
    'hr' : 'Hour',
    'holiday' : 'Holiday',
```

```
    'weekday' : 'Weekday',
    'workingday' : 'WorkingDay',
    'weathersit' : 'Weather',
    'temp' : 'Temperature',
    'atemp' : 'FeelingTemperature',
    'hum' : 'Humidity',
    'windspeed' : 'WindSpeed',
    'casual' : 'CasualCount',
    'registered' : 'RegisteredCount',
    'cnt' : 'RentalCount'
}, inplace=True)
```

In [ ]:

```
from matplotlib import rcParams
import seaborn as sns

rcParams['figure.figsize'] = 12,12
rcParams['figure.dpi'] = 100
corr = BSH.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(150, 275, s=80, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



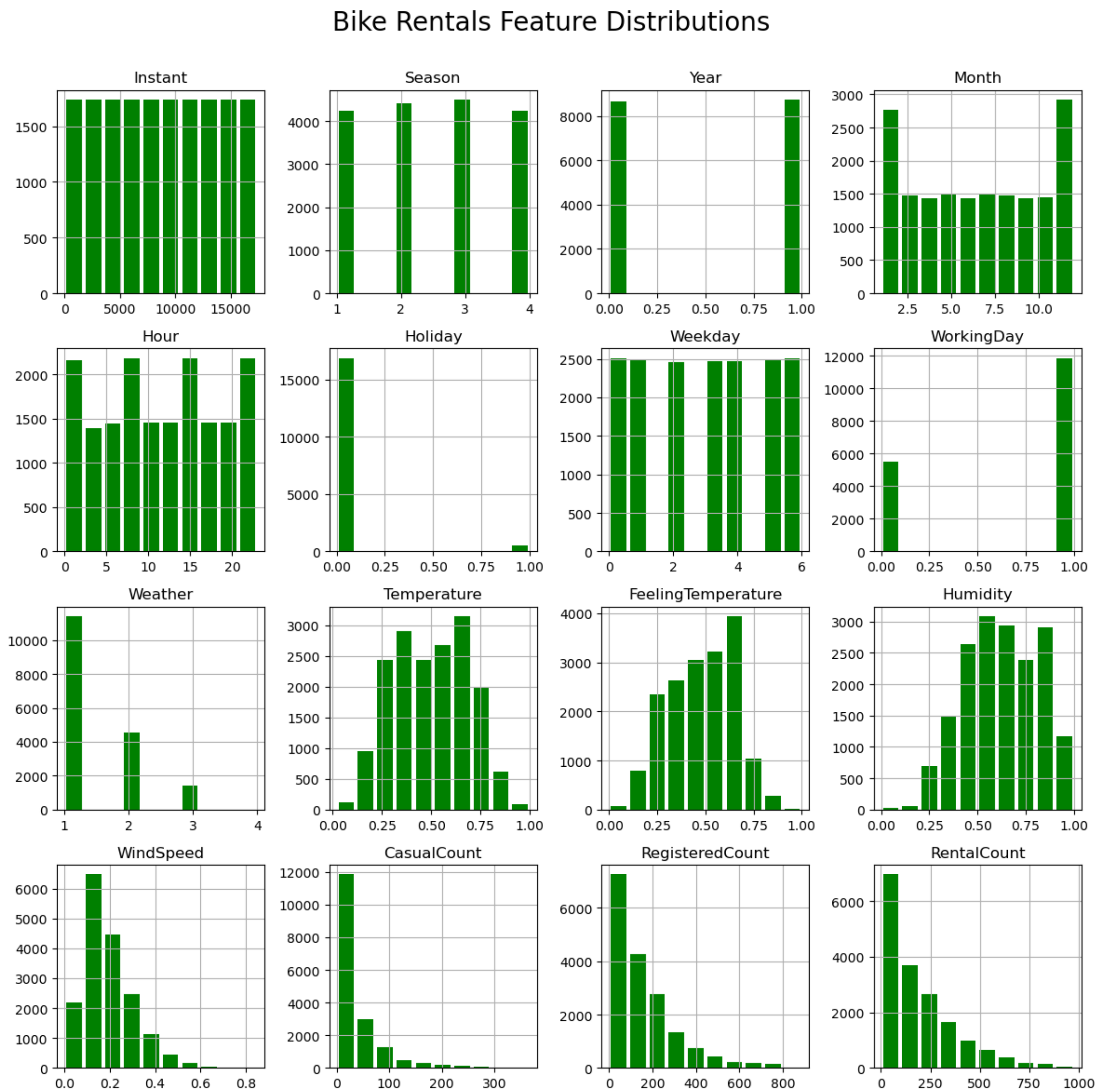
Show the distribution plots of each variable using hist function from matplotlib. Also, experiment with visual aspects of the image (not a lot, but an excellent visual will always leave a better impression. You can change color, thickness, font, font size, font color, etc.). Explain the plot distributions as much as you can. For example, you can describe the attributes of the distributions like *"From the distribution plot of variable x we can see that the mean is xx with std dev of yy and the variable seems to be skewed towards left."* (0.5 x 2)

In [ ]:

```
import matplotlib.pyplot as plt

BSH[['Instant', 'Season', 'Year',
     'Month', 'Hour', 'Holiday',
```

```
    'Weekday', 'WorkingDay', 'Weather']].astype(object)

plt.figure(figsize=(12, 12),dpi=150)
BSH.hist(color='green', rwidth=0.8)
plt.suptitle("Bike Rentals Feature Distributions\n", size=20)
plt.tight_layout()
```

```
<Figure size 1800x1800 with 0 Axes>
```



Code Snippets used in Interpretation

In [ ]:
```
season_vc = BSH['Season'].value_counts()
season_perc = (season_vc / len(BSH)) * 100
print(season_perc)
```

```
Season
3    25.870303
2    25.369699
1    24.408769
4    24.351228
Name: count, dtype: float64
```

In [ ]:
```python
year_vc = BSH['Year'].value_counts()
year_perc = (year_vc / len(BSH)) * 100
print(year_perc)
```

```
Year
1    50.256056
0    49.743944
Name: count, dtype: float64
```

In [ ]:
```python
month_vc = BSH['Month'].value_counts()
month_perc = (month_vc / len(BSH)) * 100
print(month_perc)
```

```
Month
5     8.562058
7     8.562058
12    8.533287
8     8.487255
3     8.475747
10    8.349157
6     8.285862
4     8.268600
9     8.268600
11    8.268600
1     8.222567
2     7.716209
Name: count, dtype: float64
```

In [ ]:
```python
hour_vc = BSH['Hour'].value_counts()
hour_perc = (hour_vc / len(BSH)) * 100
print(hour_perc)
```

```
Hour
17    4.200472
16    4.200472
13    4.194718
15    4.194718
14    4.194718
12    4.188964
22    4.188964
21    4.188964
20    4.188964
19    4.188964
18    4.188964
23    4.188964
11    4.183210
10    4.183210
9     4.183210
8     4.183210
7     4.183210
0     4.177456
6     4.171701
1     4.165947
5     4.125669
2     4.114161
4     4.010587
3     4.010587
Name: count, dtype: float64
```

In [ ]:

```
holiday_vc = BSH['Holiday'].value_counts()
holiday_perc = (holiday_vc / len(BSH)) * 100
print(holiday_perc)
```

```
Holiday
0    97.122964
1     2.877036
Name: count, dtype: float64
```

In [ ]:

```
weekday_vc = BSH['Weekday'].value_counts()
weekday_perc = (weekday_vc / len(BSH)) * 100
print(weekday_perc)
```

```
Weekday
6    14.454226
0    14.396686
5    14.310375
1    14.264342
3    14.241326
4    14.218309
2    14.114736
Name: count, dtype: float64
```

In [ ]:

```
wd_vc = BSH['WorkingDay'].value_counts()
wd_perc = (wd_vc / len(BSH)) * 100
print(wd_perc)
```

```
WorkingDay
1    68.272052
0    31.727948
Name: count, dtype: float64
```

In [ ]:
```
weather_vc = BSH['Weather'].value_counts()
weather_perc = (weather_vc / len(BSH)) * 100
print(weather_perc)
```

```
Weather
1    65.671212
2    26.146499
3     8.165027
4     0.017262
Name: count, dtype: float64
```

NOTE: referenced Kaggle for details on variables. Also, I skipped interpreting the Instant feature, since that just keeps track of the record index and is uninformative.

Continuous Features

Features that describe the temperature (i.e. Temperature and FeelingTemperature) are approximately normal, suggesting that the mean and standard deviation would be reasonable measures of central tendencies of the weather. These features, respectively, have means of 0.50 and 0.48 and standard deviations of 0.19 and 0.17. It should be noted that these features are min-max normalized, and the temperature measures are in Celsius.

Other measures of climate (i.e. Humidity and WindSpeed) have a slight left and right skew, respectively, suggesting that the median and IQR may be more accurate measures of central tendencies of the climate. These features, respectively, have medians of 0.63 and 0.19 and IQRs of 0.30 and 0.15. It should be noted that these features have been normalized by dividing by their raw max values.

Features that measure the number of bike rentals (i.e. CasualCount, RegisteredCount, and RentalCount) are right-skewed, suggesting that the median and IQR may be more accurate measures of central tendencies of bike rentals. These features, respectively, have medians of 17, 115, and 142 and IQRs of 44, 186, and 241.

Categorical Features

In both 2011 and 2012, bike rentals occurred relatively uniformly across seasons, months, and time of day (hour). It should be noted that the distributions are misleading for Month and Hour, most likely because of the binning.

Most bike rentals (97%) do not occur on holidays, which may point to most of these rentals being used for transportation to work.

While there's a uniform distribution of bike rentals across all days of the week, a majority of rentals (68%) occur on working days, which aligns with the insight from the Holiday feature.

Bike rentals occur most frequently when the weather is mostly clear (66%), followed by misty weather (26%) and light snow (8%).

Intermediate Steps (Essential, no points granted)

In [ ]:
```
rev_vc = OSI['Revenue'].value_counts()
rev_perc = (rev_vc / len(OSI)) * 100
print(rev_perc)
```

```
Revenue
0    84.525547
1    15.474453
Name: count, dtype: float64
```

In [ ]:
```
print(OSI.isna().sum())
```

```
Administrative               0
AdministrativeDuration       0
Informational                0
InformationalDuration        0
ProductRelated               0
ProductRelatedDuration       0
BounceRates                  0
ExitRates                    0
PageValues                   0
SpecialDay                   0
Month                      288
OperatingSystems             0
Browser                      0
Region                       0
TrafficType                  0
VisitorTypeNumeric           0
Weekend                      0
Revenue                      0
dtype: int64
```

Since Month is categorical, we'll replace the missing values with the mode of the column.

In [ ]:
```
from sklearn.impute import SimpleImputer
import numpy as np

imputer = SimpleImputer(strategy='most_frequent')
OSI_imputed = pd.DataFrame(imputer.fit_transform(OSI), columns=OSI.columns)
```

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, Baggin
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
```

```
X_OSI = OSI_imputed.loc[:, OSI.columns != 'Revenue'].to_numpy()
y_OSI = OSI_imputed.iloc[:,-1:].to_numpy()

print(X_OSI)
print(y_OSI)

OSIX_train, OSIX_test, OSIy_train, OSIy_test = train_test_split(X_OSI, y_OSI, test_
```

```
[[ 0.   0.   0.  ...  1.   1.   0.]
 [ 0.   0.   0.  ...  2.   1.   0.]
 [ 0.   0.   0.  ...  3.   1.   0.]
 ...
 [ 0.   0.   0.  ... 13.   1.   1.]
 [ 4.  75.   0.  ... 11.   1.   0.]
 [ 0.   0.   0.  ...  2.   2.   1.]]
[[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
```

In [ ]:

```
print(BSH.isna().sum())
```

```
Instant               0
Season                0
Year                  0
Month                 0
Hour                  0
Holiday               0
Weekday               0
WorkingDay            0
Weather               0
Temperature           0
FeelingTemperature    0
Humidity              0
WindSpeed             0
CasualCount           0
RegisteredCount       0
RentalCount           0
dtype: int64
```

In [ ]:

```
X_BSH = BSH.loc[:, BSH.columns != 'RentalCount'].to_numpy()
y_BSH = BSH.iloc[:,-1:].to_numpy()

print(X_BSH)
print(y_BSH)

BSHX_train, BSHX_test, BSHy_train, BSHy_test = train_test_split(X_BSH, y_BSH, test_
```

```
[[1.0000e+00 1.0000e+00 0.0000e+00 ... 0.0000e+00 3.0000e+00 1.3000e+01]
 [2.0000e+00 1.0000e+00 0.0000e+00 ... 0.0000e+00 8.0000e+00 3.2000e+01]
 [3.0000e+00 1.0000e+00 0.0000e+00 ... 0.0000e+00 5.0000e+00 2.7000e+01]
 ...
 [1.7377e+04 1.0000e+00 1.0000e+00 ... 1.6420e-01 7.0000e+00 8.3000e+01]
 [1.7378e+04 1.0000e+00 1.0000e+00 ... 1.3430e-01 1.3000e+01 4.8000e+01]
 [1.7379e+04 1.0000e+00 1.0000e+00 ... 1.3430e-01 1.2000e+01 3.7000e+01]]
[[16]
 [40]
 [32]
 ...
 [90]
 [61]
 [49]]
```

Classification (total 48)

AdaBoost

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import AdaBoostClassifier from scikit-learn.

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
```

Create the appropriate classifier, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `clf` that represents an instance of the AdaBoostClassifier model. `n_estimators` represents the number of base estimators (which, by default are decision trees of depth=1) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```
clf = AdaBoostClassifier(n_estimators=100, random_state=1859)
```

Train classifier on train data and explain what you did. (1.5)

Train the previous variable `clf` on variables `OSIX_train` and `OSIy_train` and create a variable `ada_fit` that represents the fitted AdaBoostClassifier model.

In [ ]:
```
ada_fit = clf.fit(OSIX_train, OSIy_train)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\utils\validation.py:
1141: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Test/fit classifier test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `OSIX_test` and store the predicted target values in variable `OSIy_pred`.

```
In [ ]: OSIy_pred = ada_fit.predict(OSIX_test)
```

Calculate accuracy and explain what you did. (1.5)

Generate a classification report by comparing the actual target values in `OSIy_test` to the fitted model's predicted values `OSIy_pred`.

The classification report shows a reasonable 88% accuracy in predicting whether or not an online shopper's session results in a purchase. However, considering that the target variable is heavily skewed (around an 85-15 split of Revenue=0 vs Revenue=1), the model is only performing 3% better than randomly taking guesses in predicting the target variable. The impact of the imbalanced dataset also shows in the precision, recall, and f1-scores when Revenue=0 vs Revenue=1, with Revenue=0 having much better measures than Revenue=1.

Since the data is imbalanced, it's also worth looking at the ROC AUC score to see a more holistic view of the model. The result is 0.75, which shows that the model's ability to discriminate between the target classes is reasonable.

```
In [ ]: print(classification_report(OSIy_test, OSIy_pred))
        print(roc_auc_score(OSIy_test, OSIy_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.95      0.93      3097
         1.0       0.67      0.56      0.61       602

    accuracy                           0.88      3699
   macro avg       0.80      0.75      0.77      3699
weighted avg       0.88      0.88      0.88      3699

0.7549227791979591
```

Show both text and visual confusion matrices using scikit-learn and matplotlib, explain what the graph tells you, and what you did. (2.5)

Import ConfusionMatrixDisplay from scikit-learn and create a confusion matrix using the actual target values `OSIy_test` and the predicted target values `OSIy_pred`.

Import set_palette from seaborn and create a visual confusion matrix using the fitted model `ada_fit`, the test feature set `OSIX_test`, and the actual target values `OSIy_test`

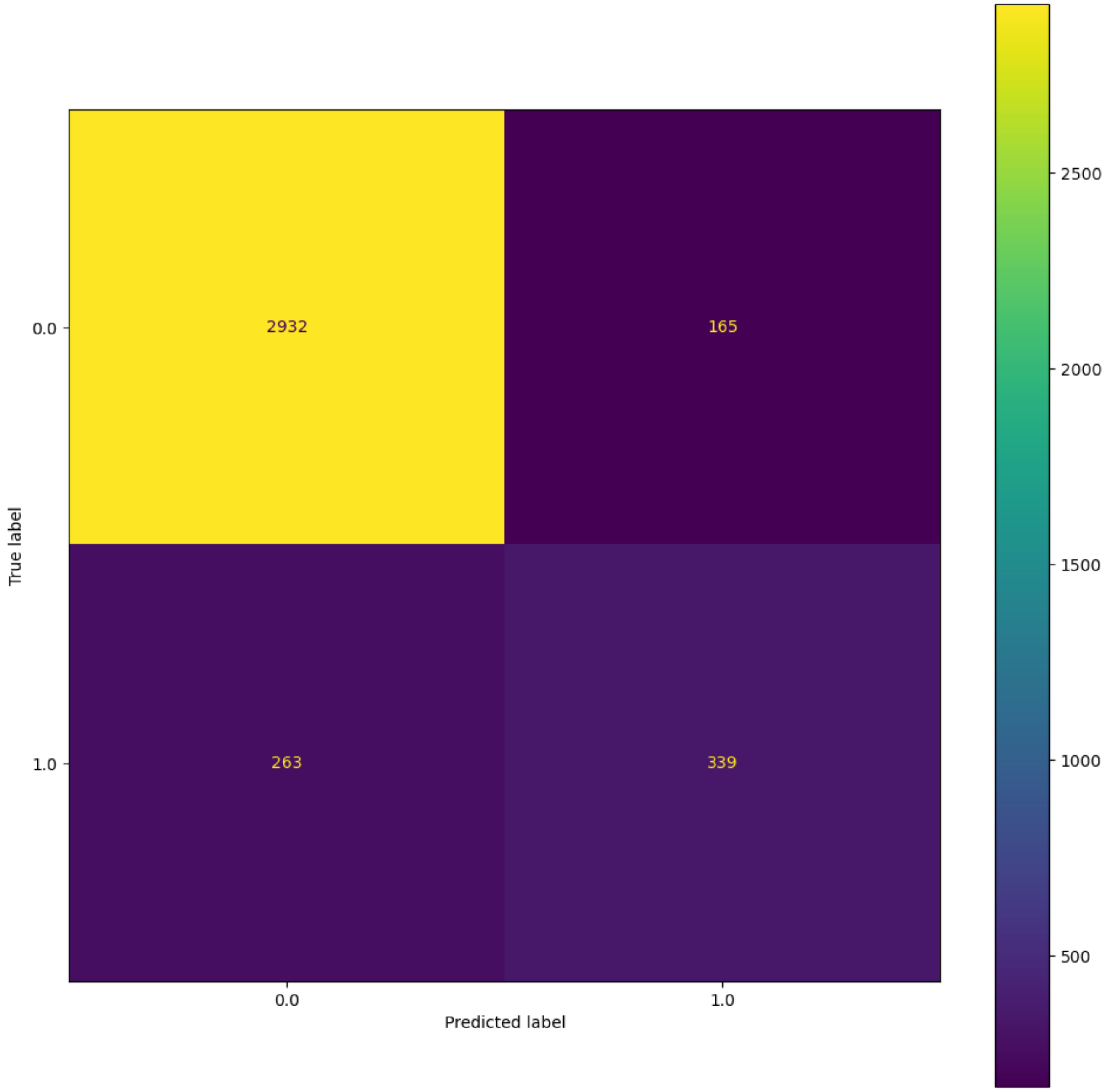
Both confusion matrices show that the model performs well in classifying True Negatives (i.e. predicting that a purchase was not made when it actually was not made). However, the model has mediocre performance in classifying True Positives (i.e. predicting that a purchase was made when it actually was made), since there's still a reasonable amount of False Negatives (i.e. predicting that a purchase was not made when it actually was made). This is most likely a result of the dataset having an imbalanced target set.

In [ ]:
```python
from sklearn.metrics import ConfusionMatrixDisplay
from seaborn import set_palette

conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(ada_fit, OSIX_test, OSIy_test)
```

```
[[2932  165]
 [ 263  339]]
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0dde1d50>
```

```
<Figure size 187.5x187.5 with 0 Axes>
```



Repeat the same with a different parameter set and compare the result (2)

Create a new AdaBoostClassifier model with more base estimators (`n_estimators=1000`) and a slower learning rate (`learning_rate=0.1`).

This new model has 10 times the number of base estimators and one-tenth the learning rate of the previous model. This new model does have a 1% higher accuracy, marginally better ROC AUC score, and the confusion matrix shows that it makes marginally better predictions than the previous model. However, these improvements in performance are negligible. Much like the first model, this model still suffers from the dataset's imbalance.

In [ ]:
```
clf = AdaBoostClassifier(n_estimators=1000, learning_rate=0.1, random_state=1859)
ada_fit = clf.fit(OSIX_train, OSIy_train)
OSIy_pred = ada_fit.predict(OSIX_test)
```

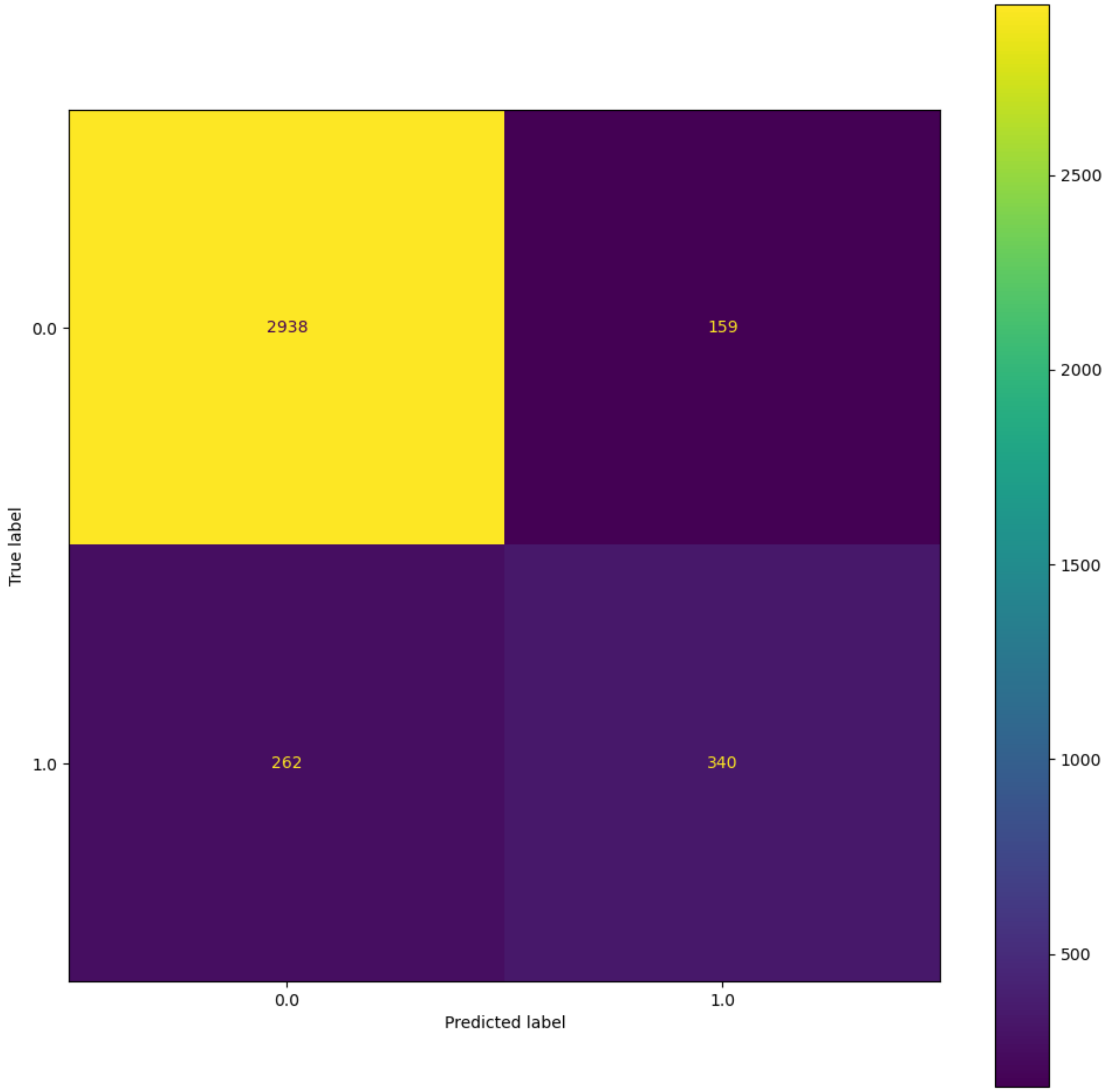
```python
print(classification_report(OSIy_test, OSIy_pred))
print(roc_auc_score(OSIy_test, OSIy_pred))

conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(ada_fit, OSIX_test, OSIy_test)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\utils\validation.py:
1141: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.95      0.93      3097
         1.0       0.68      0.56      0.62       602

    accuracy                           0.89      3699
   macro avg       0.80      0.76      0.78      3699
weighted avg       0.88      0.89      0.88      3699

0.7567220233491418
[[2938  159]
 [ 262  340]]
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0e18b580>

<Figure size 187.5x187.5 with 0 Axes>



Gradient Boost

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import GradientBoostingClassifier from scikit-learn.

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

Create the appropriate classifier, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `clf` that represents an instance of the GradientBoostingClassifier model. `n_estimators` represents the number of base estimators (which, by default are decision trees of depth=3) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```
clf = GradientBoostingClassifier(n_estimators=100, random_state=1859)
```

Train classifier on train data and explain what you did. (1.5)

Train the previous variable `clf` on variables `OSIX_train` and `OSIy_train` and create a variable `gb_fit` that represents the fitted GradientBoostingClassifier model.

In [ ]:
```
gb_fit = clf.fit(OSIX_train, OSIy_train)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_gb.py:437:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Test/fit classifier test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `OSIX_test` and store the predicted target values in variable `OSIy_pred`.

In [ ]:
```
OSIy_pred = gb_fit.predict(OSIX_test)
```

Calculate accuracy and explain what you did. (1.5)

Generate a classification report by comparing the actual target values in `OSIy_test` to the fitted model's predicted values `OSIy_pred`.

The classification report shows a reasonable 90% accuracy in predicting whether or not an online shopper's session results in a purchase. However, considering that the target variable is heavily skewed (around an 85-15 split of Revenue=0 vs Revenue=1), the model is only performing 5% better than randomly taking guesses in predicting the target variable. The impact of the imbalanced dataset also shows in the precision, recall, and f1-scores when Revenue=0 vs Revenue=1, with Revenue=0 having much better measures than Revenue=1.

Since the data is imbalanced, it's also worth looking at the ROC AUC score to see a more holistic view of the model. The result is 0.77, which shows that the model's ability to discriminate between the target classes is reasonable.

In [ ]:
```
print(classification_report(OSIy_test, OSIy_pred))
print(roc_auc_score(OSIy_test, OSIy_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94      3097
         1.0       0.73      0.59      0.65       602

    accuracy                           0.90      3699
   macro avg       0.82      0.77      0.80      3699
weighted avg       0.89      0.90      0.89      3699

0.7747163421465635
```

Show both text and visual confusion matrices using scikit-learn and matplotlib, explain what the graph tells you, and what you did. (2.5)

Both confusion matrices show that the model performs well in classifying True Negatives (i.e. predicting that a purchase was not made when it actually was not made). However, the model has mediocre performance in classifying True Positives (i.e. predicting that a purchase was made when it actually was made), since there's still a reasonable amount of False Negatives (i.e. predicting that a purchase was not made when it actually was made). This is most likely a result of the dataset having an imbalanced target set.

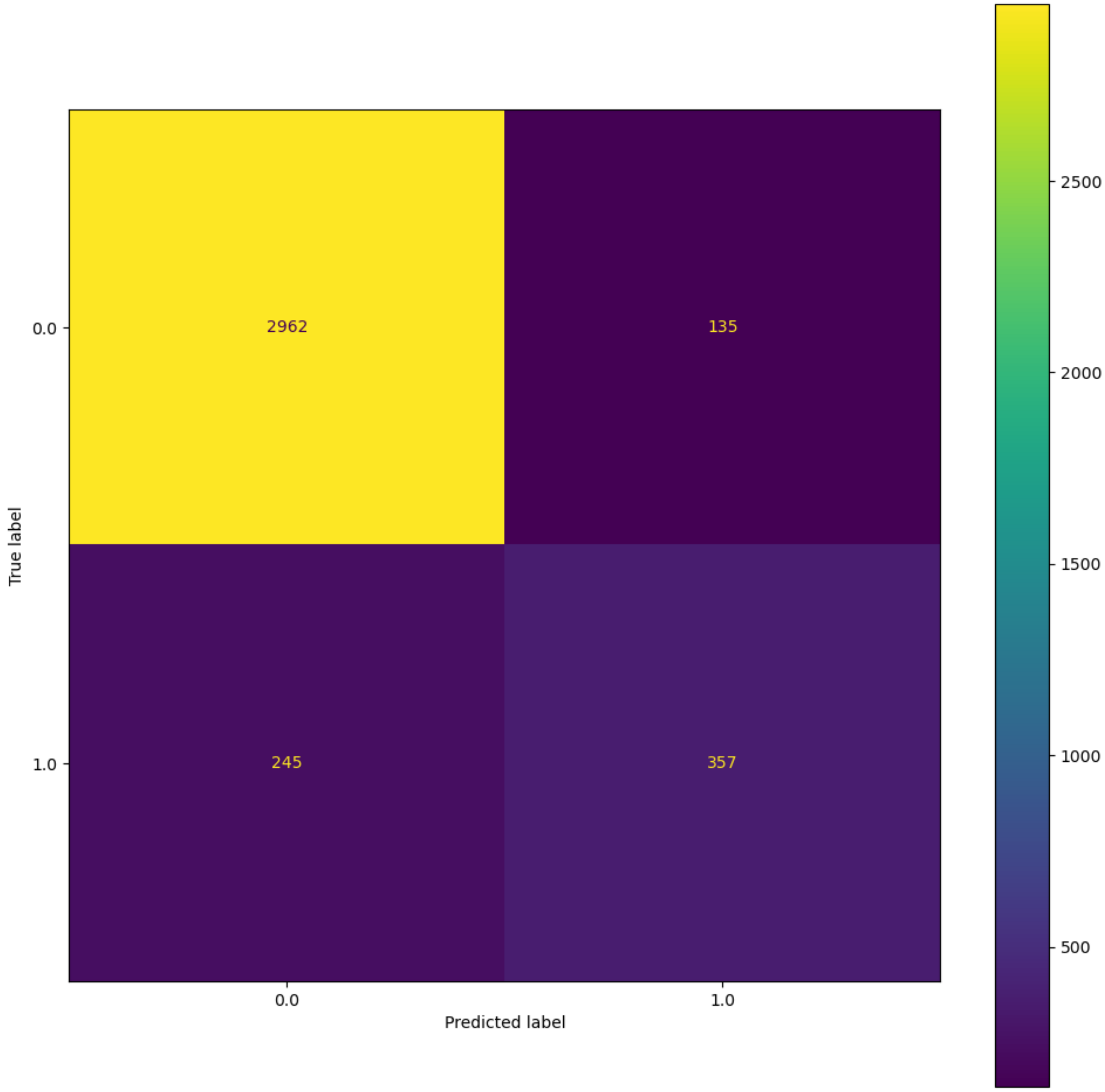
In [ ]:

```
conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(gb_fit, OSIX_test, OSIy_test)
```

```
[[2962  135]
 [ 245  357]]
```

Out[ ]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0e188910>
<Figure size 187.5x187.5 with 0 Axes>
```



Repeat the same with a different parameter set and compare the result (2)

Create a new GradientBoostingClassifier model with more base estimators (`n_estimators=1000`) and a faster learning rate (`learning_rate=1`).

This new model has 10 times the number of base estimators and the learning rate of the previous model. This new model saw a 1% reduction in accuracy, a 2% reduction in ROC AUC score, and the confusion matrix shows that it makes marginally worse predictions than the previous model. However, this decline in performance is negligible. Much like the first model, this model still suffers from the dataset's imbalance.

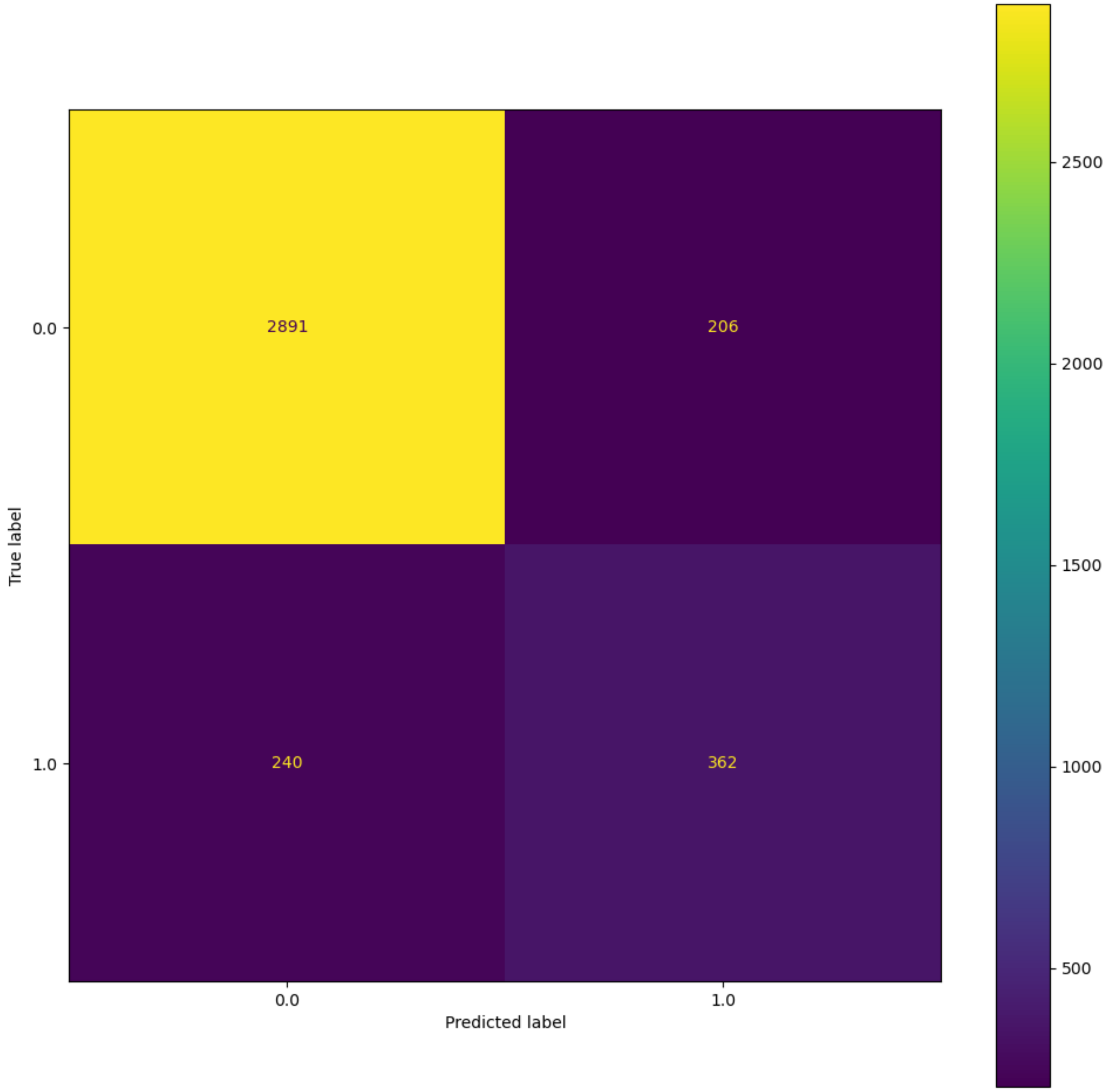
In [ ]:
```
clf = GradientBoostingClassifier(n_estimators=1000, learning_rate=1, random_state=1
gb_fit = clf.fit(OSIX_train, OSIy_train)
OSIy_pred = ada_fit.predict(OSIX_test)
```

```
print(classification_report(OSIy_test, OSIy_pred))
print(roc_auc_score(OSIy_test, OSIy_pred))

conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(gb_fit, OSIX_test, OSIy_test)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_gb.py:437:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
              precision    recall  f1-score   support

         0.0       0.92      0.95      0.93      3097
         1.0       0.68      0.56      0.62       602

    accuracy                           0.89      3699
   macro avg       0.80      0.76      0.78      3699
weighted avg       0.88      0.89      0.88      3699

0.7567220233491418
[[2938  159]
 [ 262  340]]
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0e075270>
```

```
<Figure size 187.5x187.5 with 0 Axes>
```



XG Boost

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import xgboost and XGBClassifier.

In [ ]:
```
import xgboost as xgb
from xgboost import XGBClassifier
```

Create the appropriate classifier, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `clf` that represents an instance of the XGBClassifier model.

`n_estimators` represents the number of base estimators (which, by default are decision trees of depth=3) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```
clf = XGBClassifier(n_estimators=100, random_state=1859)
```

Train classifier on train data and explain what you did. (1.5)

Train the previous variable `clf` on variables `OSIX_train` and `OSIy_train` and create a variable `xgb_fit` that represents the fitted XGBClassifier model.

In [ ]:
```
xgb_fit = clf.fit(OSIX_train, OSIy_train)
```

Test/fit classifier test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `OSIX_test` and store the predicted target values in variable `OSIy_pred`.

In [ ]:
```
OSIy_pred = xgb_fit.predict(OSIX_test)
```

Calculate accuracy and explain what you did. (1.5)

Generate a classification report by comparing the actual target values in `OSIy_test` to the fitted model's predicted values `OSIy_pred`.

The classification report shows a reasonable 89% accuracy in predicting whether or not an online shopper's session results in a purchase. However, considering that the target variable is heavily skewed (around an 85-15 split of Revenue=0 vs Revenue=1), the model is only performing 4% better than randomly taking guesses in predicting the target variable. The impact of the imbalanced dataset also shows in the precision, recall, and f1-scores when Revenue=0 vs Revenue=1, with Revenue=0 having much better measures than Revenue=1.

Since the data is imbalanced, it's also worth looking at the ROC AUC score to see a more holistic view of the model. The result is 0.77, which shows that the model's ability to discriminate between the target classes is reasonable.

In [ ]:
```
print(classification_report(OSIy_test, OSIy_pred))
print(roc_auc_score(OSIy_test, OSIy_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94      3097
         1.0       0.72      0.58      0.64       602

    accuracy                           0.89      3699
   macro avg       0.82      0.77      0.79      3699
weighted avg       0.89      0.89      0.89      3699

0.767241259090085
```

Show both text and visual confusion matrices using scikit-learn and matplotlib, explain what the graph tells you, and what you did. (2.5)

Both confusion matrices show that the model performs well in classifying True Negatives (i.e. predicting that a purchase was not made when it actually was not made). However, the model has mediocre performance in classifying True Positives (i.e. predicting that a purchase was made when it actually was made), since there's still a reasonable amount of False Negatives (i.e. predicting that a purchase was not made when it actually was made). This is most likely a result of the dataset having an imbalanced target set.

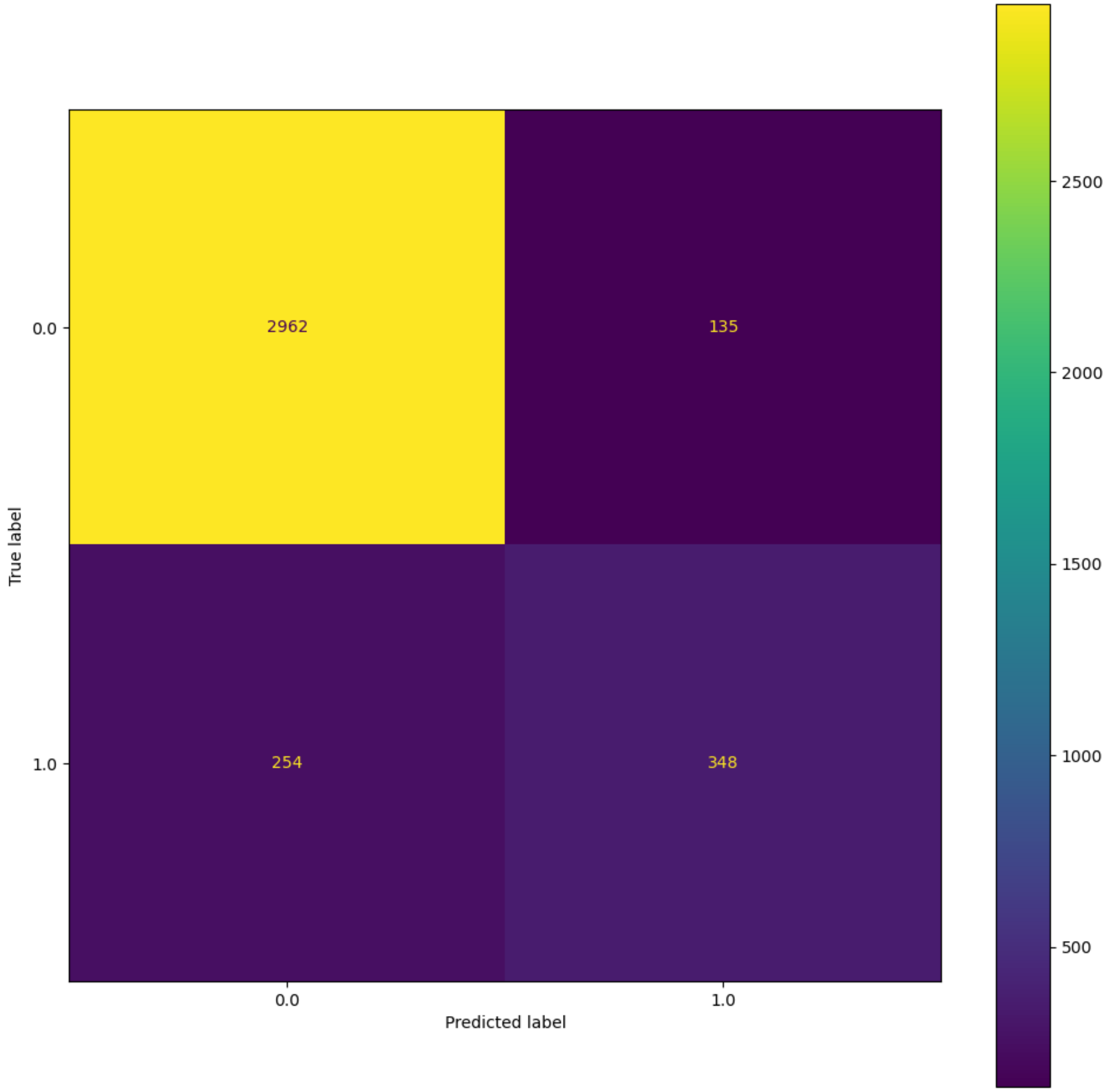
In [ ]:

```python
conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(xgb_fit, OSIX_test, OSIy_test)
```

```
[[2962  135]
 [ 254  348]]
```

Out[ ]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0e077790>

<Figure size 187.5x187.5 with 0 Axes>
```



Repeat the same with a different parameter set and compare the result (2)

Create a new XGBClassifier model with a slower learning rate (`learning_rate=0.03`).

This new model has one-tenth the learning rate of the previous model. This new model saw a 1% increase in accuracy, marginally better ROC AUC score, and the confusion matrix shows that it makes marginally better predictions than the previous model. However, this improvement in performance is negligible. Much like the first model, this model still suffers from the dataset's imbalance.

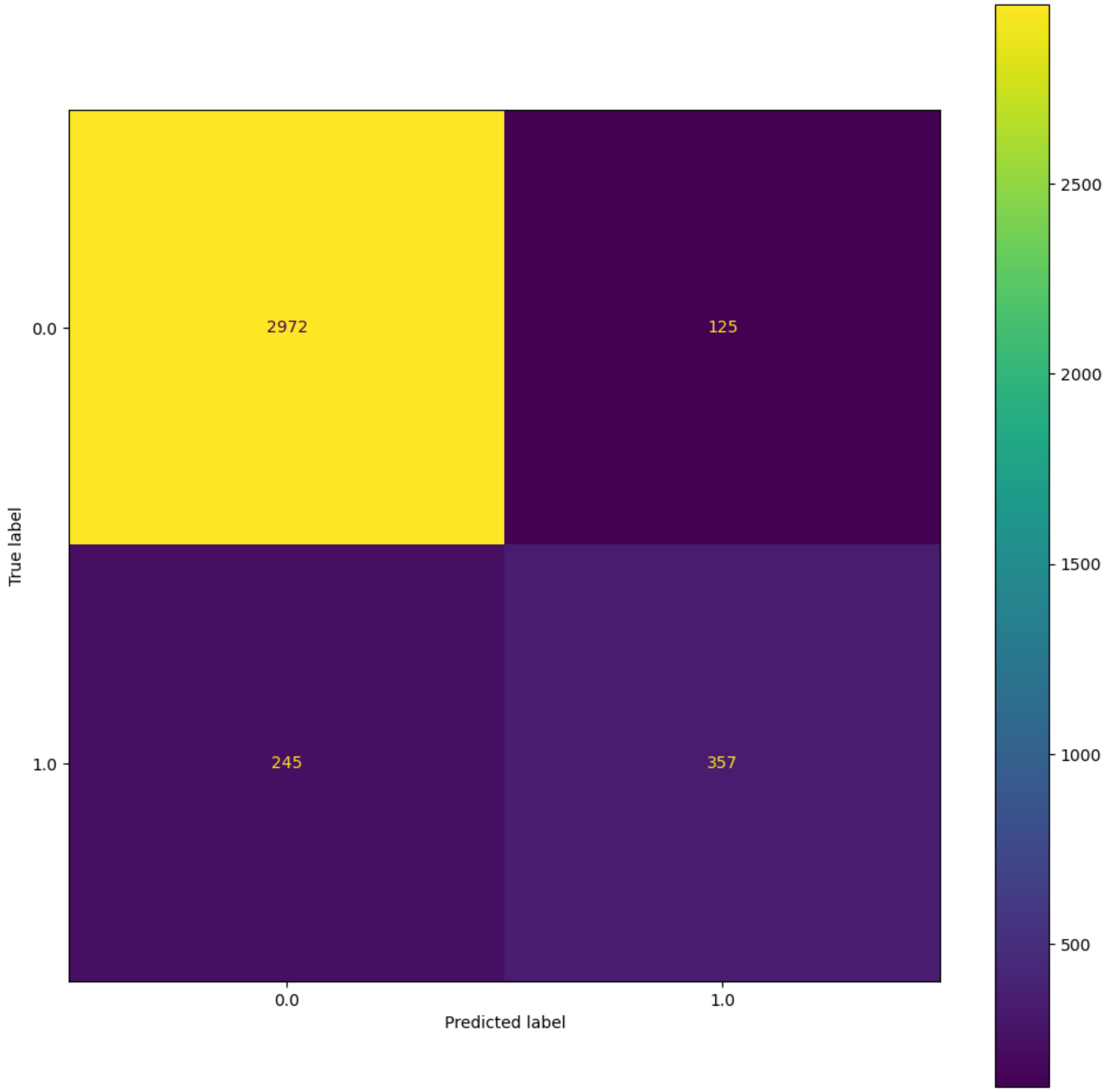
In [ ]:
```
clf = XGBClassifier(n_estimators=100, learning_rate=0.03, random_state=1859)
xgb_fit = clf.fit(OSIX_train, OSIy_train)
OSIy_pred = xgb_fit.predict(OSIX_test)

print(classification_report(OSIy_test, OSIy_pred))
```

```python
print(roc_auc_score(OSIy_test, OSIy_pred))

conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(xgb_fit, OSIX_test, OSIy_test)
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94      3097
         1.0       0.74      0.59      0.66       602

    accuracy                           0.90      3699
   macro avg       0.83      0.78      0.80      3699
weighted avg       0.89      0.90      0.90      3699

0.776330807758446
[[2972  125]
 [ 245  357]]
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0f188f40>

<Figure size 187.5x187.5 with 0 Axes>



Bagging

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import BaggingClassifier from scikit-learn.

In [ ]:
```
from sklearn.ensemble import BaggingClassifier
```

Create the appropriate classifier, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `clf` that represents an instance of the BaggingClassifier model.

`n_estimators` represents the number of base estimators (which, by default are decision trees) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

```
In [ ]: clf = BaggingClassifier(n_estimators=10, random_state=1859)
```

Train classifier on train data and explain what you did. (1.5)

Train the previous variable `clf` on variables `OSIX_train` and `OSIy_train` and create a variable `bag_fit` that represents the fitted BaggingClassifier model.

```
In [ ]: bag_fit = clf.fit(OSIX_train, OSIy_train)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:804: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Test/fit classifier test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `OSIX_test` and store the predicted target values in variable `OSIy_pred`.

```
In [ ]: OSIy_pred = bag_fit.predict(OSIX_test)
```

Calculate accuracy and explain what you did. (1.5)

Generate a classification report by comparing the actual target values in `OSIy_test` to the fitted model's predicted values `OSIy_pred`.

The classification report shows a reasonable 89% accuracy in predicting whether or not an online shopper's session results in a purchase. However, considering that the target variable is heavily skewed (around an 85-15 split of Revenue=0 vs Revenue=1), the model is only performing 4% better than randomly taking guesses in predicting the target variable. The impact of the imbalanced dataset also shows in the precision, recall, and f1-scores when Revenue=0 vs Revenue=1, with Revenue=0 having much better measures than Revenue=1.

Since the data is imbalanced, it's also worth looking at the ROC AUC score to see a more holistic view of the model. The result is 0.75, which shows that the model's ability to discriminate between the target classes is reasonable.

```
In [ ]: print(classification_report(OSIy_test, OSIy_pred))
        print(roc_auc_score(OSIy_test, OSIy_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.96      0.94      3097
         1.0       0.72      0.54      0.62       602

    accuracy                           0.89      3699
   macro avg       0.82      0.75      0.78      3699
weighted avg       0.88      0.89      0.88      3699

0.7489221698846917
```

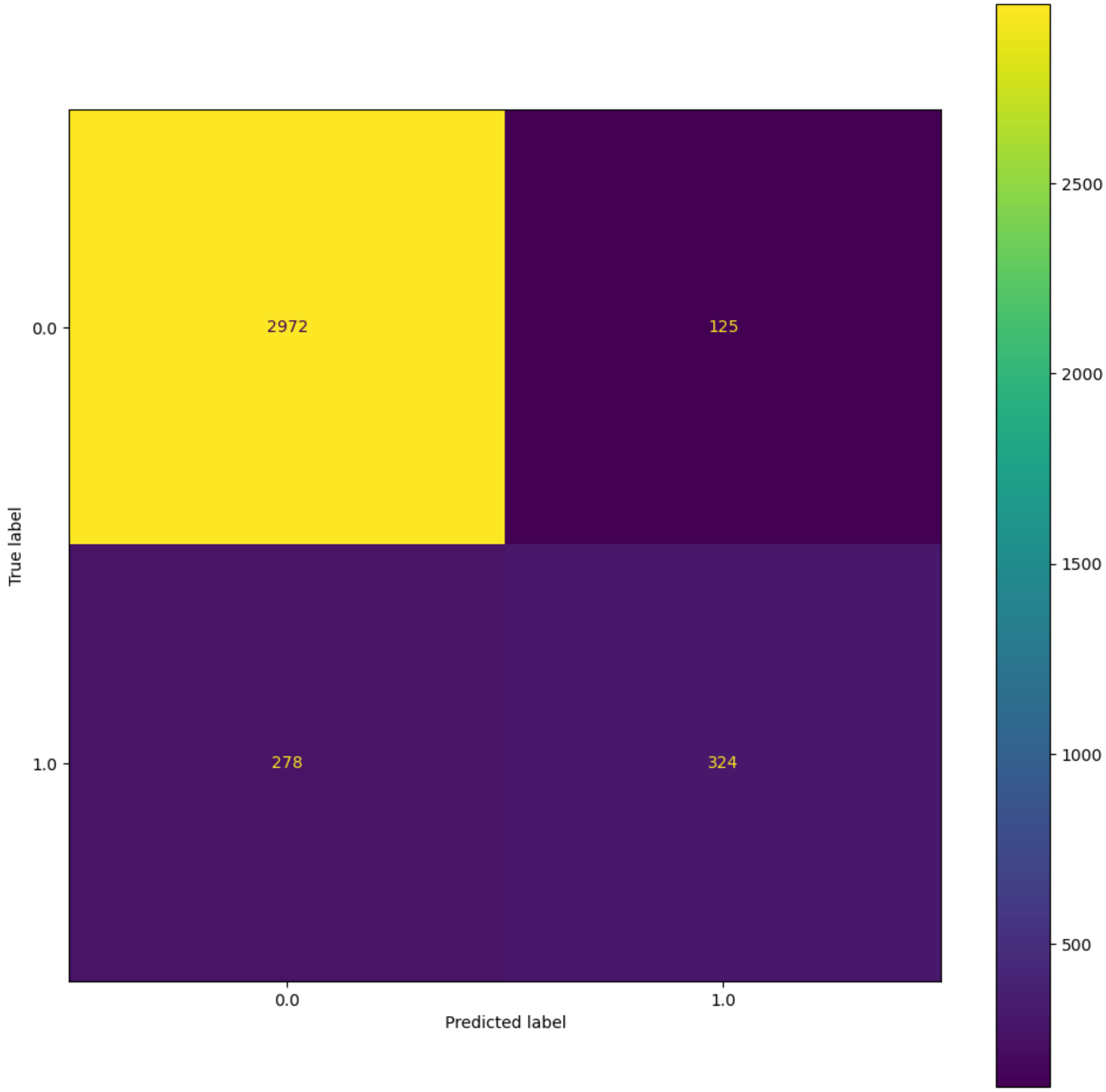
Show both text and visual confusion matrices using scikit-learn and matplotlib, explain what the graph tells you, and what you did. (2.5)

Both confusion matrices show that the model performs well in classifying True Negatives (i.e. predicting that a purchase was not made when it actually was not made). However, the model has mediocre performance in classifying True Positives (i.e. predicting that a purchase was made when it actually was made), since there's still a reasonable amount of False Negatives (i.e. predicting that a purchase was not made when it actually was made). This is most likely a result of the dataset having an imbalanced target set.

In [ ]:
```
conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(bag_fit, OSIX_test, OSIy_test)
```

```
[[2972  125]
 [ 278  324]]
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0df70910>
<Figure size 187.5x187.5 with 0 Axes>
```



Repeat the same with a different parameter set and compare the result (2)

Create a new BaggingClassifier model with a more base estimators (`n_estimators=100`).

This new model has 10 times more base estimators than the previous model. This new model saw a 1% increase in accuracy, a 2% increase in ROC AUC score, and the confusion matrix shows that it makes marginally better predictions than the previous model. However, this improvement in performance is negligible. Much like the first model, this model still suffers from the dataset's imbalance.

In [ ]:

```
clf = BaggingClassifier(n_estimators=100, n_jobs=-1, random_state=1859)
bag_fit = clf.fit(OSIX_train, OSIy_train)
OSIy_pred = bag_fit.predict(OSIX_test)

print(classification_report(OSIy_test, OSIy_pred))
```

```
print(roc_auc_score(OSIy_test, OSIy_pred))

conf_matrix = confusion_matrix(OSIy_test, OSIy_pred)
print(conf_matrix)

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
ConfusionMatrixDisplay.from_estimator(bag_fit, OSIX_test, OSIy_test)
```

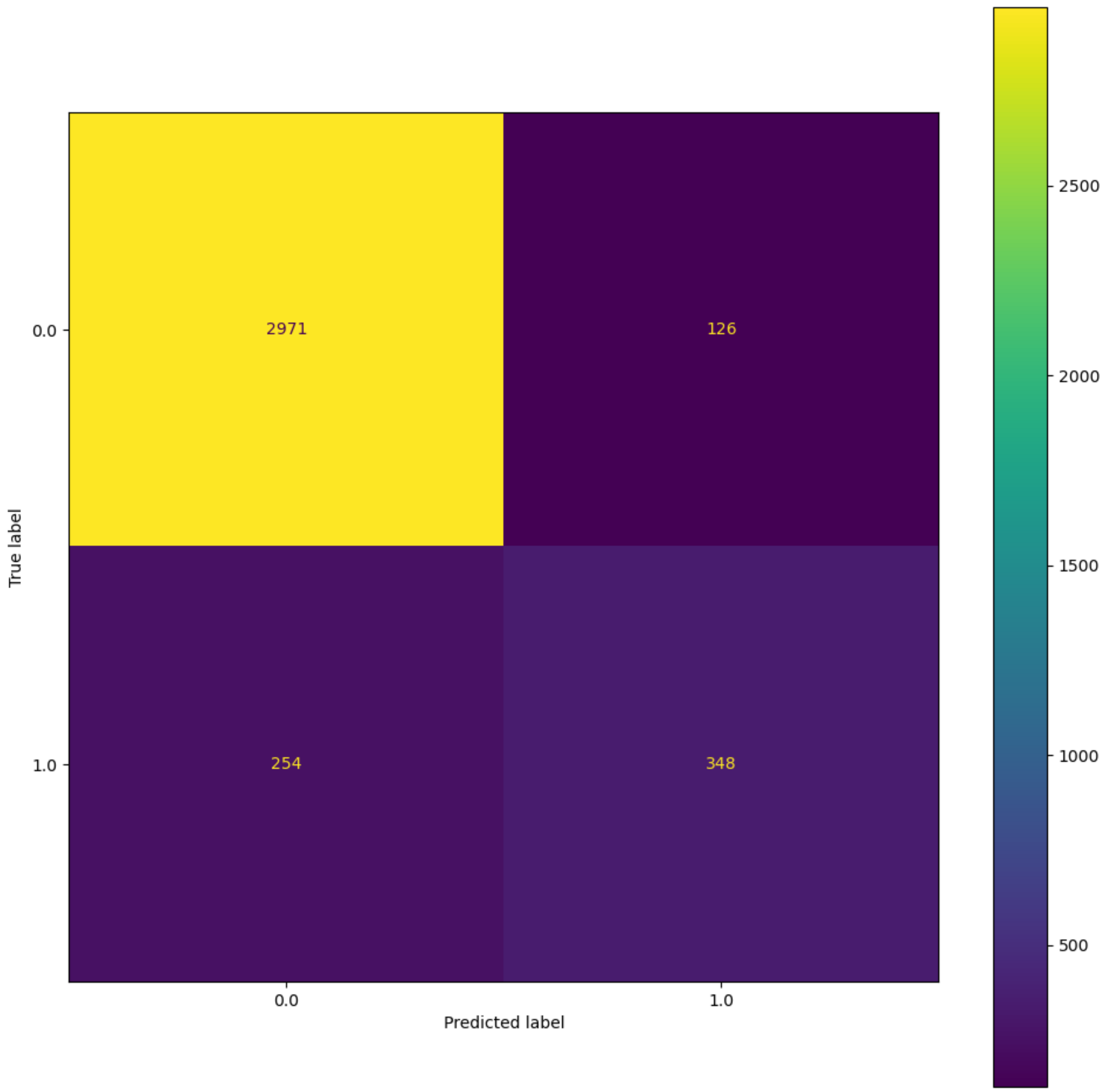
```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:804: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94      3097
         1.0       0.73      0.58      0.65       602

    accuracy                           0.90      3699
   macro avg       0.83      0.77      0.79      3699
weighted avg       0.89      0.90      0.89      3699

0.7686942781407793
[[2971  126]
 [ 254  348]]
```

Out[ ]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20a0e076bf0>
```

```
<Figure size 187.5x187.5 with 0 Axes>
```



Regression (22 Points)

Gradient Boost

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import GradientBoostingRegressor from scikit-learn.

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

Create the appropriate regressor, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `reg` that represents an instance of the GradientBoostingRegressor model. `n_estimators` represents the number of base estimators (which, by default are decision trees of depth=3) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```
reg = GradientBoostingRegressor(n_estimators=100, random_state=1859)
```

Train regressor on train data and explain what you did. (1.5)

Train the previous variable `reg` on variables `BSHX_train` and `BSHy_train` and create a variable `gb_fit` that represents the fitted GradientBoostingRegressor model.

In [ ]:
```
gb_fit = clf.fit(BSHX_train, BSHy_train)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:804: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Test/fit regressor test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `BSHX_test` and store the predicted target values in variable `BSHy_pred`.

In [ ]:
```
BSHy_pred = gb_fit.predict(BSHX_test)
```

Calculate model evaluation metrics and explain what you did. (1.5)

Import model evaluation metrics (MAE, MSE, and R^2) from scikit-learn. Calculate evaluation metrics by comparing the actual target values in `BSHy_test` to the fitted model's predicted values `BSHy_pred`.

The evaluation metrics reveal that the model is extremely robust. The MAE and RMSE are quite low, which suggest that the model, on average, makes predictions that are around 1-2 units off from the actual rental count. Additionally, an R^2 of 1 suggests that the model perfectly fits the data (i.e. 100% of the variance in the data is captured by the model).

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

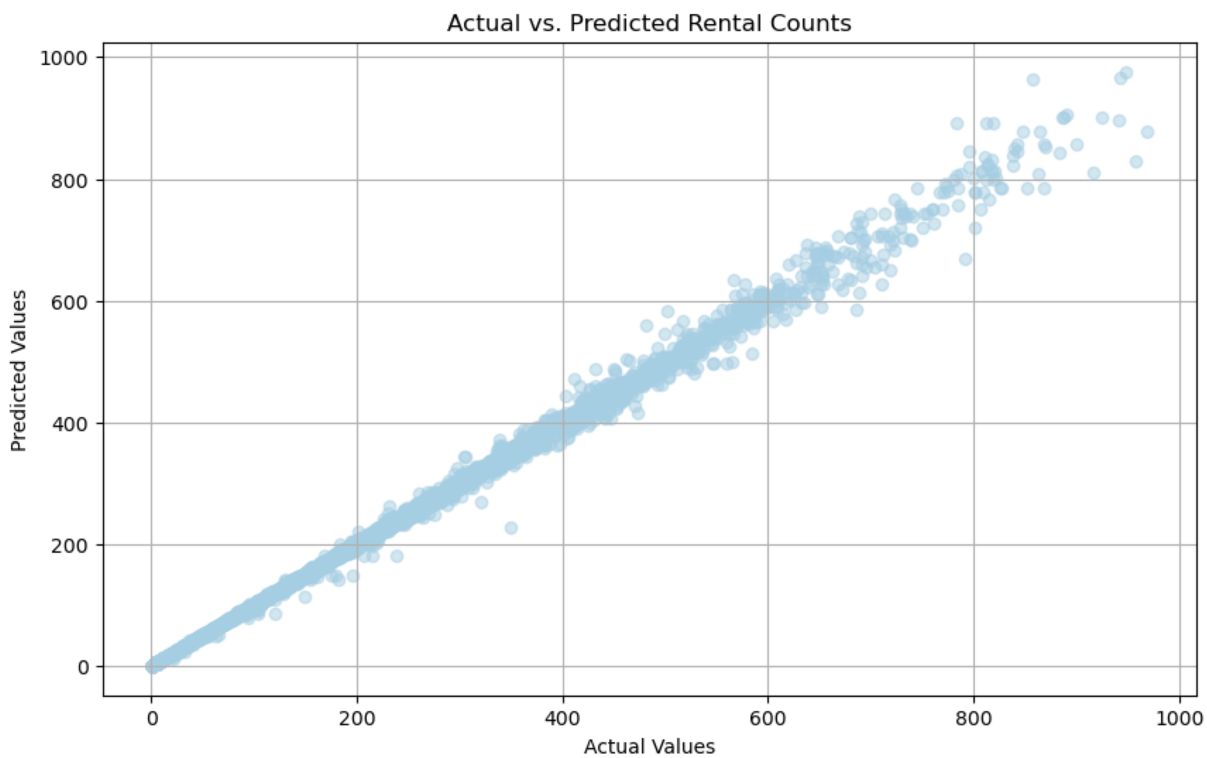
```python
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 4.10
Root Mean Squared Error (RMSE): 9.88
R-squared (R^2): 1.00
```

Plot the true vs predicted Y values using matplotlib, explain what the graph tells you, and what you did. (2.5)

The scatter plot of Actual vs. Predicted Rental Counts creates a line that tracks extremely well with y=x. This suggests that the model's predictions and the actual target values closely match, indicating low bias. Additionally, since there doesn't seem to be any outliers, the plot suggests that the model's errors are relatively symmetric and evenly distributed across the actual rental counts.

In [ ]:
```python
plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.grid(True)
plt.show()
```



Repeat the same with a different parameter set and compare the result (2)

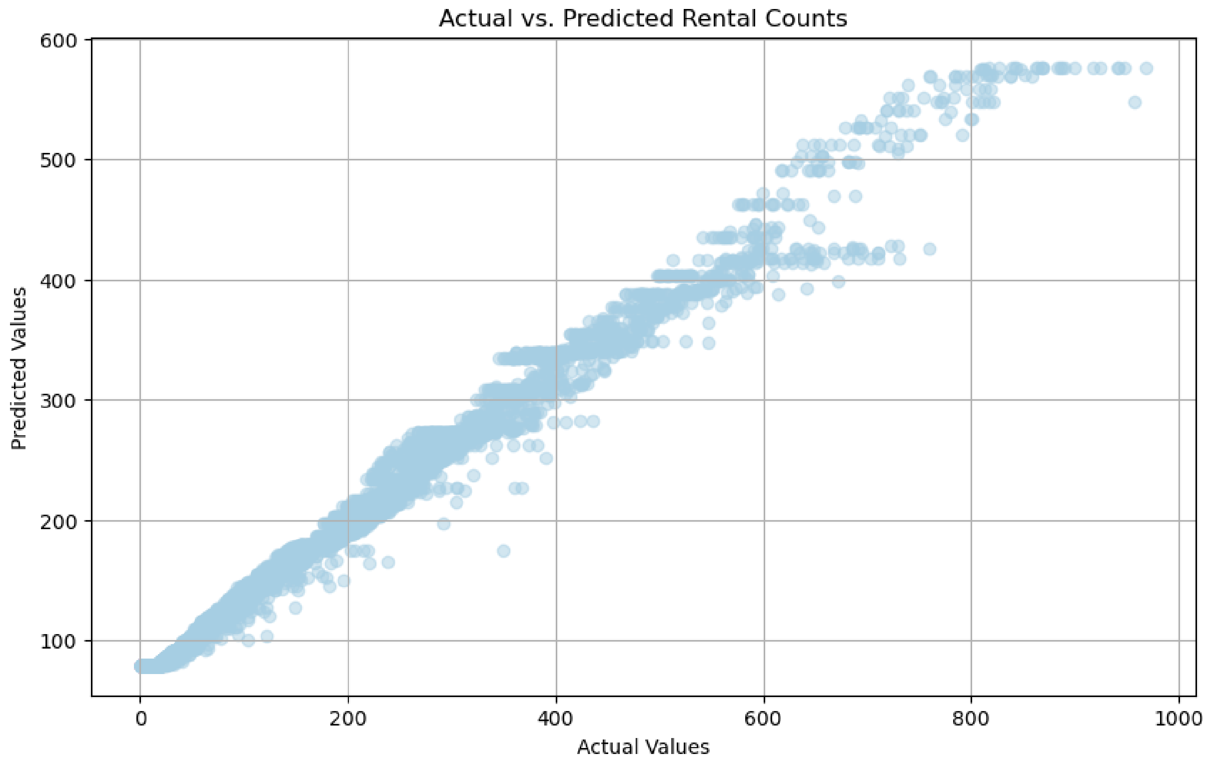
Create a new GradientBoostingRegressor model with a slower learning rate (`learning_rate=0.01`).

This new model has one-tenth the learning rate of the previous model. This new model saw quite dramatic reductions in performance than the previous model. Since a slower learning rate usually translates to better generalizability, this may suggest that the previous model is overfitted, and when new data comes in, this model may perform adequately.

In [ ]:

```
reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.01, random_state=
gb_fit = reg.fit(BSHX_train, BSHy_train)
BSHy_pred = gb_fit.predict(BSHX_test)

mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")

plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.grid(True)
plt.show()
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_gb.py:437:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Mean Absolute Error (MAE): 55.33
Root Mean Squared Error (RMSE): 72.77
R-squared (R^2): 0.84
```



XG Boost

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import XGBRegressor.

In [ ]:
```
from xgboost import XGBRegressor
```

Create the appropriate regressor, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `reg` that represents an instance of the XGBRegressor model. `n_estimators` represents the number of base estimators (which, by default are decision trees of depth=3) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```
reg = XGBRegressor(n_estimators=100, random_state=1859)
```

Train regressor on train data and explain what you did. (1.5)

Train the previous variable `reg` on variables `BSHX_train` and `BSHy_train` and create a variable `xgb_fit` that represents the fitted XGBRegressor model.

In [ ]:
```
xgb_fit = reg.fit(BSHX_train, BSHy_train)
```

Test/fit regressor test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `BSHX_test` and store the predicted target values in variable `BSHy_pred`.

In [ ]:
```
BSHy_pred = xgb_fit.predict(BSHX_test)
```

Calculate model evaluation metrics and explain what you did. (1.5)

Calculate evaluation metrics by comparing the actual target values in `BSHy_test` to the fitted model's predicted values `BSHy_pred`.

The evaluation metrics reveal that the model is extremely robust. The MAE and RMSE are quite low, which suggest that the model, on average, makes predictions that are around 2-3 units off from the actual rental count. Additionally, an R^2 of 1 suggests that the model perfectly fits the data (i.e. 100% of the variance in the data is captured by the model).

In [ ]:
```
mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")
```

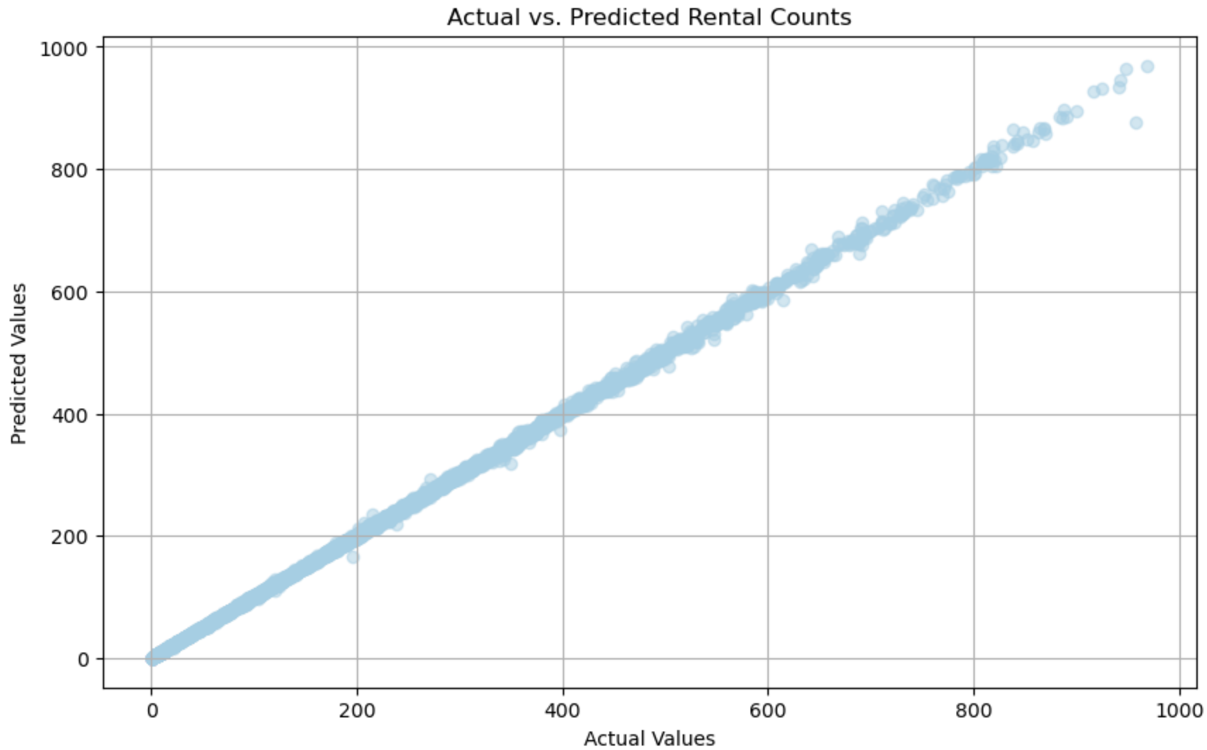
```
Mean Absolute Error (MAE): 1.89
Root Mean Squared Error (RMSE): 3.48
R-squared (R^2): 1.00
```

Plot the true vs predicted Y values using matplotlib, explain what the graph tells you, and what you did. (2.5)

The scatter plot of Actual vs. Predicted Rental Counts creates a line that tracks extremely well with y=x. This suggests that the model's predictions and the actual target values closely match, indicating low bias. Additionally, since there aren't any glaring outliers, the plot suggests that the model's errors are relatively symmetric and evenly distributed across the actual rental counts.

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
```

```
plt.grid(True)
plt.show()
```



Repeat the same with a different parameter set and compare the result (2)

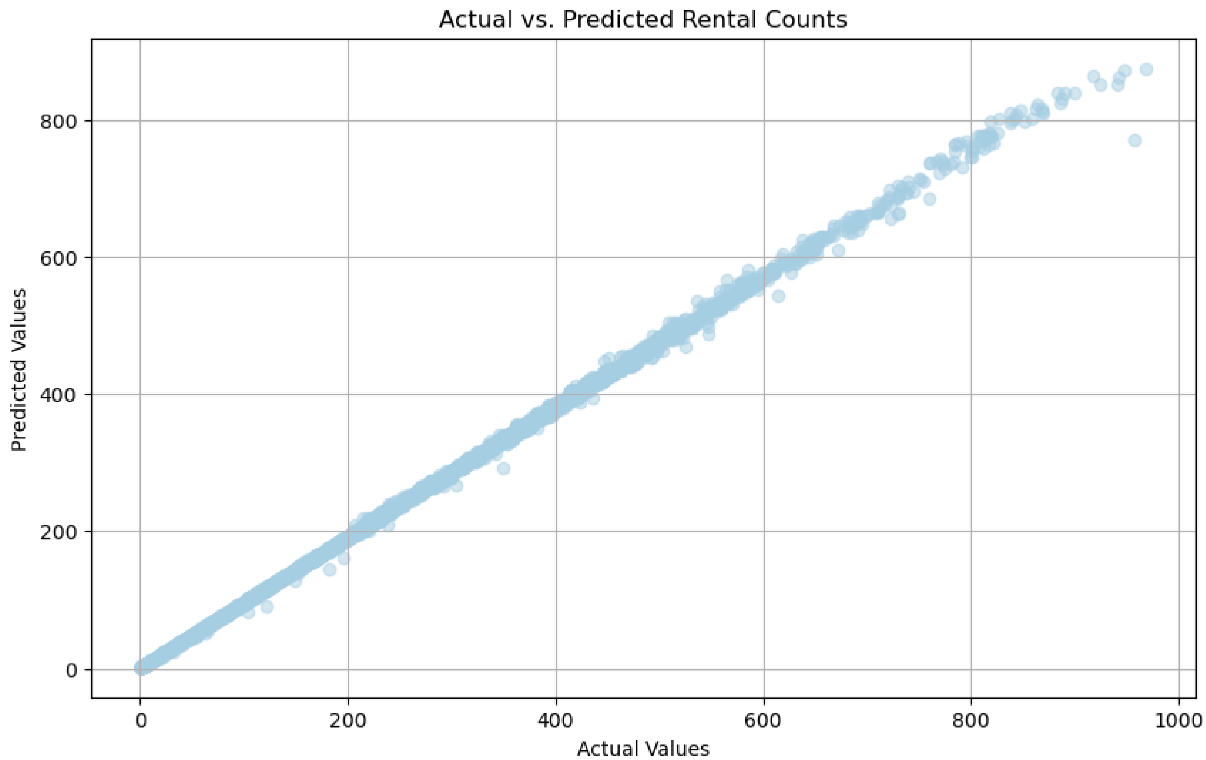
Create a new XGBRegressor model with a slower learning rate (`learning_rate=0.03`).

This new model has one-tenth the learning rate of the previous model. This new model saw moderate reductions in performance than the previous model. Since a slower learning rate usually translates to better generalizability, this may suggest that the previous model is overfitted, and when new data comes in, this model may perform adequately.

In [ ]:
```
reg = XGBRegressor(n_estimators=100, learning_rate=0.03, random_state=1859)
xgb_fit = reg.fit(BSHX_train, BSHy_train)
BSHy_pred = xgb_fit.predict(BSHX_test)

mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")

plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
```

```python
plt.ylabel("Predicted Values")
plt.grid(True)
plt.show()
```

```
Mean Absolute Error (MAE): 9.08
Root Mean Squared Error (RMSE): 13.64
R-squared (R^2): 0.99
```



Bagging

Import appropriate algorithm from scikit-learn and explain what you did. (1.5)

Import BaggingRegressor from scikit-learn.

In [ ]:
```python
from sklearn.ensemble import BaggingRegressor
```

Create the appropriate regressor, describe what the syntax represents, and what parameters you chose. (1.5)

Create a variable `reg` that represents an instance of the BaggingRegressor model. `n_estimators` represents the number of base estimators (which, by default are decision trees) that will be used for ensemble learning. The `random_state` seed is set to my net ID digits for reproducibility.

In [ ]:
```python
reg = BaggingRegressor(n_estimators=10, random_state=1859)
```

Train regressor on train data and explain what you did. (1.5)

Train the previous variable `reg` on variables `BSHX_train` and `BSHy_train` and create a variable `bag_fit` that represents the fitted BaggingRegressor model.

In [ ]:
```
bag_fit = reg.fit(BSHX_train, BSHy_train)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
```

Test/fit regressor test data and explain what you did. (1.5)

Using the trained model, attempt to make predictions from a test set of feature variables `BSHX_test` and store the predicted target values in variable `BSHy_pred`.

In [ ]:
```
BSHy_pred = bag_fit.predict(BSHX_test)
```

Calculate model evaluation metrics and explain what you did. (1.5)

Calculate evaluation metrics by comparing the actual target values in `BSHy_test` to the fitted model's predicted values `BSHy_pred`.

The evaluation metrics reveal that the model is quite robust. The MAE and RMSE are quite low, which suggest that the model, on average, makes predictions that are around 9-14 units off from the actual rental count. Additionally, an R^2 of 0.99 suggests that the model almost perfectly fits the data (i.e. 99% of the variance in the data is captured by the model).

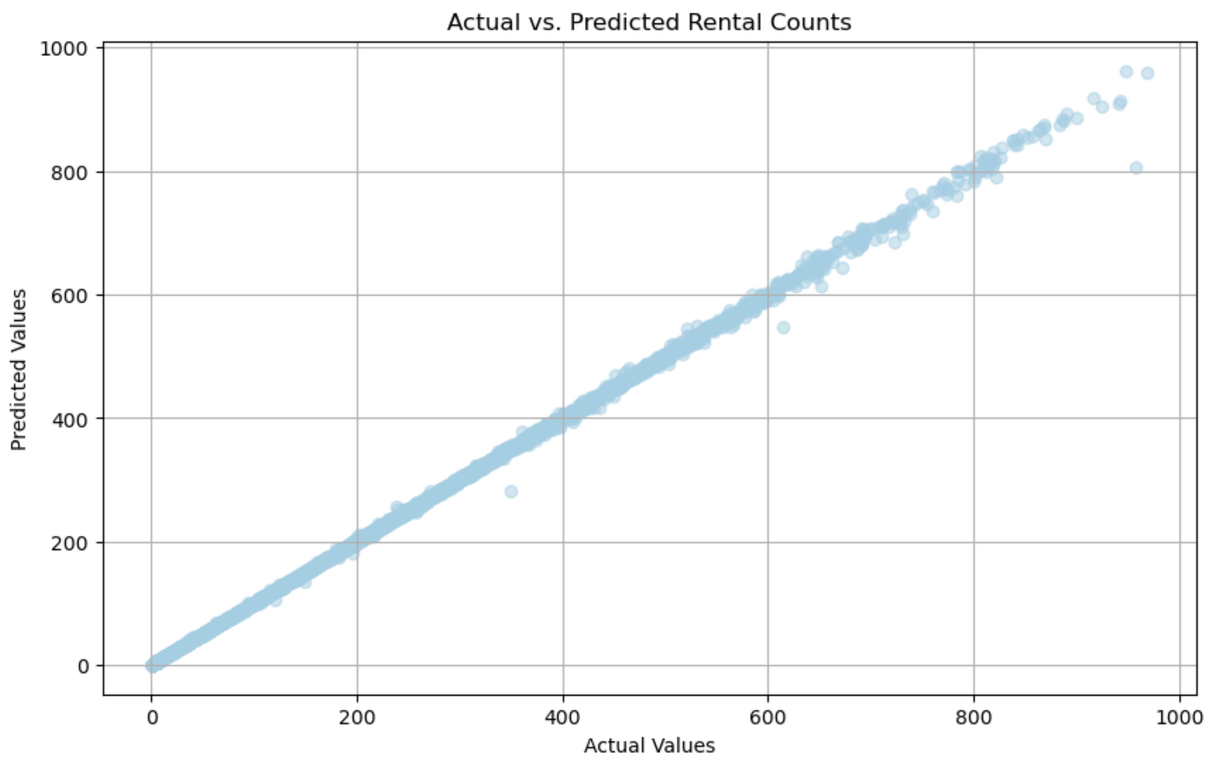
In [ ]:
```
mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 1.45
Root Mean Squared Error (RMSE): 3.88
R-squared (R^2): 1.00
```

Plot the true vs predicted Y values using matplotlib, explain what the graph tells you, and what you did. (2.5)

The scatter plot of Actual vs. Predicted Rental Counts creates a line that tracks extremely well with y=x. This suggests that the model's predictions and the actual target values closely match, indicating low bias. Additionally, since there aren't any glaring outliers, the plot suggests that the model's errors are relatively symmetric and evenly distributed across the actual rental counts.

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.grid(True)
plt.show()
```



Repeat the same with a different parameter set and compare the result (2)

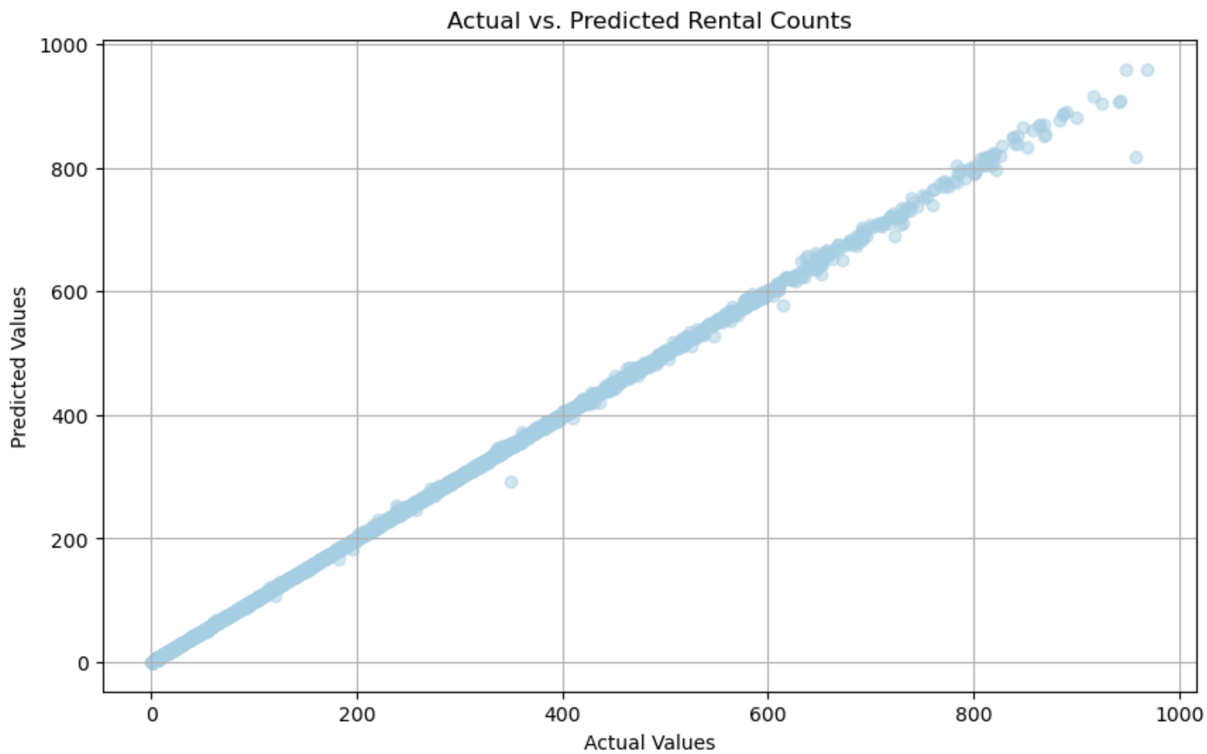
Create a new BaggingRegressor model with a more base estimators (`n_estimators=100`).

This new model has 10 times more base estimators than the previous model. This new model saw a noticeable increase in performance than the previous model. However, this improvement cwould also indicated that the new model is overfitting, and when new data comes in, this model may be unable to generalize quite as well as the previous.

In [ ]:
```
reg = BaggingRegressor(n_estimators=100, n_jobs=-1, random_state=1859)
bag_fit = reg.fit(BSHX_train, BSHy_train)
BSHy_pred = bag_fit.predict(BSHX_test)

mae = mean_absolute_error(BSHy_test, BSHy_pred)
mse = mean_squared_error(BSHy_test, BSHy_pred)
rmse = np.sqrt(mse)
```

```
r2 = r2_score(BSHy_test, BSHy_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R^2): {r2:.2f}")

plt.figure(figsize=(10, 6))
plt.scatter(BSHy_test, BSHy_pred, alpha=0.5)
plt.title("Actual vs. Predicted Rental Counts")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.grid(True)
plt.show()
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
Mean Absolute Error (MAE): 1.03
Root Mean Squared Error (RMSE): 3.17
R-squared (R^2): 1.00
```



Bonus Question (5)

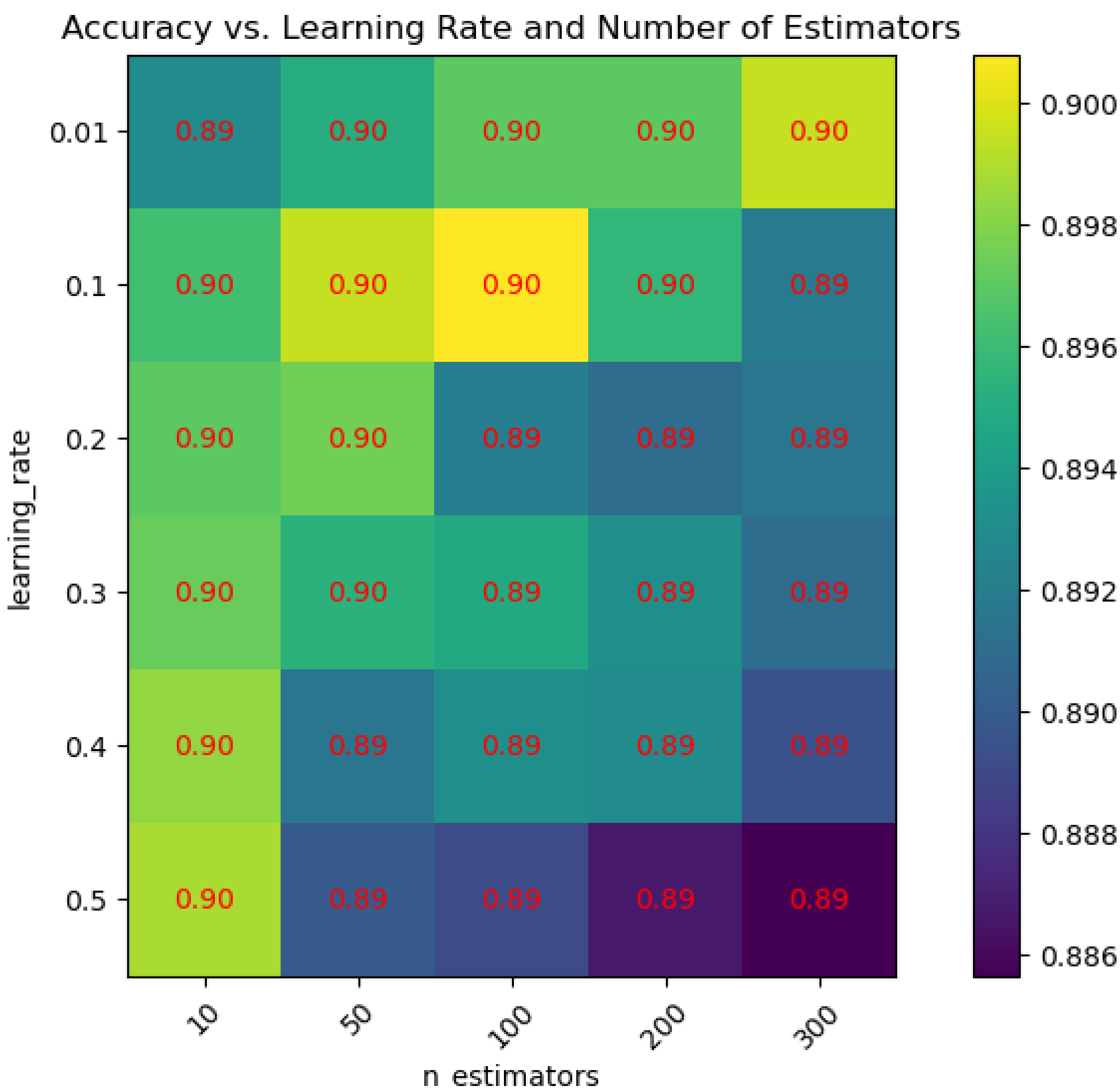
For all the given classifiers (Q3), evaluate the different parameter sets including (njobs, learning rate, etc).

For boosting and bagging compare the trade off between n jobs and learning rate. Plot the graph of

different learning rates vs number of jobs (Label the plot correctly. It should show title, x and y tick labels, and x and y axis labels). (1)

Boosting: Trade off b/w `learning_rate` and `n_estimators`

In [ ]:
```
# set up learning_rates and n_estimators
learning_rates = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5]
n_estimators = [10, 50, 100, 200, 300]

# dummy array to store values later
results = np.zeros((len(learning_rates), len(n_estimators)))

# iterate over every combination of learning_rates and n_estimators
for i, learning_rate in enumerate(learning_rates):
    for j, n_estimator in enumerate(n_estimators):
        clf = XGBClassifier(learning_rate=learning_rate, n_estimators=n_estimator,
        xgb_fit = clf.fit(OSIX_train, OSIy_train)
        OSIy_pred = xgb_fit.predict(OSIX_test)

        accuracy = accuracy_score(OSIy_test, OSIy_pred)
        results[i, j] = accuracy

# create heatmap
plt.figure(figsize=(10, 6))
plt.imshow(results, interpolation='nearest', cmap='viridis')
plt.colorbar()
plt.xticks(np.arange(len(n_estimators)), n_estimators, rotation=45)
plt.yticks(np.arange(len(learning_rates)), learning_rates)
plt.xlabel('n_estimators')
plt.ylabel('learning_rate')
plt.title('Accuracy vs. Learning Rate and Number of Estimators')

# add accuracy measures on the plot
for i in range(len(learning_rates)):
    for j in range(len(n_estimators)):
        plt.annotate(f'{results[i, j]:.2f}', (j, i), color='r', # round to 2 digits
                     ha='center', va='center', fontsize=10)

plt.show();
```



Bagging: Trade off b/w `n_jobs` and `n_estimators`

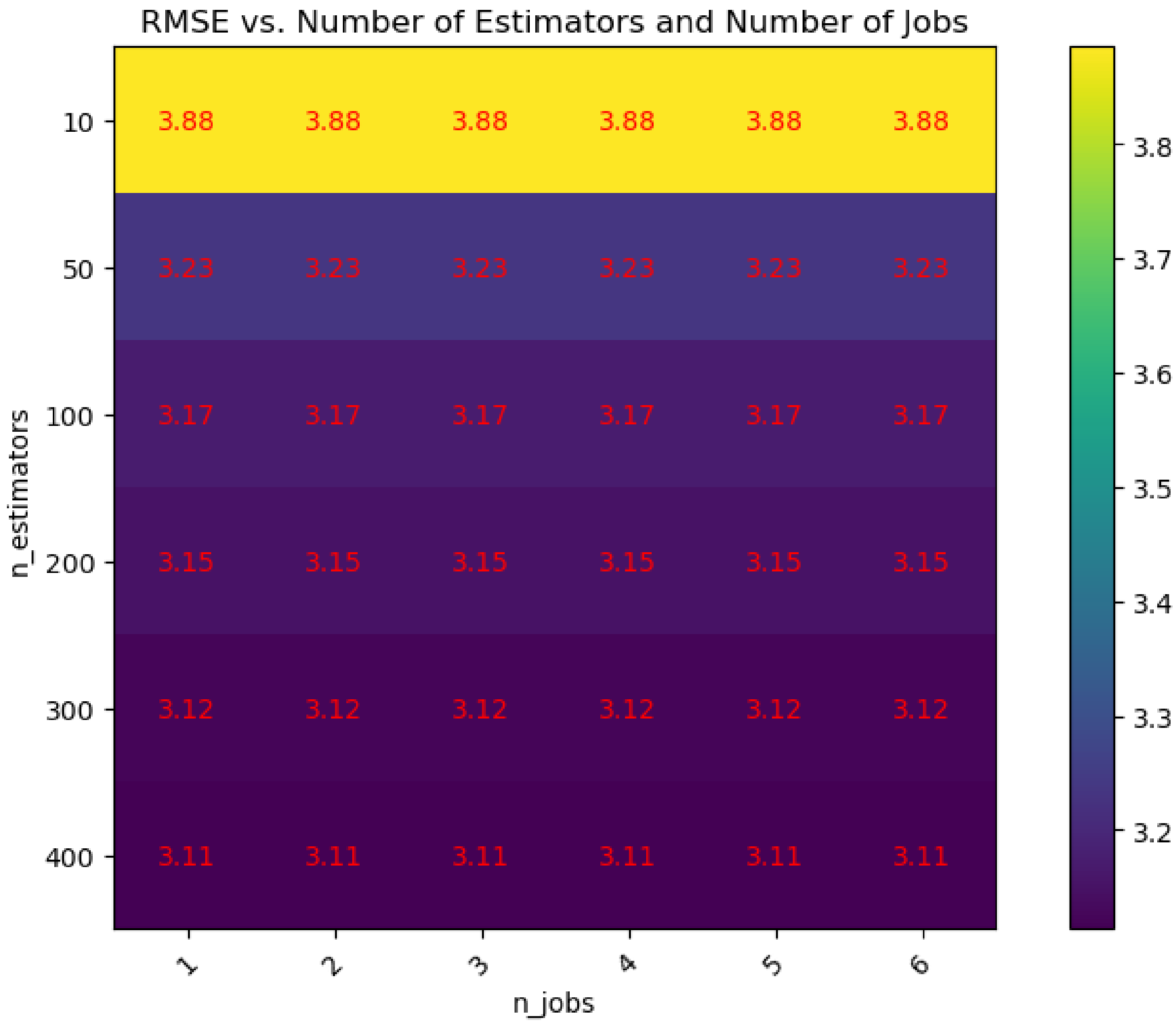
In [ ]:
```
# set up n_jobs and n_estimators
n_jobs = [1, 2, 3, 4, 5, 6] # I have a 6-core machine, so n_jobs=6 == n_jobs=-1. Us
n_estimators = [10, 50, 100, 200, 300, 400]

# dummy array to store values later
results = np.zeros((len(n_jobs), len(n_estimators)))

# iterate over every combination of learning_rates and n_estimators
for i, n_job in enumerate(n_jobs):
    for j, n_estimator in enumerate(n_estimators):
        reg = BaggingRegressor(n_estimators=n_estimator, n_jobs=n_job, random_state
        bag_fit = reg.fit(BSHX_train, BSHy_train)
        BSHy_pred = bag_fit.predict(BSHX_test)

        rmse = np.sqrt(mean_squared_error(BSHy_test, BSHy_pred))
        results[i, j] = rmse

# create heatmap
plt.figure(figsize=(10, 6))
```

```python
plt.imshow(results.T, interpolation='nearest', cmap='viridis')
plt.colorbar()
plt.xticks(np.arange(len(n_jobs)), n_jobs, rotation=45)
plt.yticks(np.arange(len(n_estimators)), n_estimators)
plt.xlabel('n_jobs')
plt.ylabel('n_estimators')
plt.title('RMSE vs. Number of Estimators and Number of Jobs')

# add rmse measures on the plot
for i in range(len(n_jobs)):
    for j in range(len(n_estimators)):
        plt.annotate(f'{results[i, j]:.2f}', (i, j), color='r', # round to 2 digits
                     ha='center', va='center', fontsize=10)

plt.show();
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
```



Explain the graph in detail. Specifically, describe the trade off between the learning rate and n jobs. Also, comment on the evolution of error for each combination (1 paragraph at least, 1.5).

The boosting trade off heatmap shows that there's some trade off in accuracy between learning rate and number of estimators. The best combination for this particular instance was learning_rate=0.1 and n_estimators=100, and the worst combination was learning_rate=0.5 and n_estimators=300. Overall, though, the variation in accuracy seems relatively negligible, being maybe a little over a percent at best.

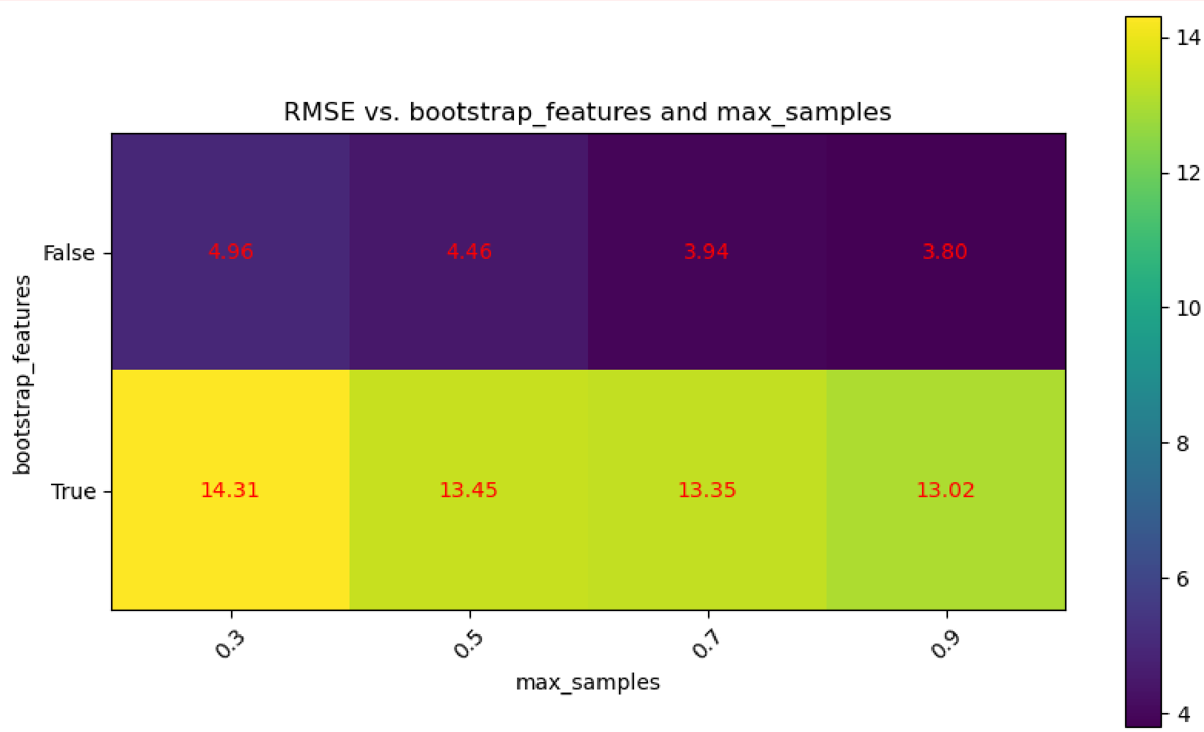
The bagging trade off heatmap shows that there's no trade off in performance (RMSE) between n_estimators and n_jobs. In fact, the only difference from run-to-run would be its time for execution (since `n_jobs` leverages parallelization). As expected, the models perform better as n_estimators increase.

For bagging, compare the trade off between the bootstrap features and max samples. Plot the graph of different combination of bootstrap features and max samples (Label the plot correctly. It should show title, x and y tick labels, and x and y axis labels). (1)

In [ ]:

```
from itertools import product

# set up bootstrap_features and max_samples
bootstrap_features = [False, True]
max_samples = [0.3, 0.5, 0.7, 0.9]

# dummy array to store values later
results = np.zeros((len(bootstrap_features), len(max_samples)))

# generate combinations of bootstrap_features and max_samples
combinations = product(bootstrap_features, max_samples)

# iterate over every combination
for (bootstrap, max_sample) in combinations:
    reg = BaggingRegressor(bootstrap_features=bootstrap, max_samples=max_sample, ra
    bag_fit = reg.fit(BSHX_train, BSHy_train)
    BSHy_pred = bag_fit.predict(BSHX_test)

    rmse = np.sqrt(mean_squared_error(BSHy_test, BSHy_pred))
    i = bootstrap_features.index(bootstrap)
    j = max_samples.index(max_sample)
    results[i, j] = rmse

# create heatmap
plt.figure(figsize=(10, 6))
plt.imshow(results, interpolation='nearest', cmap='viridis')
plt.colorbar()
plt.xticks(np.arange(len(max_samples)), max_samples, rotation=45)
plt.yticks(np.arange(len(bootstrap_features)), ['False', 'True'])
plt.xlabel('max_samples')
plt.ylabel('bootstrap_features')
plt.title('RMSE vs. bootstrap_features and max_samples')

# add rmse measures on the plot
for i in range(len(bootstrap_features)):
    for j in range(len(max_samples)):
        plt.annotate(f'{results[i, j]:.2f}', (j, i), color='r', # round to 2 digits
                     ha='center', va='center', fontsize=10)

plt.show();
```

```
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
c:\Users\Eric\anaconda3\envs\DSAN6700\lib\site-packages\sklearn\ensemble\_bagging.p
y:510: DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel().
  return column_or_1d(y, warn=True)
```



Explain the graph in detail. Specifically, describe the trade off between bootstrap features and max samples (1 paragraph at least, 1.5)

The bagging trade off heatmap shows that there's quite a noticeable trade off in performance between bootstrap_features and max_samples. When bootstrap_features is false, the model consistently outperforms models when bootstrap_features is true for all values of max_samples. This may be a result of there being a relatively large sample size, so bootstraping, in this case, is unnecessary and actually harms the predictions. Additionally, we can see as max_samples increases, RMSE decreases across the board, which suggests, unsurprisingly, that as the number of features to train each base estimator increases, the model's performance also increases.