Assignment 02 – Generative and Non-Generative Methods

Group 5 - Kaiyue Wei - kw823

Group 5 - Xulu Wang - xw370

Group 5 - Haoqi Wang - hw506

Group 5 - Yanyan Li - yl1494

Initialization

Load the dataset. (0.5 x 2)

In [ ]:
```
import urllib.request
url_g05 = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00468/online_shop
filename_g05 = 'datasets/online_shoppers_intention.csv'
urllib.request.urlretrieve(url_g05, filename_g05)
```

Out[ ]:
```
('datasets/online_shoppers_intention.csv',
 <http.client.HTTPMessage at 0x234e766afd0>)
```

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

In [ ]:
```
import pandas as pd
filename_g05 = 'datasets/online_shoppers_intention.csv'
OSI_g05 = pd.read_csv(filename_g05, header=0)
OSI_g05["Weekend"] = OSI_g05["Weekend"].astype(int)
OSI_g05["Revenue"] = OSI_g05["Revenue"].astype(int)

month_dict_g05 = {"Jan":1, "Feb":2, "Mar":3, "Apr":4, "May":5, "June":6,
                  "Jul":7, "Aug":8, "Sep":9, "Oc t":10, "Nov":11, "Dec":12}
OSI_g05["Month"] = OSI_g05["Month"].map(month_dict_g05)

OSI_g05.insert(loc=16,
               column="VisitorTypeNumeric",
               value=pd.factorize(OSI_g05['VisitorType'])[0] + 1)
OSI_g05 = OSI_g05.drop('VisitorType', axis=1)

OSI_g05.head(6)
```

Out[ ]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	Pr
0	0	0.0	0	0.0	1	
1	0	0.0	0	0.0	2	
2	0	0.0	0	0.0	1	
3	0	0.0	0	0.0	2	
4	0	0.0	0	0.0	10	
5	0	0.0	0	0.0	19	

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

In [ ]:
```
OSI_g05.describe()
```

Out[ ]:

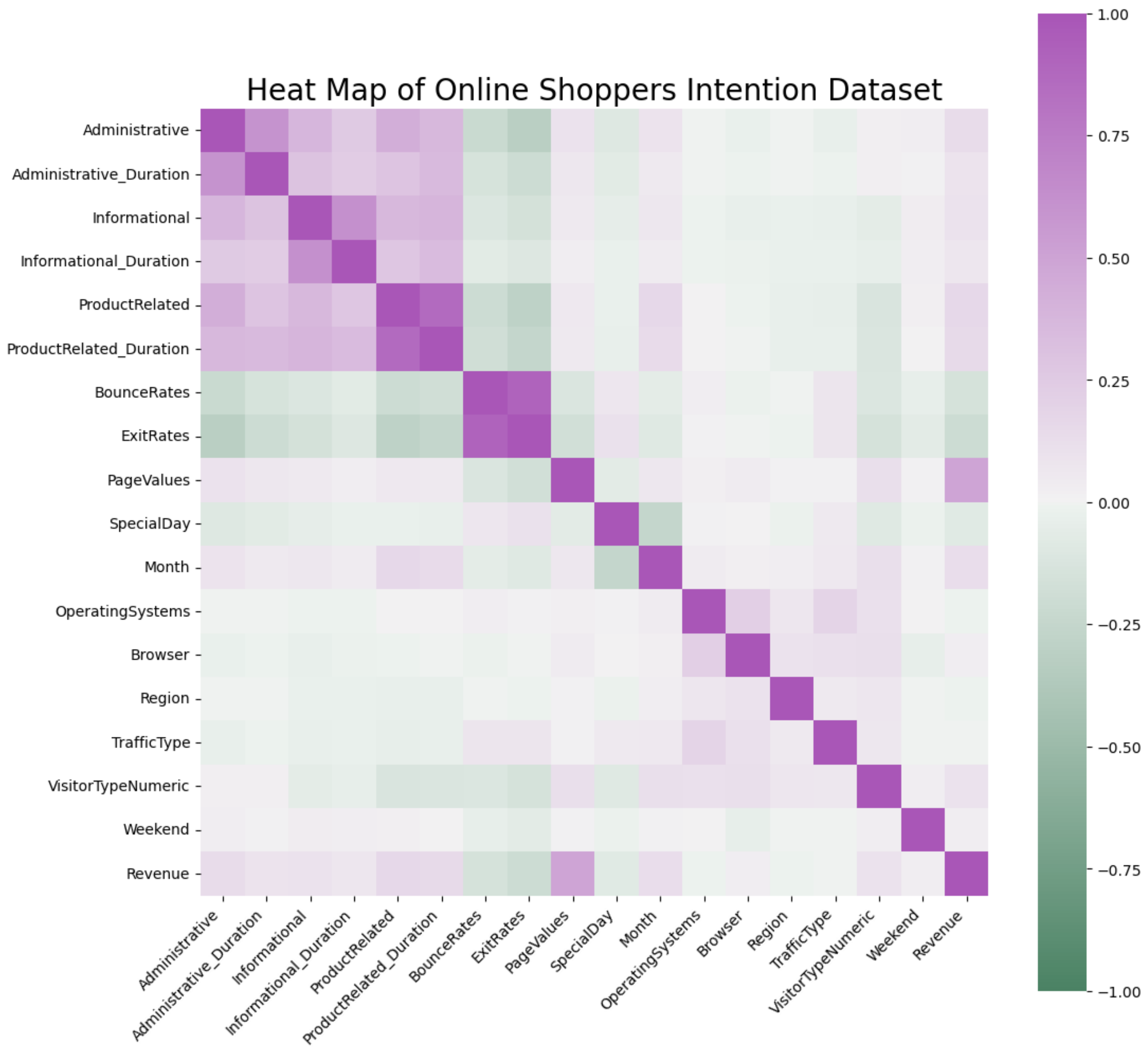
	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468
std	3.321784	176.779107	1.270156	140.749294	44.475503
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000
50%	1.000000	7.500000	0.000000	0.000000	18.000000
75%	4.000000	93.256250	0.000000	0.000000	38.000000
max	27.000000	3398.750000	24.000000	2549.375000	705.000000

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

rcParams['figure.figsize'] = 12,12
rcParams['figure.dpi'] = 100
corr_OSI_g05 = OSI_g05.corr()
ax_OSI_g05 = sns.heatmap(
    corr_OSI_g05,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(145, 300, s=60, as_cmap=True),
    square=True
)
ax_OSI_g05.set_xticklabels(
    ax_OSI_g05.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
)
ax_OSI_g05.set_title(
    "Heat Map of Online Shoppers Intention Dataset",
    fontdict={"size":20}
)
```

Out[ ]:

```
Text(0.5, 1.0, 'Heat Map of Online Shoppers Intention Dataset')
```

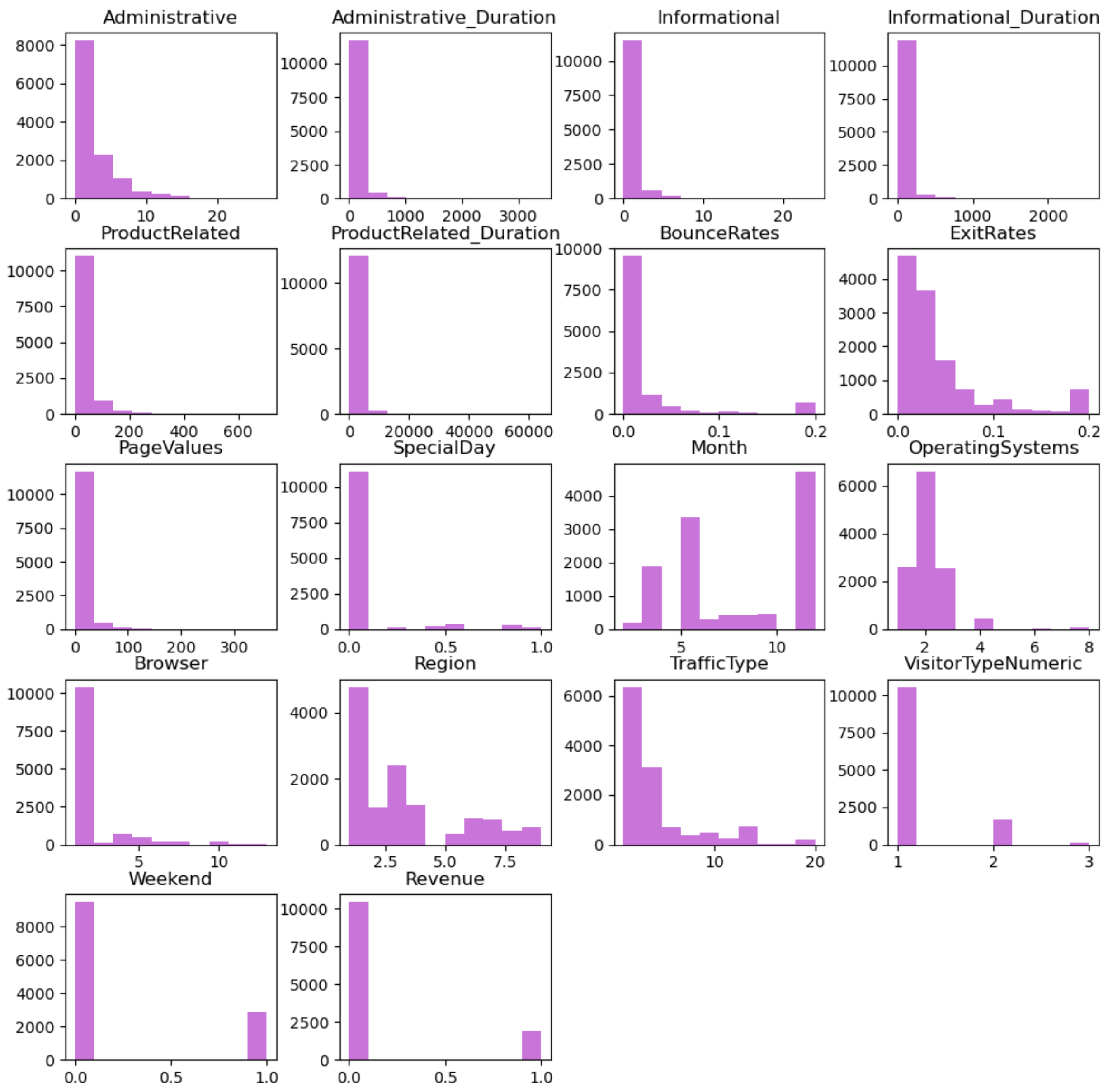


Show the distribution plots of each variable using hist function from pandas+matplotlib. Also, experiment with visual aspects of the image (not a lot, but an excellent visual will always leave a better impression. you can change color, thickness, font, font size, font color, etc.). Explain the plot distributions 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

plt.figure(figsize=(12, 12),dpi=150)
OSI_g05.hist(color='#C874D9', grid=False)
```

Out[ ]:
```
array([[<Axes: title={'center': 'Administrative'}>,
        <Axes: title={'center': 'Administrative_Duration'}>,
        <Axes: title={'center': 'Informational'}>,
        <Axes: title={'center': 'Informational_Duration'}>],
       [<Axes: title={'center': 'ProductRelated'}>,
        <Axes: title={'center': 'ProductRelated_Duration'}>,
        <Axes: title={'center': 'BounceRates'}>,
        <Axes: title={'center': 'ExitRates'}>],
       [<Axes: title={'center': 'PageValues'}>,
        <Axes: title={'center': 'SpecialDay'}>,
        <Axes: title={'center': 'Month'}>,
        <Axes: title={'center': 'OperatingSystems'}>],
       [<Axes: title={'center': 'Browser'}>,
        <Axes: title={'center': 'Region'}>,
        <Axes: title={'center': 'TrafficType'}>,
        <Axes: title={'center': 'VisitorTypeNumeric'}>],
       [<Axes: title={'center': 'Weekend'}>,
        <Axes: title={'center': 'Revenue'}>, <Axes: >, <Axes: >]],
      dtype=object)
<Figure size 1800x1800 with 0 Axes>
```



In [ ]:
```
print("Unique values of Revenue variable: ", OSI_g05['Revenue'].unique())
print("Unique values of Special_Day variable: ", sorted(OSI_g05['SpecialDay'].unique()
print("Unique values of Browser variable: ", sorted(OSI_g05['Browser'].unique()))
print("Unique values of TrafficType variable: ", sorted(OSI_g05['TrafficType'].unique
```

```
Unique values of Revenue variable:  [0 1]
Unique values of Special_Day variable:  [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
Unique values of Browser variable:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
Unique values of TrafficType variable:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
4, 15, 16, 17, 18, 19, 20]
```

Numerical:

Administrative

- The distribution plot of variable Administrative shows that the mean is around 2.3 with a standard deviation of around 3.3. The distribution is skewed towards the right, which means that there are more values on the right side of the distribution than on the left side.

Administrative Duration & Informational & Informational Duration & ProductRelated & ProductRelated Duration & Page Values

- The distribution plot of these variables have very similar distribution. Their distributions are skewed towards the right with a sharp peak at the minimum value, most of their values are on the right side.
- `Administrative Duration` variable has the mean around 80.82 with a standard deviation around 176.78.
- `Informational` variable has the mean around 0.5 with a standard deviation around 1.27.
- `Informational Duration` variable has the mean around 34.47 with a standard deviation around 140.75.
- `ProductRelated` variable has the mean around 31.73 with a standard deviation around 44.48.
- `ProductRelated Duration` variable has the mean around 1194.75 with a standard deviation around 1913.67.
- `Page Values` variable has the mean around 5.89 with a standard deviation around 18.57.

BounceRates & ExitRates

- The distribution plot of these variables have similar distribution. Their distributions are skewed towards the right, which means that there are more values on the right side of the distribution with small values in the other bins and a slightly higher value in the last right bin.
- `BounceRates` variable has the mean around 0.02 with a standard deviation around 0.05.
- `ExitRates` variable has the mean around 0.04 with a standard deviation around 0.05.

Categorical:

Special Day

- From the distribution plot of Special Day, the most common value is 0. Very few data points in the other categories.

Operating Systems

- From the distribution plot of Operating Systems, the most common value is 2, followed by 1 and 3. Very few data points in the other categories.

Browser

- From the distribution plot of Browser, the most common value is 1. Very few data points in the other categories.

Region

- From the distribution plot of Region, the most common value is 1, followed by 3 and then 2 & 4. Some data points in the other categories.

Traffic Type

- From the distribution plot of Traffic Type, the distribution is skewed towards the right. The most common value is 1, followed by 2. Few data points in the other categories.

Visitor Type Numeric

- From the distribution plot of Visitor Type Numeric, the most common value is 1, followed by 2 and then 3.

Weekend

- Weekend varible is binary category. From its distribution plot, the most common value is 0.

Revenue

- Revenue varible is binary category. From its distribution plot, the most common value is 0.

Load the dataset. (0.5 x 2)

```
In [ ]: import pandas as pd
        filename_BSH_g05 = 'datasets/Bike-Sharing-Hour.csv'
        BSH_g05 = pd.read_csv(filename_BSH_g05, header=0)
```

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

```
In [ ]: import pandas as pd
        BSH_g05.head(6)
```

Out[ ]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879
5	6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2576

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

In [ ]:
```
BSH_g05.describe()
```

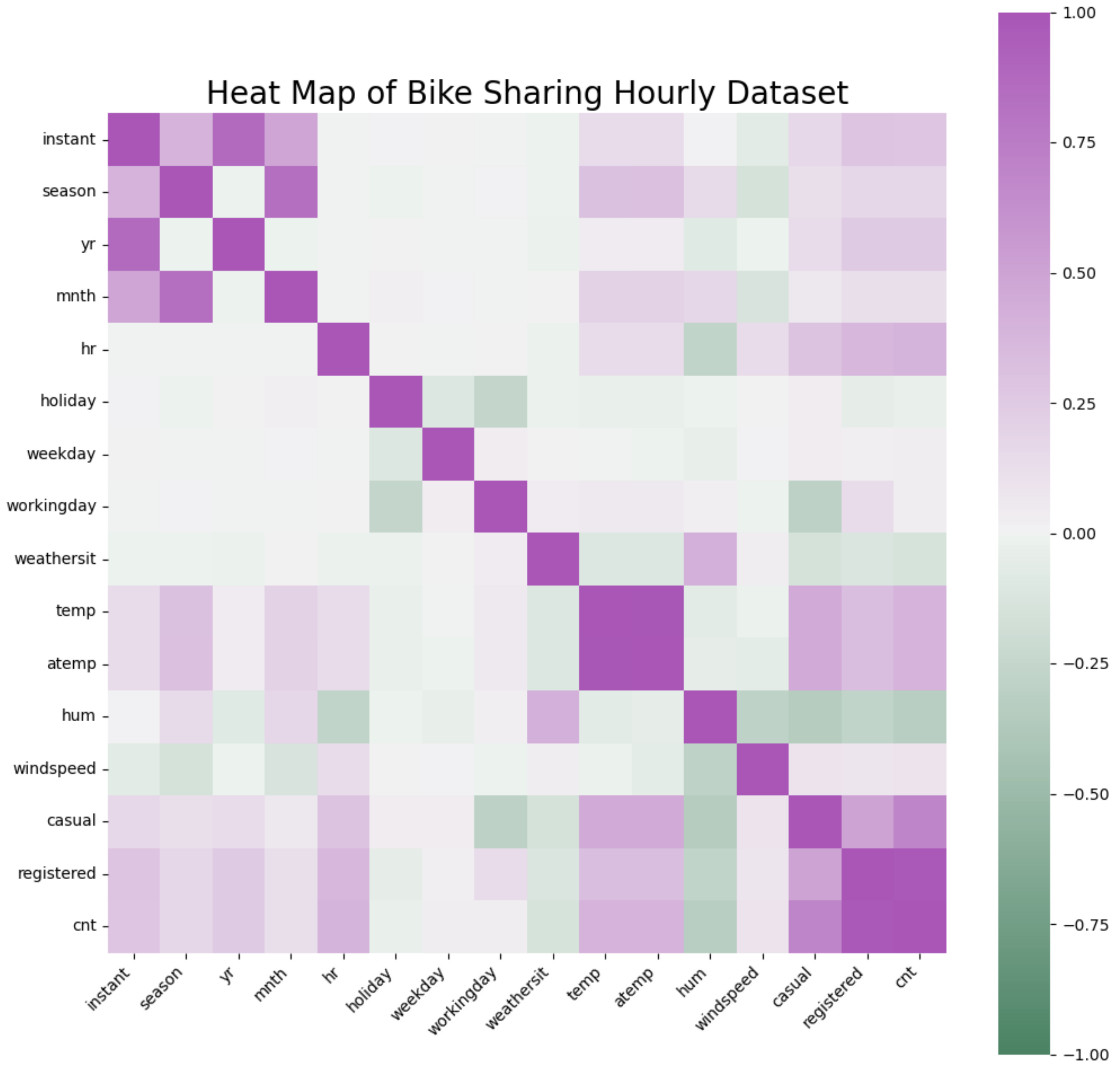
Out[ ]:

	instant	season	yr	mnth	hr	holiday	weekda
count	17379.0000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.00000
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	3.00368
std	5017.0295	1.106918	0.500008	3.438776	6.914405	0.167165	2.00577
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	0.00000
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	1.00000
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	3.00000
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	5.00000
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	6.00000

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

rcParams['figure.figsize'] = 12,12
rcParams['figure.dpi'] = 100
corr_BSH_g05 = BSH_g05.corr()
ax_BSH_g05 = sns.heatmap(
    corr_BSH_g05,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(145, 300, s=60, as_cmap=True),
    square=True
)
ax_BSH_g05.set_xticklabels(
    ax_BSH_g05.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
)
ax_BSH_g05.set_title(
    "Heat Map of Bike Sharing Hourly Dataset",
    fontdict={"size":20}
)
```

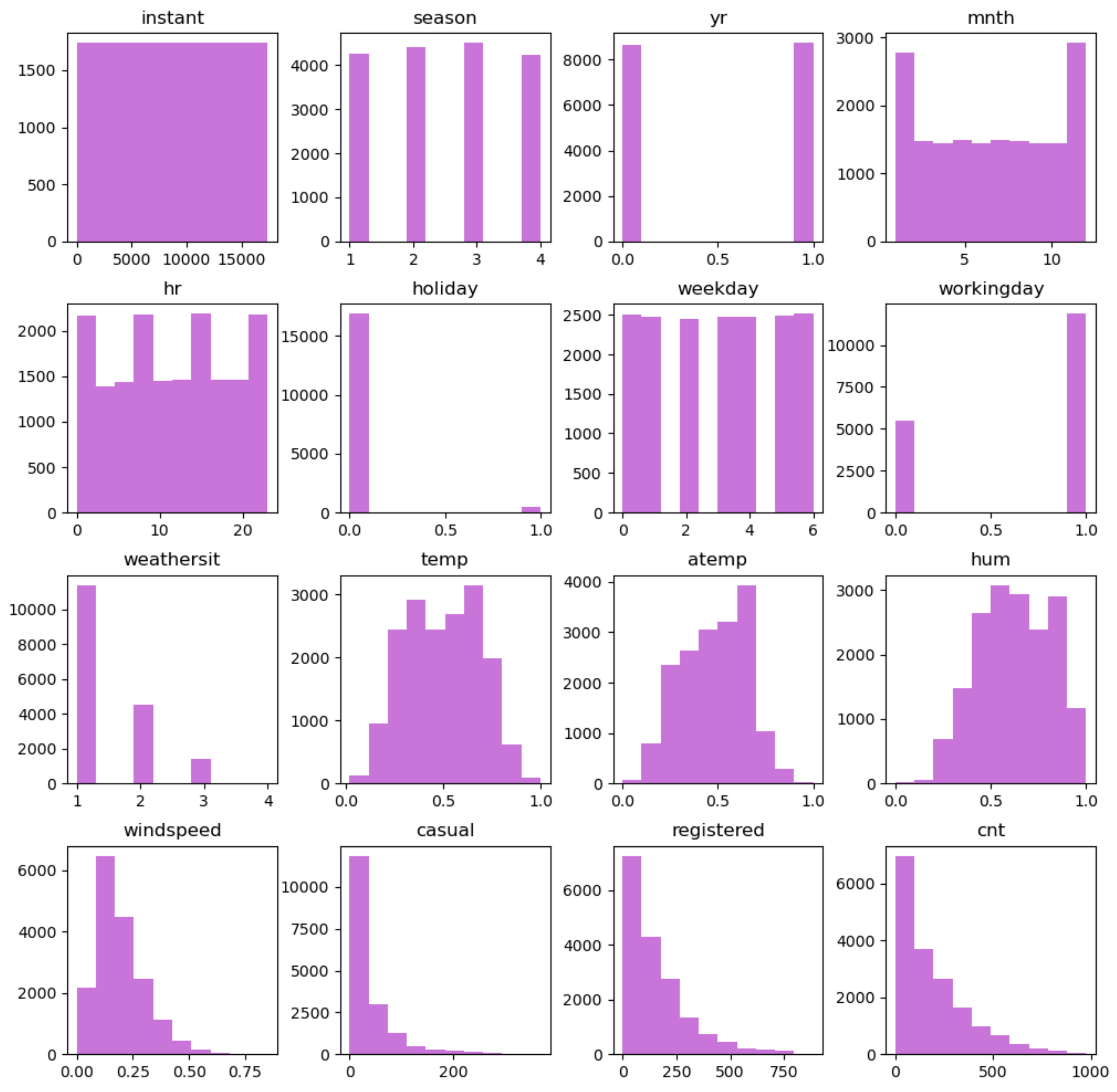
Out[ ]:
```
Text(0.5, 1.0, 'Heat Map of Bike Sharing Hourly Dataset')
```



Show the distribution plots of each variable using hist function from pandas+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

plt.figure(figsize=(12, 12),dpi=150)
BSH_g05.hist(color='#C874D9', grid=False)
```

```
Out[ ]: array([[<Axes: title={'center': 'instant'}>,
                <Axes: title={'center': 'season'}>,
                <Axes: title={'center': 'yr'}>, <Axes: title={'center': 'mnth'}>],
               [<Axes: title={'center': 'hr'}>,
                <Axes: title={'center': 'holiday'}>,
                <Axes: title={'center': 'weekday'}>,
                <Axes: title={'center': 'workingday'}>],
               [<Axes: title={'center': 'weathersit'}>,
                <Axes: title={'center': 'temp'}>,
                <Axes: title={'center': 'atemp'}>,
                <Axes: title={'center': 'hum'}>],
               [<Axes: title={'center': 'windspeed'}>,
                <Axes: title={'center': 'casual'}>,
                <Axes: title={'center': 'registered'}>,
                <Axes: title={'center': 'cnt'}>]], dtype=object)
        <Figure size 1800x1800 with 0 Axes>
```



Numerical:

hr:

- More data is collected during the time periods 9pm to 2am, 7am to 9am and 2pm to 4pm. The rest of the hours have roughly equal numbers of records.

temp (normalized temperature in Celsius):

- The majority of records take place when the temp is between 0.3 (6.1 °C) and 0.8 (29.6 °C), but its distribution is slightly different from the normal distribution, because there are two peaks at about 0.4 and 0.7, corresponding to roughly 11 °C and 25 °C.
- Its standard deviation is about 1 °C

atemp:

- The number of records for the feeling temperature ranging from 0.3 (6.1 °C) to 0.7 (24.9 °C) gradually increase, constituting the majority of all records.

hum:

- The distribution of normalized humidity generally conforms to the normal distribution with a mean of 0.627 and a standard deviation of 0.193, except when the normalized humidity is around 0.8.

windspeed:

- The distribution of normalized windspeed is right-skewed, with a mean of 0.19 and a standard deviation of 0.12. However, the mode occurs within the range of 0.085 to 0.17.

casual & registered & cnt:

- These three variables exhibit a similar distribution, which aligns with their intended meaning, representing the number of users on a given day. Based on the histograms, the highest number of records is found in the bin with the lowest count value. As the number of users increases, the number of records decreases rapidly.

Categorical:

season ((1:winter, 2:spring, 3:summer, 4:fall)):

- The number of records from different seasons is about equal, but slightly more data come from the summer.

yr:

- The number of records from 2011 (0) and 2012 (1) is about equal.

mnth:

- The number of records from different weekdays is about equal.

holiday:

- Data from non-holiday days dominates.

weekday (day of the week):

- The number of records from different weekdays is about equal.

workingday (if day is neither weekend nor holiday is 1, otherwise is 0.):

- The numder of records from workingdays is more than twice that of non-workingdays.

weathersit:

> - 1 - Clear, Few clouds, Partly cloudy, Partly cloudy
> - 2 - Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
> - 3 - Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
> - 4 - Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
> - Data from category 1 make up the majority. Catergory 2 and 3 follow behind. Few data come from the 'Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog' days.

Intermediate Steps (Essential, no points granted)

In [ ]:
```python
from sklearn.ensemble import AdaBoostClassifier, \
    GradientBoostingClassifier, BaggingClassifier
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, roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC

OSI_g05.dropna(inplace=True)
X_OSI_g05 = OSI_g05.loc[:, OSI_g05.columns != 'Revenue'].to_numpy()
y_OSI_g05 = OSI_g05.iloc[:,-1:].to_numpy()

# print(X_OSI_g05)
# print(y_OSI_g05)

OSIX_train_g05, OSIX_test_g05, OSIy_train_g05, OSIy_test_g05 = \
    train_test_split(X_OSI_g05, y_OSI_g05, test_size=0.30, random_state=3316)
```

In [ ]:
```python
# X_BSH_g05 = BSH_g05.loc[:, BSH_g05.columns != 'cnt'].to_numpy()
X_BSH_g05 = BSH_g05.loc[:, (BSH_g05.columns != 'cnt') & \
                        (BSH_g05.columns != 'dteday')].to_numpy()
y_BSH_g05 = BSH_g05.iloc[:,-1:].to_numpy()

# print(X_BSH_g05)
# print(y_BSH_g05)
```

```
BSHX_train_g05, BSHX_test_g05, BSHy_train_g05, BSHy_test_g05 = \
    train_test_split(X_BSH_g05, y_BSH_g05, test_size=0.30, random_state=3316)
```

Classification (total 48)

AdaBoost

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

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

Import the AdaBoost Classifier Algorithm from the sklearn package(class) under the ensemble method/category.

Create the appropriate classifier and describe what the syntax represents and what parameters you choose (1.5)

In [ ]:
```
OSI_clf_ada_g05 = AdaBoostClassifier(n_estimators=100, random_state=3316)
```

For the AdaBoostClassifier, we do not specify the base learner, so it will use the default decision tree algorithm.

`n_estimators` , which is the number of base estimators, sets to 100, so that 100 decision trees will be used to build the ensemble adaboost.

`random_state` sets to 3316 which represent the GUID of one of the group members in order to ensure to get the same result with same random state number.

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

In [ ]:
```
OSI_ada_fit_g05 = OSI_clf_ada_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
```

`OSI_clf_ada_g05` classifier will be used to train the ada boosting model with input variables training x `OSIX_train_g05` , on target label training y `OSIy_train_g05` .

- `ravel()` used to convert a 2D array into a 1D array

`OSI_ada_fit_g05` used to store the trained classifier

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

In [ ]:
```
OSIy_pred_g05 = OSI_ada_fit_g05.predict(OSIX_test_g05)
```

Use the trained AdaBoost classifier `OSI_ada_fit_g05` to make predictions on the test data `OSIX_test_g05`, and store the predicted labels in `OSIy_pred_g05`.

Calculate accuracy and explain what you did. (1.5)

In [ ]:
```
OSI_Ada_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_g05)
print("AdaBoosting Classifier Accuracy: {:.2%}".format(OSI_Ada_accuracy_g05))
```

```
AdaBoosting Classifier Accuracy: 89.22%
```

Calculate the accuracy of the classifier's predictionswith `accuracy_score` function.

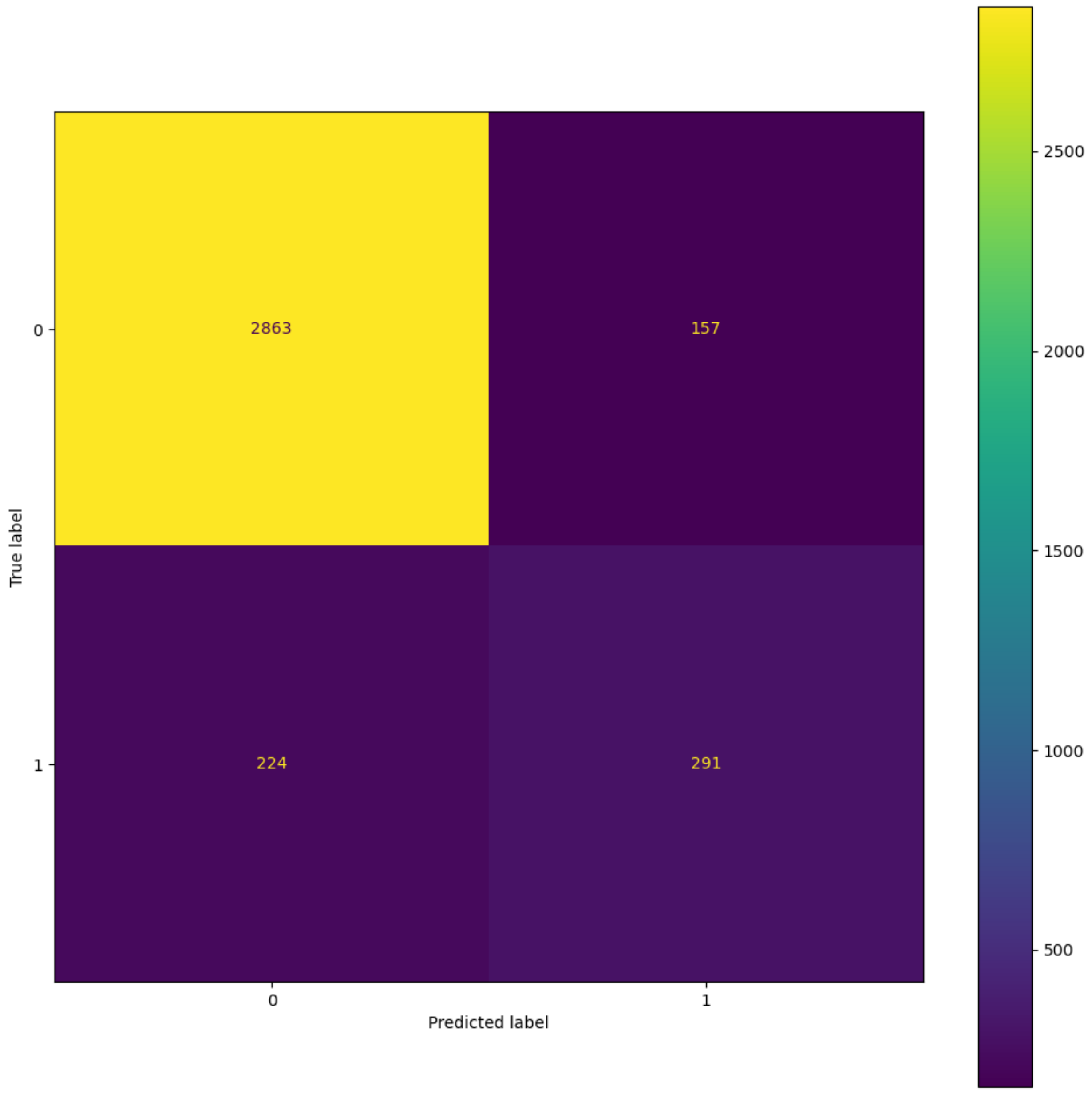
`OSIy_test_g05` y labels testing data as the `y_true` and compared with `OSIy_pred_g05` y labels predicted on testing features x from the previous step.

`OSI_Ada_accuracy_g05` stores the accuracy score calculated, and then print the result.

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

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

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
OSI_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_ada_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_pred_g05))
```

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      3020
           1       0.65      0.57      0.60       515

    accuracy                           0.89      3535
   macro avg       0.79      0.76      0.77      3535
weighted avg       0.89      0.89      0.89      3535

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



Get confusion matrix based on actual testing data y label `OSIy_test_g05`, and predcted y label from x feature testing data `OSIy_pred_g05` and store in `OSI_conf_matrix_g05`.

Use `ConfusionMatrixDisplay` to visualize the confusion matrix.

Print the classification report with `OSIy_test_g05` and `OSIy_pred_g05`, whcih provides precision, recall, F1-score, and support for each class.

The result:

The overall accuracy is 89.22%.

The model correctly predicted the negative class (label 0) very well with 2,863 data points correctly classified as negative, and it has a relatively small number of false positives (157 data points).

This model has some ability to recognize positive instances. It correctly predicted the positive class (label 1) for 291 data points, but it has 224 false negatives.

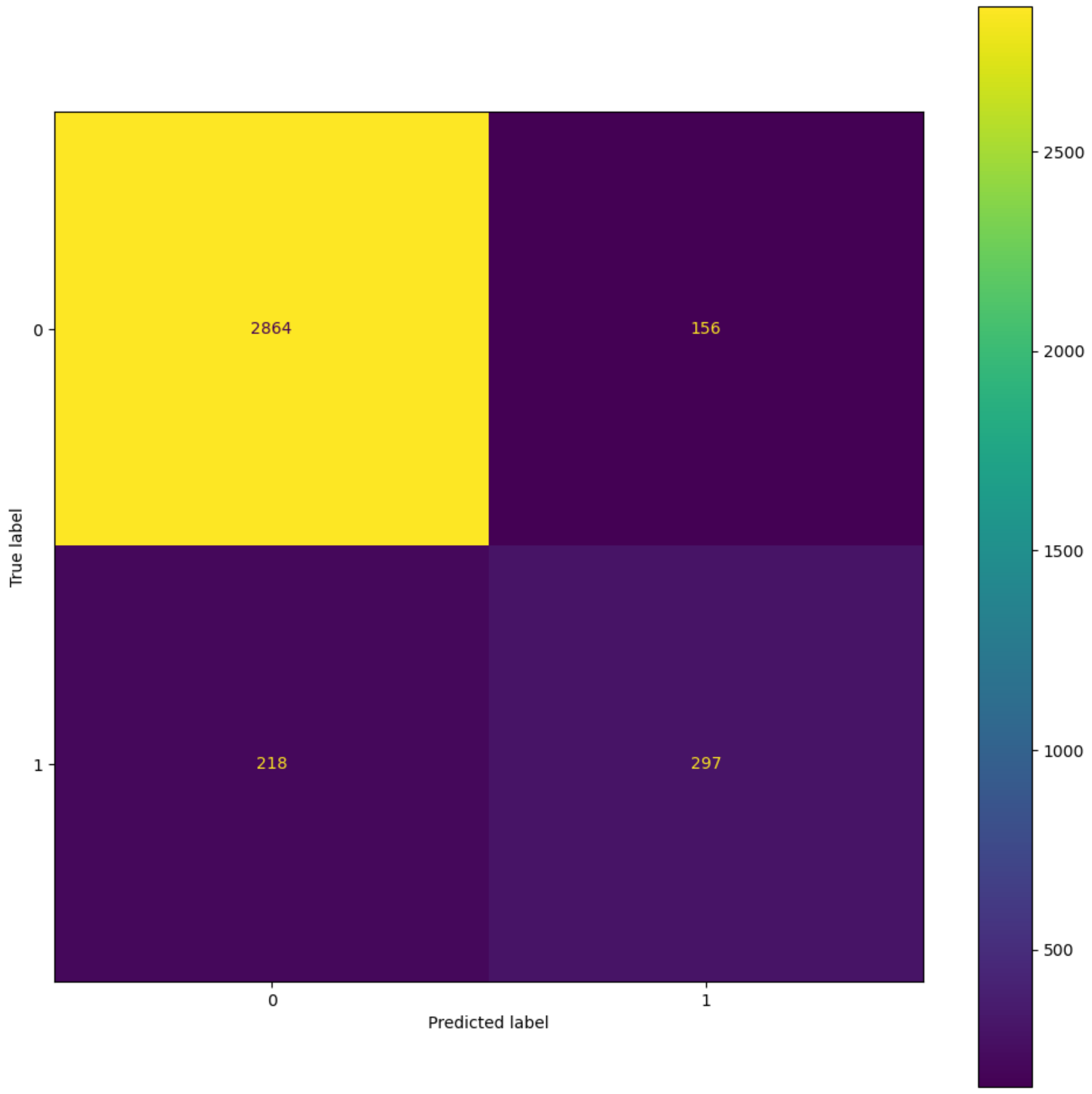
It also shows that most of the data points are labeled as 0. The dataset is imbalanced, which can impact the model's performance.

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

In [ ]:
```
clf_ada_500_g05 = AdaBoostClassifier(n_estimators=500, random_state=3316)

ada_fit_g05 = clf_ada_500_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
OSIy_pred_500_g05 = ada_fit_g05.predict(OSIX_test_g05)

Ada_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_500_g05)
print("AdaBoosting Classifier Accuracy: {:.2%}".format(Ada_accuracy_g05))
print(classification_report(OSIy_test_g05, OSIy_pred_500_g05))

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")

conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_500_g05)
ConfusionMatrixDisplay.from_estimator(ada_fit_g05, OSIX_test_g05, OSIy_test_g05)
```

```
AdaBoosting Classifier Accuracy: 89.42%
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      3020
           1       0.66      0.58      0.61       515

    accuracy                           0.89      3535
   macro avg       0.79      0.76      0.78      3535
weighted avg       0.89      0.89      0.89      3535
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x234f627ee50>

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



The Ada boosting model with 500 number of estimators performs better with a higher accuracy 89.42% than 100 number of estimators with the accuracy of 89.22%.

As the number of estimators increase from 100 to 500, the predictions on both true positive and negative do not change much.

Gradient Boost

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

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

Import the GradientBoostingClassifier class from ensemble module of sklearn library.

Create the appropriate classifier and describe what the syntax represents and what parameters you choose (1.5)

In [ ]:
```
OSI_clf_gb_g05 = GradientBoostingClassifier(n_estimators=100, random_state=3316)
```

What the syntax above does is initializing the gradient boosting classifier. Here are two parameters that are choosen for initialization.

- `n_estimators` specifies the number of boosting stages to perform. It's set to 100 here, meaning that 100 boosting iterations should be processed.
- `random_state` is set to 3316 which represents the GUID of one of the group members in order to control the random permutation of the features at each split and control the random splitting of the training data to ensure that we can get the same result with same random state number.

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

In [ ]:
```
OSI_gb_fit_g05 = OSI_clf_gb_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
```

Feed training dataset to the classifier to train the initialized model.

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

In [ ]:
```
OSIy_pred_g05 = OSI_gb_fit_g05.predict(OSIX_test_g05)
```

Based on the trained/fitted model, here is making predictions for the test predictors.

Calculate accuracy and explain what you did. (1.5)

In [ ]:
```
OSI_gb_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_g05)
print("GradientBoost Classifier Accuracy: {:.2%}".format(OSI_gb_accuracy_g05))
```

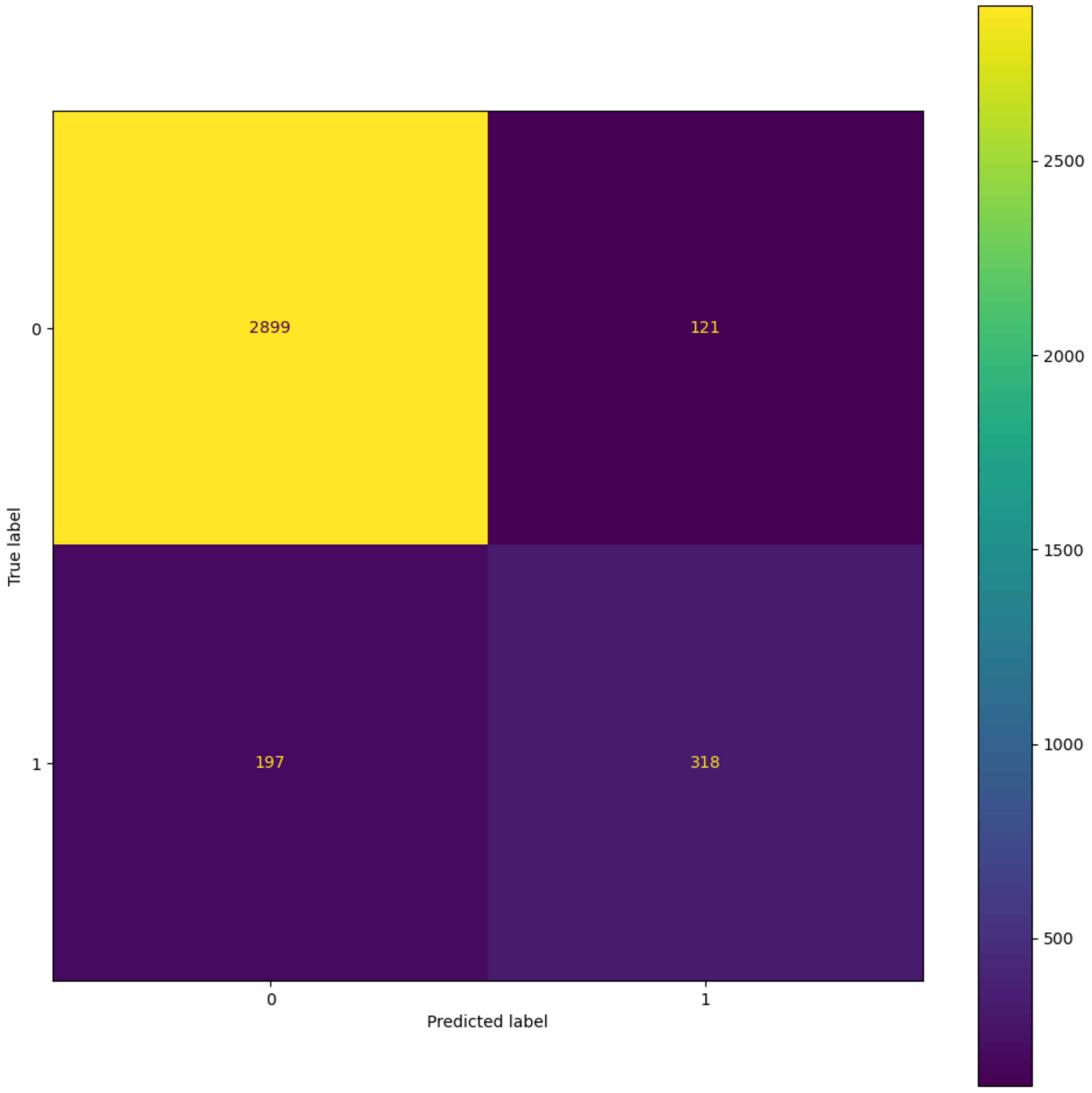
```
GradientBoost Classifier Accuracy: 91.00%
```

Taking `y_true` and `y_pred` as parameters, `accuracy_score` is used to measure the proportion of correctly predicted labels or classes by a classification model. Then we get the accuracy of Gradient Boosting model at 91%.

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

In [ ]:
```
plt.figure(figsize=(2,2),dpi=75)
set_palette("Paired")
OSI_gb_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_gb_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_pred_g05))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      3020
           1       0.72      0.62      0.67       515

    accuracy                           0.91      3535
   macro avg       0.83      0.79      0.81      3535
weighted avg       0.91      0.91      0.91      3535

<Figure size 150x150 with 0 Axes>
```

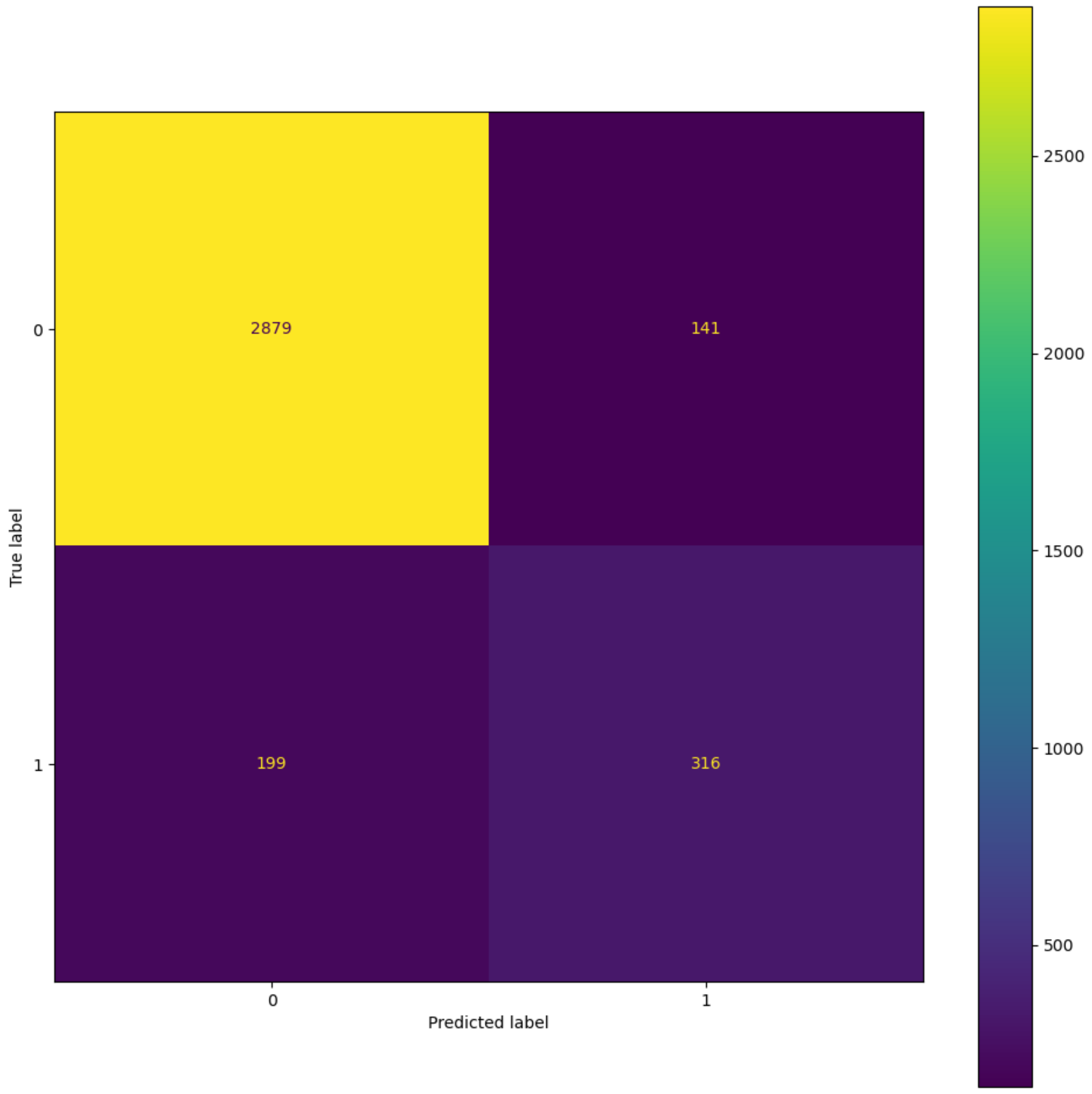


The overall accuracy of the gradient boosting classifier is decent. It's especially good at making predictions for lable 0. However, for label 1, both precision and recall are relatively low. Low precision means that a relatively small proportion of the positive predictions made by the classifier for that label were correct, and low recall means that the classifier is failing to capture a significant portion of the actual positive instances for that label. Low precision and recall lead to higher false positive and false negtive respectively.

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

In [ ]:
```
# Initialize classifier
OSI_clf_gb_g05 = GradientBoostingClassifier(n_estimators=500, random_state=3316)
# Train the model using training data
OSI_gb_fit_g05 = OSI_clf_gb_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
# Make predictions for X_test
OSIy_pred_g05 = OSI_gb_fit_g05.predict(OSIX_test_g05)
# Accuracy
OSI_gb_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_g05)
print("GradientBoost Classifier Accuracy: {:.2%}".format(OSI_gb_accuracy_g05))
# Plot confusion matrix
OSI_gb_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_gb_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_pred_g05))
```

```
GradientBoost Classifier Accuracy: 90.38%
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      3020
           1       0.69      0.61      0.65       515

    accuracy                           0.90      3535
   macro avg       0.81      0.78      0.80      3535
weighted avg       0.90      0.90      0.90      3535
```



After changing the number of boosting iterations (`n_estimators`) to 500, the accuracy of this model slightly decreases to 90.38% from 91%, which may indicate the beginning of overfitting.

XG Boost

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

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

Import the XGBClassifier class from xgboost library.

Create the appropriate classifier and describe what the syntax represents and what parameters you choose (1.5)

In [ ]:
```
OSI_clf_xgb_g05 = XGBClassifier(n_estimators=100, random_state=3316)
```

What the syntax above does is initializing the XGBoost classifier. Here are two parameters that are choosen for initialization.

- `n_estimators` specifies the number of gradient boosted trees, which is equivalent to number of boosting rounds. It's set to 100 here, meaning that 100 boosting iterations should be processed.
- `random_state` is set to 3316 which represents the GUID of one of the group members in order to control the random permutation of the features at each split and control the random splitting of the training data to ensure that we can get the same result with same random state number.

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

In [ ]:
```
OSI_xgb_fit_g05 = OSI_clf_xgb_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
```

Feed training dataset to the classifier to train the initialized model.

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

In [ ]:
```
OSIy_pred_g05 = OSI_xgb_fit_g05.predict(OSIX_test_g05)
```

Based on the trained/fitted model, here is making predictions for the test predictors.

Calculate accuracy and explain what you did. (1.5)

In [ ]:
```
OSI_xgb_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_g05)
print("XGBoost Classifier Accuracy: {:.2%}".format(OSI_xgb_accuracy_g05))
```

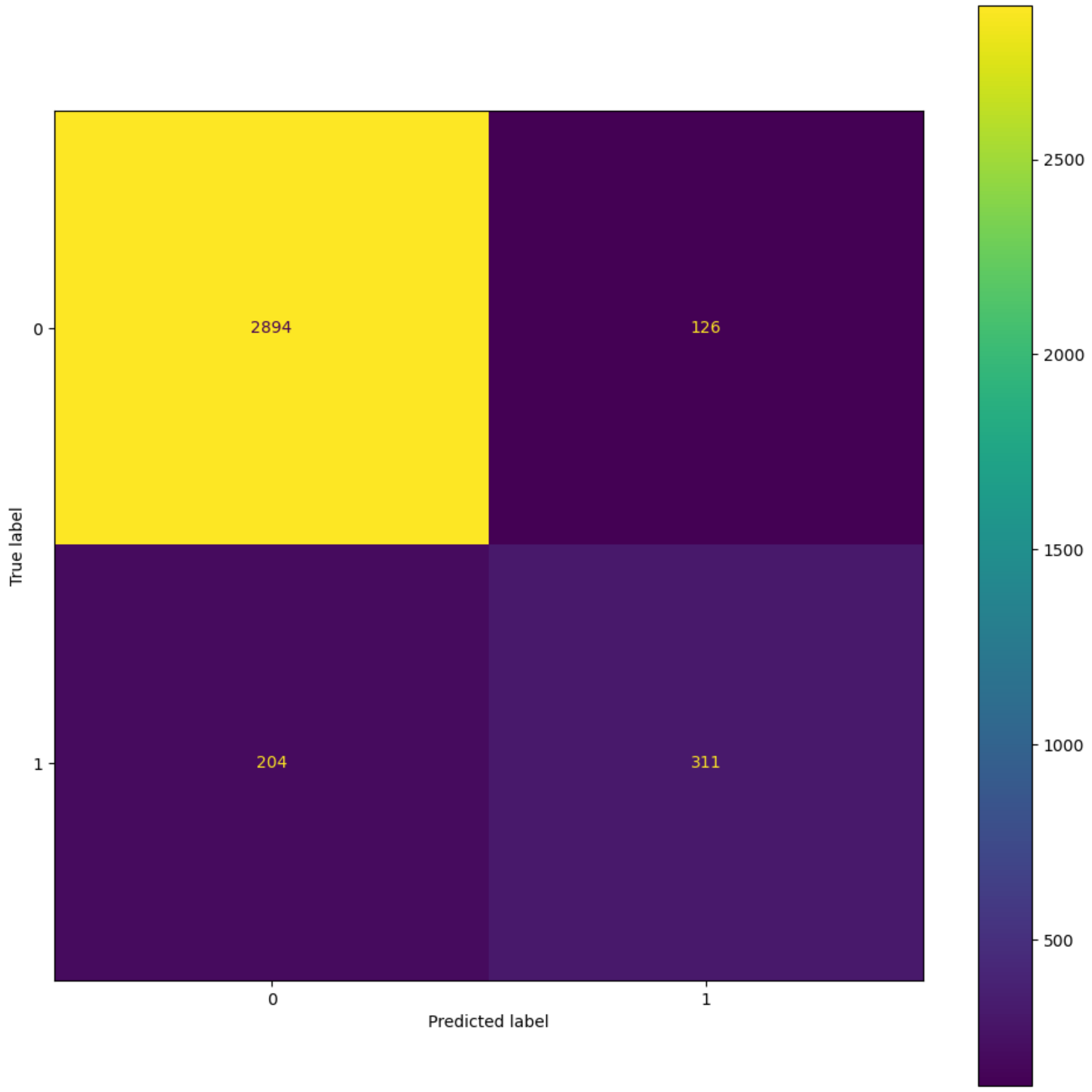
```
XGBoost Classifier Accuracy: 90.66%
```

Taking `y_true` and `y_pred` as parameters, `accuracy_score` is used to measure the proportion of correctly predicted labels or classes by a classification model. Then we get the accuracy of XGBoost model at 90.66%.

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

In [ ]:
```
OSI_xgb_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_xgb_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_pred_g05))
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      3020
           1       0.71      0.60      0.65       515

    accuracy                           0.91      3535
   macro avg       0.82      0.78      0.80      3535
weighted avg       0.90      0.91      0.90      3535
```



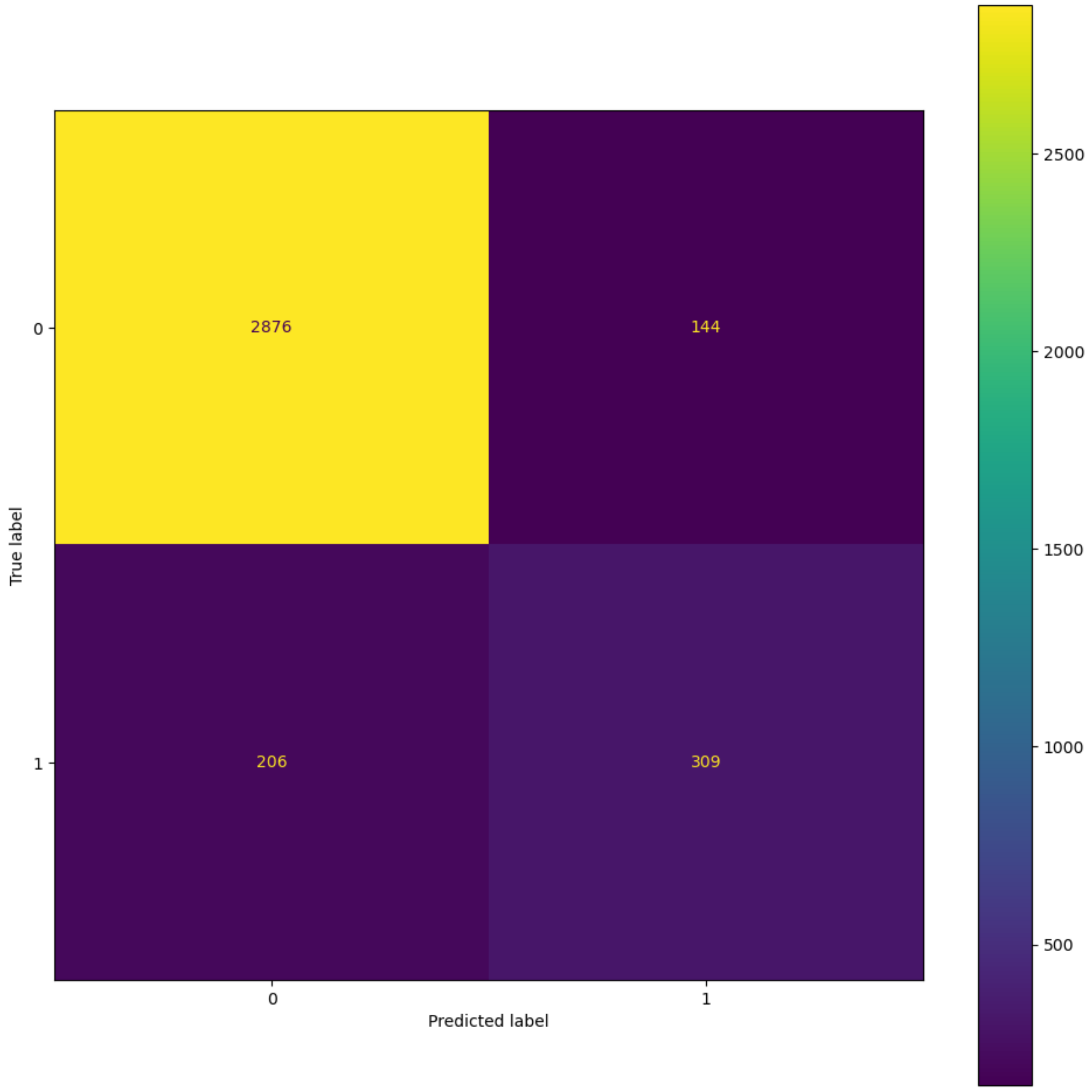
The accuracy at 90.66% indicates the model is performing well in terms of overall correctness in its predictions. Also, the high precision and recall means the model is minimizing false positives and false negatives as well.

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

```
In [ ]: OSI_clf_xgb_g05 = XGBClassifier(n_estimators=500, random_state=3316)
        OSI_xgb_fit_g05 = OSI_clf_xgb_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
```

```
OSIy_pred_g05 = OSI_xgb_fit_g05.predict(OSIX_test_g05)
OSI_xgb_accuracy_g05 = accuracy_score(y_true=OSIy_test_g05, y_pred=OSIy_pred_g05)
print("GradientBoost Classifier Accuracy: {:.2%}".format(OSI_xgb_accuracy_g05))
OSI_xgb_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_xgb_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_pred_g05))
```

```
GradientBoost Classifier Accuracy: 90.10%
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      3020
           1       0.68      0.60      0.64       515

    accuracy                           0.90      3535
   macro avg       0.81      0.78      0.79      3535
weighted avg       0.90      0.90      0.90      3535
```



While the number of iteration rounds increases, the accuracy experiences a slight decrease, coinciding with a reduction in precision for label 1. This phenomenon may stem from the dominance of label 0, leading the model to place more emphasis on data associated with label 0. Consequently, it becomes more prone to producing predictions of 0.

Bagging

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

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

Import the RandomForestClassifier class from scikit-learn, which is an ensemble learning method based on bagging.

Create the appropriate classifier and describe what the syntax represents and what parameters you choose (1.5)

```
In [ ]: OSI_clf_bag_g05 = RandomForestClassifier(n_estimators=100,
                                                  max_depth=None,
                                                  random_state=3316)
```

Create a RandomForestClassifier object with the following parameters:

`n_estimators` : The number of decision trees in the ensemble setting to 100.

`max_depth` : The maximum depth of the individual decision trees. Here, I've set it to None, which means the trees will expand until they contain less than min_samples_split samples in each leaf node.

`random_state` : A random seed for reproducibility.

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

```
In [ ]: OSI_bag_fit_g05=OSI_clf_bag_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
```

`OSI_clf_bag_g05` classifier will be used to train the bagging model with input variables training x `OSIX_train_g05` , on target label training y `OSIy_train_g05` .

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

```
In [ ]: OSIy_bag_pred_g05 = OSI_clf_bag_g05.predict(OSIX_test_g05)
```

Use the predict method of `OSI_clf_bag_g05` classifier to obtain predictions for the test data `OSIX_test_g05` .

Calculate accuracy and explain what you did. (1.5)

In [ ]:
```
from sklearn.metrics import accuracy_score

OSI_bag_accuracy_g05 = accuracy_score(OSIy_test_g05, OSIy_bag_pred_g05)
print("Bagging Classifier Accuracy: {:.2%}".format(OSI_bag_accuracy_g05))
```

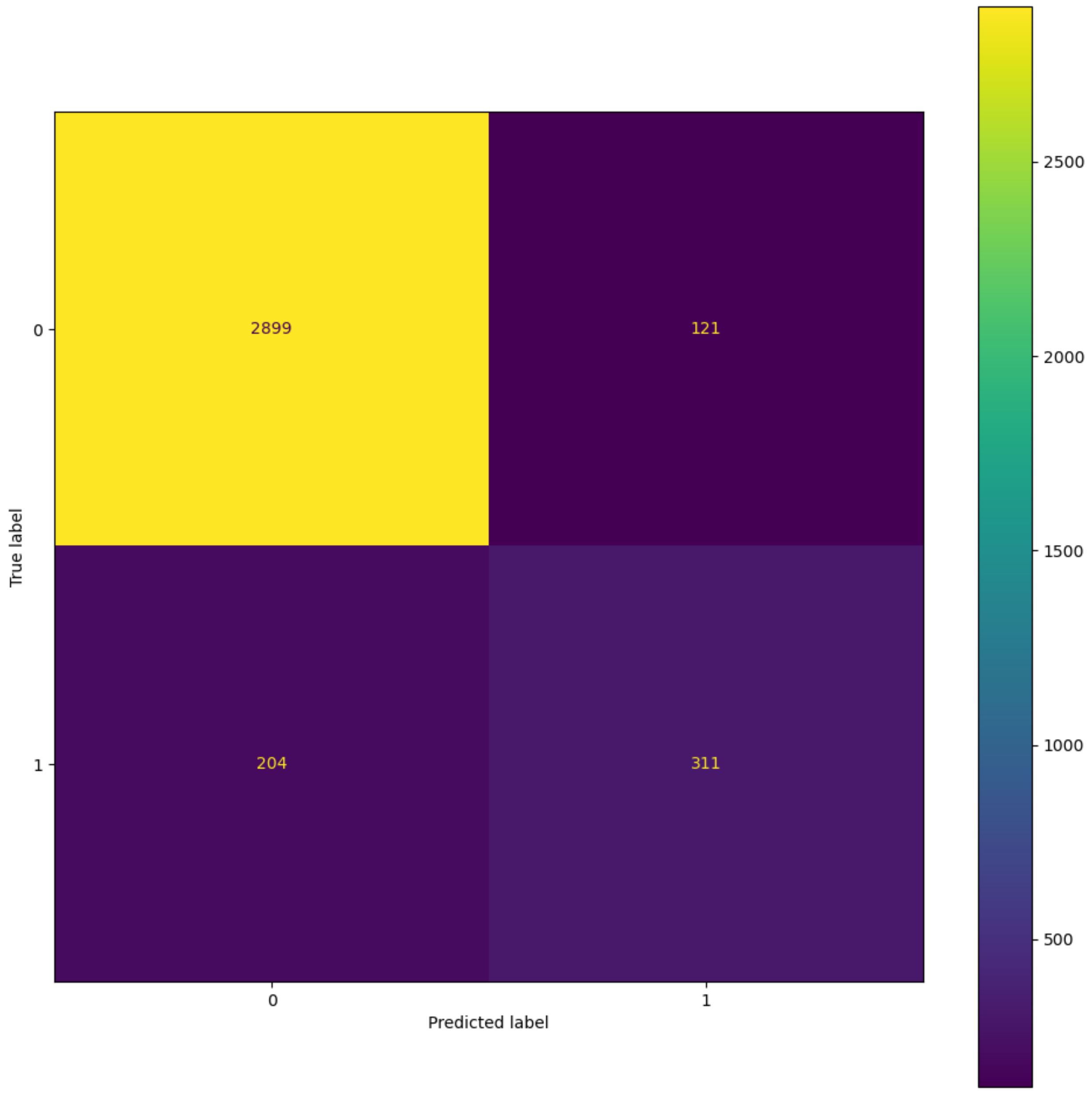
```
Bagging Classifier Accuracy: 90.81%
```

Use the `accuracy_score` function to calculate the accuracy of the classifier by comparing the predicted labels (`OSIy_bag_pred_g05`) with the true labels (`OSIy_test_g05`)

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

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

plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
OSI_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_bag_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_bag_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_bag_pred_g05))
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      3020
           1       0.72      0.60      0.66       515

    accuracy                           0.91      3535
   macro avg       0.83      0.78      0.80      3535
weighted avg       0.90      0.91      0.90      3535

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



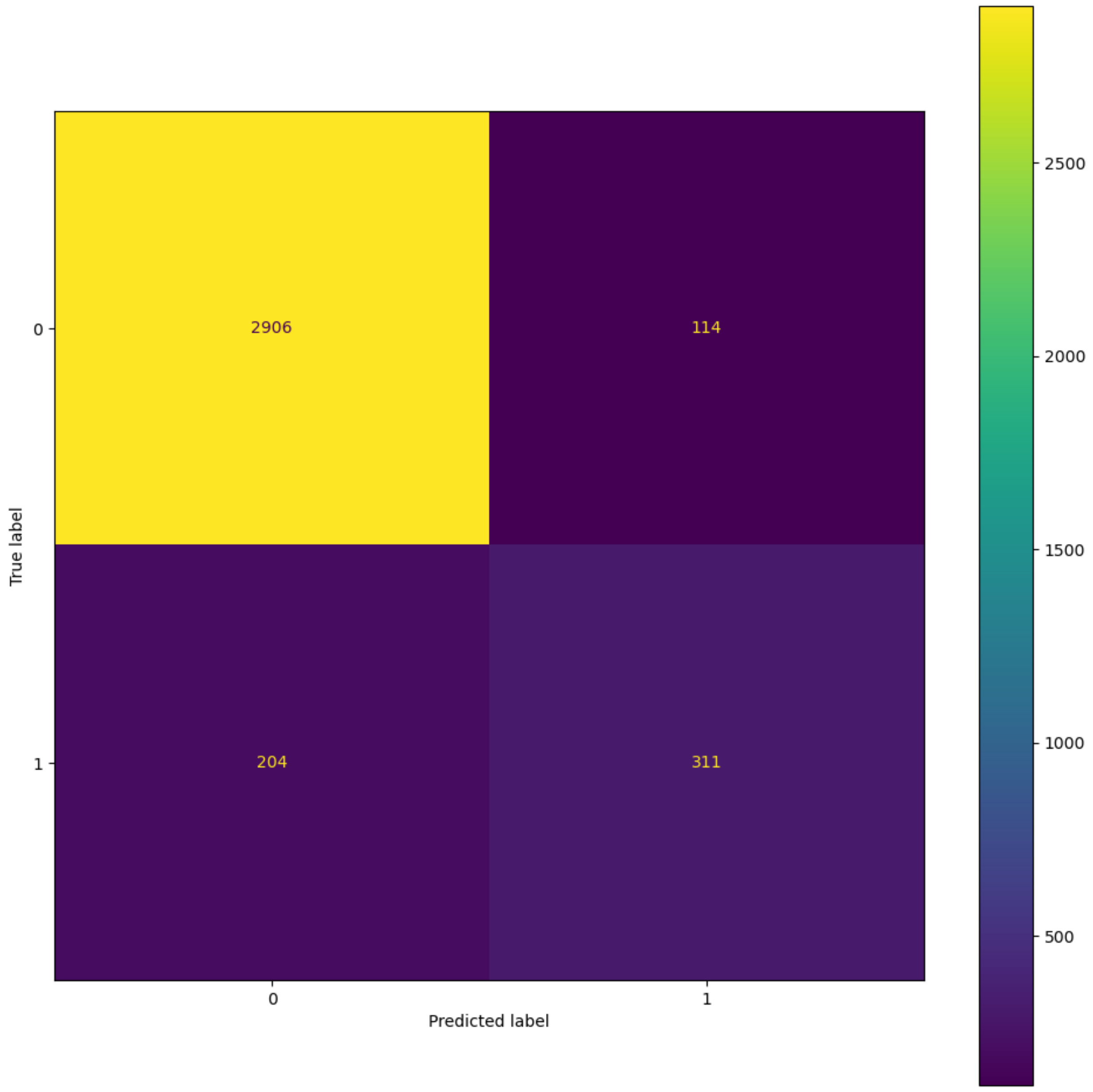
The overall acuracy is 90.81%. This classification report indicates that the model performs well for class 0 with high `precision`, `recall`, and `F1-score`. However, for class 1, the precision and recall are lower, indicating that there might be some difficulty in correctly classifying instances of class 1, possibly due to class imbalance or other factors.

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

In [ ]:
```
OSI_clf_bag_g05 = RandomForestClassifier(n_estimators=500,
                                         max_depth=None,
                                         random_state=3316)
OSI_bag_fit_g05=OSI_clf_bag_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
OSIy_bag_pred_g05 = OSI_clf_bag_g05.predict(OSIX_test_g05)
OSI_bag_accuracy_g05 = accuracy_score(OSIy_test_g05, OSIy_bag_pred_g05)
print("Bagging Classifier Accuracy: {:.2%}".format(OSI_bag_accuracy_g05))
plt.figure(figsize=(2.5,2.5),dpi=75)
set_palette("Paired")
```

```
OSI_conf_matrix_g05 = confusion_matrix(OSIy_test_g05, OSIy_bag_pred_g05)
ConfusionMatrixDisplay.from_estimator(OSI_bag_fit_g05, OSIX_test_g05, OSIy_test_g05)
print(classification_report(OSIy_test_g05, OSIy_bag_pred_g05))
```

```
Bagging Classifier Accuracy: 91.00%
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      3020
           1       0.73      0.60      0.66       515

    accuracy                           0.91      3535
   macro avg       0.83      0.78      0.80      3535
weighted avg       0.90      0.91      0.91      3535

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



With the number of estimators improving, the new results show a slight improvement in precision for class 0 and a slight improvement in recall for class 1. The macro average metrics also show slight improvements. Overall, the two sets of results are quite similar, with minor differences in precision and recall. These changes may not be substantial and could be within the variability of the model's performance.

Regression (22 Points)

Gradient Boost

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

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

Create the appropriate regressor and describe what the syntax represents and what parameters you choose (1.5)

```
In [ ]: gb_regressor_g05 = GradientBoostingRegressor(n_estimators=100,
                                                     learning_rate=0.1,
                                                     max_depth=3,
                                                     random_state=3316,
                                                     loss='squared_error')
```

n_estimators=100: We're specifying the boosting process to consist of 100 trees. Depending on the problem, this might need to be adjusted.

learning_rate=0.1: This means each tree will contribute 10% of its prediction as a correction to the previous tree's errors.

max_depth=3: The trees will be of a maximum depth of 3, preventing them from becoming too complex and overfitting the data.

random_state=3316: Ensures reproducibility of results.

loss='squared_error': This specifies that we're using the least squares regression as the loss function to be minimized.

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

```
In [ ]: gb_regressor_g05.fit(BSHX_train_g05, BSHy_train_g05.ravel())
```

```
Out[ ]: ▾ GradientBoostingRegressor
        GradientBoostingRegressor(random_state=3316)
```

By training the regressor on the BSHX_train_g05 and BSHy_train_g05 datasets, we're teaching it to understand the patterns in the bike sharing data. This trained model can then predict the number of bike rentals

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

```
In [ ]: # Predict on Test Data
BSHy_pred_g05 = gb_regressor_g05.predict(BSHX_test_g05)
```

Im making predictions using the Gradient Boost regressor on the test set BSHX_test_g05 and store the predictions in BSHy_pred_g05.

Calculate MSE,MAE,R^2 and explain what you did. (1.5)

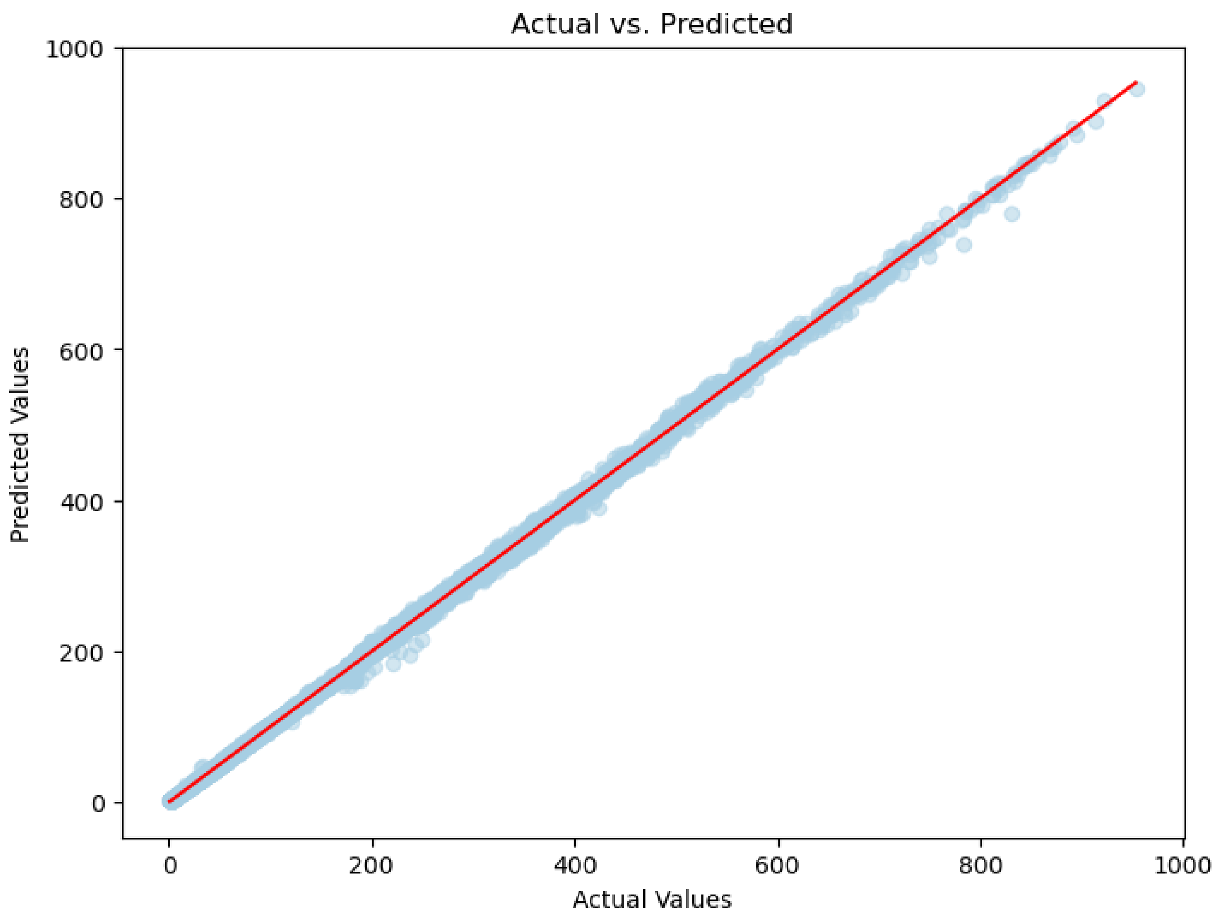
```
In [ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


mse_g05 = mean_squared_error(BSHy_test_g05, BSHy_pred_g05)
mae_g05 = mean_absolute_error(BSHy_test_g05, BSHy_pred_g05)
r2_g05 = r2_score(BSHy_test_g05, BSHy_pred_g05)

print(f"Mean Squared Error (MSE): {mse_g05}")
print(f"Mean Absolute Error (MAE): {mae_g05}")
print(f"R^2 Score: {r2_g05}")
```

```
Mean Squared Error (MSE): 24.436737029647364
Mean Absolute Error (MAE): 3.0599832861618785
R^2 Score: 0.9992338160525185
```

I calcuated the MSE, MAE and R^2 and those stats can tell us how good our model is. R^2 score is approximately 0.9992, which is very close to 1. This indicates that the model explains 99.92% of the variance in the target variable. Such a high R^2 score typically suggests an excellent fit to the data

Show both text and visual Actual value vs Predicted values using scikit learn and matplotlib and explain what the graph tells you and what you did. (2.5)

```
In [ ]: def plot_actual_vs_predicted(y_true_g05, y_pred_g05):
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true_g05, y_pred_g05, alpha=0.5)
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title('Actual vs. Predicted')
    plt.plot([min(y_true_g05), max(y_true_g05)], [min(y_true_g05), max(y_true_g05)], 
    plt.show()

plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
```



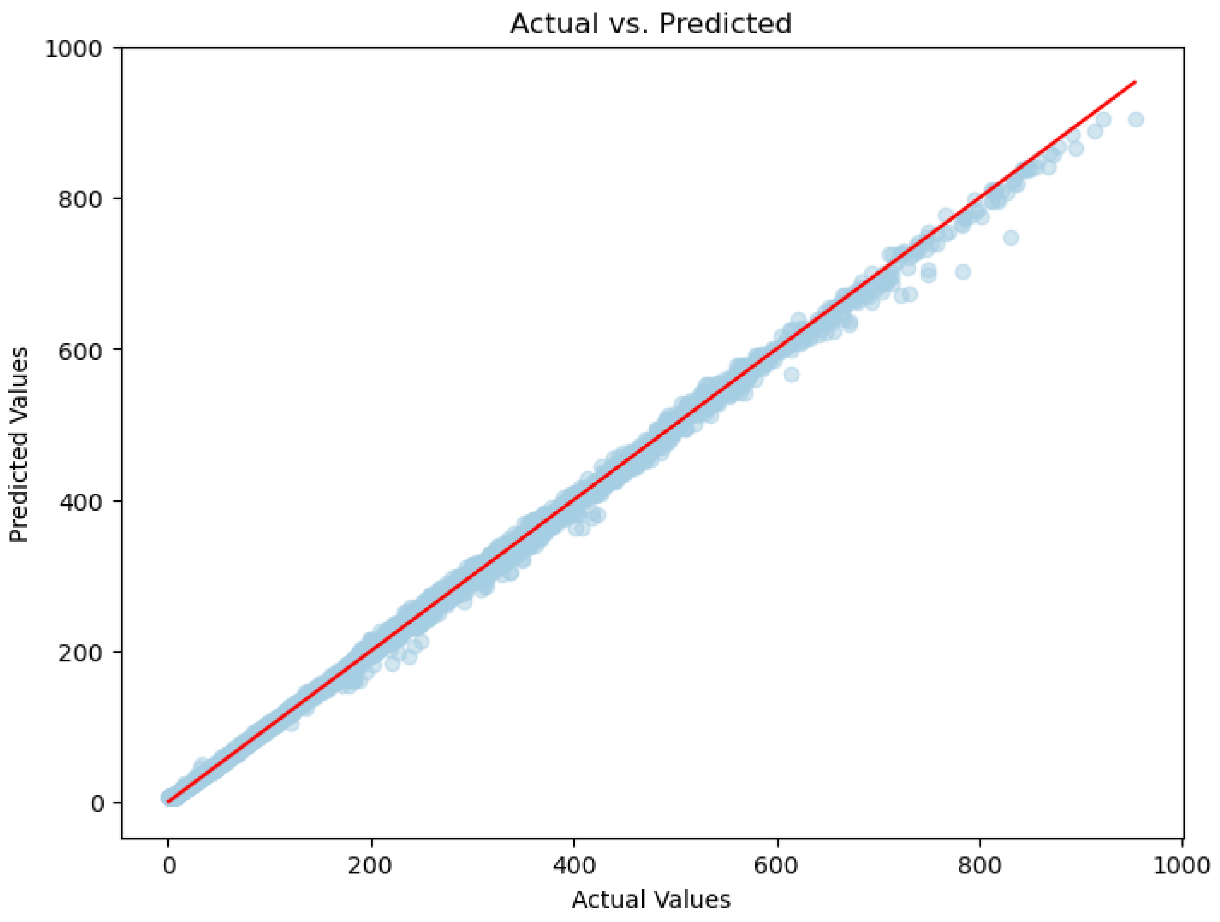
In [ ]:
```
import numpy as np
def regression_metrics(y_true_g05, y_pred_g05):
    """
    Calculate common regression metrics.

    Parameters:
    - y_true: Actual target values.
    - y_pred: Predicted target values.

    Returns:
    - A dictionary containing MAE, MSE, RMSE, and R^2.
    """

    mae_g05 = mean_absolute_error(y_true_g05, y_pred_g05)
    mse_g05 = mean_squared_error(y_true_g05, y_pred_g05)
    rmse_g05 = np.sqrt(mse_g05)
    r2_g05 = r2_score(y_true_g05, y_pred_g05)

    return {
        'MAE': mae_g05,
        'MSE': mse_g05,
        'RMSE': rmse_g05,
        'R^2': r2_g05
    }

print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```

```
{'MAE': 3.0599832861618785, 'MSE': 24.436737029647364, 'RMSE': 4.943352812580482, 'R^
2': 0.9992338160525185}
```

The scatter plot of actual versus predicted values reveals that most data points lie along a 45-degree line, signifying that the predicted values are in close agreement with the true values. This pattern is further corroborated by a high R^2 value, emphasizing that our model has successfully captured a significant proportion of the variance in the target variable. Such alignment between actual and predicted values, coupled with the high R^2, underscores the robustness and predictive accuracy of our regression model

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

In [ ]:
```
gb_regressor_g05 = GradientBoostingRegressor(n_estimators=50,
                                              learning_rate=0.1,
                                              max_depth=3,
                                              random_state=3316,
                                              loss='squared_error')
gb_regressor_g05.fit(BSHX_train_g05, BSHy_train_g05.ravel())

BSHy_pred_g05 = gb_regressor_g05.predict(BSHX_test_g05)
plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```



```
{'MAE': 3.9418149962768307, 'MSE': 40.87939074722536, 'RMSE': 6.3936993006572775, 'R^
2': 0.9987182767922186}
```

MAE, MSE, and RMSE are slightly higher for a regressor with a 100 number of estimators compared to one with 50 estimators. R^2 is slightly lower for 50 estimator model compared to

100 estimator model, it means that the model explains a smaller proportion of the variance in the dependent variable.

XG Boost

Import appropriate algorithm from xgboost and explain what you did. (1.5)

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

I import XGBoost in the Python code

Create the appropriate regressor and describe what the syntax represents and what parameters you choose (1.5)

In [ ]:
```
xgb_regressor_g05 = xgb.XGBRegressor(learning_rate=0.05, random_state=3316, objective=
```

Im initializing an XGBRegressor from the xgboost library with a specified learning rate = 0.05 and objective function = square error.

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

In [ ]:
```
xgb_regressor_g05.fit(BSHX_train_g05, BSHy_train_g05)
```

Out[ ]:
```
▾ XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
             enable_categorical=False, eval_metric=None, feature_types=No
ne,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=No
ne,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
```

By training the xg boost regressor on the BSHX_train_g05 and BSHy_train_g05 datasets, we're teaching it to understand the patterns in the bike sharing data. This trained model can then predict the number of bike rentals

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

In [ ]:
```
BSHy_pred_g05 = xgb_regressor_g05.predict(BSHX_test_g05)
```

Calcuated the number of bkie rentals based on the test data

Calculate MSE,MAE,R^2 and explain what you did. (1.5)

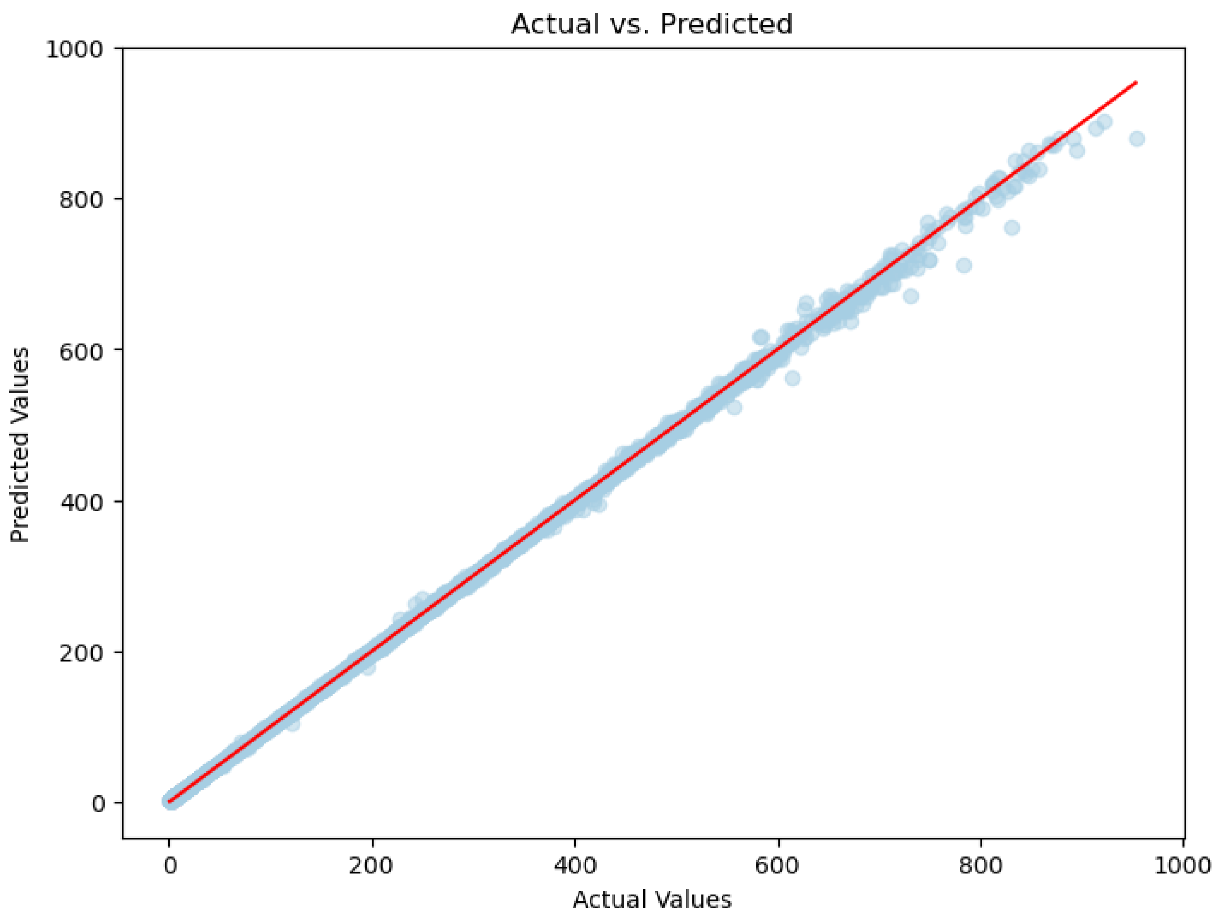
In [ ]:
```
mse_g05 = mean_squared_error(BSHy_test_g05, BSHy_pred_g05)
mae_g05 = mean_absolute_error(BSHy_test_g05, BSHy_pred_g05)
r2_g05 = r2_score(BSHy_test_g05, BSHy_pred_g05)

print(f"Mean Squared Error (MSE): {mse_g05}")
print(f"Mean Absolute Error (MAE): {mae_g05}")
print(f"R^2 Score: {r2_g05}")
```

```
Mean Squared Error (MSE): 16.278192097956545
Mean Absolute Error (MAE): 2.0047899855019993
R^2 Score: 0.9994896172322703
```

I calcuated the MSE, MAE and R^2 and those stats can tell us how good our model is. R^2 score is approximately 0.9994, which is very close to 1. This indicates that the model explains 99.94% of the variance in the target variable. Such a high R^2 score typically suggests an excellent fit to the data

Show both text and visual Actual value vs Predicted values using scikit learn and matplotlib and explain what the graph tells you and what you did. (2.5)

In [ ]:
```
def plot_actual_vs_predicted(y_true_g05, y_pred_g05):
    plt.figure(figsize=(8, 6))
    plt.scatter(y_true_g05, y_pred_g05, alpha=0.5)
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title('Actual vs. Predicted')
    plt.plot([min(y_true_g05), max(y_true_g05)], [min(y_true_g05), max(y_true_g05)], 
    plt.show()

plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
```



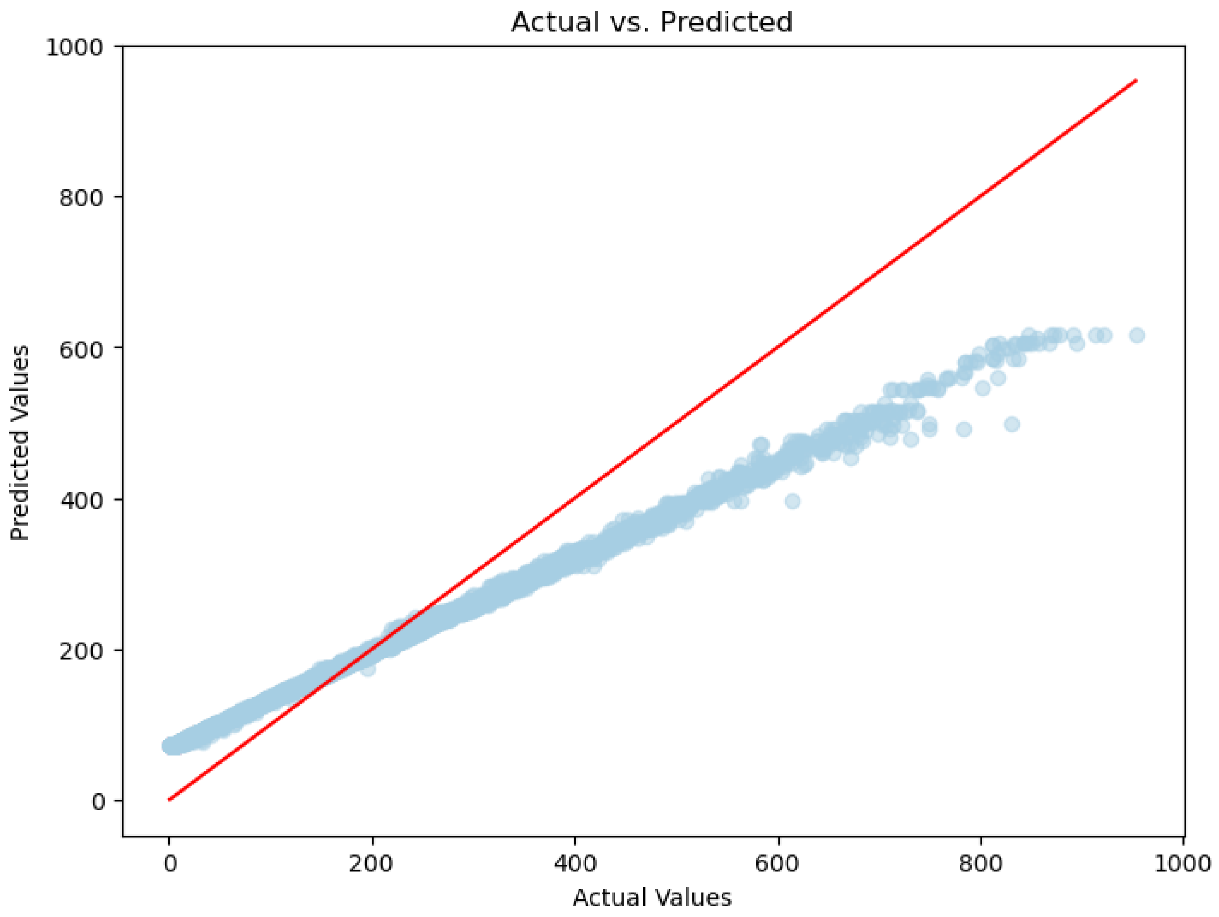
In [ ]:
```
print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```

```
{'MAE': 2.0047899855019993, 'MSE': 16.278192097956545, 'RMSE': 4.034624158203158, 'R^
2': 0.9994896172322703}
```

The near-perfect alignment of our data points with the 45-degree line on a scatter plot of actual versus predicted values is indicative of our model's predictive prowess. The strong R^2 value reinforces this, indicating that our model not only predicts well but also captures the essence of the data's variability effectively.

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

In [ ]:
```
# start a new regressor with leanring rate =0.01
xgb_regressor_g05 = xgb.XGBRegressor(learning_rate=0.01, random_state=3316, objective=
xgb_regressor_g05.fit(BSHX_train_g05, BSHy_train_g05.ravel())
BSHy_pred_g05 = xgb_regressor_g05.predict(BSHX_test_g05)
plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```



```
{'MAE': 52.233063488935755, 'MSE': 4441.035492520655, 'RMSE': 66.64109462276753, 'R^
2': 0.8607567736872176}
```

The model with a learning rate of 0.01 exhibits suboptimal performance. The actual vs. predicted plot reveals considerable deviations from true values. Additionally, metrics indicate deterioration: MAE, MSE, and RMSE have risen, while R^2 has declined. This suggests the model might be underfitting or the learning rate may be too conservative for effective convergence.

Bagging

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

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

Improt the bagging regressor from sklearn

Create the appropriate regressor and describe what the syntax represents and what parameters you choose (1.5)

In [ ]:
```
n_estimators = 100  # Number of base estimators in the ensemble
bagging_model_g05 = BaggingRegressor(n_estimators=n_estimators, random_state=3316)
```

Im initialized a Bagging Regressor with 100 base estimators

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

In [ ]:
```
bagging_model_g05.fit(BSHX_train_g05, BSHy_train_g05.ravel())
```

Out[ ]:
```
▾ BaggingRegressor
BaggingRegressor(n_estimators=100, random_state=3316)
```

By training the bagging regressor on the BSHX_train_g05 and BSHy_train_g05 datasets, we're teaching it to understand the patterns in the bike sharing data. This trained model can then predict the number of bike rentals

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

In [ ]:
```
# Predict on the test set
BSHy_pred_g05 = bagging_model_g05.predict(BSHX_test_g05)
```

predicted the number of bkie rentals based on the test data

Calculate MSE,MAE,R^2 and explain what you did. (1.5)

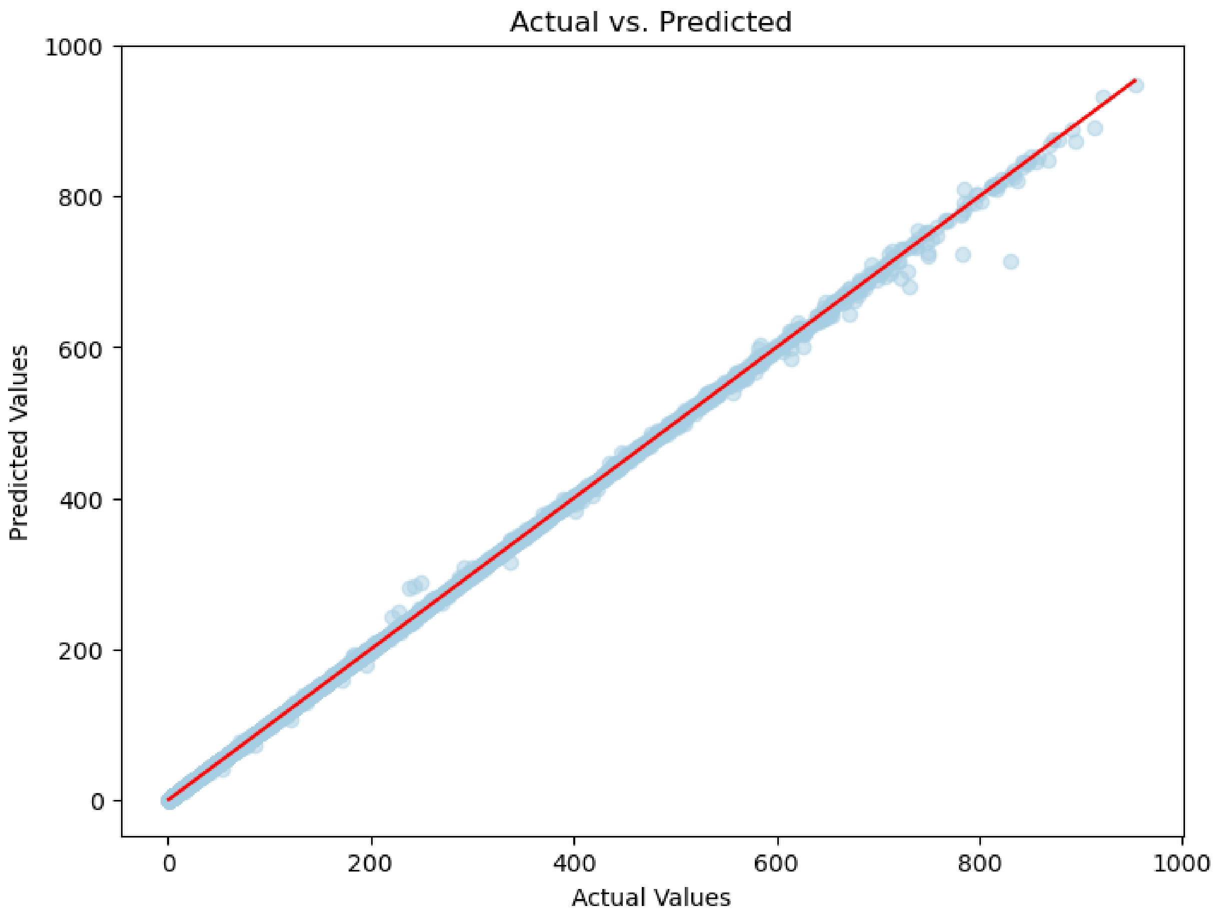
In [ ]:
```
# Compute MSE
mse_g05 = mean_squared_error(BSHy_test_g05, BSHy_pred_g05)
mae_g05 = mean_absolute_error(BSHy_test_g05, BSHy_pred_g05)
r2_g05 = r2_score(BSHy_test_g05, BSHy_pred_g05)

print(f"Mean Squared Error (MSE): {mse_g05}")
print(f"Mean Absolute Error (MAE): {mae_g05}")
print(f"R^2 Score: {r2_g05}")
```

```
Mean Squared Error (MSE): 9.988577560414276
Mean Absolute Error (MAE): 1.0733985423858845
R^2 Score: 0.9996868203894947
```

I calcuated the MSE, MAE and R^2 and those stats can tell us how good our model is. R^2 score is approximately 0.9996, which is very close to 1. This indicates that the model explains 99.96% of the variance in the target variable. Such a high R^2 score typically suggests an excellent fit to the data.

Show both text and visual Actual value vs Predicted values using scikit learn and matplotlib and explain what the graph tells you and what you did. (2.5)

In [ ]:
```
plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
```



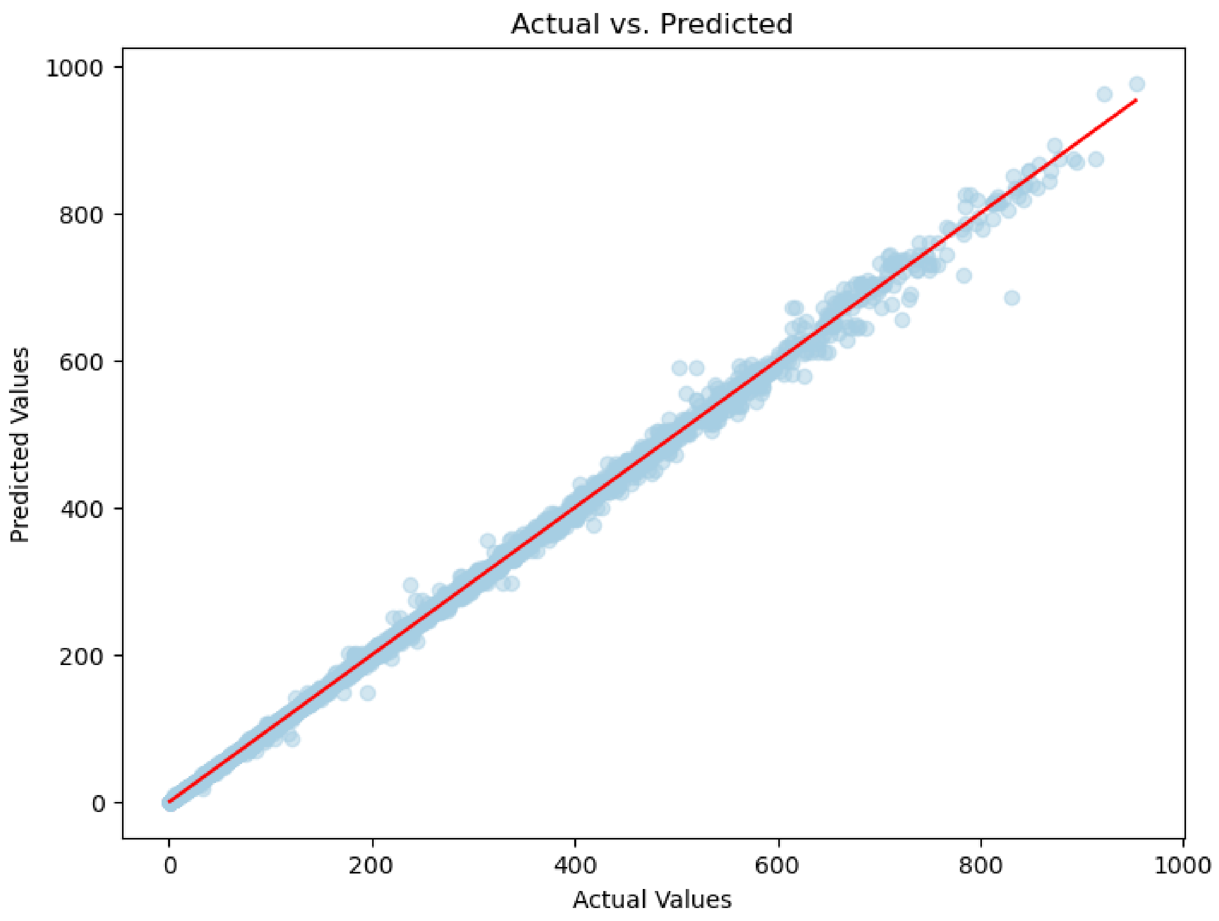
In [ ]:
```
print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```

```
{'MAE': 1.0733985423858845, 'MSE': 9.988577560414276, 'RMSE': 3.1604710978609307, 'R^
2': 0.9996868203894947}
```

When plotting our actual against predicted values, we observe a close alignment with a 45-degree angle, which is a hallmark of precise predictions. Our model's high R^2 value further attests to its ability to capture the inherent variability of the data. This combination of visual and quantitative metrics highlights the model's efficacy in making accurate forecasts.

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

In [ ]:
```
bagging_model_g05 = BaggingRegressor(n_estimators=1, random_state=3316)
bagging_model_g05.fit(BSHX_train_g05, BSHy_train_g05.ravel())
BSHy_pred_g05 = bagging_model_g05.predict(BSHX_test_g05)
plot_actual_vs_predicted(BSHy_test_g05,BSHy_pred_g05)
print(regression_metrics(BSHy_test_g05,BSHy_pred_g05))
```



```
{'MAE': 3.149597238204833, 'MSE': 45.22056003068661, 'RMSE': 6.724623411811743, 'R^
2': 0.9985821647485259}
```

MAE, MSE, and RMSE are slightly higher for a regressor with a 100 number of estimators compared to one with 50 estimators. R^2 is slightly lower for 1 estimator model compared to 100 estimator model, it means that the model explains a smaller proportion of the variance in the dependent variable.

Visually, the graph also highlights that there are more outliers or points deviating from the expected line, underscoring that model accuracy and fit may not necessarily improve with an increasing number of estimators.

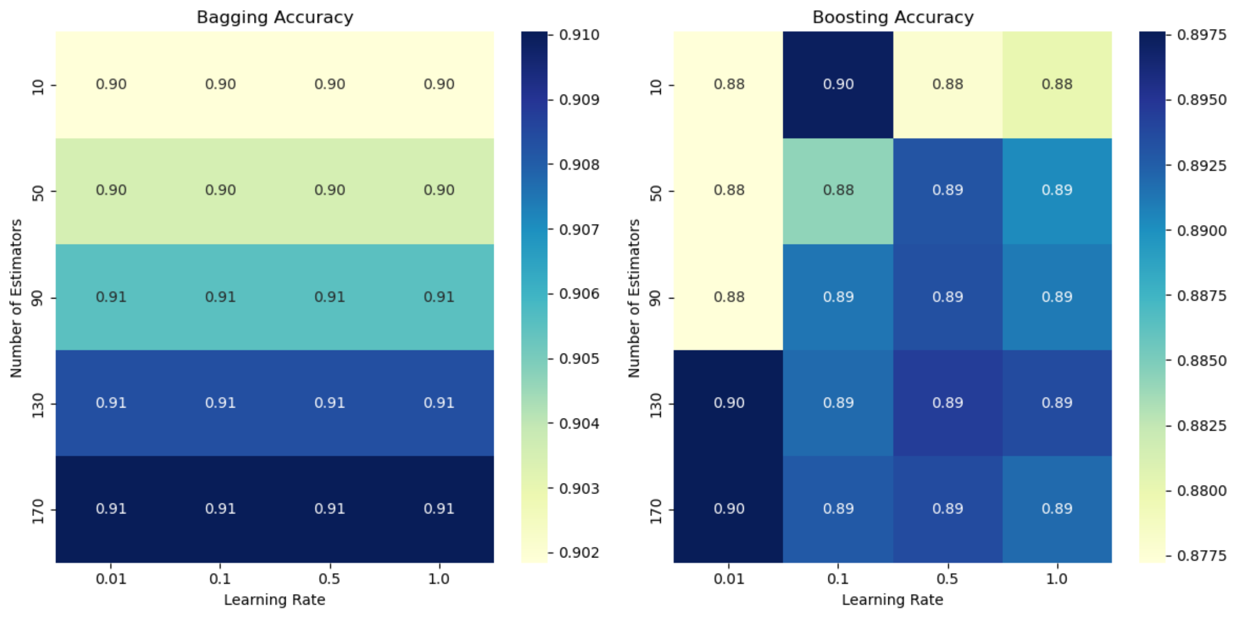
Bonus Question (5)

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

For boosting and bagging compare the tradeoff between njobs 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 tik labels, and x and y axis labels). (1)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score
import warnings
from sklearn.exceptions import DataConversionWarning

# Suppress DataConversionWarning
warnings.filterwarnings("ignore", category=DataConversionWarning)

# Define a range of n_estimators and learning rates to try
n_estimators_values_g05 = [10, 50, 90, 130, 170]
learning_rates_g05 = [0.01, 0.1, 0.5, 1.0]

bagging_accuracies_g05 = []
boosting_accuracies_g05 = []

# Iterate over different combinations of n_estimators and learning rates
for n_estimators_g05 in n_estimators_values_g05:
    for learning_rate_g05 in learning_rates_g05:
        # Bagging Classifier (Random Forest)
        clf_bag_g05 = RandomForestClassifier(n_estimators=n_estimators_g05,
                                             max_depth=None,
                                             random_state=3316)
        clf_bag_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
        y_bag_pred_g05 = clf_bag_g05.predict(OSIX_test_g05)
        bag_accuracy_g05 = accuracy_score(OSIy_test_g05, y_bag_pred_g05)
        bagging_accuracies_g05.append(bag_accuracy_g05)

        # Boosting Classifier (AdaBoost)
        clf_boost_g05 = AdaBoostClassifier(n_estimators=n_estimators_g05,
                                           learning_rate=learning_rate_g05,
                                           random_state=3316)
        clf_boost_g05.fit(OSIX_train_g05, OSIy_train_g05.ravel())
        y_boost_pred_g05 = clf_boost_g05.predict(OSIX_test_g05)
        boost_accuracy_g05 = accuracy_score(OSIy_test_g05, y_boost_pred_g05)
        boosting_accuracies_g05.append(boost_accuracy_g05)

# Reshape the accuracy lists into grids for plotting
bagging_accuracies_g05 = np.array(bagging_accuracies_g05)\
    .reshape(len(n_estimators_values_g05), len(learning_rates_g05))
boosting_accuracies_g05 = np.array(boosting_accuracies_g05)\
    .reshape(len(n_estimators_values_g05), len(learning_rates_g05))

# Create a heatmap for bagging
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.heatmap(bagging_accuracies_g05, annot=True, fmt=".2f", cmap="YlGnBu",
            xticklabels=learning_rates_g05, yticklabels=n_estimators_values_g05)
```

```
plt.title("Bagging Accuracy")
plt.xlabel("Learning Rate")
plt.ylabel("Number of Estimators")

# Create a heatmap for boosting
plt.subplot(1, 2, 2)
sns.heatmap(boosting_accuracies_g05, annot=True, fmt=".2f", cmap="YlGnBu",
            xticklabels=learning_rates_g05, yticklabels=n_estimators_values_g05)
plt.title("Boosting Accuracy")
plt.xlabel("Learning Rate")
plt.ylabel("Number of Estimators")

plt.tight_layout()
plt.show()
```



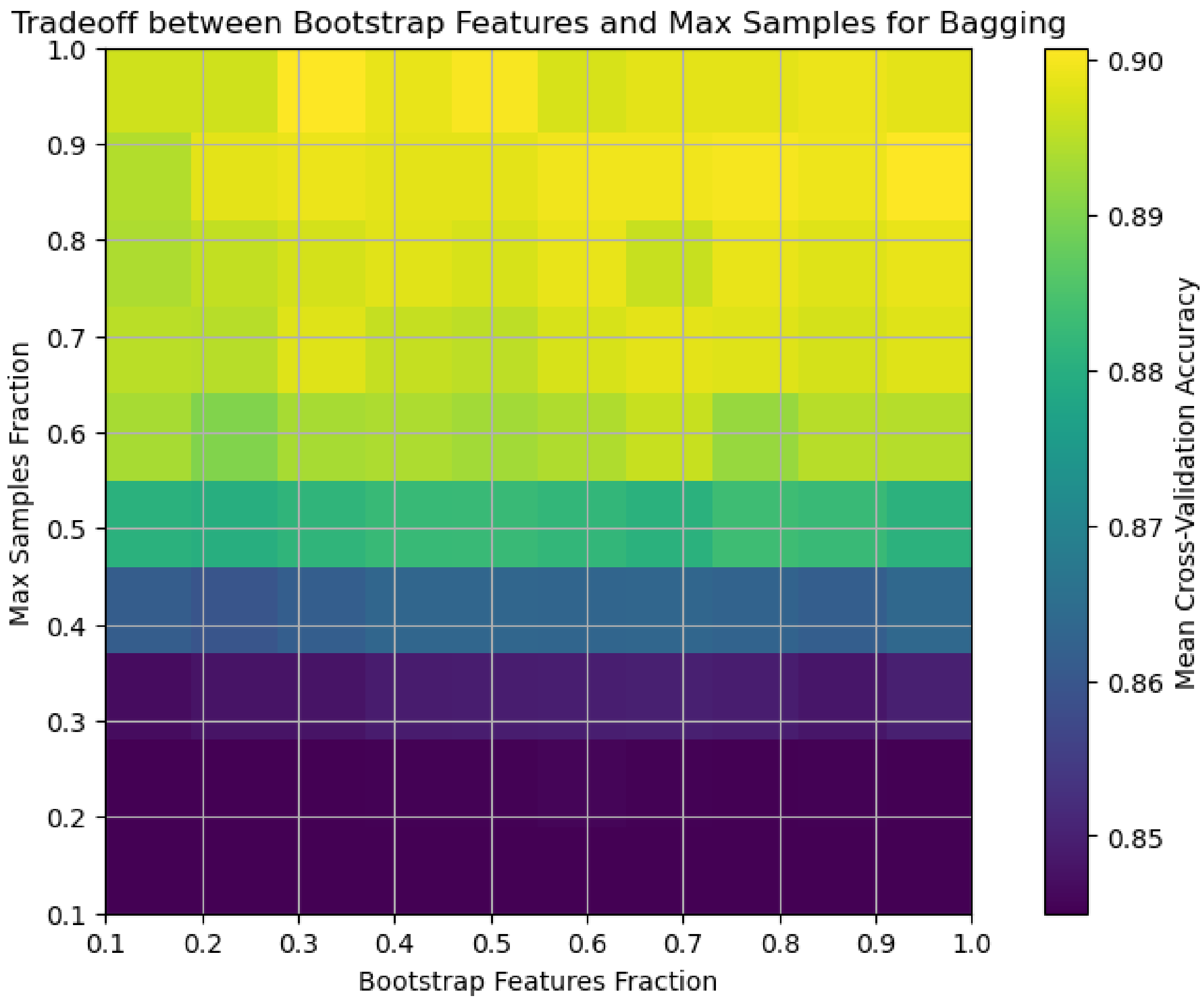
Explain the graph in details, specifically describe the trade off between thelearning rate and n jobs. Also comment on the eolution of error for each combination(1 paragraph at least, 1.5).

The provided code generates two heatmaps to show how ensemble classifiers, notably Bagging (Random Forest) and Boosting (AdaBoost), perform for various combinations of the two hyperparameters learning rate and the number of estimators (trees) in the ensemble.The accuracy of the Bagging Classifier (Random Forest) is depicted on the left heatmap as a function of the learning rate (x-axis) and the number of estimators (y-axis).Each heatmap pixel displays the accuracy score that the Bagging Classifier was able to attain for a particular learning rate and estimator count combination.

The resulting plots show that while there is a distinct and notable pattern emerging, the variation in accuracy across different learning rates is quite low. It becomes clear that both Bagging (Random Forest) and Boosting (AdaBoost) classifiers perform much better when there are more estimators. This pattern implies that the prediction accuracy improves steadily as we use more base models in the ensemble. This result highlights the significance of include a sufficient number of estimators to improve the overall accuracy of the ensemble models, even when fine-tuning other hyperparameters is crucial.Higher learning rates can speed up convergence but may also increase instability in the model's learning process. Learning rates, however, have a more subtle influence. Consequently, adjusting learning rates together with the the number of estimators is essential for achieving the best model performance.

For bagging compare the tradeoff 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 tik labels, and x and y axis labels). (1)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")

# Initialize variables for the plot
bootstrap_features_values_g05 = np.arange(0.1, 1.1, 0.1)
max_samples_values_g05 = np.arange(0.1, 1.1, 0.1)
scores_g05 = np.zeros((len(bootstrap_features_values_g05), len(max_samples_values_g05)

# Loop through different combinations of bootstrap features and max samples
for i_g05, bootstrap_features_g05 in enumerate(bootstrap_features_values_g05):
    for j_g05, max_samples_g05 in enumerate(max_samples_values_g05):
        # Create a BaggingClassifier with the current settings
        clf_g05 = BaggingClassifier(n_estimators=50,
                                    max_samples=max_samples_g05,
                                    max_features=bootstrap_features_g05,
                                    random_state=3316)

        # Calculate cross-validation score
        cv_scores_g05 = cross_val_score(clf_g05, OSIX_train_g05, OSIy_train_g05, cv=5
        scores_g05[i_g05, j_g05] = np.mean(cv_scores_g05)

# Create the heatmap
plt.figure(figsize=(10, 6))
plt.imshow(scores_g05, cmap='viridis', origin='lower', extent=[0.1, 1.0, 0.1, 1.0])
plt.colorbar(label='Mean Cross-Validation Accuracy')
plt.title('Tradeoff between Bootstrap Features and Max Samples for Bagging')
plt.xlabel('Bootstrap Features Fraction')
plt.ylabel('Max Samples Fraction')
plt.xticks(bootstrap_features_values_g05)
plt.yticks(max_samples_values_g05)
plt.grid(True)
plt.show()
```



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

The graph illustrates a comprehensive analysis of the tradeoff between two critical hyperparameters, "Bootstrap Features Fraction" and "Max Samples Fraction," in a Bagging ensemble. The x-axis represents the proportion of features randomly selected for each base estimator, ranging from 0.1 to 1.0, while the y-axis represents the fraction of samples randomly chosen for each base estimator, also varying from 0.1 to 1.0. The color-coded heatmap reveals the mean cross-validation accuracy achieved by the Bagging ensemble for different combinations of these hyperparameters.

In essence, the tradeoff emerges as follows: When both the bootstrap features fraction and max samples fraction are set to lower values (towards the bottom-left corner of the graph), the Bagging ensemble becomes more restrained. It employs fewer features and samples for each base estimator, potentially resulting in higher model variance but lower bias. Conversely, when both fractions are increased (moving towards the upper-right corner), the ensemble employs more features and samples, potentially reducing model variance but increasing bias and achieve higher accuracy.