Classification of Body Performance

Motivation:

Body health is closely related to our life. Staying healthy represents stronger bodies, more energy helping us accomplish goals and greater pride in ourselves. Maintaining a good healthy body also leads to better emotions, reducing anxiety and stress, since lack of sleep, unhealthy diets and any other irregular lifestyle will directly affect us mentally.

With the significance of maintaining health, our project is aimed at building a model to predict the assessment of body health performance based on regular physical indicators and some exercise performance data.

Data:

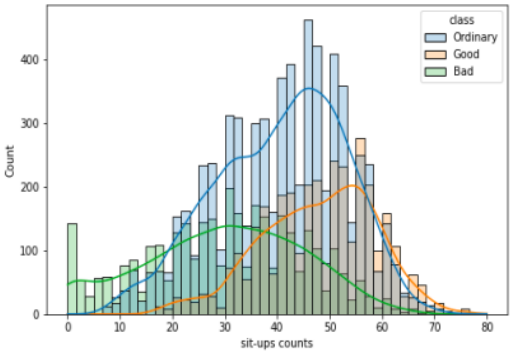
a. Data Collection

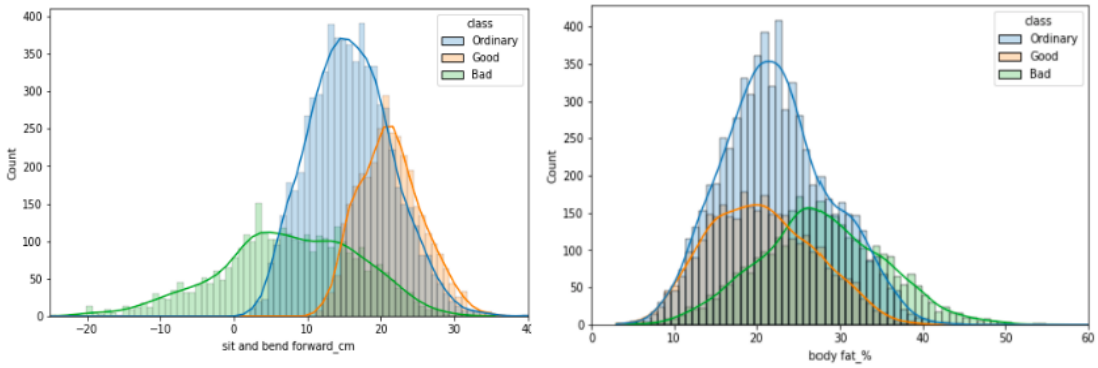
We collected our data from the sports promotion foundation website. Given various body indexes, we choose 11 indicators and exercise performance data as our independent variables among 50 features from the raw data.

Here is the link of our data source: https://www.kaggle.com/kukuroo3/body-performance-data

b. Data Preprocessing

We redefined “class”: “A” as “Good”; “B、C”as”Ordinary”; “D” as “Bad”. Additionally, we did EDA to see the data performance. From the pictures we generate we can see that in general, females outperform males in this body performance assessment and the class of good people outperform the average significantly.





c. Dataset

Our dataset contains a total 13393 observations aged from 20 ~ 64. There are 12 variables in our dataset described in the table below. We will predict **class** based on other 11 independent variables, such as age, gender, weight and some exercise performance metrics.

Table 1:

Variable	Description	Variable	Description
age	20 ~ 64 (in years)	systolic	Systolic blood pressure
gender	F(Female); M(Male)	gripForce	Grip strength:the force applied by the hand to pull on or suspend from objects
height_cm	Height (cm)	sit and bend forward _cm	Exercise sit and bend forward (cm)
weight_kg	Weight (kg)	sit-ups counts	Number of exercise sit-ups
body_fat_%	Total fat mass divided by total body mass	broad jump_cm	Exercise broad jump (cm)
diastolic	Diastolic blood pressure	class	A (Good); B、C (Ordinary); D (Bad)

Analytic Models:

Since we are dealing with the problem of classification, we will have confusion matrices for all models. Based on that, we may obtain metrics including accuracy, precision, recall, and F-score using the following equations to evaluate which model performs the best.

$$Accuracy = (TP + TN)/(TP + FP + FN + TN)$$
$$Precision = TP/(TP + FP)$$
$$Recall = TP/(TP + FN)$$
$$F - Score = 2\times(Recall\times Precision)/(Recall + Precision)$$

Baseline Model

The majority of classes are ordinary, consisting 50.17% of the observations in the training dataset. For our baseline model, we will predict all the testing data as ordinary. The accuracy of the baseline model is 0.49, precision is 0.24, recall is 0.49, and F-score is 0.33.

Logistic Regression

Since we have three classes as our dependent variable, we utilized the sklearn library to perform multiclass linear regression. For the coefficient of the model, we select the best performed parameter from the GridSearchCV method. From a geometric progression, a log scale spacing, and a value ranging from 0.01 to 2, we found that the inverse of regularization rate equals to 2 generates the best result. See Appendix Figure A.1 for plot of accuracy vs. inverse of regularization rate.

Figure A.2 in Appendix is the confusion matrix of our fitted logistic regression model. The model rarely falsely predicts good class as bad class. This observation can be justified from the ROC curve for multi-class data, see Figure A.3 in Appendix. Our model performs well predicting either good or bad class, whereas less accurately predicting ordinary class as we adjust the threshold, shown in lower Area Under the Curve compared to other two classes. The performance metric is shown in the summary Table 2. above. Accuracy is 0.72, which greatly exceeds the performance of the baseline model.

Random Forest

Similar to the logistic regression, we perform GridSearchCV on coefficient ccp-alpha to find the best parameter for our random forest classifier model. We used logarithmic spacing ranging 0.0001 to 1 for ccp-alpha. See appendix for plot of accuracy vs. ccp-alpha. We found that ccp-alpha equals 0.0001, which is the least pruned and most complex tree, performing the best, see Figure B.1 in the Appendix section. The confusion matrix (Appendix Figure B.2) behaves similarly to the logistic regression, and the performance metric is listed in Table 2.

Bagging

For our bagging model (bootstrap aggregation), we select a number of max features ranging from 1 to 11, the maximum number of features available, as our grid search values. From the plot accuracy vs. max features, Appendix Figure C.1, we can tell that the bagging model performs the best when max features equals to 2.

Gradient Boosting

Gradient boosting follows a similar routine as random forest, setting ccp-alpha as the coefficient in grid search. We found that ccp-alpha equals 0.0001 results the best. See Appendix Figure D.1.

Neural Network

In our neural network, we used a multi-layer perceptron classifier as our classification model; it uses stochastic gradient descent to optimize the log-loss function. We've provided

some combinations of hidden layers to GridSearchCV in order to find the best hidden layer for our model. From Figure E.1 in Appendix, we can conclude that hidden layer size with (9, 6) outperforms the rest of parameters.

Model Evaluation:

Following figure is the visualization of accuracy of different models, with darker color representing higher accuracy. The plot on the right excludes the significantly underperformed baseline model. From the plot, we may conclude that the random forest model performs the best. It has the highest accuracy, precision, recall and F-score. Random forest model is also easily explainable, useful in diagnosing observations based on decision tree algorithms.

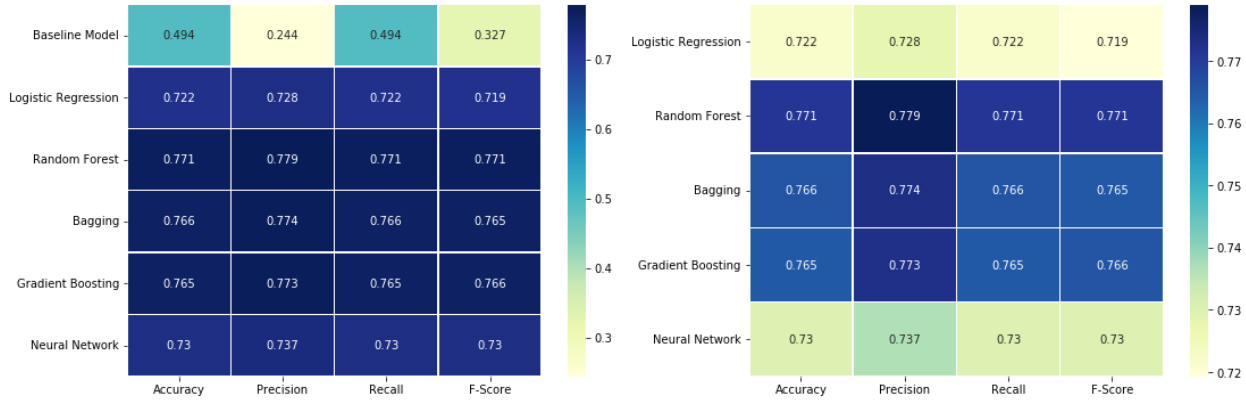


Figure F.1: Model comparison on accuracy

We use bootstrap on our best performed random forest model to access the quality of our final model. Figure F.2 shows the histogram of the bootstrapped random forest model. Black lines are the confidence interval of the model. Model has final accuracy equal to 0.7709 with standard deviation of 0.007154. 95% confidence interval of accuracy is [0.7557, 0.7841].

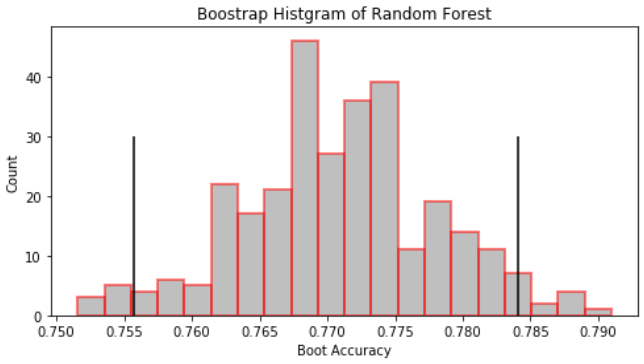


Figure F.2: Bootstrap of Random Forest

Conclusion:

We performed several models, such as logistic regression, random forest, and bagging on the dataset that we collected and cleaned to classify people's body performance. After carefully doing model comparison and model selection, we choose the random forest model which can achieve 77% accuracy and has pretty small standard deviation in accuracy on

average. The model is pretty stable and powerful to predict people's body performance by providing enough information to the model.

Potential improvements of the model include adding more physical indicators such as BMI, number of cigarettes per day, blood glucose level or other exercise performance data like long jump in place, 800m long run to our independent variables.

Impact:

Our model aims to assess people's body performance by using some required data, such as gender, height, weight, body fat percentage and so on, to see which class they will fall into. In this way, people can test their own body performance by themselves. And they will be able to judge their performance with pretty simple tests.

Appendix: Figures

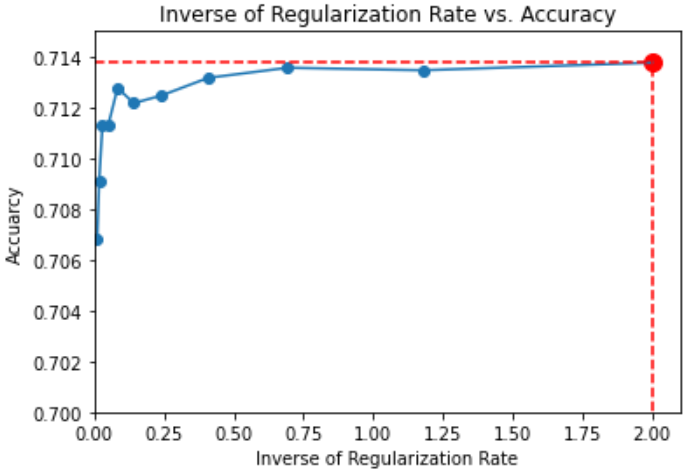


Figure A.1: GridSearchCV on accuracy vs. inverse of regularization rate

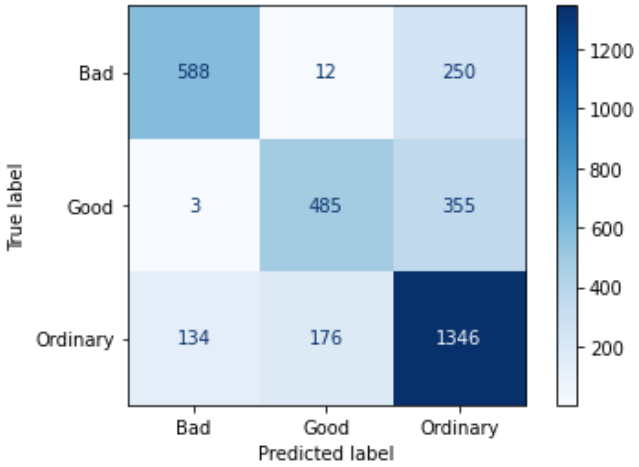


Figure A.2: Confusion Matrix of Logistic Regression

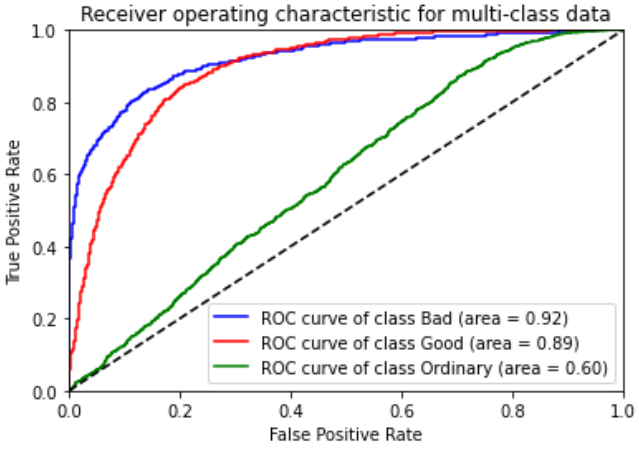


Figure A.3: ROC curve of logistic regression model

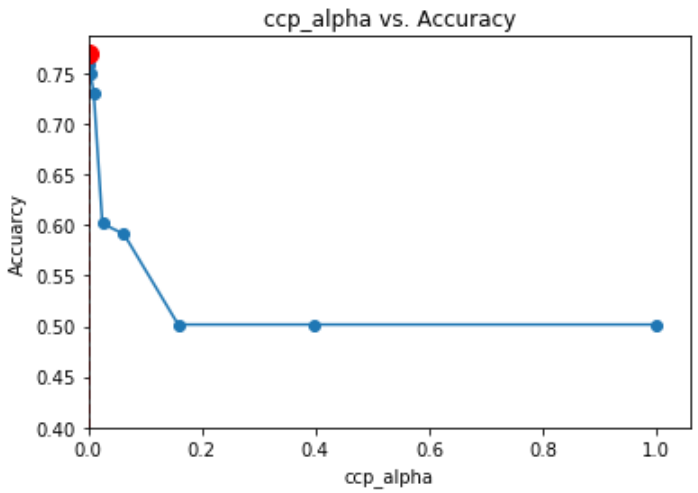


Figure B.1: GridSearchCV on accuracy vs. ccp-alpha

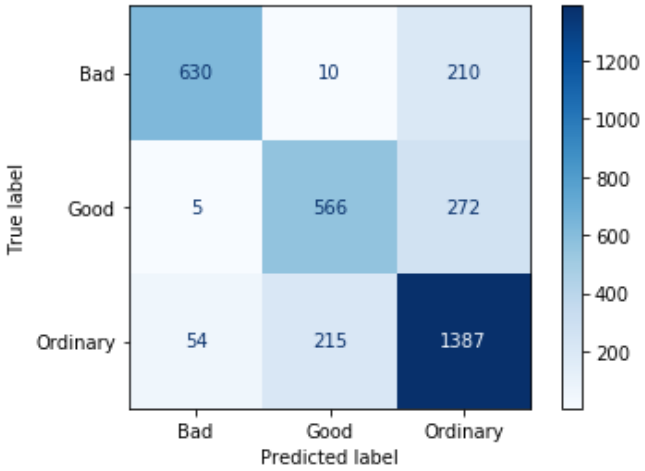


Figure B.2: Confusion Matrix of Random Forest Model

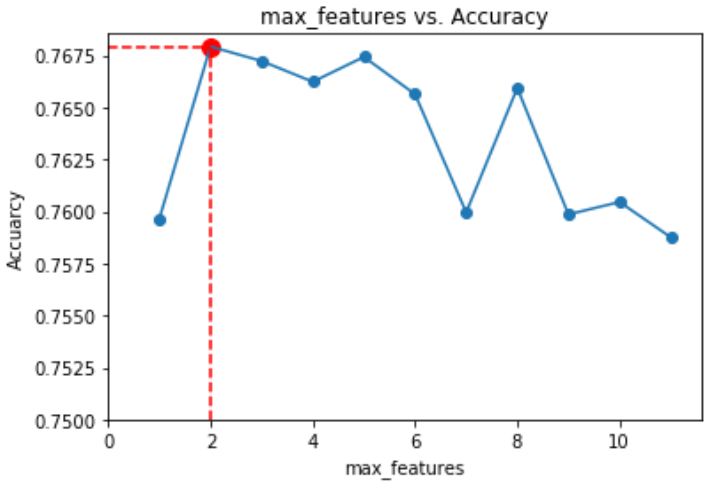


Figure C.1: GridSearchCV on accuracy vs. max features

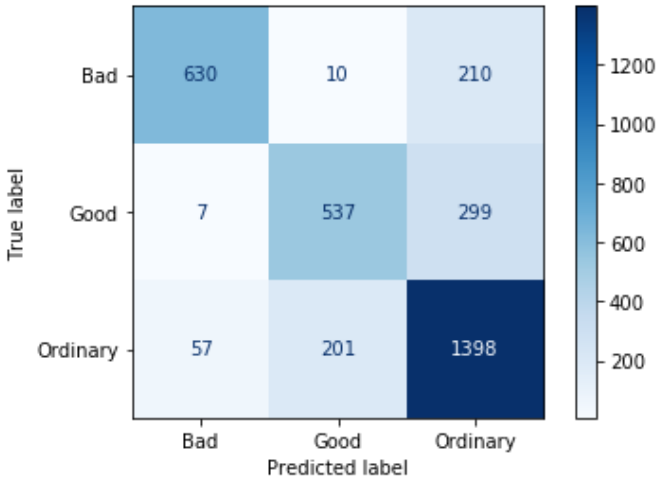


Figure C.2: Confusion Matrix of Bagging Method

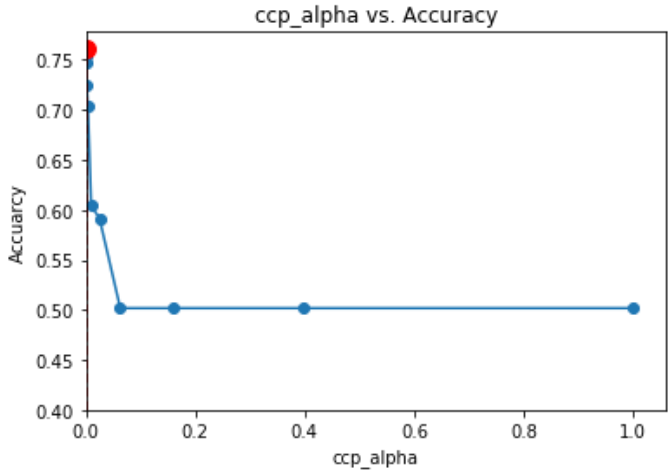


Figure D.1: GridSearchCV on accuracy vs. ccp-alpha

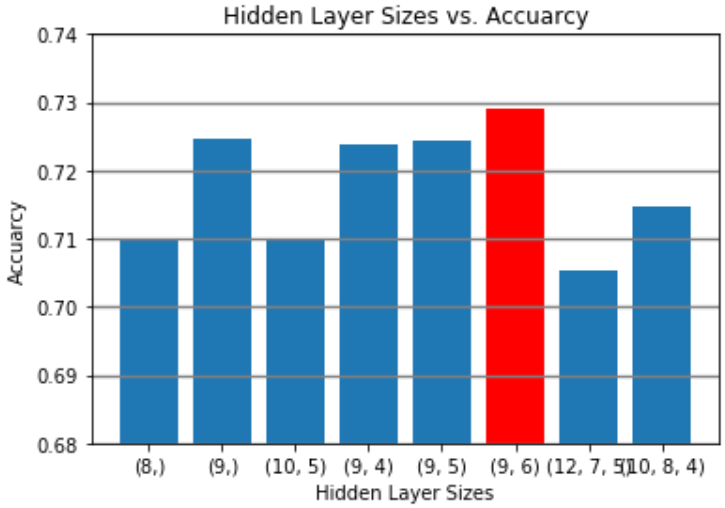


Figure E.1: Accuracy vs. Hidden Layer Sizes

Appendix: Code

IEOR242_FinalProject_Team5_21Fall_12071430

December 16, 2021

1 Classification of Body Performance

Team 5

Name	Email
Anni Wang	anniwang1998@berkeley.edu
Tianhao Wu	thwu@berkeley.edu
Yueting Wu	yueting_wu@berkeley.edu
Zhiyong Jiang	joe_jiangzhiyong@berkeley.edu
Zihe Yan	sxakyzh0805@berkeley.edu

[ ]:
```
import numpy as np
import pandas as pd
# !pip install -U seaborn
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.metrics import roc_curve, auc, plot_confusion_matrix,␣
 ↪confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression, Lasso
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.svm import SVC
from sklearn import svm, datasets
from sklearn.multiclass import OneVsRestClassifier

import statsmodels.api as sm
import statsmodels.formula.api as smf

from itertools import cycle
!pip install bootstrapped
import bootstrapped.bootstrap as bs
```

```
import bootstrapped.stats_functions as bs_stats

# from google.colab import files
import io
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
Collecting bootstrapped
  Downloading bootstrapped-0.0.2.tar.gz (11 kB)
Requirement already satisfied: matplotlib>=1.5.3 in
/usr/local/lib/python3.7/dist-packages (from bootstrapped) (3.2.2)
Requirement already satisfied: numpy>=1.11.1 in /usr/local/lib/python3.7/dist-
packages (from bootstrapped) (1.19.5)
Requirement already satisfied: pandas>=0.18.1 in /usr/local/lib/python3.7/dist-
packages (from bootstrapped) (1.1.5)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.3->bootstrapped)
(1.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.3->bootstrapped)
(3.0.6)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.5.3->bootstrapped)
(2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib>=1.5.3->bootstrapped) (0.11.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.18.1->bootstrapped) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.1->matplotlib>=1.5.3->bootstrapped) (1.15.0)
Building wheels for collected packages: bootstrapped
  Building wheel for bootstrapped (setup.py) … done
  Created wheel for bootstrapped: filename=bootstrapped-0.0.2-py2.py3-none-
any.whl size=13954
sha256=d4b3c23b31b671c0e80122424b2dc70192be611ee8746e3dd35209282ce19cdc
  Stored in directory: /root/.cache/pip/wheels/15/55/6a/9a722f067ac4c3dfab359ed2
ec7906b9cc6649156d9886bd59
Successfully built bootstrapped
Installing collected packages: bootstrapped
Successfully installed bootstrapped-0.0.2
```

2 1. Data

2.1 1.1 Data Import and Integration

[ ]:
```
# Run this cell if the file has not been uploaded, otherwise skip this cell
# uploaded = files.upload()
# df = pd.read_csv(io.StringIO(uploaded['bodyPerformance.csv'].decode('utf-8')))
url = 'https://raw.githubusercontent.com/6shun/IEOR242_Project_BodyHealth/main/
 ↪bodyPerformance.csv'
#df = pd.read_csv('bodyPerformance.csv')
df = pd.read_csv(url)
df
```

[ ]:

	age	gender	height_cm	…	sit-ups counts	broad jump_cm	class
0	27.0	M	172.3	…	60.0	217.0	C
1	25.0	M	165.0	…	53.0	229.0	A
2	31.0	M	179.6	…	49.0	181.0	C
3	32.0	M	174.5	…	53.0	219.0	B
4	28.0	M	173.8	…	45.0	217.0	B
…	…	…	…	…	…	…	…
13388	25.0	M	172.1	…	47.0	198.0	C
13389	21.0	M	179.7	…	48.0	167.0	D
13390	39.0	M	177.2	…	45.0	229.0	A
13391	64.0	F	146.1	…	0.0	75.0	D
13392	34.0	M	164.0	…	51.0	180.0	C

[13393 rows x 12 columns]

2.2 1.2 Data Cleaning

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13393 entries, 0 to 13392
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      13393 non-null  float64
 1   gender                   13393 non-null  object
 2   height_cm                13393 non-null  float64
 3   weight_kg                13393 non-null  float64
 4   body fat_%               13393 non-null  float64
 5   diastolic                13393 non-null  float64
 6   systolic                 13393 non-null  float64
 7   gripForce                13393 non-null  float64
 8   sit and bend forward_cm  13393 non-null  float64
 9   sit-ups counts           13393 non-null  float64
 10  broad jump_cm            13393 non-null  float64
 11  class                    13393 non-null  object
dtypes: float64(10), object(2)
```

```
memory usage: 1.2+ MB
```

To make the label more clear, we chang A to “Good”, B and C to “Ordinary”, and D to “Bad”.

[ ]:
```
df['class'] = df['class'].map({"A":"Good", "B":"Ordinary", "C":"Ordinary", "D":
↪"Bad"})
```

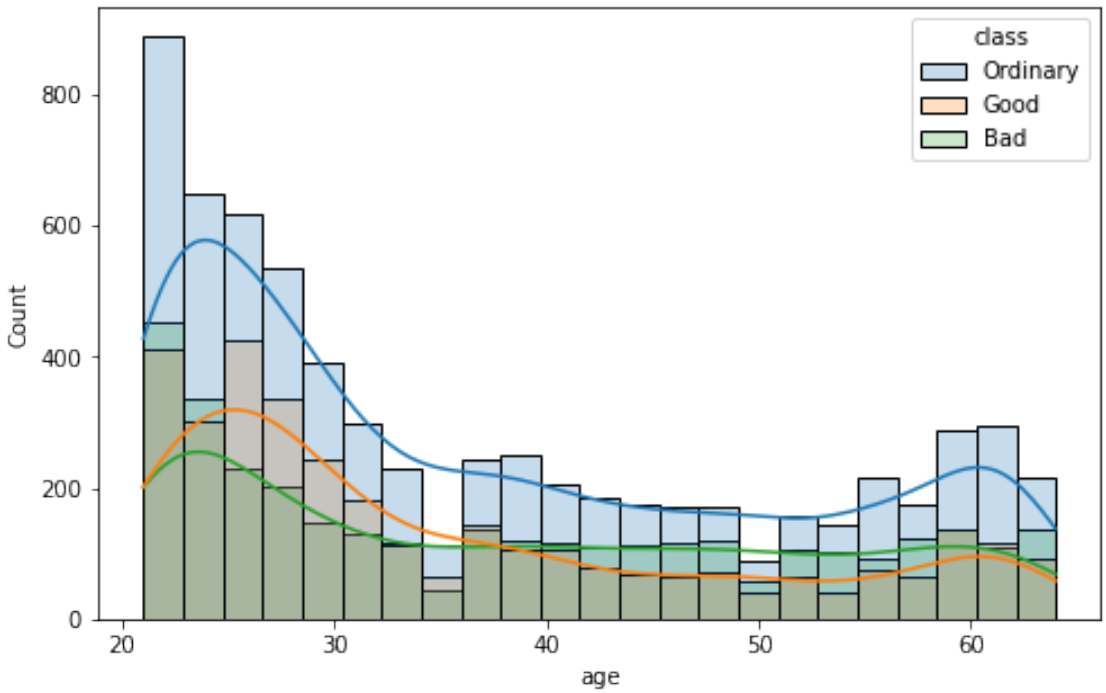
3 2. Exploratory Data Analysis

3.1 2.1 Visualization

[ ]:
```
# # Examine covariance
# sns.pairplot(df, hue='class');
```

Plot the age distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='age', hue='class', kde=True, alpha=0.25);
```

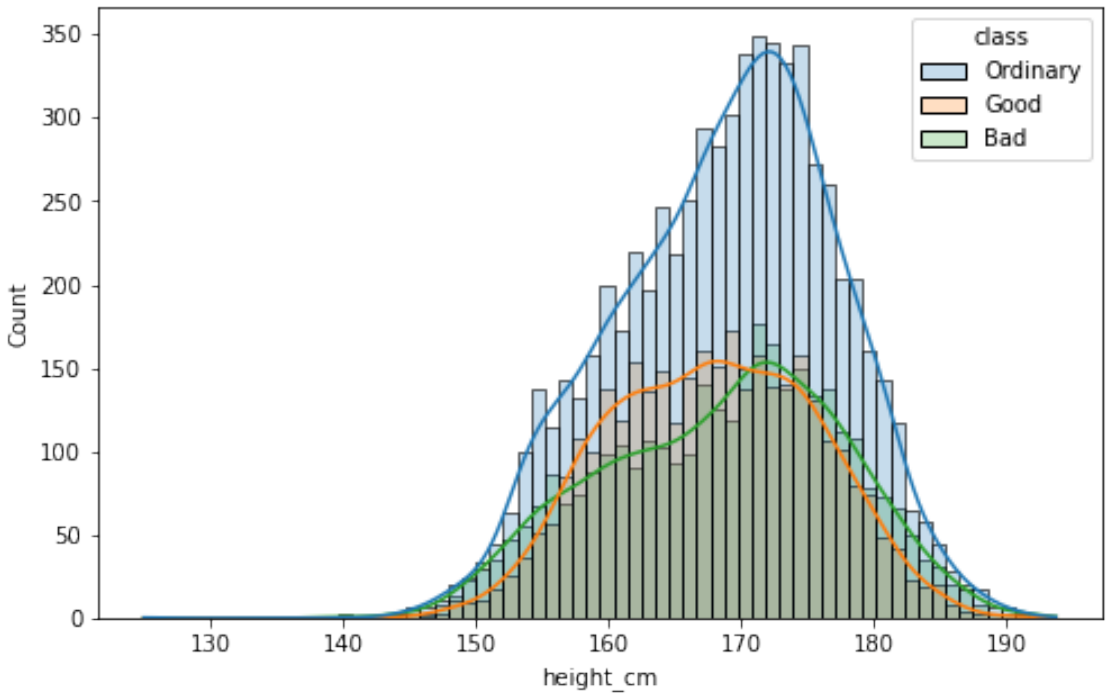


Plot the gender percentage of each class.

[ ]:
```
tmp = pd.pivot_table(df, index='gender', columns='class', aggfunc='size')
# Normalize by F and M
tmp = tmp / tmp.sum()
sns.heatmap(tmp, cmap="GnBu", annot=True, linewidths=.5);
```

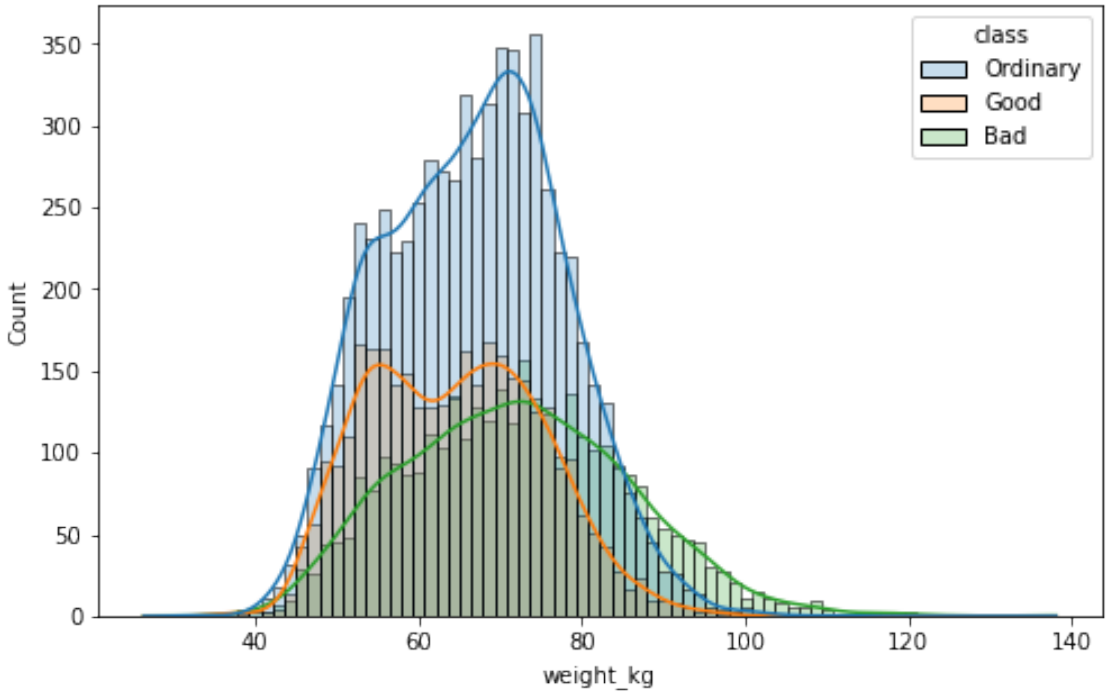
Plot the height distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='height_cm', hue='class', kde=True, alpha=0.25);
```



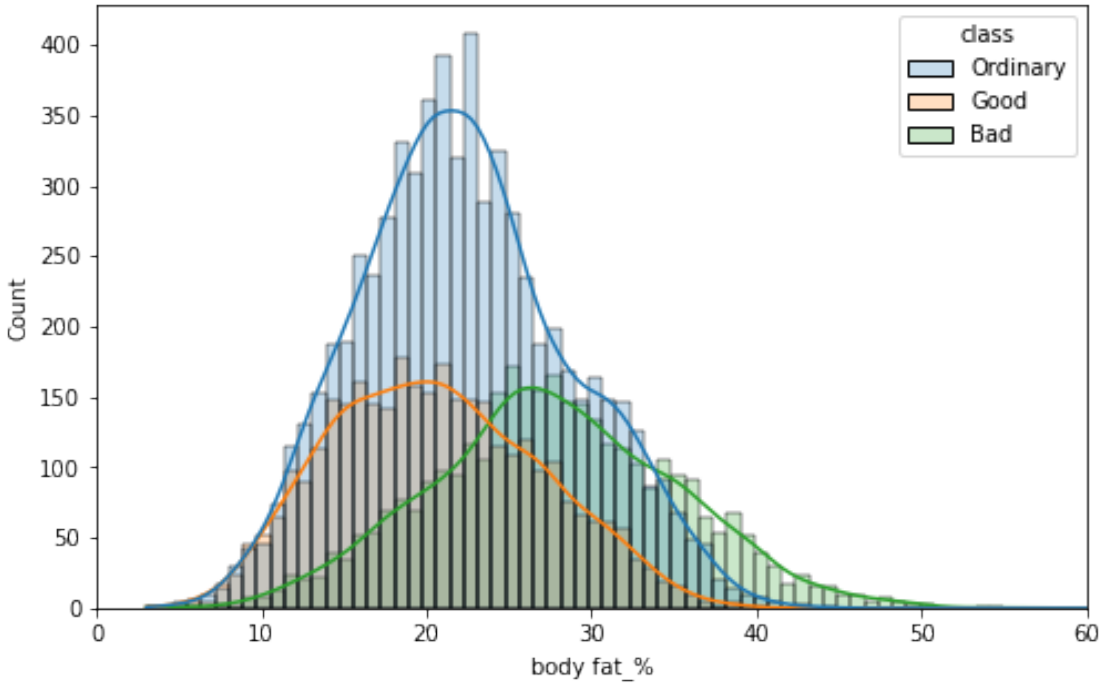
Plot the weight distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='weight_kg', hue='class', kde=True, alpha=0.25);
```



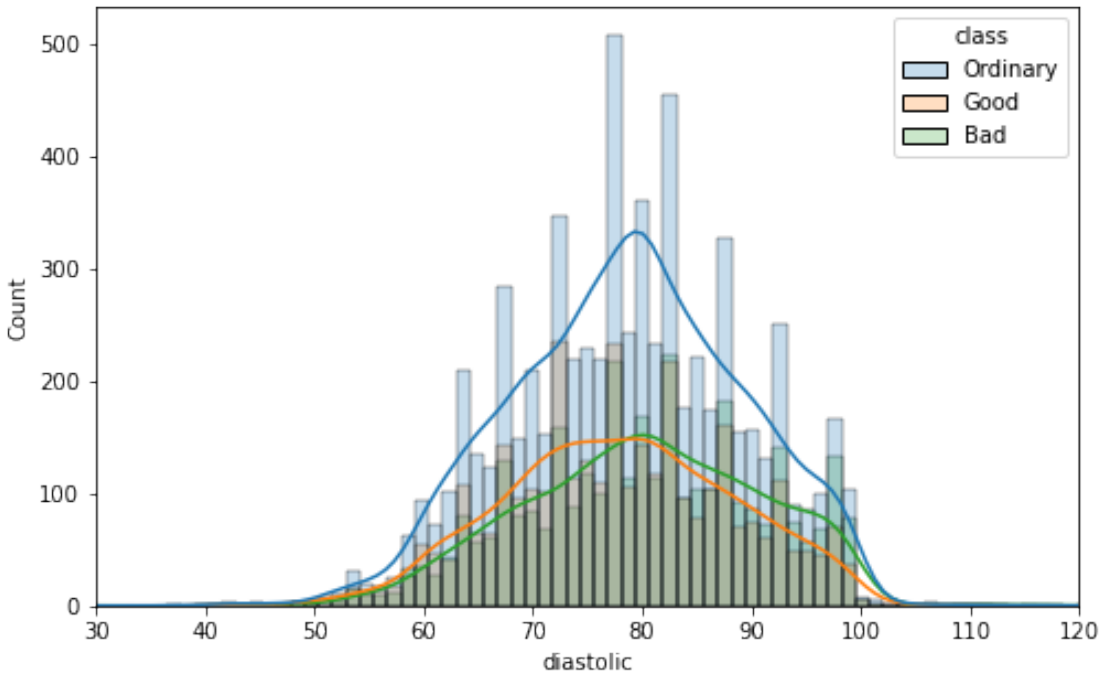
Plot the body fat percentage distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='body fat_%', hue='class', kde=True, alpha=0.25);
plt.xlim(left=0, right=60);
```



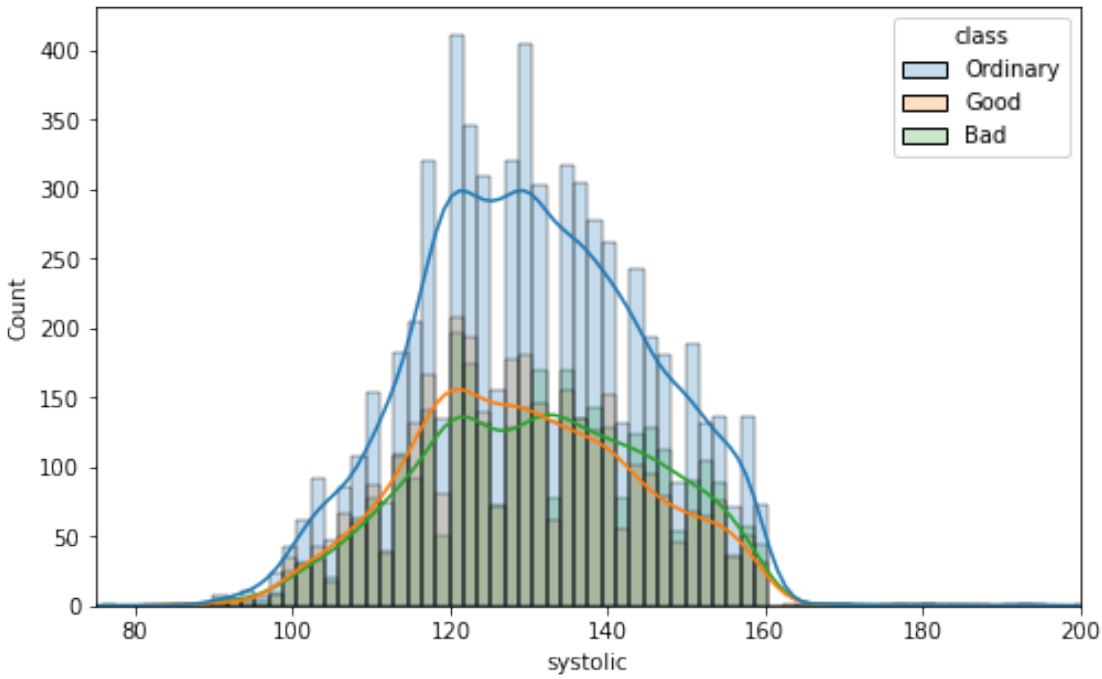
Plot the diastolic distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='diastolic', hue='class', kde=True, alpha=0.25);
plt.xlim(left=30, right=120);
```



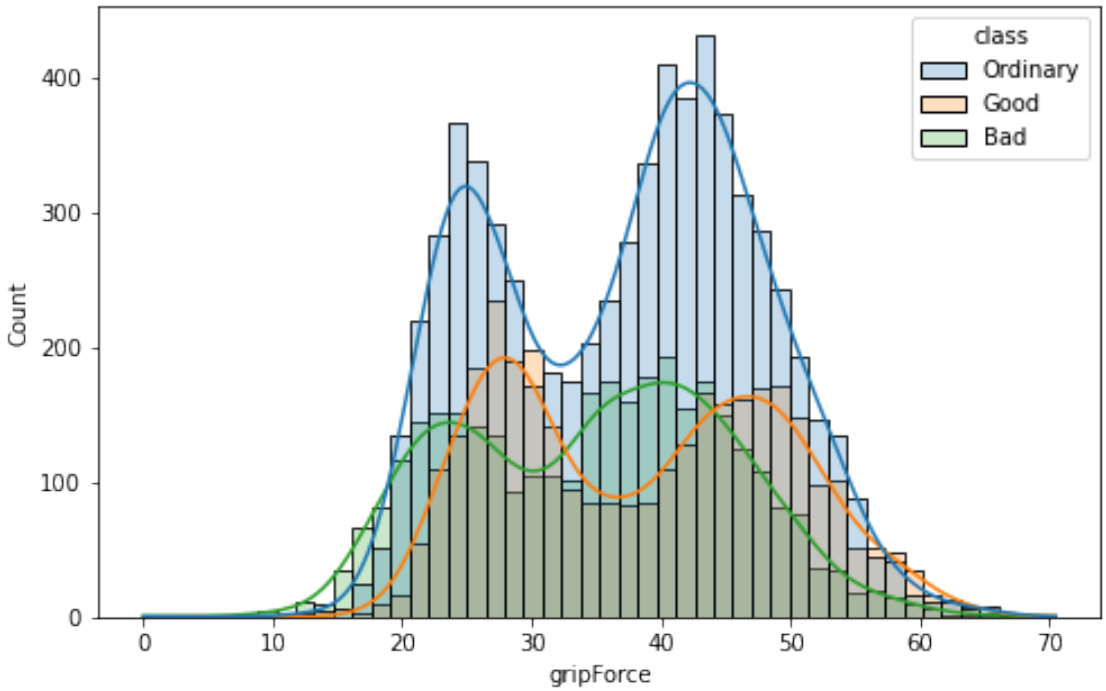
Plot the systolic distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='systolic', hue='class', kde=True, alpha=0.25);
plt.xlim(left=75, right=200);
```



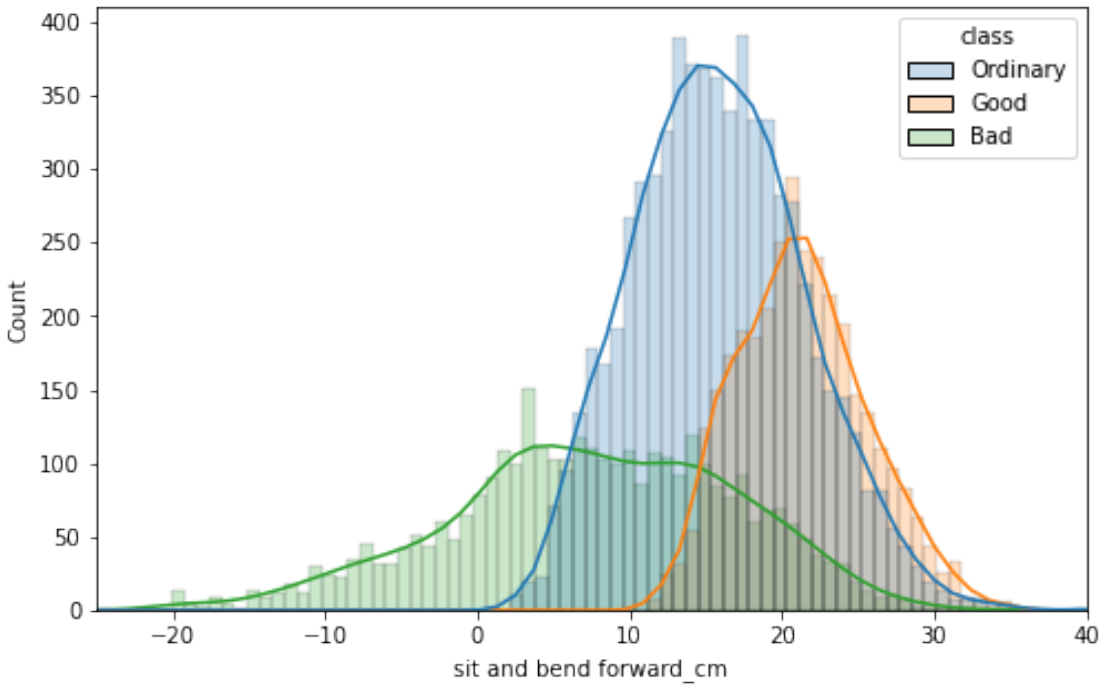
Plot the gripForce distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='gripForce', hue='class', kde=True, alpha=0.25);
```



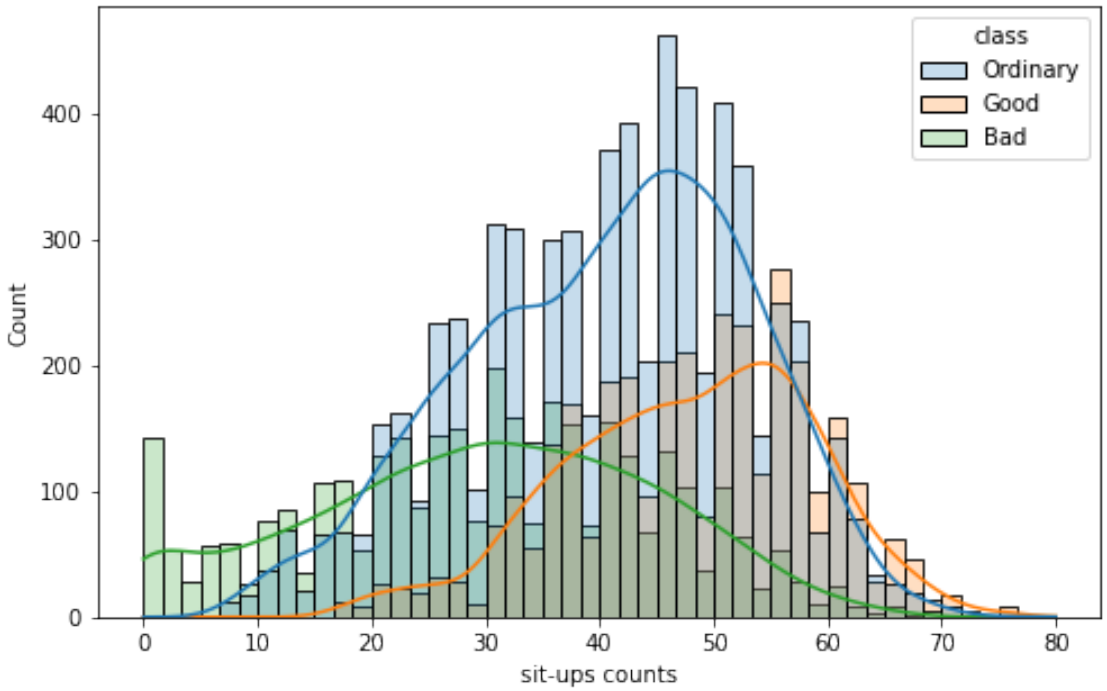
Plot the sit and bend forward distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='sit and bend forward_cm', hue='class', kde=True,␣
↪alpha=0.25);
plt.xlim(left=-25, right=40);
```



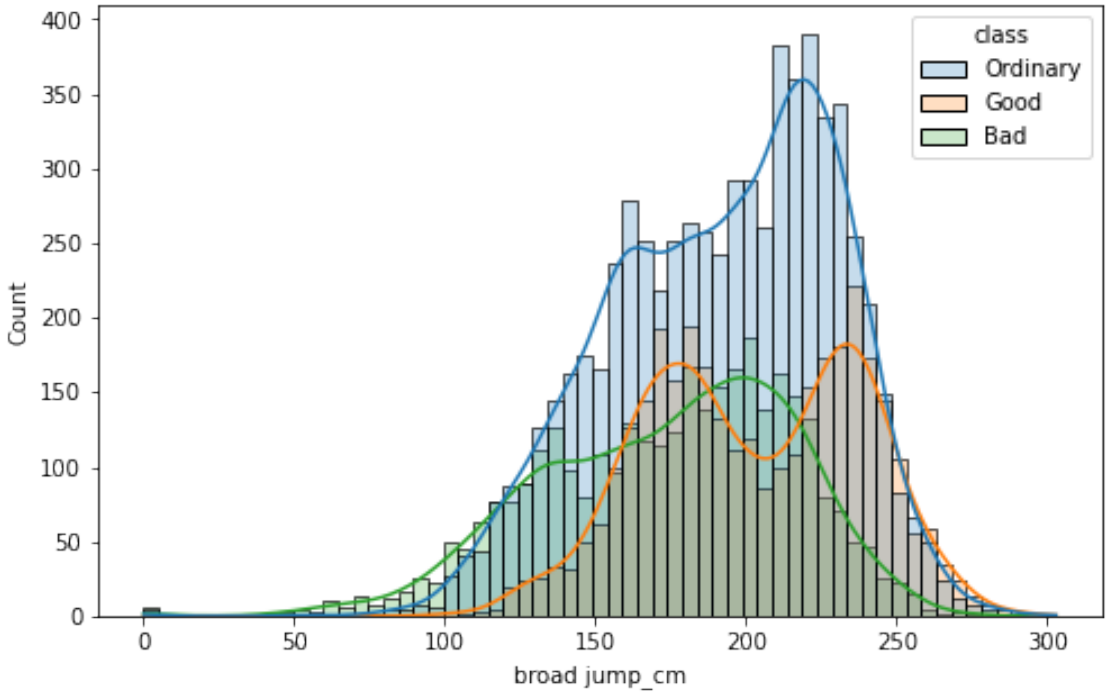
Plot the sit-ups counts distribution of each class.

[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='sit-ups counts', hue='class', kde=True, alpha=0.25);
```



Plot the broad jump distribution of each class.

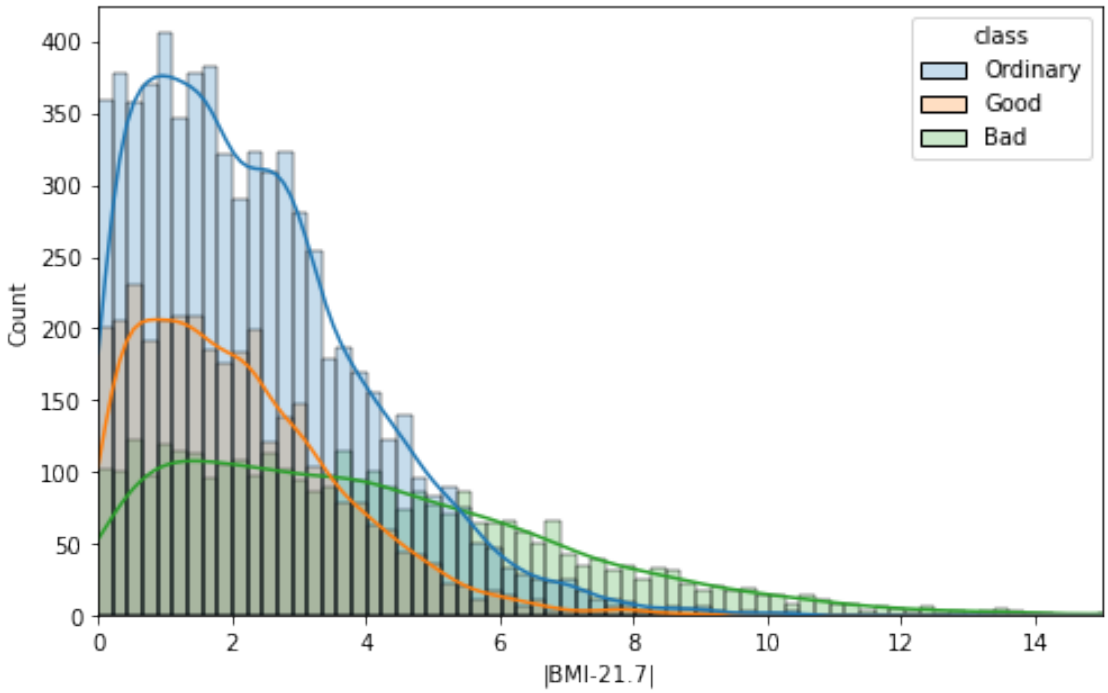
[ ]:
```
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='broad jump_cm', hue='class', kde=True, alpha=0.25);
```



3.2 2.2 Feature Engineering

Body mass index (BMI) is a measure of body fat based on height and weight.The best BMI lies in the range from 18.5 to 24.9. So we create a new feature to calculate the absolute difference between an individual's BMI and the midpoint 21.7.

[ ]:
```
# BMI
df['BMI'] = df['weight_kg'] / (df['height_cm']/100)**2
df['|BMI-21.7|'] = abs(df['BMI'] - 21.7)
plt.figure(figsize=(8, 5));
sns.histplot(data=df, x='|BMI-21.7|', hue='class', kde=True, alpha=0.25);
plt.xlim(left=0, right=15);
```



4 3. Modeling

[ ]:
```
# Change "M" to 1 and "F" to 0 for convenient purpose
df['gender'] = df['gender'].apply(lambda x: 1 if x == "M" else 0)

# Rename variables in case of error
df = df.rename({"body fat_%":"body_fat_percent",
            "sit and bend forward_cm":"sit_and_bend_forward_cm",
            "sit-ups counts":"sit_ups_counts",
            "broad jump_cm":"broad_jump_cm",
            "|BMI-21.7|":"abs_BMI_minus_21dot7"}, axis=1)

# We will choose the whole features without feature selection
features = ['gender', 'height_cm', 'weight_kg', 'body_fat_percent', 'diastolic',
            'systolic', 'gripForce', 'sit_and_bend_forward_cm',␣
 ↪'sit_ups_counts',
            'broad_jump_cm', 'abs_BMI_minus_21dot7']

X = df[features]
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=242)
```

[ ]:
```
# Define a function to generate classification performace metrics [accuracy,␣
 ↪precision, recall, F_Score]
def generate_performance(y_pred=None):
  '''accuracy_score(y_test, y_pred), precision, recall, F_Score'''
    # accuracy, precision, recall, F-Score
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    F_Score = f1_score(y_test, y_pred, average='weighted')
    res = [accuracy_score(y_test, y_pred), precision, recall, F_Score]
    return [round(num, 4) if num else None for num in res]
```

4.1 3.1 Baseline Model

[ ]:
```
y_pred_test_baseline = [y_train.value_counts().index[0]] * len(y_test)

print("Test Score:", round(accuracy_score(y_test, y_pred_test_baseline), 4))

baseline_compare = generate_performance(y_pred_test_baseline)
baseline_compare
```

```
Test Score: 0.4945
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
[0.4945, 0.2445, 0.4945, 0.3272]
```

4.2 3.2 Logistic Regression

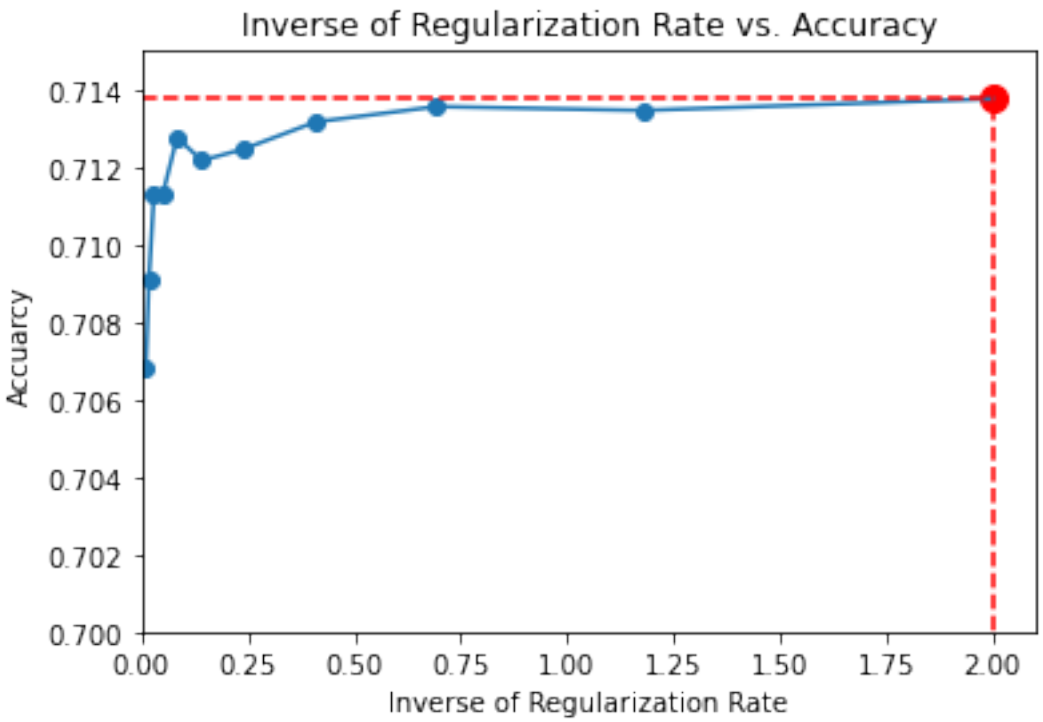
[ ]:
```
# tmp = pd.concat([X_train, pd.DataFrame(y_train)], axis=1)

# tmp['class'] = pd.concat([X_train, pd.DataFrame(y_train)], axis=1)['class'].
 ↪apply(lambda x: 1 if x == "Good" else 0)
# mod = smf.logit(formula = "class ~ gender + height_cm + weight_kg +␣
 ↪body_fat_percent + diastolic + systolic + gripForce +␣
 ↪sit_and_bend_forward_cm + sit_ups_counts + broad_jump_cm +␣
 ↪abs_BMI_minus_21dot7",
#                 data=tmp)

# res = mod.fit()
# print(res.summary())
```

[ ]:
```
# In linear regression, "C" is the inverse of regularization rate, we can use␣
 ↪grid search to get the optimal value
grid_values = {'C': np.geomspace(start=1, stop=200, num=11) / 100}
```

```
clf = LogisticRegression(random_state=242, solver='liblinear')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0, n_jobs=-1)
clf_cv.fit(X_train, y_train);
```

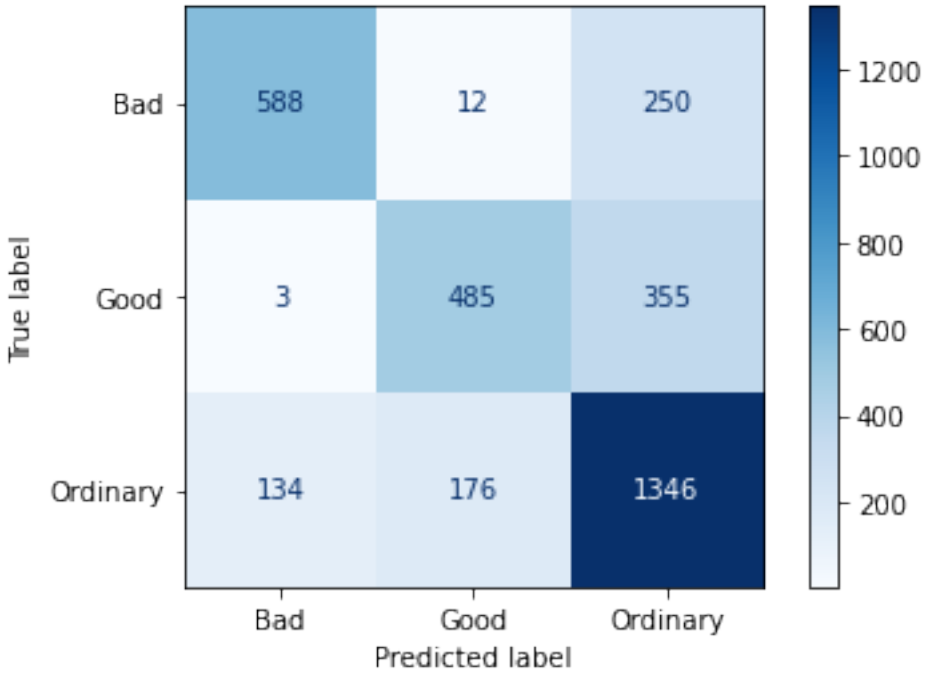
[ ]:
```
accuarcy = clf_cv.cv_results_['mean_test_score']
C = clf_cv.cv_results_['param_C'].data #
best_parameter = clf_cv.best_params_['C'] #

# plt.figure(figsize=(8,5));
plt.plot(C, accuarcy);
plt.scatter(C, accuarcy);
plt.ylim(bottom=0.7, top=0.715); #
plt.xlim(left=0); #
plt.xlabel("Inverse of Regularization Rate");
plt.ylabel("Accuarcy");
plt.scatter(best_parameter, max(accuarcy), c='r',s=100);
plt.plot(np.linspace(0.0, best_parameter, 101), [max(accuarcy) for i in␣
 ↪range(101)], c='r', linestyle='dashed');
plt.plot([best_parameter for i in range(101)], np.linspace(0.0, max(accuarcy),␣
 ↪101),  c='r', linestyle='dashed');
plt.title("Inverse of Regularization Rate vs. Accuracy"); #
print("Best accuarcy:", max(accuarcy))
print("Best C:", best_parameter) #
```

```
Best accuarcy: 0.713759248719408
Best C: 2.0000000000000004
```



[ ]:
```
log = LogisticRegression(random_state=242, solver='liblinear', C=best_parameter)
y_pred_test_logistic = log.fit(X_train, y_train).predict(X_test) #
y_pred = y_pred_test_logistic

disp = plot_confusion_matrix(log, X_test, y_test, display_labels=["Bad",␣
↪"Good", "Ordinary"], cmap=plt.cm.Blues)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



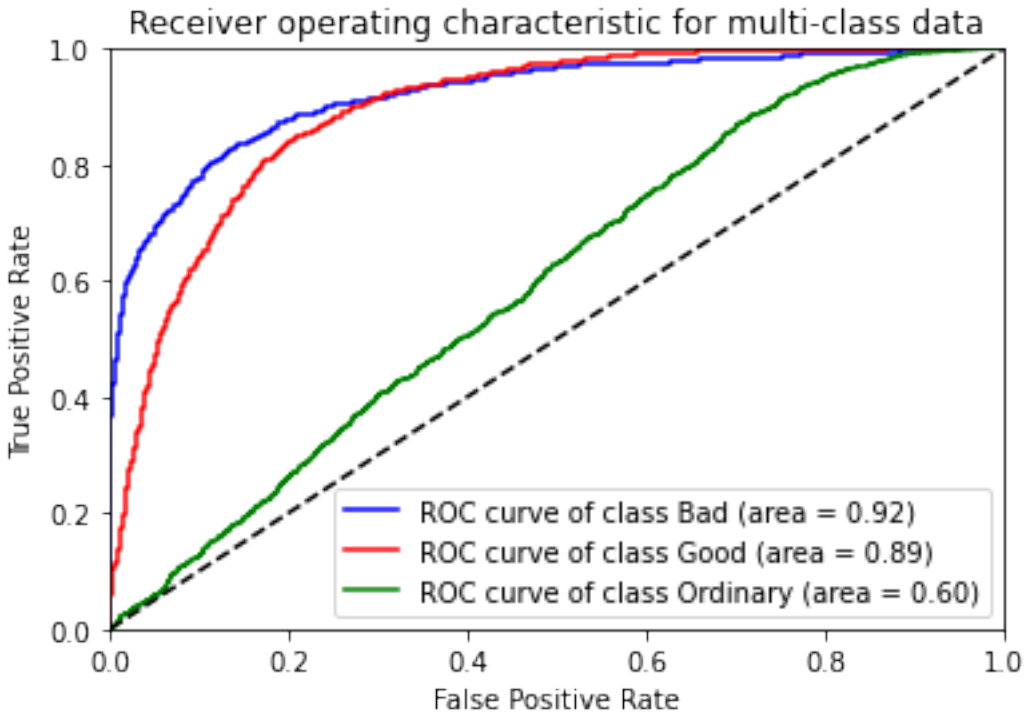
[ ]:
```
tempmodel = log.fit(X_train, y_train)
# tempmodel.summary()
```

[ ]:
```
logistic_compare = generate_performance(y_pred_test_logistic)
logistic_compare
```

[ ]:
```
[0.7223, 0.7284, 0.7223, 0.7196]
```

[ ]:
```
X = df[features]
y = df['class']

y = label_binarize(y, classes=["Bad", "Good", "Ordinary"])
n_classes = y.shape[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
↪random_state=242)

classifier = OneVsRestClassifier(LogisticRegression(random_state=242,␣
↪solver='liblinear'))
y_score = classifier.fit(X_train, y_train).decision_function(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()
```

```
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
colors = cycle(['blue', 'red', 'green'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, label='ROC curve of class {0} (area =␣
 ↪{1:0.2f})'
             ''.format(["Bad", "Good", "Ordinary"][i], roc_auc[i]))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0, 1.0])
plt.ylim([0.0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for multi-class data')
plt.legend(loc="lower right")
plt.show()

X = df[features]
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=242)
```

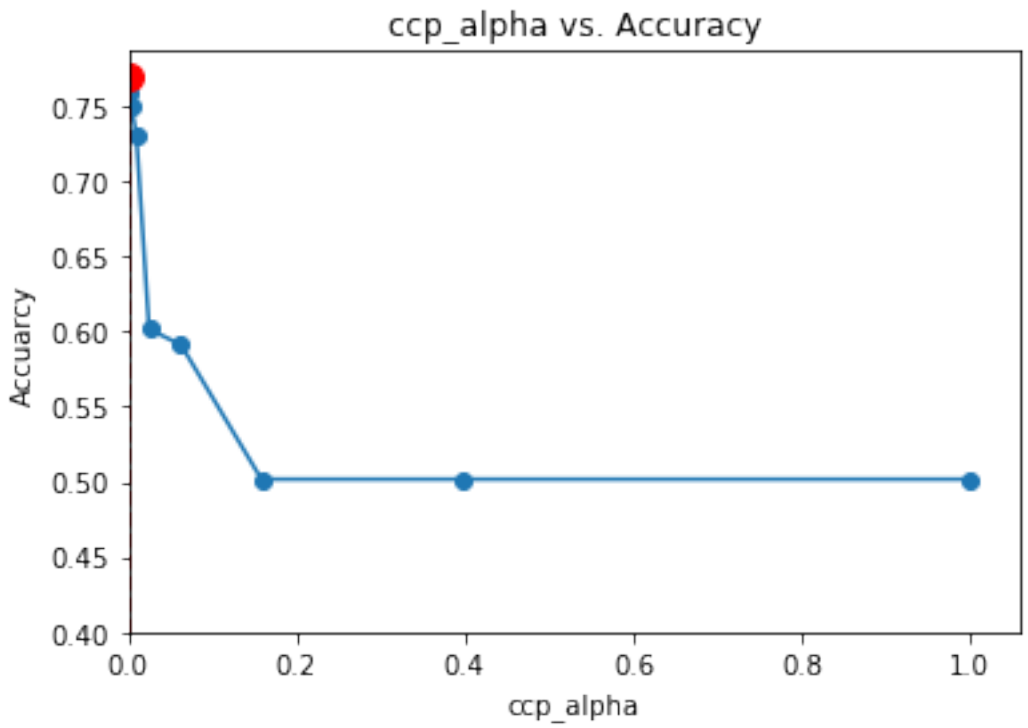


4.3 3.3 Random Forest

[ ]:
```
# In random forest, "ccp_alpha" represents the complexity, we can use grid␣
 ↪search to get the optimal value
grid_values = {'ccp_alpha': np.geomspace(start=1, stop=10000, num=11) / 10000}
clf = RandomForestClassifier(random_state=242)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0, n_jobs=-1)
clf_cv.fit(X_train, y_train);
```

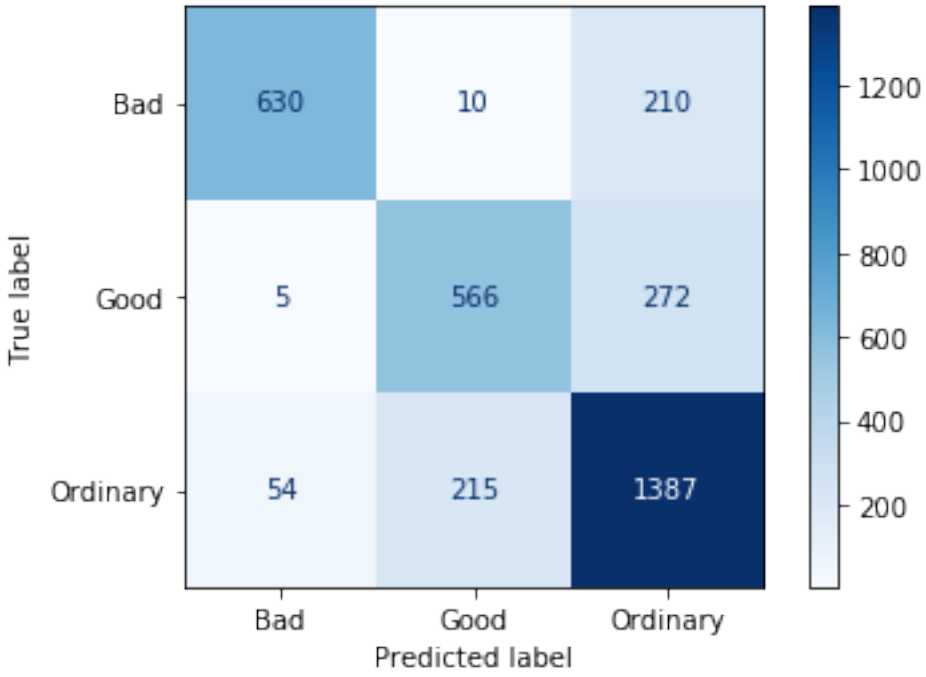
[ ]:
```
accuarcy = clf_cv.cv_results_['mean_test_score']
ccp = clf_cv.cv_results_['param_ccp_alpha'].data #
best_parameter = clf_cv.best_params_['ccp_alpha'] #

# plt.figure(figsize=(8,5));
plt.plot(ccp, accuarcy);
plt.scatter(ccp, accuarcy);
plt.ylim(bottom=0.4); #
plt.xlim(left=0); #
plt.xlabel("ccp_alpha"); #
plt.ylabel("Accuarcy");
plt.scatter(best_parameter, max(accuarcy), c='r',s=100);
plt.plot(np.linspace(0.0, best_parameter, 101), [max(accuarcy) for i in␣
 ↪range(101)], c='r', linestyle='dashed');
plt.plot([best_parameter for i in range(101)], np.linspace(0.0, max(accuarcy),␣
 ↪101),  c='r', linestyle='dashed');
plt.title("ccp_alpha vs. Accuracy"); #
print("Best accuarcy:", max(accuarcy))
print("Best ccp_alpha:", best_parameter) #
```

```
Best accuarcy: 0.7684189573215351
Best ccp_alpha: 0.0001
```



[ ]:
```
rf = RandomForestClassifier(random_state=242, ccp_alpha=best_parameter)
y_pred_test_rf = rf.fit(X_train, y_train).predict(X_test) #
y_pred = y_pred_test_rf

disp = plot_confusion_matrix(rf, X_test, y_test, display_labels=["Bad", "Good",␣
↪"Ordinary"], cmap=plt.cm.Blues)
plt.show()
```

```
C:\Users\Joe\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[ ]:
```
rf_compare = generate_performance(y_pred_test_rf)
rf_compare
```

[ ]:
```
[0.7713, 0.7791, 0.7713, 0.7713]
```

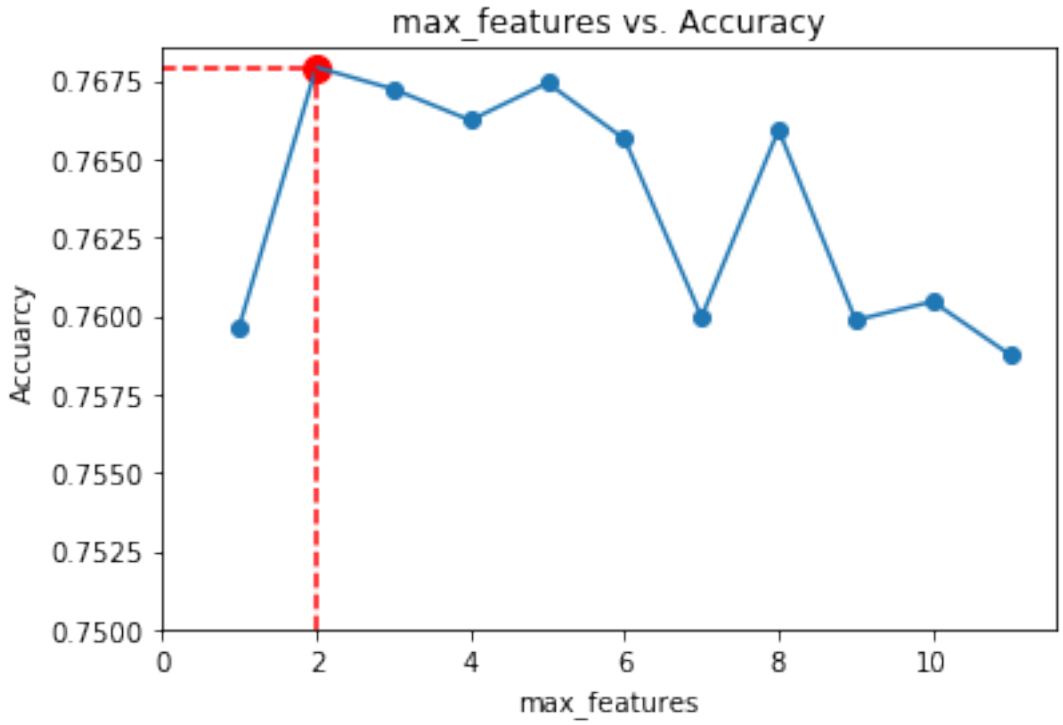
4.4 3.4 Bagging

[ ]:
```
# In bagging, "max_features" represents the maximum number of features to␣
 ↪choose, we can use grid search to get the optimal value
grid_values = {'max_features': range(1, len(X_train.columns) + 1)}
clf = RandomForestClassifier(random_state=242)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0, n_jobs=-1)
clf_cv.fit(X_train, y_train);
```

[ ]:
```
accuarcy = clf_cv.cv_results_['mean_test_score']
max_features = clf_cv.cv_results_['param_max_features'].data #
best_parameter = clf_cv.best_params_['max_features'] #

# plt.figure(figsize=(8, 5));
plt.plot(max_features, accuarcy);
plt.scatter(max_features, accuarcy);
plt.ylim(bottom=0.75); #
plt.xlim(left=0); #
```

```
plt.xlabel("max_features"); #
plt.ylabel("Accuarcy");
plt.scatter(best_parameter, max(accuarcy), c='r',s=100);
plt.plot(np.linspace(0.0, best_parameter, 101), [max(accuarcy) for i in␣
 ↪range(101)], c='r', linestyle='dashed');
plt.plot([best_parameter for i in range(101)], np.linspace(0.0, max(accuarcy),␣
 ↪101),  c='r', linestyle='dashed');
plt.title("max_features vs. Accuracy"); #
print("Best accuarcy:", max(accuarcy))
print("Best max_features:", best_parameter) #
```

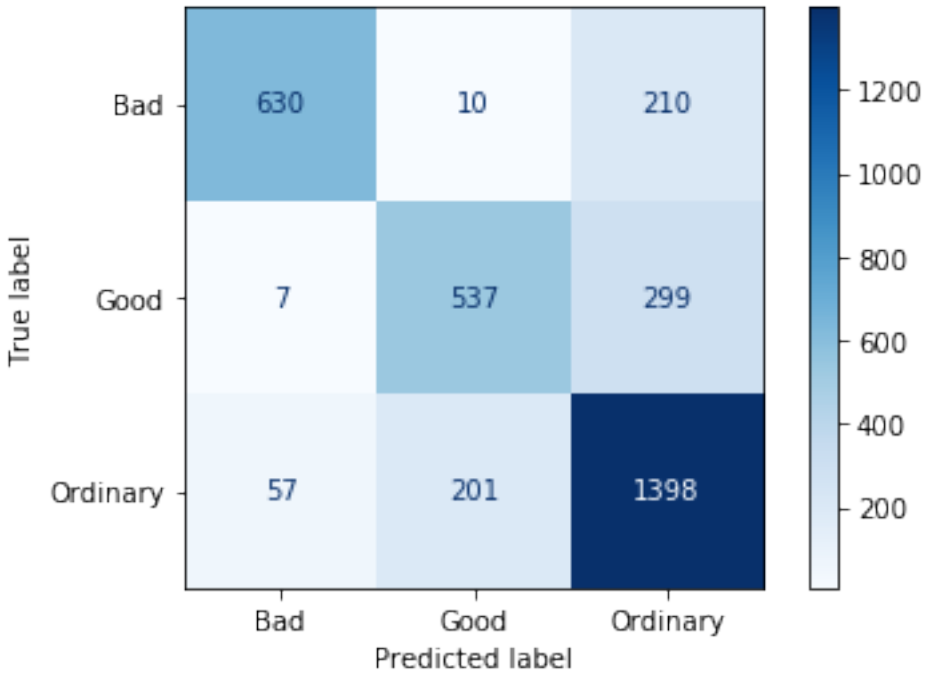
```
Best accuarcy: 0.7679206023095274
Best max_features: 2
```



[ ]:
```
bag = RandomForestClassifier(random_state=242, max_features=best_parameter)
y_pred_test_bag = bag.fit(X_train, y_train).predict(X_test) #
y_pred = y_pred_test_bag

disp = plot_confusion_matrix(bag, X_test, y_test, display_labels=["Bad",␣
 ↪"Good", "Ordinary"], cmap=plt.cm.Blues)
plt.show()
```

```
C:\Users\Joe\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
```

```
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[ ]:
```
bag_compare = generate_performance(y_pred_test_bag)
bag_compare
```

[ ]:
```
[0.7659, 0.7736, 0.7659, 0.7651]
```

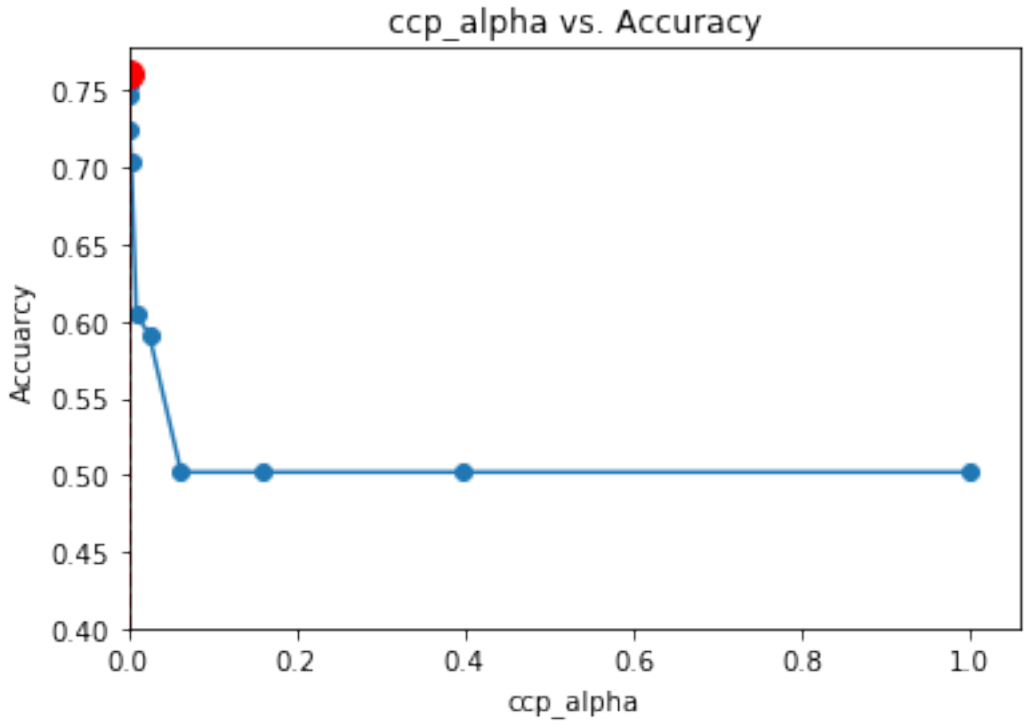
4.5 3.5 Gradient Boosting

[ ]:
```
# In gradinet boosting, "ccp_alpha" represents the complexity, we can use grid␣
 ↪search to get the optimal value
grid_values = {'ccp_alpha': np.geomspace(start=1, stop=10000, num=11) / 10000}
clf = GradientBoostingClassifier(random_state=242)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0, n_jobs=-1)
clf_cv.fit(X_train, y_train);
```

[ ]:
```
accuarcy = clf_cv.cv_results_['mean_test_score']
ccp = clf_cv.cv_results_['param_ccp_alpha'].data #
best_parameter = clf_cv.best_params_['ccp_alpha'] #

# plt.figure(figsize=(8,5));
```

```
plt.plot(ccp, accuarcy);
plt.scatter(ccp, accuarcy);
plt.ylim(bottom=0.4); #
plt.xlim(left=0); #
plt.xlabel("ccp_alpha"); #
plt.ylabel("Accuarcy");
plt.scatter(best_parameter, max(accuarcy), c='r',s=100);
plt.plot(np.linspace(0.0, best_parameter, 101), [max(accuarcy) for i in␣
 ↪range(101)], c='r', linestyle='dashed');
plt.plot([best_parameter for i in range(101)], np.linspace(0.0, max(accuarcy),␣
 ↪101),  c='r', linestyle='dashed');
plt.title("ccp_alpha vs. Accuracy"); #
print("Best accuarcy:", max(accuarcy))
print("Best ccp_alpha:", best_parameter) #
```

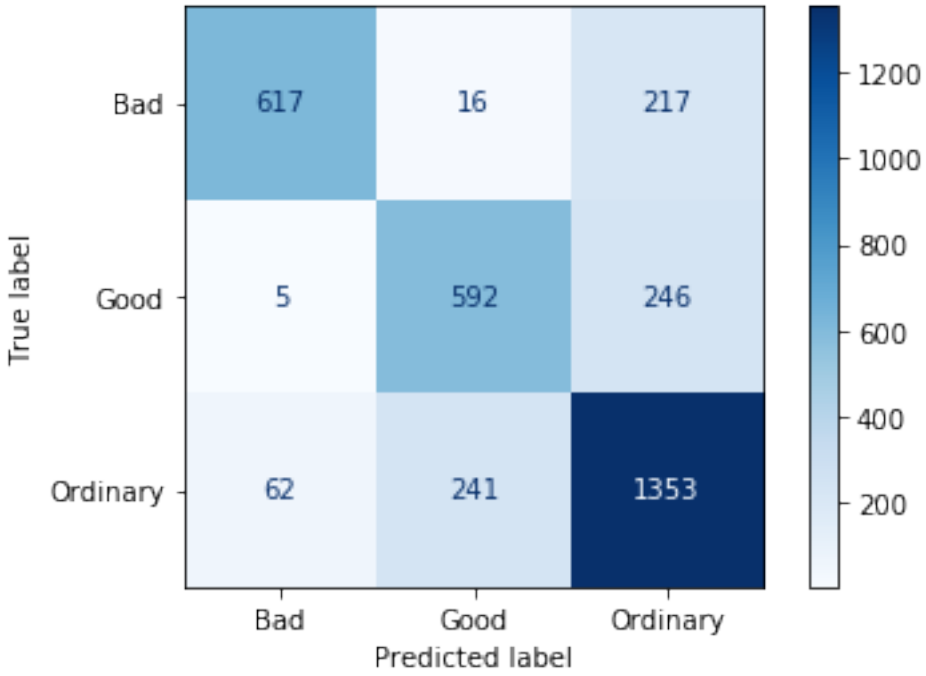
```
Best accuarcy: 0.760552711999191
Best ccp_alpha: 0.0001
```



[ ]:
```
gb = GradientBoostingClassifier(random_state=242, ccp_alpha=best_parameter)
y_pred_test_gb = gb.fit(X_train, y_train).predict(X_test) #
y_pred = y_pred_test_gb
```

```
disp = plot_confusion_matrix(gb, X_test, y_test, display_labels=["Bad", "Good",␣
↪"Ordinary"], cmap=plt.cm.Blues)
plt.show()
```

```
C:\Users\Joe\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[ ]:
```
gb_compare = generate_performance(y_pred_test_gb)
gb_compare
```

[ ]:
```
[0.765, 0.7729, 0.765, 0.7657]
```

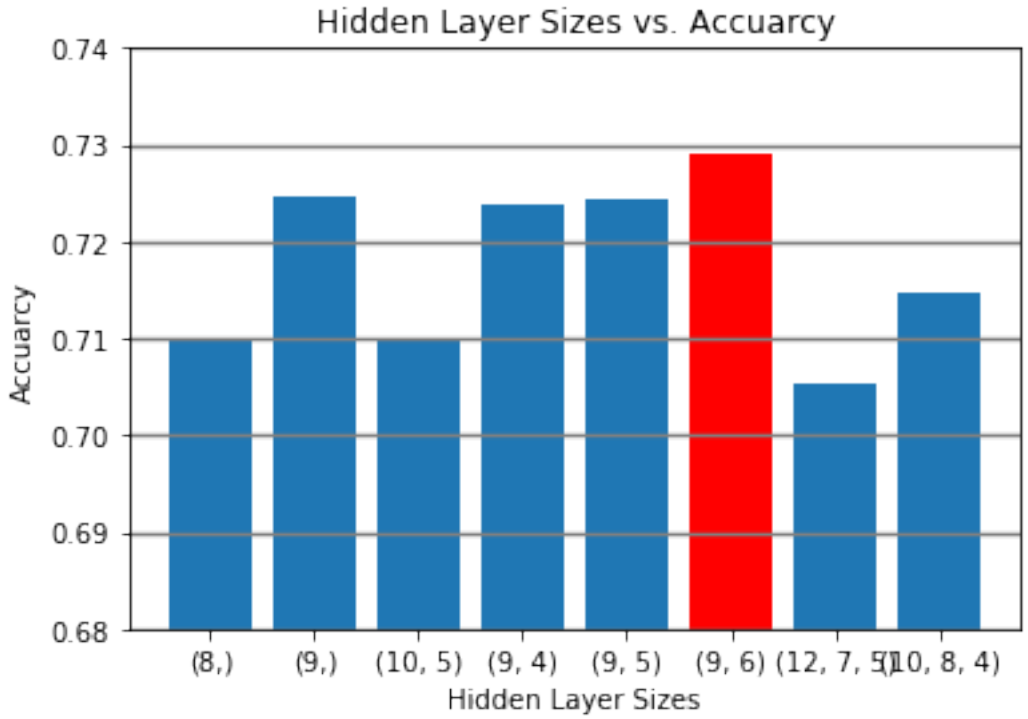
4.6 3.6 Neural Network

[ ]:
```
# In neural network, "hidden_layer_sizes" represents hidden layer structure, we␣
↪can use grid search to get the optimal value
grid_values = {'hidden_layer_sizes': [(8,), (9,), (10, 5),(9, 4), (9, 5), (9,␣
↪6), (12, 7, 5), (10, 8, 4)]}
clf = MLPClassifier(random_state=242, max_iter=1000)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
↪verbose=0, n_jobs=-1)
```

```
clf_cv.fit(X_train, y_train);
```

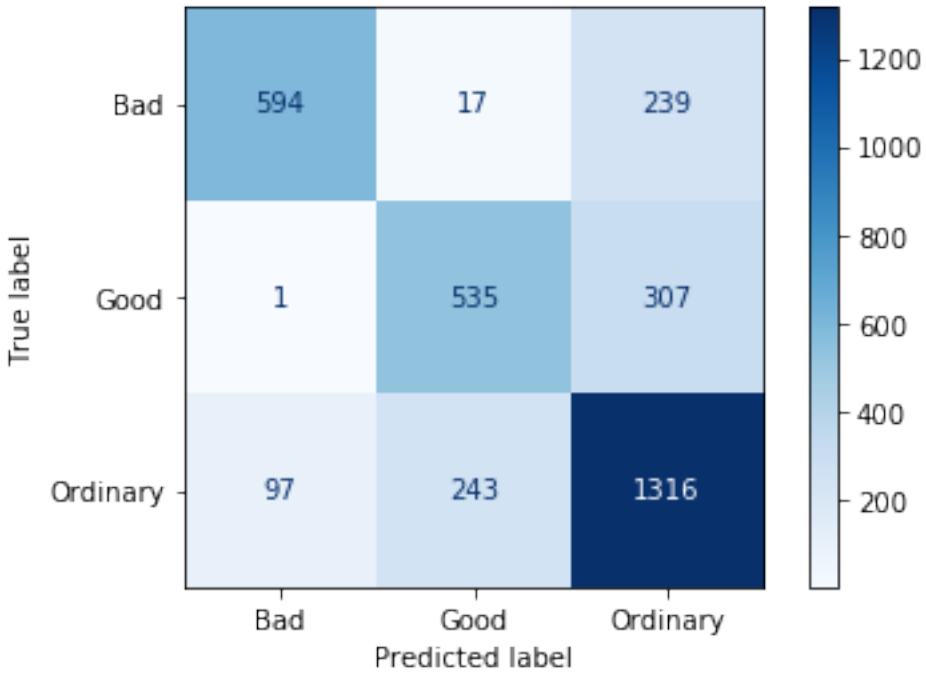
[ ]:
```
accuarcy = clf_cv.cv_results_['mean_test_score']
hidden_layer_sizes = clf_cv.cv_results_['param_hidden_layer_sizes'].data #
best_parameter = clf_cv.best_params_['hidden_layer_sizes'] #

# plt.figure(figsize=(8,5));
plt.grid(True, axis='y', color = "grey", linewidth = "1.4")
x_pos = [i for i, _ in enumerate(hidden_layer_sizes)]
plt.bar(x=x_pos, height=accuarcy);
accuarcy = [num if num == max(accuarcy) else 0 for num in accuarcy]
plt.bar(x=x_pos, height=accuarcy, color='r');

plt.xticks(x_pos, hidden_layer_sizes)
plt.ylim(bottom=0.68, top=0.74); #
plt.xlabel("Hidden Layer Sizes"); #
plt.ylabel("Accuarcy");
plt.title("Hidden Layer Sizes vs. Accuarcy"); #
print("Best accuarcy:", max(accuarcy))
print("Best hidden_layer_sizes:", best_parameter) #
```

```
Best accuarcy: 0.7290915481924962
Best hidden_layer_sizes: (9, 6)
```



[ ]:
```
nn = MLPClassifier(random_state=242, max_iter=1000,␣
 ↪hidden_layer_sizes=best_parameter)
y_pred_test_nn = nn.fit(X_train, y_train).predict(X_test) #
y_pred = y_pred_test_nn

disp = plot_confusion_matrix(nn, X_test, y_test, display_labels=["Bad", "Good",␣
 ↪"Ordinary"], cmap=plt.cm.Blues)
plt.show()
```

```
C:\Users\Joe\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[ ]:
```
nn_compare = generate_performance(y_pred_test_nn)
nn_compare
```

[ ]: [0.7301, 0.7367, 0.7301, 0.7299]

5 4. Model Evaluation and Comparison

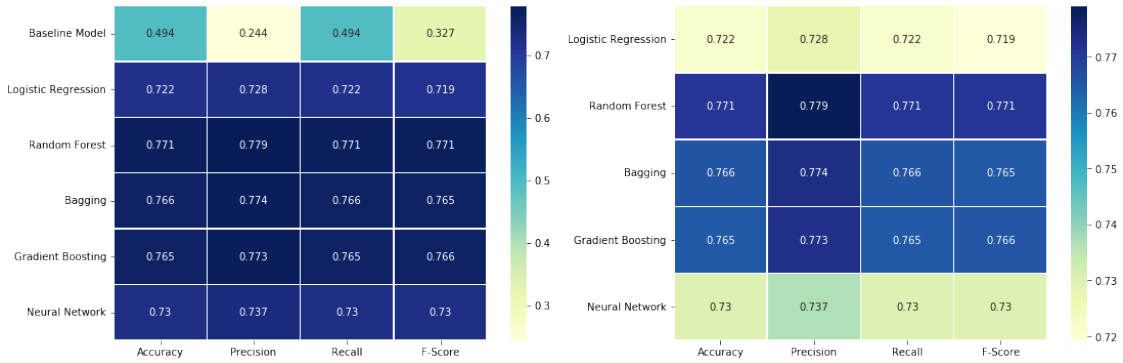
5.1 4.1 Model Comparison

[ ]:
```
comparison = pd.DataFrame(data=[baseline_compare, logistic_compare, rf_compare,
                                bag_compare, gb_compare, nn_compare],
             index=['Baseline Model', 'Logistic Regression', 'Random Forest',
                    'Bagging', 'Gradient Boosting', 'Neural Network'],
             columns=['Accuracy', 'Precision', 'Recall', 'F-Score'])
comparison
```

[ ]:

	Accuracy	Precision	Recall	F-Score
Baseline Model	0.4945	0.2445	0.4945	0.3272
Logistic Regression	0.7220	0.7281	0.7220	0.7193
Random Forest	0.7713	0.7791	0.7713	0.7713
Bagging	0.7659	0.7736	0.7659	0.7651
Gradient Boosting	0.7650	0.7729	0.7650	0.7657
Neural Network	0.7301	0.7367	0.7301	0.7299

[ ]:
```
plt.figure(figsize=(18, 6));
plt.subplot(1,2,1);
sns.heatmap(comparison, cmap="YlGnBu", annot=True, fmt='.3g', linewidths=.5);

plt.subplot(1,2,2);
sns.heatmap(comparison.iloc[1:,:], cmap="YlGnBu", annot=True, fmt='.3g',␣
 ↪linewidths=.5);
```



5.2 4.2 Model Evaluation

[ ]:
```
# Use the bootstrap to carefully test which model performs best.
# Which model would you recommend for this problem? For your chosen model,
# again use the bootstrap to construct a confidence interval for its accuracy.
def accuracy(y_pred, y_test, y_train):
    accuracy = np.mean(y_pred == y_test)
    return accuracy
```

```
def bootstrap_validation(test_data, test_label, train_label, model,␣
↪metrics_list, sample=100, random_state=0):
    n_sample = sample
    n_metrics = len(metrics_list)
    output_array = np.zeros([n_sample, n_metrics])
    output_array[:] = np.nan

    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index),␣
↪replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_predicted = model.predict(bs_data)
        for metrics_iter in range(n_metrics):
            metrics = metrics_list[metrics_iter]
            output_array[bs_iter, metrics_iter] = metrics(bs_predicted,␣
↪bs_label, train_label)

    output_df = pd.DataFrame(output_array)
    return output_df
```

[ ]:
```
models = [rf.fit(X_train, y_train),
          bag.fit(X_train, y_train),
          gb.fit(X_train, y_train)]

output_table = pd.DataFrame()
for model in models:
    tic = time.time()
    bs_output = bootstrap_validation(X_test, y_test, y_train, model,
                                     metrics_list=[accuracy],
                                     sample=300) #
    output_table = pd.concat([output_table, bs_output], axis=1)
    toc = time.time()
    print(round(toc - tic, 2), "seconds")

performance_table = pd.DataFrame(zip(np.mean(output_table), np.
↪std(output_table)),
                                 index=['Random Forest', 'Bagging', 'Gradient␣
↪Boosting'],
                                 columns=['Accuracy Mean', 'Accuracy std']).T
performance_table
```

```
21.54 seconds
23.0 seconds
4.55 seconds
```

[ ]:

	Random Forest	Bagging	Gradient Boosting
Accuracy Mean	0.770897	0.765462	0.764934
Accuracy std	0.007154	0.007310	0.007365

[ ]:
```
# Choose Random Forest,because it has best accuracy mean and accuracy standard␣
 ↪deviation
CI = np.quantile(output_table.iloc[:,0], np.array([0.025, 0.975]))
print("95-percent CI of accuracy is %s" % CI)

fig, axs = plt.subplots(ncols=1, figsize=(8, 4));
axs.set_xlabel('Boot Accuracy');
axs.set_ylabel('Count');
axs.hist(output_table.iloc[:,0], bins=20, edgecolor='red', linewidth=2,␣
 ↪color="grey", alpha=0.5);
axs.vlines(x=CI[0], ymin=0, ymax=30, color="black");
axs.vlines(x=CI[1], ymin=0, ymax=30, color="black");
plt.title("Boostrap Histgram of Random Forest");
```

```
95-percent CI of accuracy is [0.75573305 0.78411466]
```