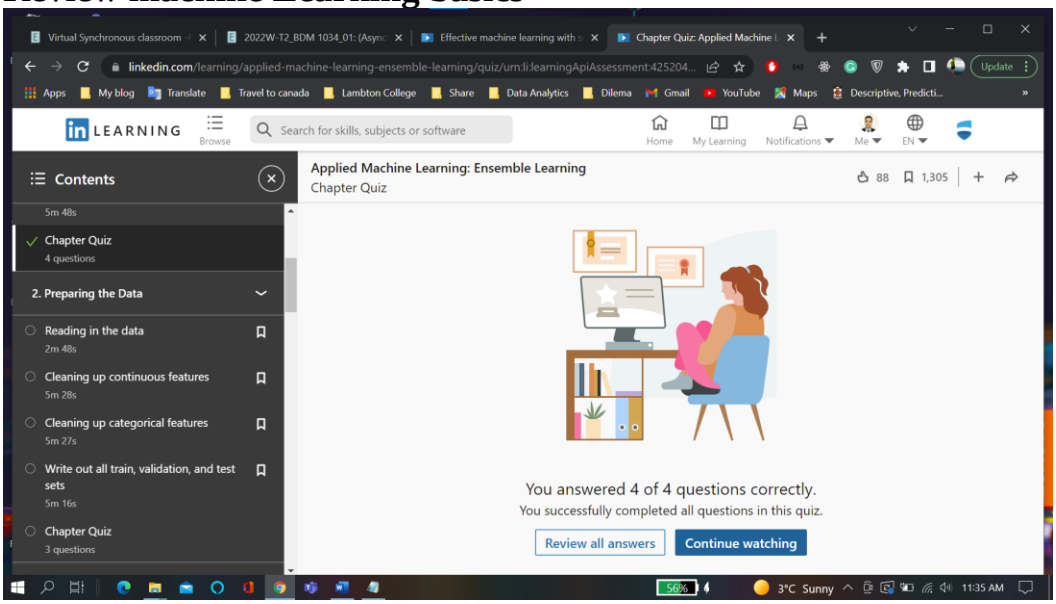
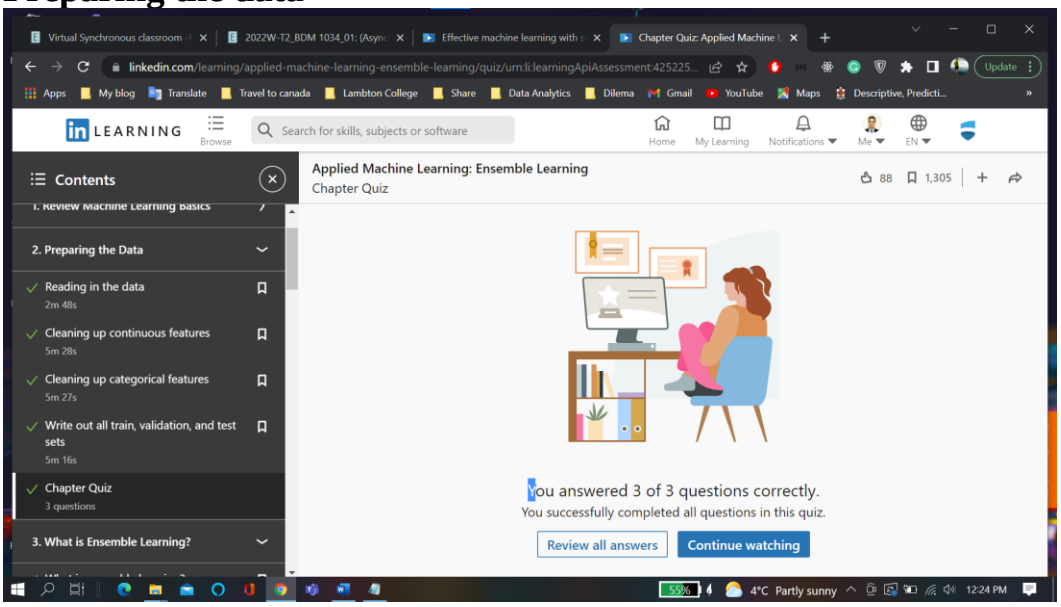
BDM 1034 - Application Design for Big Data
Week11
Submitted by: Aadarsha Chapagain
Student ID:C0825975
Submitted to: Prof. Teresa Zhu

Here I have attached the chapter quiz and exercise I have done from Linkedin learning for Course "**Applied Machine Learning: Ensemble Learning"** along with the Screenshot of score I achieved.

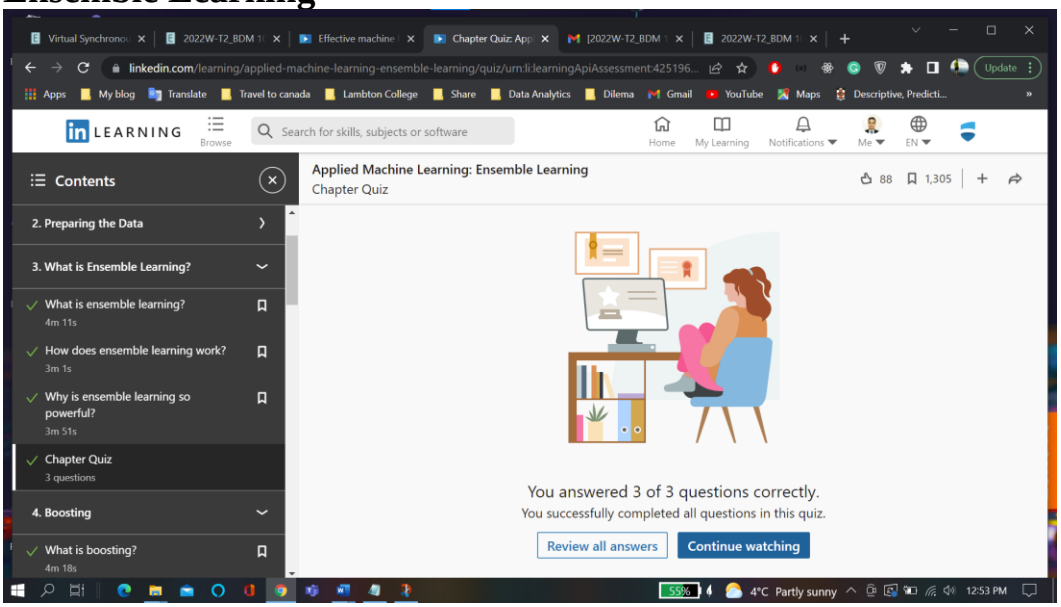
Chapter 1:
Review machine Learning basics



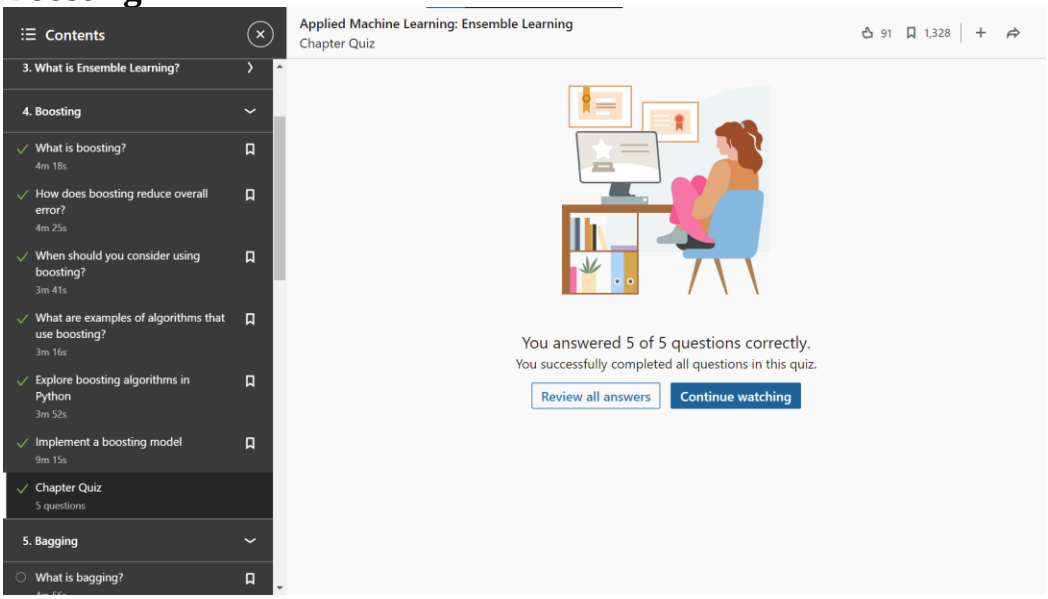
Chapter 2
Preparing the data



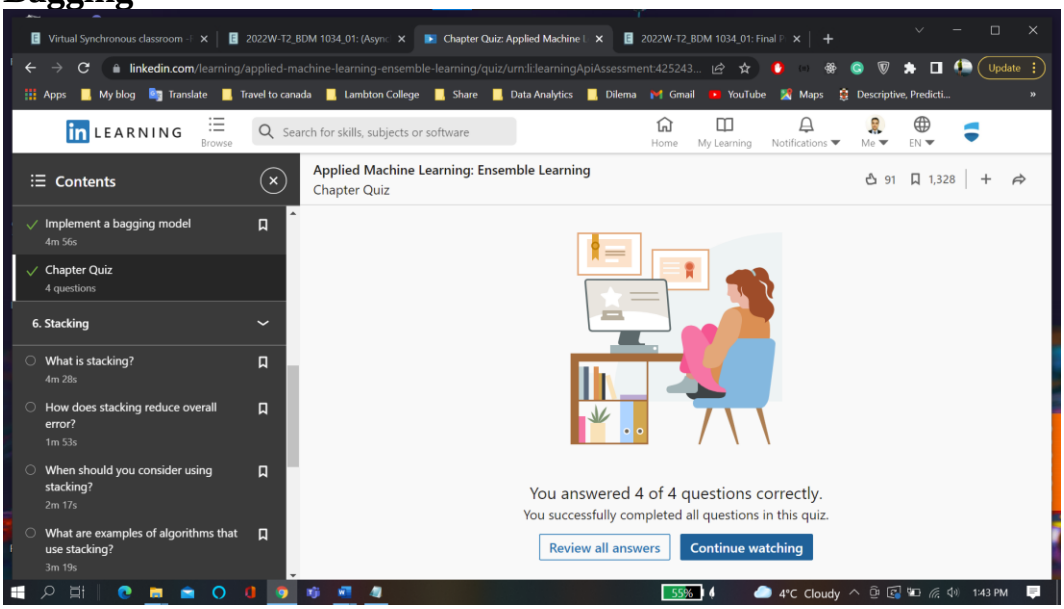
Chapter 3:
Ensemble Learning



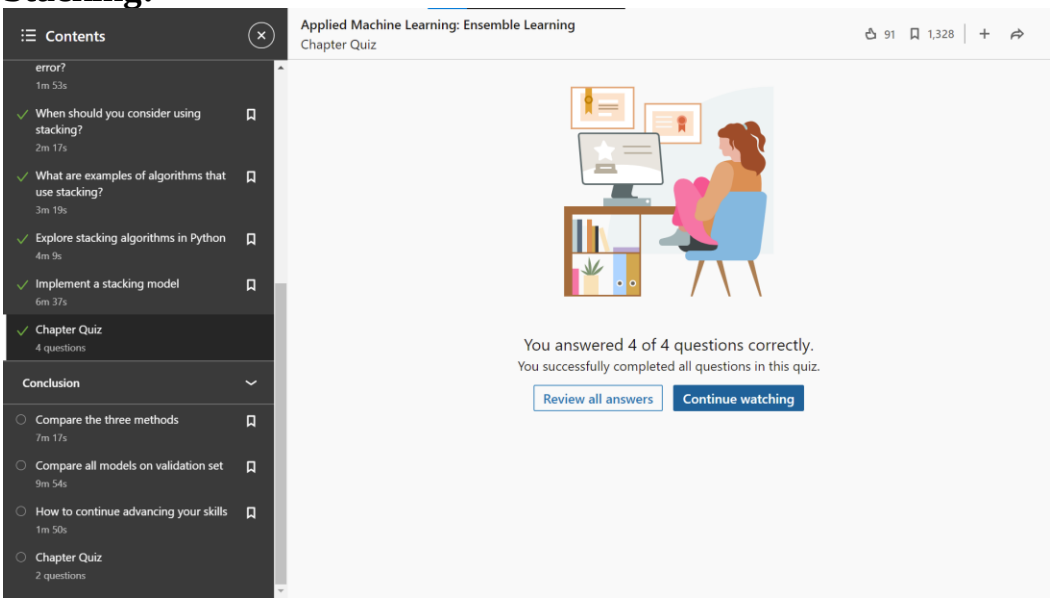
Chapter 4:
Boosting



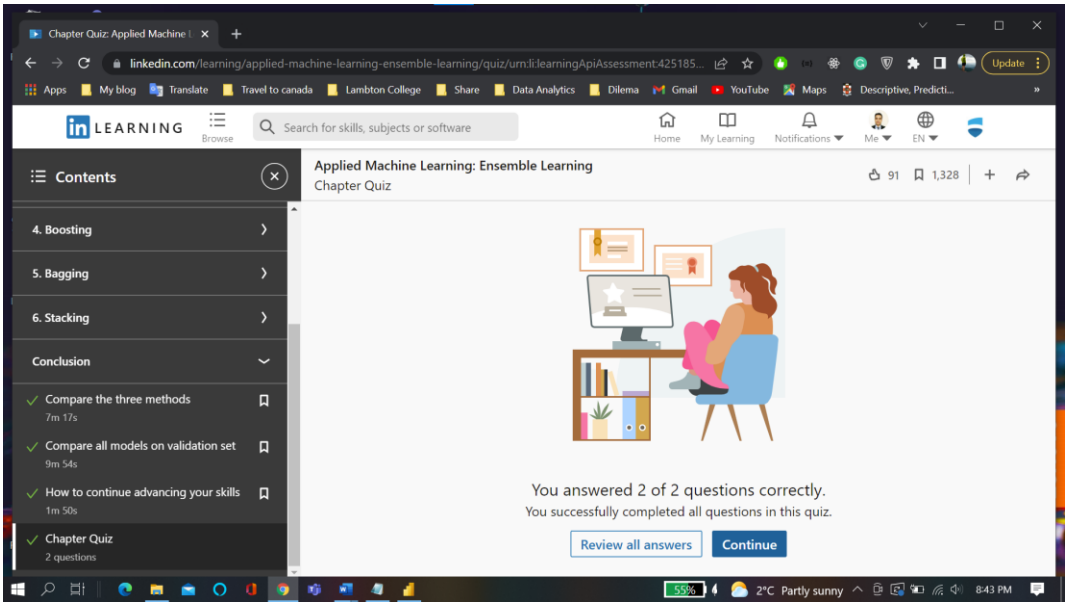
Chapter 5:
Bagging



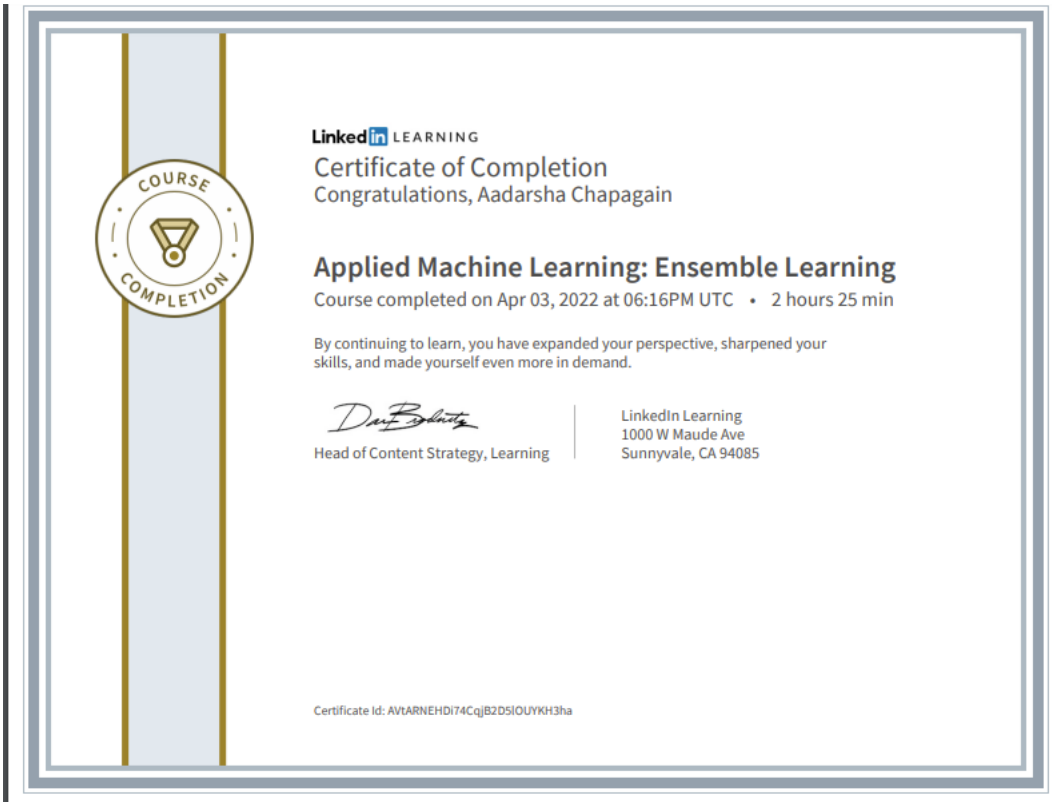
Chapter 6:
Stacking:



Conclusion



Certificate



LinkedIn LEARNING

Certificate of Completion

Congratulations, Aadarsha Chapagain

Applied Machine Learning: Ensemble Learning

Course completed on Apr 03, 2022 at 06:16PM UTC • 2 hours 25 min

By continuing to learn, you have expanded your perspective, sharpened your skills, and made yourself even more in demand.

Head of Content Strategy, Learning

LinkedIn Learning
1000 W Maude Ave
Sunnyvale, CA 94085

Certificate Id: AVtARNEHDi74CqjB2D5lOUYKH3ha

Chapter 2

Prepare Data: Read in the data

Using the Titanic dataset from this Kaggle competition (we are only using the training set).

This dataset contains information about 891 people who were on board the ship when departed on April 15th, 1912. As noted in the description on Kaggle's website, some people aboard the ship were more likely to survive the wreck than others. There were not enough lifeboats for everybody so women, children, and the upper-class were prioritized. Using the information about these 891 passengers, the challenge is to build a model to predict which people would survive based on the following fields:

- **Name** (str) - Name of the passenger
- **Pclass** (int) - Ticket class
- **Sex** (str) - Sex of the passenger
- **Age** (float) - Age in years
- **SibSp** (int) - Number of siblings and spouses aboard
- **Parch** (int) - Number of parents and children aboard
- **Ticket** (str) - Ticket number
- **Fare** (float) - Passenger fare
- **Cabin** (str) - Cabin number
- **Embarked** (str) - Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

Read in Data

In [2]:
```
import pandas as pd

titanic = pd.read_csv('./titanic.csv')
titanic.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Clean continuous variables

Fill missing for Age

In [4]:
```
titanic.isnull().sum()
```

Out[4]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [5]:
```
titanic['Age'].fillna(titanic['Age'].mean(), inplace=True)
titanic.head(10)
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	NaN	S

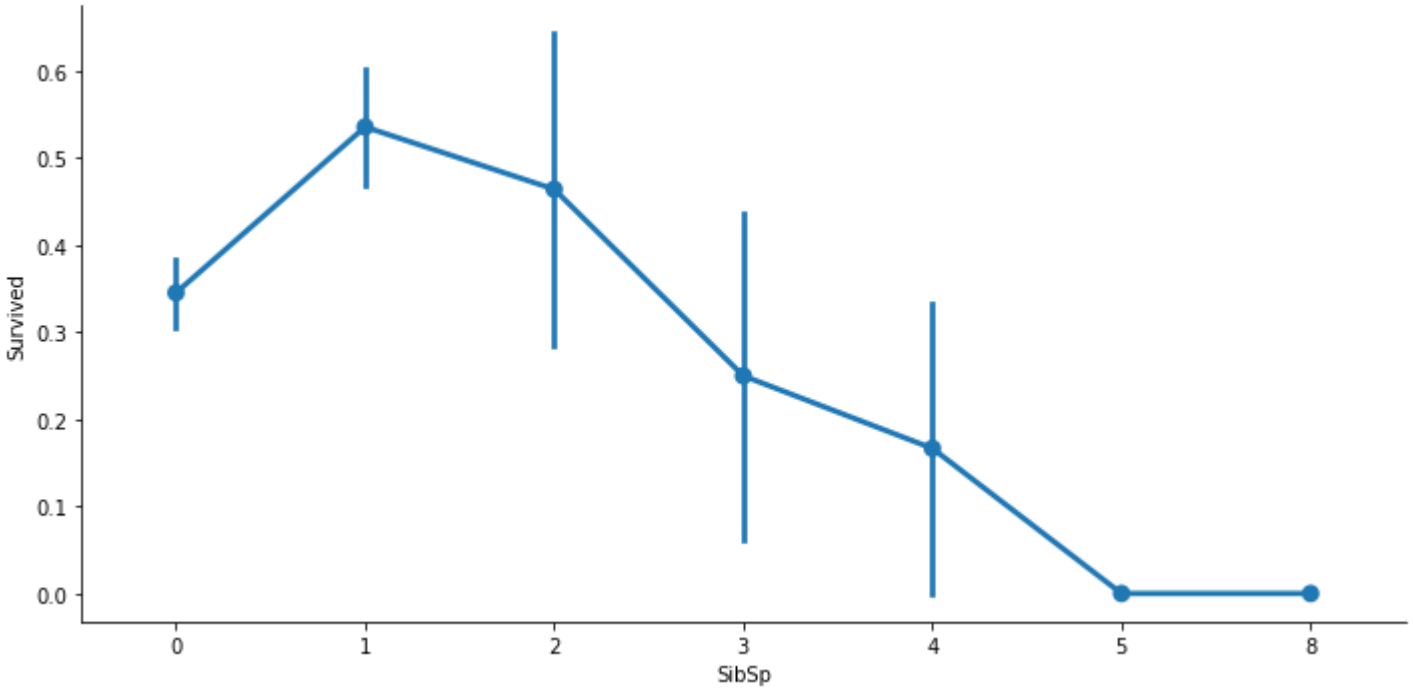
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	29.699118	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.0708	NaN	C

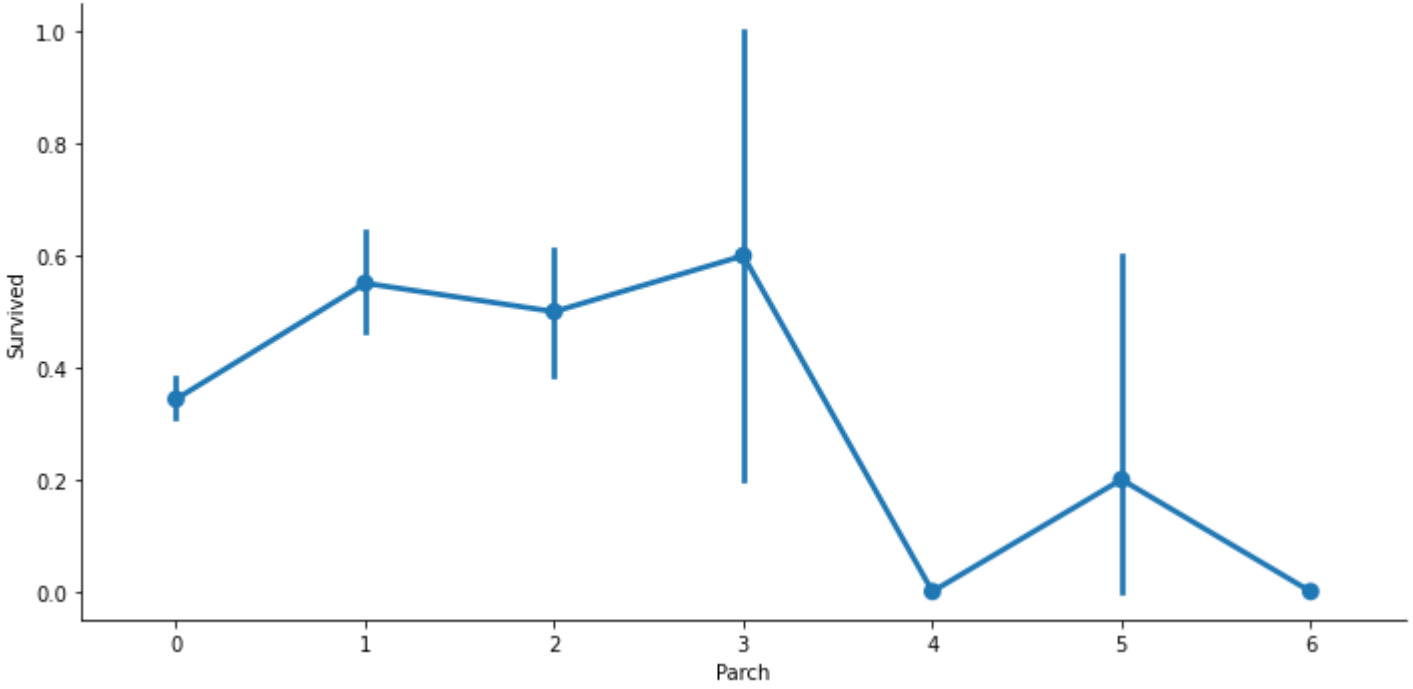
Combine SibSp & Parch

In [8]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
%matplotlib inline

for i, col in enumerate(['SibSp', 'Parch']):
    plt.figure(i)
    sns.catplot(x=col, y='Survived', data=titanic, kind='point', aspect=2, )
```

```
<Figure size 432x288 with 0 Axes>
```


Survived
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4
5
8
SibSp


Survived
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
Parch

In [9]:
```
titanic['Family_cnt'] = titanic['SibSp'] + titanic['Parch']
```

Drop unnnecessary variables

In [11]:
```
titanic.drop(['PassengerId', 'SibSp', 'Parch'], axis=1, inplace=True)
```

In [12]:
```
titanic.head()
```

Out[12]:

	Survived	Pclass	Name	Sex	Age	Ticket	Fare	Cabin	Embarked	Family_cnt
0	0	3	Braund, Mr. Owen Harris	male	22.0	A/5 21171	7.2500	NaN	S	1
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	PC 17599	71.2833	C85	C	1
2	1	3	Heikkinen, Miss. Laina	female	26.0	STON/O2. 3101282	7.9250	NaN	S	0
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	113803	53.1000	C123	S	1
4	0	3	Allen, Mr. William Henry	male	35.0	373450	8.0500	NaN	S	0

Clean categorical variables

Fill in missing & create indicator for `Cabin`

In [17]:
```
titanic.isnull().sum()
```

Out[17]:
```
Survived      0
Pclass        0
Name          0
Sex           0
Age           0
Ticket        0
Fare          0
Cabin       687
Embarked      2
Family_cnt    0
dtype: int64
```

In [20]:
```
titanic.groupby(titanic['Cabin'].isnull())['Survived'].mean()
```

Out[20]:
```
Cabin
False    0.666667
True     0.299854
Name: Survived, dtype: float64
```

In [21]:
```
titanic['Cabin_ind'] = np.where(titanic['Cabin'].isnull(), 0, 1)
titanic.head()
```

Out[21]:

	Survived	Pclass	Name	Sex	Age	Ticket	Fare	Cabin	Embarked	Family_cnt	Cabin_ind
0	0	3	Braund, Mr. Owen Harris	male	22.0	A/5 21171	7.2500	NaN	S	1	0
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	PC 17599	71.2833	C85	C	1	1
2	1	3	Heikkinen, Miss. Laina	female	26.0	STON/O2. 3101282	7.9250	NaN	S	0	0
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	113803	53.1000	C123	S	1	1
4	0	3	Allen, Mr. William Henry	male	35.0	373450	8.0500	NaN	S	0	0

Convert `Sex` to numeric

In [22]:
```
gender_num = {'male': 0, 'female': 1}

titanic['Sex'] = titanic['Sex'].map(gender_num)
titanic.head()
```

Out[22]:

	Survived	Pclass	Name	Sex	Age	Ticket	Fare	Cabin	Embarked	Family_cnt	Cabin_ind
0	0	3	Braund, Mr. Owen Harris	0	22.0	A/5 21171	7.2500	NaN	S	1	0
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	PC 17599	71.2833	C85	C	1	1
2	1	3	Heikkinen, Miss. Laina	1	26.0	STON/O2. 3101282	7.9250	NaN	S	0	0
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	113803	53.1000	C123	S	1	1
4	0	3	Allen, Mr. William Henry	0	35.0	373450	8.0500	NaN	S	0	0

Drop unnecessary variables

In [23]:
```
titanic.drop(['Cabin', 'Embarked', 'Name', 'Ticket'], axis=1, inplace=True)
titanic.head()
```

Out[23]:

	Survived	Pclass	Sex	Age	Fare	Family_cnt	Cabin_ind
0	0	3	0	22.0	7.2500	1	0
1	1	1	1	38.0	71.2833	1	1
2	1	3	1	26.0	7.9250	0	0
3	1	1	1	35.0	53.1000	1	1
4	0	3	0	35.0	8.0500	0	0

Write out cleaned data

In [27]:
```
titanic.to_csv('./titanic_cleaned.csv')
```

Prepare Data: Split data into train, validation, and test set

Using the Titanic dataset from this Kaggle competition (we are only using the training set).

In this section, we will split the data into train, validation, and test set in preparation for fitting a basic model in the next section.

Read in Data

In [30]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

titanic = pd.read_csv('./titanic_cleaned.csv')
titanic.head()
```

Out[30]:

	Unnamed: 0	Survived	Pclass	Sex	Age	Fare	Family_cnt	Cabin_ind
0	0	0	3	0	22.0	7.2500	1	0

	Unnamed: 0	Survived	Pclass	Sex	Age	Fare	Family_cnt	Cabin_ind
1	1	1	1	1	38.0	71.2833	1	1
2	2	1	3	1	26.0	7.9250	0	0
3	3	1	1	1	35.0	53.1000	1	1
4	4	0	3	0	35.0	8.0500	0	0

Split into train, validation, and test set

In [31]:
```
features = titanic.drop('Survived', axis=1)
labels = titanic['Survived']

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.4, random_state = 42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state = 42)
```

In [32]:
```
for dataset in [y_train, y_val, y_test]:
    print(round(len(dataset) / len(labels), 2))
```

```
0.6
0.2
0.2
```

Write out all data

In [35]:
```
X_train.to_csv('./train_features.csv', index=False)
X_val.to_csv('./val_features.csv', index=False)
X_test.to_csv('./test_features.csv', index=False)

y_train.to_csv('./train_labels.csv', index=False)
y_val.to_csv('./val_labels.csv', index=False)
y_test.to_csv('./test_labels.csv', index=False)
```

Chapter 4

Boosting: Explore boosting algorithms in Python

Import `GradientBoostingClassifier` and `AdaBoostClassifier` from `sklearn` and explore the hyperparameters.

Import Boosting Algorithm for Classification

In [36]:
```
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier

GradientBoostingClassifier().get_params()
```

Out[36]:
```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [37]:
```
AdaBoostClassifier().get_params()
```

Out[37]:
```
{'algorithm': 'SAMME.R',
 'base_estimator': None,
 'learning_rate': 1.0,
 'n_estimators': 50,
 'random_state': None}
```

Read in Data

In [39]:
```
import joblib
import pandas as pd
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

tr_features = pd.read_csv('./train_features.csv')
tr_labels = pd.read_csv('./train_labels.csv')
```

Hyperparameter tuning

In [41]:

```
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

In [42]:

```
gb = GradientBoostingClassifier()
parameters = {
    'n_estimators': [5, 50, 250, 500],
    'max_depth': [1, 3, 5, 7, 9],
    'learning_rate': [0.01, 0.1, 1, 10, 100]
}
cv = GridSearchCV(gb, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
BEST PARAMS: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}

0.624 (+/-0.007) for {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 5}
0.796 (+/-0.115) for {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 50}
0.796 (+/-0.115) for {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 250}
0.811 (+/-0.117) for {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}
0.624 (+/-0.007) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 5}
0.811 (+/-0.069) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
0.824 (+/-0.086) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 250}
0.828 (+/-0.074) for {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}
0.624 (+/-0.007) for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 5}
0.809 (+/-0.046) for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 50}
0.822 (+/-0.059) for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 250}
0.816 (+/-0.042) for {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 500}
0.624 (+/-0.007) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 5}
```

```
0.817 (+/-0.053) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 50}
0.807 (+/-0.026) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 250}
0.798 (+/-0.022) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 500}
0.624 (+/-0.007) for {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 5}
0.798 (+/-0.049) for {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 50}
0.787 (+/-0.036) for {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 250}
0.773 (+/-0.021) for {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 500}
0.796 (+/-0.115) for {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 5}
0.815 (+/-0.119) for {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 50}
0.818 (+/-0.109) for {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 250}
0.818 (+/-0.118) for {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}
0.813 (+/-0.071) for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 5}
0.824 (+/-0.074) for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
0.803 (+/-0.032) for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 250}
0.788 (+/-0.042) for {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
0.811 (+/-0.061) for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 5}
0.817 (+/-0.041) for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
0.798 (+/-0.048) for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 250}
0.794 (+/-0.056) for {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
0.822 (+/-0.056) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 5}
0.796 (+/-0.017) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50}
0.802 (+/-0.036) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 250}
0.8 (+/-0.034) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 500}
0.798 (+/-0.052) for {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 5}
0.798 (+/-0.045) for {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 50}
0.796 (+/-0.041) for {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 250}
0.798 (+/-0.039) for {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 500}
0.818 (+/-0.099) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 5}
0.818 (+/-0.114) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 50}
0.815 (+/-0.069) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 250}
0.79 (+/-0.08) for {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 500}
0.813 (+/-0.065) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 5}
0.788 (+/-0.069) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 50}
0.802 (+/-0.047) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 250}
0.807 (+/-0.053) for {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 500}
0.775 (+/-0.049) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 5}
0.798 (+/-0.067) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
0.807 (+/-0.057) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 250}
0.803 (+/-0.047) for {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 500}
0.766 (+/-0.013) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 5}
0.798 (+/-0.041) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 50}
0.792 (+/-0.027) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 250}
0.772 (+/-0.045) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 500}
0.783 (+/-0.049) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 5}
0.794 (+/-0.058) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 50}
```

```
0.783 (+/-0.04) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 250}
0.766 (+/-0.026) for {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 500}
0.204 (+/-0.115) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 5}
0.204 (+/-0.115) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 50}
0.204 (+/-0.115) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 250}
0.204 (+/-0.115) for {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 500}
0.369 (+/-0.374) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 5}
0.369 (+/-0.374) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 50}
0.369 (+/-0.374) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 250}
0.369 (+/-0.374) for {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 500}
0.489 (+/-0.15) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 5}
0.489 (+/-0.156) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 50}
0.494 (+/-0.154) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 250}
0.49 (+/-0.145) for {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 500}
0.607 (+/-0.172) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 5}
0.605 (+/-0.187) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 50}
0.62 (+/-0.141) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 250}
0.59 (+/-0.204) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 500}
0.706 (+/-0.103) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 5}
0.715 (+/-0.162) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 50}
0.732 (+/-0.13) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 250}
0.721 (+/-0.105) for {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 500}
0.376 (+/-0.007) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 5}
0.376 (+/-0.007) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 50}
0.376 (+/-0.007) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 250}
0.376 (+/-0.007) for {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 500}
0.281 (+/-0.118) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 5}
0.281 (+/-0.118) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 50}
0.281 (+/-0.118) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 250}
0.281 (+/-0.118) for {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 500}
0.47 (+/-0.192) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 5}
0.468 (+/-0.188) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 50}
0.472 (+/-0.178) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 250}
0.475 (+/-0.185) for {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 500}
0.565 (+/-0.11) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 5}
0.563 (+/-0.129) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 50}
0.584 (+/-0.164) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 250}
0.539 (+/-0.114) for {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 500}
0.646 (+/-0.12) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 5}
0.646 (+/-0.082) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 50}
0.588 (+/-0.203) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 250}
0.61 (+/-0.187) for {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 500}
```

In [43]:

```
cv.best_estimator_
```

Out[43]:
```
GradientBoostingClassifier(learning_rate=0.01, n_estimators=500)
```

Write out pickled model

In [44]:
```
joblib.dump(cv.best_estimator_, './models/GB_model.pkl')
```

Out[44]:
```
['./models/GB_model.pkl']
```

Chapter 5

Bagging: Explore bagging algorithms in Python

Import `RandomForestClassifier` from `sklearn` and explore the hyperparameters.

Import Random Forest Algorithm for Classification

In [45]:
```
from sklearn.ensemble import RandomForestClassifier

RandomForestClassifier().get_params()
```

Out[45]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [47]:
```
import joblib
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

tr_features = pd.read_csv('./train_features.csv')
tr_labels = pd.read_csv('./train_labels.csv')
```

Hyperparameter tuning

In [49]:
```
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

In [50]:
```
rf = RandomForestClassifier()
parameters = {
    'n_estimators': [5, 50, 250, 500],
    'max_depth': [4, 8, 16, 32, None]
}
cv = GridSearchCV(rf, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
BEST PARAMS: {'max_depth': 4, 'n_estimators': 500}

0.811 (+/-0.086) for {'max_depth': 4, 'n_estimators': 5}
0.818 (+/-0.119) for {'max_depth': 4, 'n_estimators': 50}
0.809 (+/-0.141) for {'max_depth': 4, 'n_estimators': 250}
0.822 (+/-0.103) for {'max_depth': 4, 'n_estimators': 500}
0.813 (+/-0.067) for {'max_depth': 8, 'n_estimators': 5}
0.818 (+/-0.107) for {'max_depth': 8, 'n_estimators': 50}
0.813 (+/-0.087) for {'max_depth': 8, 'n_estimators': 250}
0.813 (+/-0.101) for {'max_depth': 8, 'n_estimators': 500}
0.798 (+/-0.079) for {'max_depth': 16, 'n_estimators': 5}
0.815 (+/-0.073) for {'max_depth': 16, 'n_estimators': 50}
0.807 (+/-0.077) for {'max_depth': 16, 'n_estimators': 250}
```

```
0.805 (+/-0.075) for {'max_depth': 16, 'n_estimators': 500}
0.794 (+/-0.09) for {'max_depth': 32, 'n_estimators': 5}
0.807 (+/-0.081) for {'max_depth': 32, 'n_estimators': 50}
0.805 (+/-0.08) for {'max_depth': 32, 'n_estimators': 250}
0.813 (+/-0.069) for {'max_depth': 32, 'n_estimators': 500}
0.79 (+/-0.052) for {'max_depth': None, 'n_estimators': 5}
0.807 (+/-0.088) for {'max_depth': None, 'n_estimators': 50}
0.809 (+/-0.075) for {'max_depth': None, 'n_estimators': 250}
0.803 (+/-0.069) for {'max_depth': None, 'n_estimators': 500}
```

In [51]:
```
cv.best_estimator_
```

Out[51]:
```
RandomForestClassifier(max_depth=4, n_estimators=500)
```

Write out pickled model

In [53]:
```
joblib.dump(cv.best_estimator_, './models/RF_model.pkl')
```

Out[53]:
```
['./models/RF_model.pkl']
```

Chapter 6

Stacking: Explore stacking algorithms in Python

Import `StackingClassifier` from `sklearn` and explore the hyperparameters.

In [55]:
```
from sklearn.ensemble import StackingClassifier, GradientBoostingClassifier, RandomForestClassifier

estimators = [('gb', GradientBoostingClassifier()), ('rf', RandomForestClassifier())]
StackingClassifier(estimators = estimators).get_params()
```

Out[55]:
```
{'cv': None,
 'estimators': [('gb', GradientBoostingClassifier()),
  ('rf', RandomForestClassifier())],
 'final_estimator': None,
 'n_jobs': None,
 'passthrough': False,
 'stack_method': 'auto',
 'verbose': 0,
```

```
'gb': GradientBoostingClassifier(),
'rf': RandomForestClassifier(),
'gb__ccp_alpha': 0.0,
'gb__criterion': 'friedman_mse',
'gb__init': None,
'gb__learning_rate': 0.1,
'gb__loss': 'deviance',
'gb__max_depth': 3,
'gb__max_features': None,
'gb__max_leaf_nodes': None,
'gb__min_impurity_decrease': 0.0,
'gb__min_impurity_split': None,
'gb__min_samples_leaf': 1,
'gb__min_samples_split': 2,
'gb__min_weight_fraction_leaf': 0.0,
'gb__n_estimators': 100,
'gb__n_iter_no_change': None,
'gb__random_state': None,
'gb__subsample': 1.0,
'gb__tol': 0.0001,
'gb__validation_fraction': 0.1,
'gb__verbose': 0,
'gb__warm_start': False,
'rf__bootstrap': True,
'rf__ccp_alpha': 0.0,
'rf__class_weight': None,
'rf__criterion': 'gini',
'rf__max_depth': None,
'rf__max_features': 'auto',
'rf__max_leaf_nodes': None,
'rf__max_samples': None,
'rf__min_impurity_decrease': 0.0,
'rf__min_impurity_split': None,
'rf__min_samples_leaf': 1,
'rf__min_samples_split': 2,
'rf__min_weight_fraction_leaf': 0.0,
'rf__n_estimators': 100,
'rf__n_jobs': None,
'rf__oob_score': False,
'rf__random_state': None,
'rf__verbose': 0,
'rf__warm_start': False}
```

Read in Data

Hyperparameter tuning

In [56]:
```
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

In [57]:
```
estimators = [('rf', RandomForestClassifier()),
              ('gb', GradientBoostingClassifier())]

sc = StackingClassifier(estimators=estimators)
sc.get_params()
```

Out[57]:
```
{'cv': None,
 'estimators': [('rf', RandomForestClassifier()),
  ('gb', GradientBoostingClassifier())],
 'final_estimator': None,
 'n_jobs': None,
 'passthrough': False,
 'stack_method': 'auto',
 'verbose': 0,
 'rf': RandomForestClassifier(),
 'gb': GradientBoostingClassifier(),
 'rf__bootstrap': True,
 'rf__ccp_alpha': 0.0,
 'rf__class_weight': None,
 'rf__criterion': 'gini',
 'rf__max_depth': None,
 'rf__max_features': 'auto',
 'rf__max_leaf_nodes': None,
 'rf__max_samples': None,
 'rf__min_impurity_decrease': 0.0,
 'rf__min_impurity_split': None,
 'rf__min_samples_leaf': 1,
 'rf__min_samples_split': 2,
 'rf__min_weight_fraction_leaf': 0.0,
 'rf__n_estimators': 100,
 'rf__n_jobs': None,
 'rf__oob_score': False,
 'rf__random_state': None,
```

```
'rf__verbose': 0,
'rf__warm_start': False,
'gb__ccp_alpha': 0.0,
'gb__criterion': 'friedman_mse',
'gb__init': None,
'gb__learning_rate': 0.1,
'gb__loss': 'deviance',
'gb__max_depth': 3,
'gb__max_features': None,
'gb__max_leaf_nodes': None,
'gb__min_impurity_decrease': 0.0,
'gb__min_impurity_split': None,
'gb__min_samples_leaf': 1,
'gb__min_samples_split': 2,
'gb__min_weight_fraction_leaf': 0.0,
'gb__n_estimators': 100,
'gb__n_iter_no_change': None,
'gb__random_state': None,
'gb__subsample': 1.0,
'gb__tol': 0.0001,
'gb__validation_fraction': 0.1,
'gb__verbose': 0,
'gb__warm_start': False}
```

In [59]:

```
import joblib
import pandas as pd
from sklearn.ensemble import StackingClassifier, GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

tr_features = pd.read_csv('./train_features.csv')
tr_labels = pd.read_csv('./train_labels.csv')
```

Hyperparameter tuning

In [60]:

```
def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

In [61]:
```
estimators = [('rf', RandomForestClassifier()),
              ('gb', GradientBoostingClassifier())]

sc = StackingClassifier(estimators=estimators)
sc.get_params()
```

Out[61]:
```
{'cv': None,
 'estimators': [('rf', RandomForestClassifier()),
  ('gb', GradientBoostingClassifier())],
 'final_estimator': None,
 'n_jobs': None,
 'passthrough': False,
 'stack_method': 'auto',
 'verbose': 0,
 'rf': RandomForestClassifier(),
 'gb': GradientBoostingClassifier(),
 'rf__bootstrap': True,
 'rf__ccp_alpha': 0.0,
 'rf__class_weight': None,
 'rf__criterion': 'gini',
 'rf__max_depth': None,
 'rf__max_features': 'auto',
 'rf__max_leaf_nodes': None,
 'rf__max_samples': None,
 'rf__min_impurity_decrease': 0.0,
 'rf__min_impurity_split': None,
 'rf__min_samples_leaf': 1,
 'rf__min_samples_split': 2,
 'rf__min_weight_fraction_leaf': 0.0,
 'rf__n_estimators': 100,
 'rf__n_jobs': None,
 'rf__oob_score': False,
 'rf__random_state': None,
 'rf__verbose': 0,
 'rf__warm_start': False,
 'gb__ccp_alpha': 0.0,
 'gb__criterion': 'friedman_mse',
 'gb__init': None,
 'gb__learning_rate': 0.1,
 'gb__loss': 'deviance',
 'gb__max_depth': 3,
 'gb__max_features': None,
 'gb__max_leaf_nodes': None,
 'gb__min_impurity_decrease': 0.0,
```

```
'gb__min_impurity_split': None,
'gb__min_samples_leaf': 1,
'gb__min_samples_split': 2,
'gb__min_weight_fraction_leaf': 0.0,
'gb__n_estimators': 100,
'gb__n_iter_no_change': None,
'gb__random_state': None,
'gb__subsample': 1.0,
'gb__tol': 0.0001,
'gb__validation_fraction': 0.1,
'gb__verbose': 0,
'gb__warm_start': False}
```

In [62]:

```
parameters = {
    'gb__n_estimators': [50, 100],
    'rf__n_estimators': [50, 100],
    'final_estimator': [LogisticRegression(C=0.1),
                        LogisticRegression(C=1),
                        LogisticRegression(C=10)],
    'passthrough': [True, False]
}
cv = GridSearchCV(sc, parameters, cv=5)
cv.fit(tr_features, tr_labels.values.ravel())

print_results(cv)
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
BEST PARAMS: {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_estimato
rs': 50}

0.822 (+/-0.13) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_es
timators': 50}
0.815 (+/-0.133) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_e
stimators': 100}
0.824 (+/-0.062) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_
estimators': 50}
0.824 (+/-0.062) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_
estimators': 100}
0.83 (+/-0.105) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_e
stimators': 50}
0.815 (+/-0.126) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_
estimators': 100}
0.82 (+/-0.058) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_
estimators': 50}
0.82 (+/-0.05) for {'final_estimator': LogisticRegression(C=0.1), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_e
stimators': 100}
0.817 (+/-0.099) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_est
imators': 50}
0.818 (+/-0.093) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_est
imators': 100}
0.818 (+/-0.071) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_es
timators': 50}
0.817 (+/-0.07) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_est
imators': 100}
0.824 (+/-0.095) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_es
timators': 50}
0.818 (+/-0.102) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_es
timators': 100}
0.82 (+/-0.061) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_es
timators': 50}
0.822 (+/-0.063) for {'final_estimator': LogisticRegression(C=1), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_e
stimators': 100}
0.815 (+/-0.088) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_es
timators': 50}
0.817 (+/-0.091) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 50, 'passthrough': True, 'rf__n_es
timators': 100}
0.822 (+/-0.061) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_e
stimators': 50}
0.817 (+/-0.069) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 50, 'passthrough': False, 'rf__n_e
stimators': 100}
0.824 (+/-0.082) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_e
stimators': 50}
```

```
0.82 (+/-0.083) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 100, 'passthrough': True, 'rf__n_es
timators': 100}
0.824 (+/-0.065) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_
estimators': 50}
0.818 (+/-0.064) for {'final_estimator': LogisticRegression(C=10), 'gb__n_estimators': 100, 'passthrough': False, 'rf__n_
estimators': 100}
S:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Write out pickled model

In [63]:
```
joblib.dump(cv.best_estimator_, './models/stacked_model.pkl')
```

Out[63]:
```
['./models/stacked_model.pkl']
```

Conclusion: Compare model results and final model selection

Using the Titanic dataset from this Kaggle competition.

In this section, we will do the following:

1. Evaluate all of our saved models on the validation set
2. Select the best model based on performance on the validation set
3. Evaluate that model on the holdout test set

Read in Data

In [66]:
```
import joblib
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score
from time import time
```

```
val_features = pd.read_csv('./val_features.csv')
val_labels = pd.read_csv('./val_labels.csv')

te_features = pd.read_csv('./test_features.csv')
te_labels = pd.read_csv('./test_labels.csv')
```

In [67]:
```
gb_mdl = joblib.load('./models/GB_model.pkl')
rf_mdl = joblib.load('./models/RF_model.pkl')
stacked_mdl = joblib.load('./models/stacked_model.pkl')
```

Evaluate models on the validation set

In [68]:
```
def evaluate_model(model, features, labels):
    start = time()
    pred = model.predict(features)
    end = time()
    accuracy = round(accuracy_score(labels, pred), 3)
    precision = round(precision_score(labels, pred), 3)
    recall = round(recall_score(labels, pred), 3)
    print('{} -- Accuracy: {} / Precision: {} / Recall: {} / Latency: {}ms'.format(str(model).split('(')[0],
                                                                                   accuracy,
                                                                                   precision,
                                                                                   recall,
                                                                                   round((end - start)*1000, 1)))
```

In [69]:
```
for mdl in [gb_mdl, rf_mdl, stacked_mdl]:
    evaluate_model(mdl, val_features, val_labels)
```

```
GradientBoostingClassifier -- Accuracy: 0.809 / Precision: 0.804 / Recall: 0.631 / Latency: 8.3ms
RandomForestClassifier -- Accuracy: 0.809 / Precision: 0.816 / Recall: 0.615 / Latency: 52.1ms
StackingClassifier -- Accuracy: 0.803 / Precision: 0.778 / Recall: 0.646 / Latency: 9.1ms
```

Evaluate best model on test set

In [70]:
```
evaluate_model(rf_mdl, te_features, te_labels)
```

```
RandomForestClassifier -- Accuracy: 0.799 / Precision: 0.87 / Recall: 0.618 / Latency: 54.0ms
```

In [ ]:

BDM 1034 - Application Design for Big Data
Week11
Submitted by: Aadarsha Chapagain
Student ID:C0825975
Submitted to: Prof. Teresa Zhu

Here I have attached the certificate I achieved from Linkedin learning for Course "**Machine learning with Using Scikit-Learn"** along with the demo.

Certificate:

Chapter 2

In [1]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
```

In [2]:
```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['species'] = data.target
df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [3]:
```
feature_names = ['sepal length (cm)',
                 'sepal width (cm)',
                 'petal length (cm)',
                 'petal width (cm)']
```

In [4]:
```
# Multiple column features matrix to convert to NumPy Array
df.loc[:, feature_names]
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [5]:
```
# Convert to numpy array
x = df.loc[:, feature_names].values
```

In [6]:
```
# Make sure NumPy array is two dimensional
x.shape
```

Out[6]:
```
(150, 4)
```

In [7]:
```
# Pandas series to convert to NumPy Array
df.loc[:, 'species']
```

Out[7]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
```

```
147    2
148    2
149    2
Name: species, Length: 150, dtype: int32
```

In [8]:
```
y = df.loc[:, 'species'].values
```

In [9]:
```
y.shape
```

Out[9]:
```
(150,)
```

2_3

In [10]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [11]:
```
df = pd.read_csv("data/linear.csv")
df.head()
```

Out[11]:

	x	y
0	0.000000	-51.000000
1	25.000000	-12.000000
2	117.583220	134.907414
3	108.922466	134.085180
4	69.887445	NaN

In [12]:
```
# Look at the shape of the dataframe
df.shape
```

Out[12]:
```
(102, 2)
```

In [13]:
```
# There are missing values in the y column which is what we will predict
df.isnull().sum()
```

Out[13]:
```
x    0
y    8
dtype: int64
```

In [14]:
```
# Remove entire rows from dataframe if they contain any nans in them or 'all'
# this may not be the best strategy for our dataset
df = df.dropna(how = 'any')
```

In [15]:
```
# There are no more missing values
df.isnull().sum()
```

Out[15]:
```
x    0
y    0
dtype: int64
```

In [16]:
```
df.shape
```

Out[16]:
```
(94, 2)
```

In [17]:
```
# Convert x column to numpy array
X = df.loc[:, ['x']].values
```

In [18]:
```
# Features Matrix needs to be at 2 dimensional
X.shape
```

Out[18]:
```
(94, 1)
```

In [19]:
```
y = df.loc[:, 'y'].values
```

In [20]:
```
y.shape
```

Out[20]:
```
(94,)
```

In [21]:
```
# Make a linear regression instance
reg = LinearRegression(fit_intercept=True)
```

In [22]:
```
reg.fit(X,y)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
# Input needs to be two dimensional (reshape makes input two dimensional )
reg.predict(X[0].reshape(-1,1))
```

Out[23]:
```
array([-50.99119328])
```

In [24]:
```
reg.predict(X[0:10])
```

Out[24]:
```
array([-50.99119328, -11.39905237, 135.223663  , 121.50775193,
       102.37289634,  31.0056196 ,   4.46431068,  74.84474012,
        20.82088826,  72.16749711])
```

In [25]:
```
score = reg.score(X, y)
print(score)
```

```
0.979881836115762
```

In [26]:
```
reg.coef_
```

Out[26]:
```
array([1.58368564])
```

In [27]:
```
reg.intercept_
```

Out[27]:
```
-50.99119328333397
```

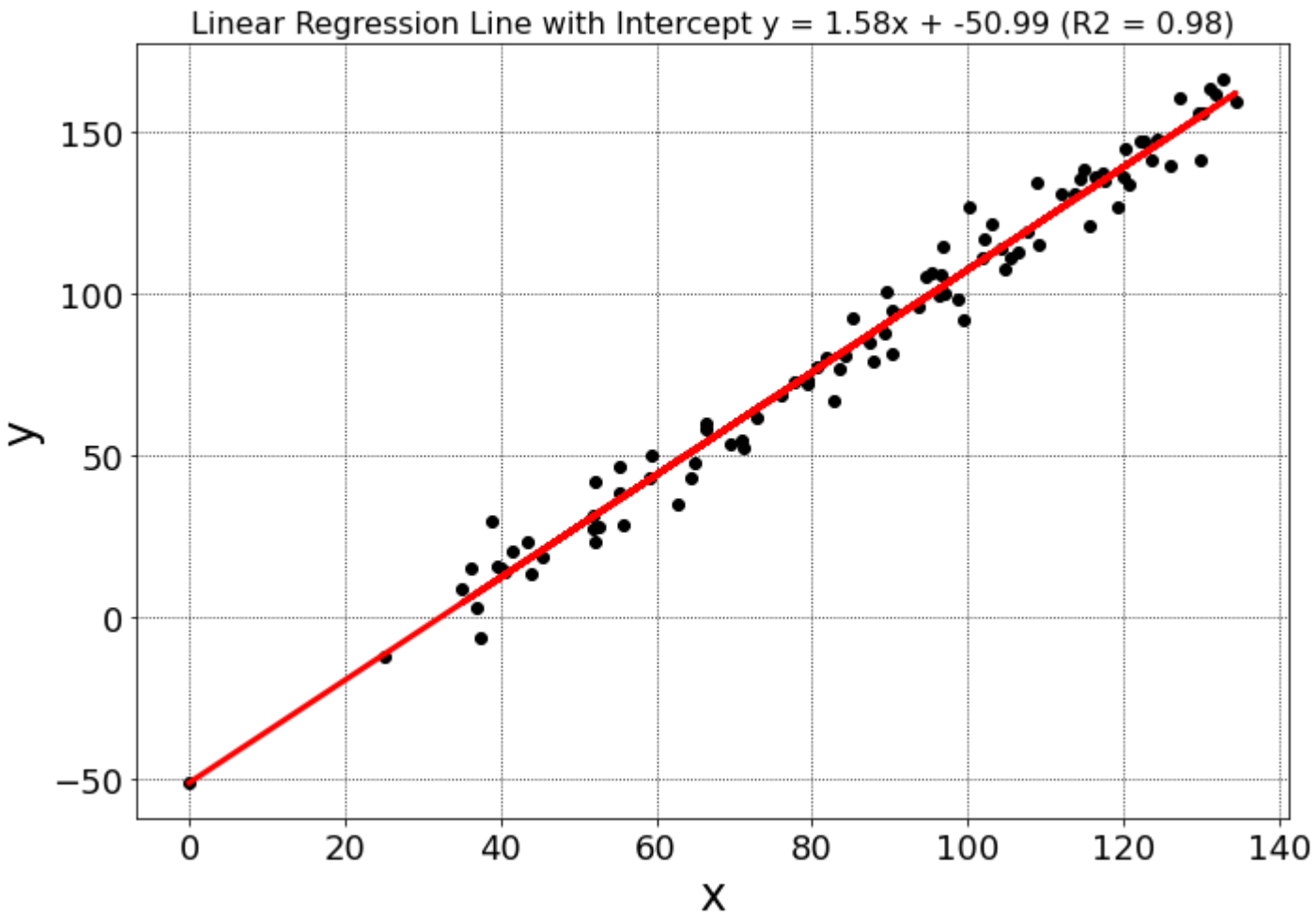
In [28]:
```
m = reg.coef_[0]

b = reg.intercept_
```

```
# following slope intercept form
print("formula: y = {:.2f}x + {:.2f}".format(m, b) )
```

```
formula: y = 1.58x + -50.99
```

In [29]:

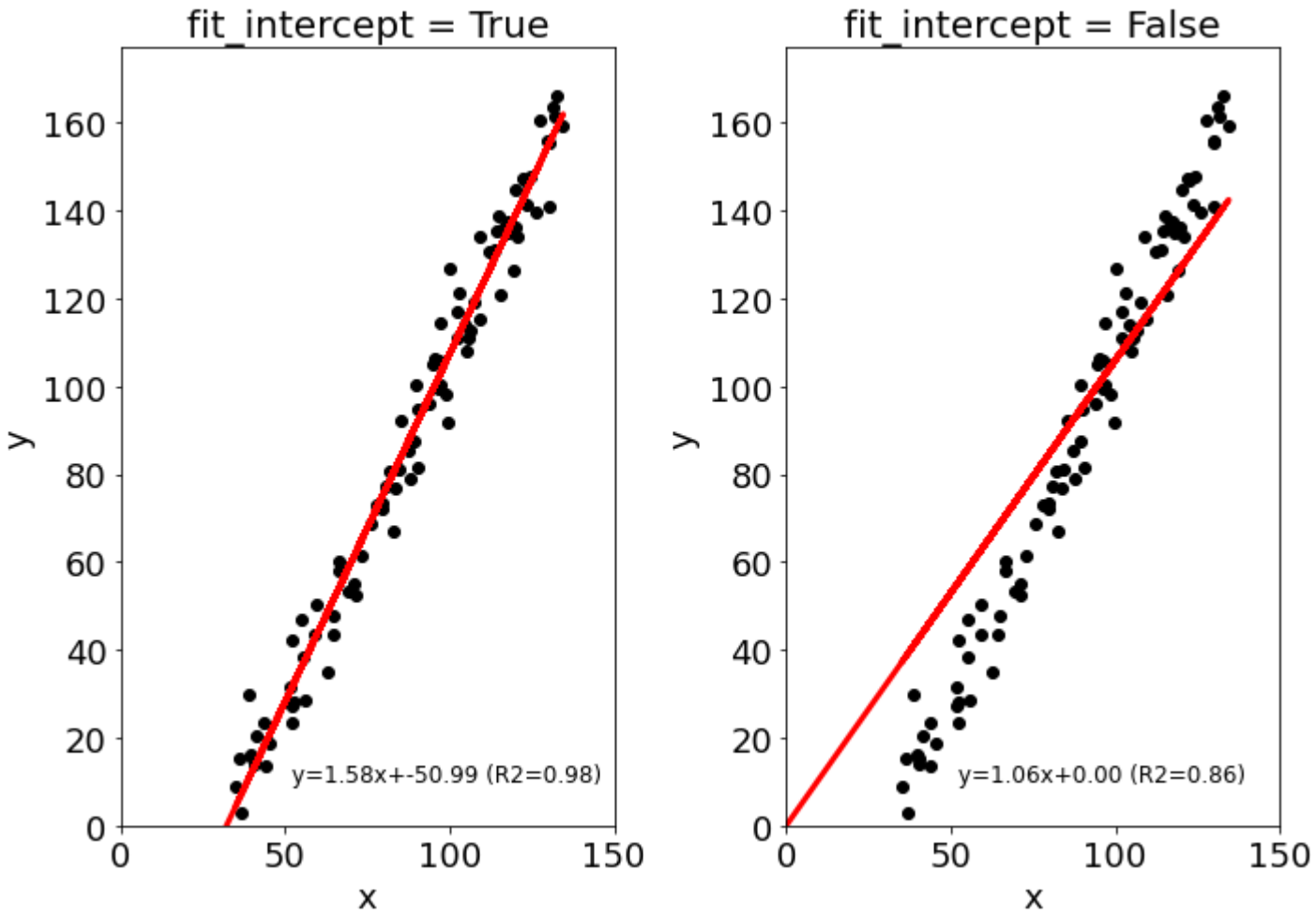
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));

ax.scatter(X, y, color='black');
ax.plot(X, reg.predict(X), color='red',linewidth=3);
ax.grid(True,
        axis = 'both',
        zorder = 0,
        linestyle = ':',
        color = 'k')
ax.tick_params(labelsize = 18)
ax.set_xlabel('x', fontsize = 24)
ax.set_ylabel('y', fontsize = 24)
ax.set_title("Linear Regression Line with Intercept y = {:.2f}x + {:.2f} (R2 = {:.2f})".format(m, b, score), fontsize = 1
fig.tight_layout()
#fig.savefig('images/linearregression', dpi = 300)
```



In [30]:
```
# Model with Intercept (like earlier in notebook)
reg_inter = LinearRegression(fit_intercept=True)
reg_inter.fit(X,y)
predictions_inter = reg_inter.predict(X)
score_inter = reg_inter.score(X, y)
```

In [31]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (10,7));

for index, model in enumerate([LinearRegression(fit_intercept=True), LinearRegression(fit_intercept=False)]):
    model.fit(X,y)
    predictions = model.predict(X)
```

```
    score = model.score(X, y)
    m = model.coef_[0]
    b = model.intercept_

    ax[index].scatter(X, y, color='black');
    ax[index].plot(X, model.predict(X), color='red',linewidth=3);

    ax[index].tick_params(labelsize = 18)
    ax[index].set_xlabel('x', fontsize = 18)
    ax[index].set_ylabel('y', fontsize = 18)
    ax[index].set_xlim(left = 0, right = 150)
    ax[index].set_ylim(bottom = 0)

    ax[index].text(50, 10, " y={:.2f}x+{:.2f} (R2={:.2f})".format(m, b, score), fontsize = 12)

ax[0].set_title('fit_intercept = True', fontsize = 20)
ax[1].set_title('fit_intercept = False',  fontsize = 20)
fig.tight_layout()
```



Train test split

In [32]:

```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
```

In [33]:
```
data = load_boston()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[33]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [34]:
```
X = df.loc[:, ['RM', 'LSTAT', 'PTRATIO']].values
```

In [35]:
```
y = df.loc[:, 'target'].values
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=3)
```

In [37]:
```
# Make a linear regression instance
reg = LinearRegression(fit_intercept=True)

# Train the model on the training set.
reg.fit(X_train, y_train)
```

Out[37]:
```
LinearRegression()
```

Model Performance

In [ ]:
```
# Test the model on the testing set and evaluate the performance
score = reg.score(X_test, y_test)
print(score)
```

2_5

In [38]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn import metrics
```

In [39]:
```
df = pd.read_csv('data/modifiedIris2Classes.csv')
```

In [40]:
```
df.shape
```

Out[40]:
```
(100, 5)
```

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(df[['petal length (cm)']], df['target'], random_state=0)
```

In [42]:
```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [43]:
```
clf = LogisticRegression()
```

In [44]:
```
clf.fit(X_train, y_train)
```

Out[44]:
```
LogisticRegression()
```

In [45]:
```
# One observation's petal length after standardization
X_test[0].reshape(1,-1)
```

Out[45]:
```
array([[-0.12093628]])
```

In [46]:
```
print('prediction', clf.predict(X_test[0].reshape(1,-1))[0])
print('probability', clf.predict_proba(X_test[0].reshape(1,-1)))
```

```
prediction 0
probability [[0.52720087 0.47279913]]
```

In [47]:
```
example_df = pd.DataFrame()
example_df.loc[:, 'petal length (cm)'] = X_test.reshape(-1)
example_df.loc[:, 'target'] = y_test.values
example_df['logistic_preds'] = pd.DataFrame(clf.predict_proba(X_test))[1]
```

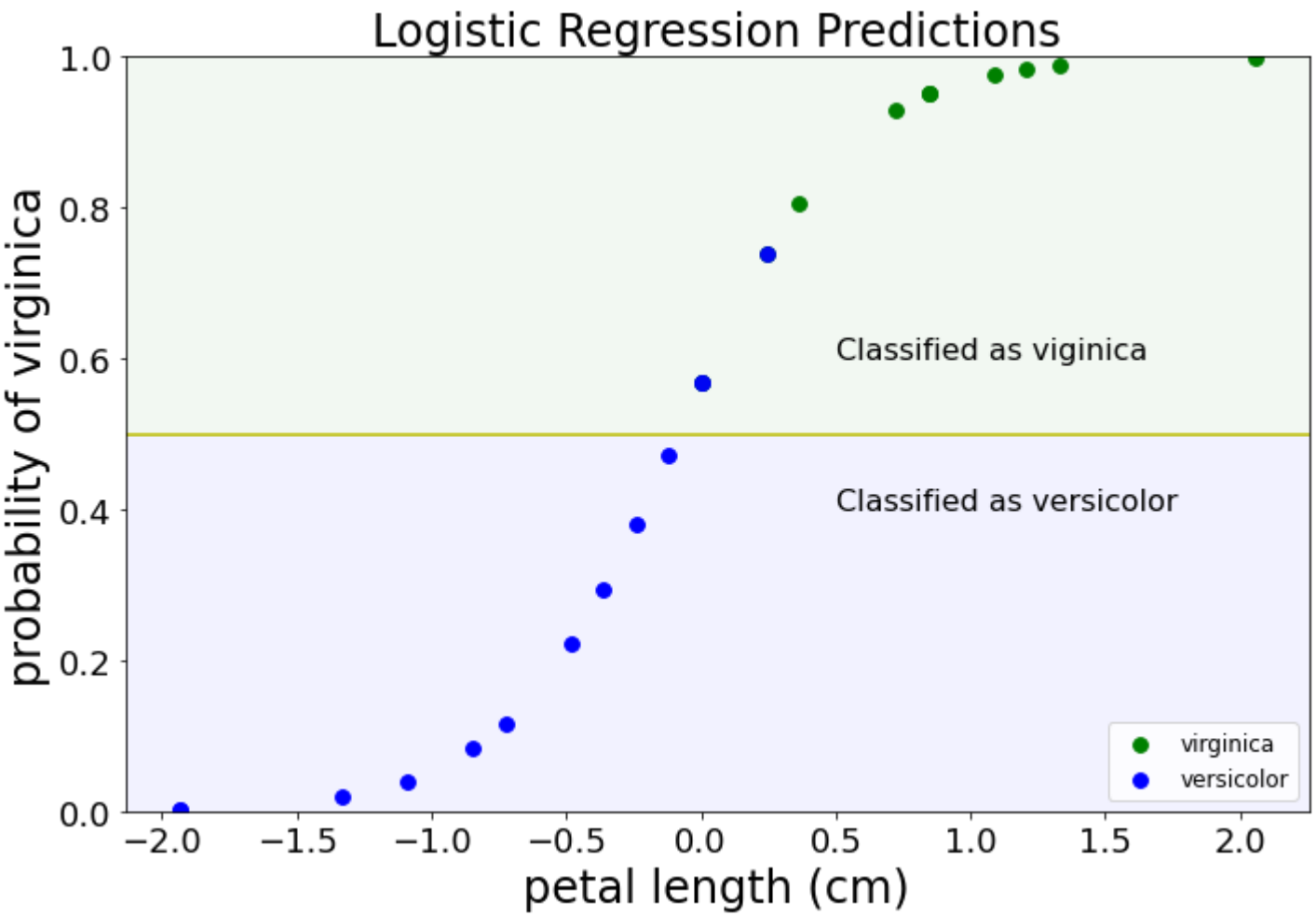
In [48]:
```
example_df.head()
```

Out[48]:

	petal length (cm)	target	logistic_preds
0	-0.120936	0	0.472799
1	0.846554	1	0.950658
2	0.000000	0	0.568197
3	2.055917	1	0.998879
4	1.330299	1	0.988926

In [49]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));


virginicaFilter = example_df['target'] == 1
versicolorFilter = example_df['target'] == 0

ax.scatter(example_df.loc[virginicaFilter, 'petal length (cm)'].values,
```

```
            example_df.loc[virginicaFilter, 'logistic_preds'].values,
           color = 'g',
           s = 60,
           label = 'virginica')


ax.scatter(example_df.loc[versicolorFilter, 'petal length (cm)'].values,
            example_df.loc[versicolorFilter, 'logistic_preds'].values,
           color = 'b',
           s = 60,
           label = 'versicolor')

ax.axhline(y = .5, c = 'y')

ax.axhspan(.5, 1, alpha=0.05, color='green')
ax.axhspan(0, .4999, alpha=0.05, color='blue')
ax.text(0.5, .6, 'Classified as viginica', fontsize = 16)
ax.text(0.5, .4, 'Classified as versicolor', fontsize = 16)

ax.set_ylim(0,1)
ax.legend(loc = 'lower right', markerscale = 1.0, fontsize = 12)
ax.tick_params(labelsize = 18)
ax.set_xlabel('petal length (cm)', fontsize = 24)
ax.set_ylabel('probability of virginica', fontsize = 24)
ax.set_title('Logistic Regression Predictions', fontsize = 24)
fig.tight_layout()
```

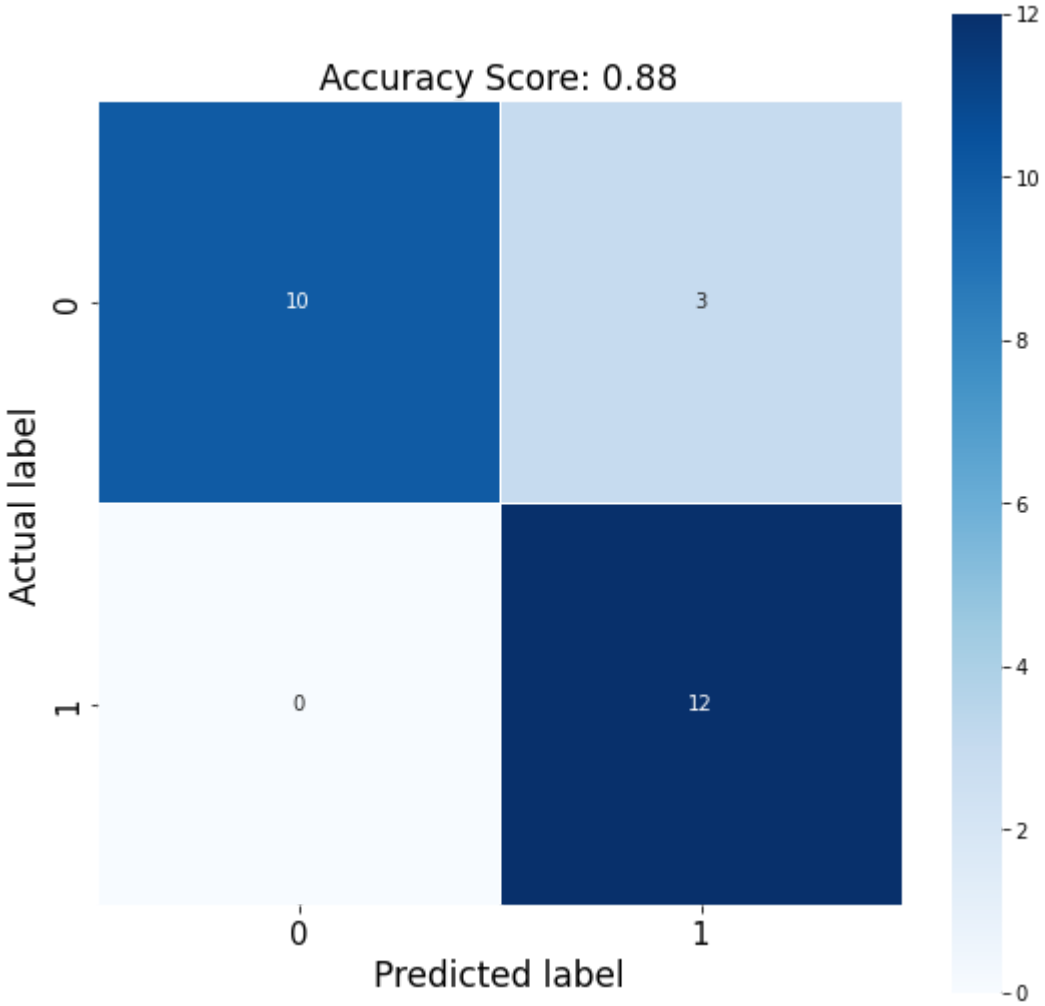


In [50]:
```
score = clf.score(X_test, y_test)
print(score)
```

```
0.88
```

In [51]:
```
cm = metrics.confusion_matrix(y_test, clf.predict(X_test))

plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True,
            fmt=".0f",
            linewidths=.5,
            square = True,
```

```
            cmap = 'Blues');
plt.ylabel('Actual label', fontsize = 17);
plt.xlabel('Predicted label', fontsize = 17);
plt.title('Accuracy Score: {}'.format(score), size = 17);
plt.tick_params(labelsize= 15)
```



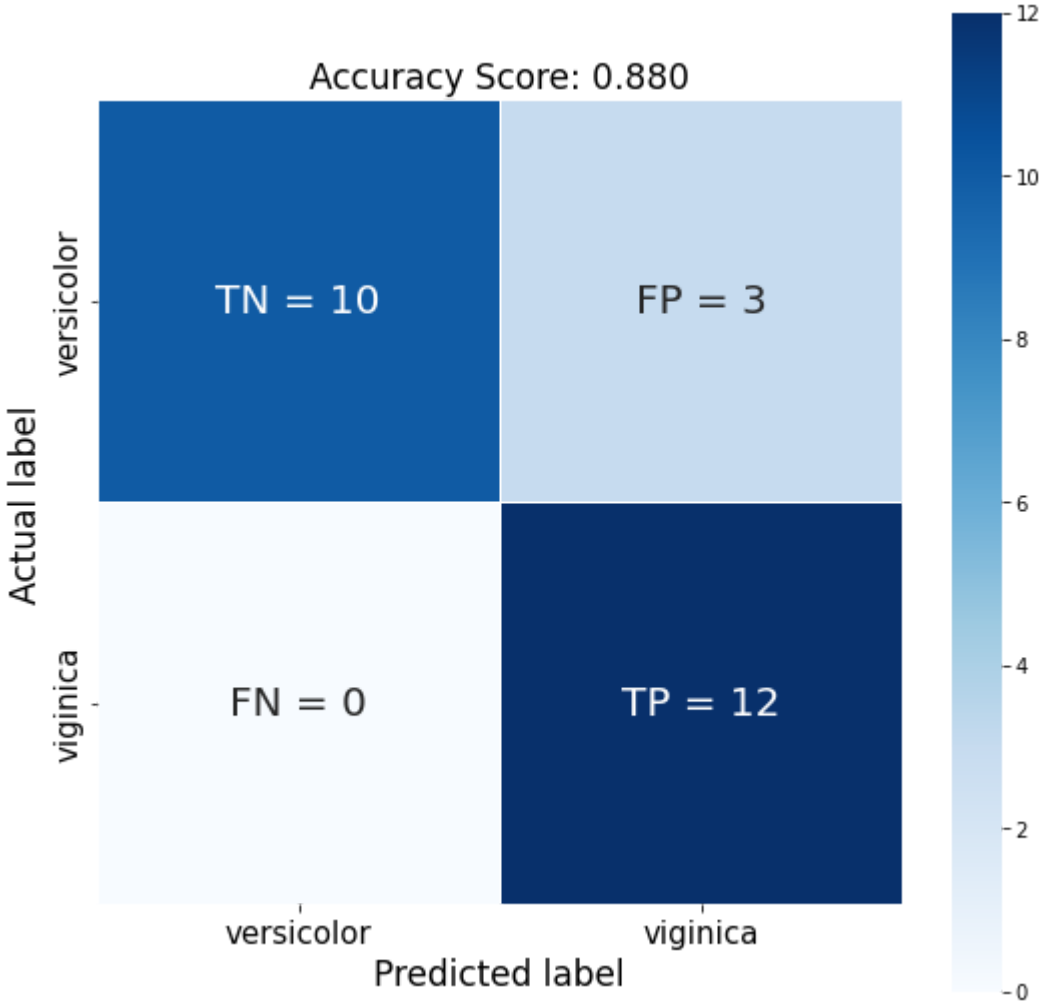
In [52]:

```
# ignore this code

modified_cm = []
for index,value in enumerate(cm):
    if index == 0:
        modified_cm.append(['TN = ' + str(value[0]), 'FP = ' + str(value[1])])
```

```
    if index == 1:
        modified_cm.append(['FN = ' + str(value[0]), 'TP = ' + str(value[1])])
```

In [53]:

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=np.array(modified_cm),
            fmt="",
            annot_kws={"size": 20},
            linewidths=.5,
            square = True,
            cmap = 'Blues',
            xticklabels = ['versicolor', 'viginica'],
            yticklabels = ['versicolor', 'viginica'],
            );

plt.ylabel('Actual label', fontsize = 17);
plt.xlabel('Predicted label', fontsize = 17);
plt.title('Accuracy Score: {:.3f}'.format(score), size = 17);
plt.tick_params(labelsize= 15)
```



2_6 one vs Rest

In [54]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
```

In [55]:
```
df = pd.read_csv('data/modifiedDigits4Classes.csv')
```

In [56]:
```
df.head()
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	label
0	0	0	5	13	9	1	0	0	0	0	...	0	0	0	6	13	10	0	0	0	0
1	0	0	0	12	13	5	0	0	0	0	...	0	0	0	0	11	16	10	0	0	1
2	0	0	0	4	15	12	0	0	0	0	...	0	0	0	0	3	11	16	9	0	2
3	0	0	7	15	13	1	0	0	0	8	...	0	0	0	7	13	13	9	0	0	3
4	0	0	1	9	15	11	0	0	0	0	...	0	0	0	1	10	13	3	0	0	0

5 rows × 65 columns

In [57]:
```
df.shape
```

Out[57]:
```
(720, 65)
```

Visualize Each Digit

In [59]:
```
pixel_colnames = df.columns[:-1]
```

In [60]:
```
pixel_colnames
```

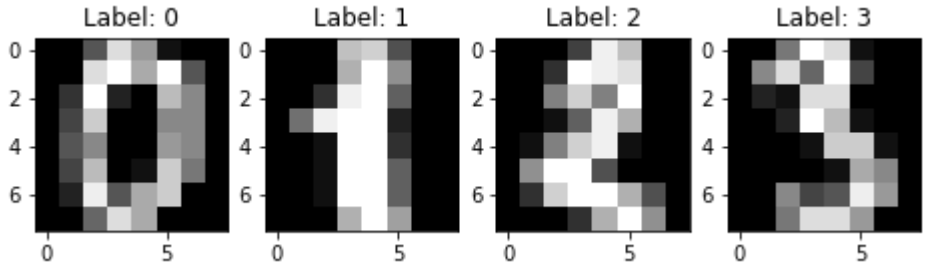
Out[60]:
```
Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
       '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48',
       '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60',
       '61', '62', '63'],
      dtype='object')
```

In [61]:
```
# Get all columns except the label column for the first image
```

```
image_values = df.loc[0, pixel_colnames].values
```

In [62]:

```
plt.figure(figsize=(10,2))
for index in range(0, 4):

    plt.subplot(1, 5, 1 + index )
    image_values = df.loc[index, pixel_colnames].values
    image_label = df.loc[index, 'label']
    plt.imshow(image_values.reshape(8,8), cmap ='gray')
    plt.title('Label: ' + str(image_label))
```



In [63]:

```
X_train, X_test, y_train, y_test = train_test_split(df[pixel_colnames], df['label'], random_state=0)
```

In [64]:

```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [65]:

```
# multi_class is specifying one versus rest
clf = LogisticRegression(solver='liblinear',
                         multi_class='ovr',
                         random_state = 0)

clf.fit(X_train, y_train)
print('Training accuracy:', clf.score(X_train, y_train))
print('Test accuracy:', clf.score(X_test, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

In [66]:
```
clf.intercept_
```

Out[66]:
```
array([-2.712674  , -3.54379096, -3.18367757, -2.623974  ])
```

In [67]:
```
clf.coef_.shape
```

Out[67]:
```
(4, 64)
```

In [68]:
```
clf.predict_proba(X_test[0:1])
```

Out[68]:
```
array([[0.00183123, 0.98368966, 0.00536378, 0.00911533]])
```

In [69]:
```
clf.predict(X_test[0:1])
```

Out[69]:
```
array([1], dtype=int64)
```

2_7 Decision trees

In [71]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [72]:
```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[72]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [74]:
```
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state = 0)
```

In [75]:
```
clf.fit(X_train, y_train)
```

Out[75]:
```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

In [76]:
```
# Predict for One Observation
clf.predict(X_test.iloc[0].values.reshape(1, -1))
```

Out[76]:
```
array([2])
```

In [77]:
```
clf.predict(X_test[0:10])
```

Out[77]:
```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1])
```

In [78]:
```
score = clf.score(X_test, y_test)
print(score)
```

```
0.8947368421052632
```

In [79]:
```
# List of values to try for max_depth:
max_depth_range = list(range(1, 6))
```

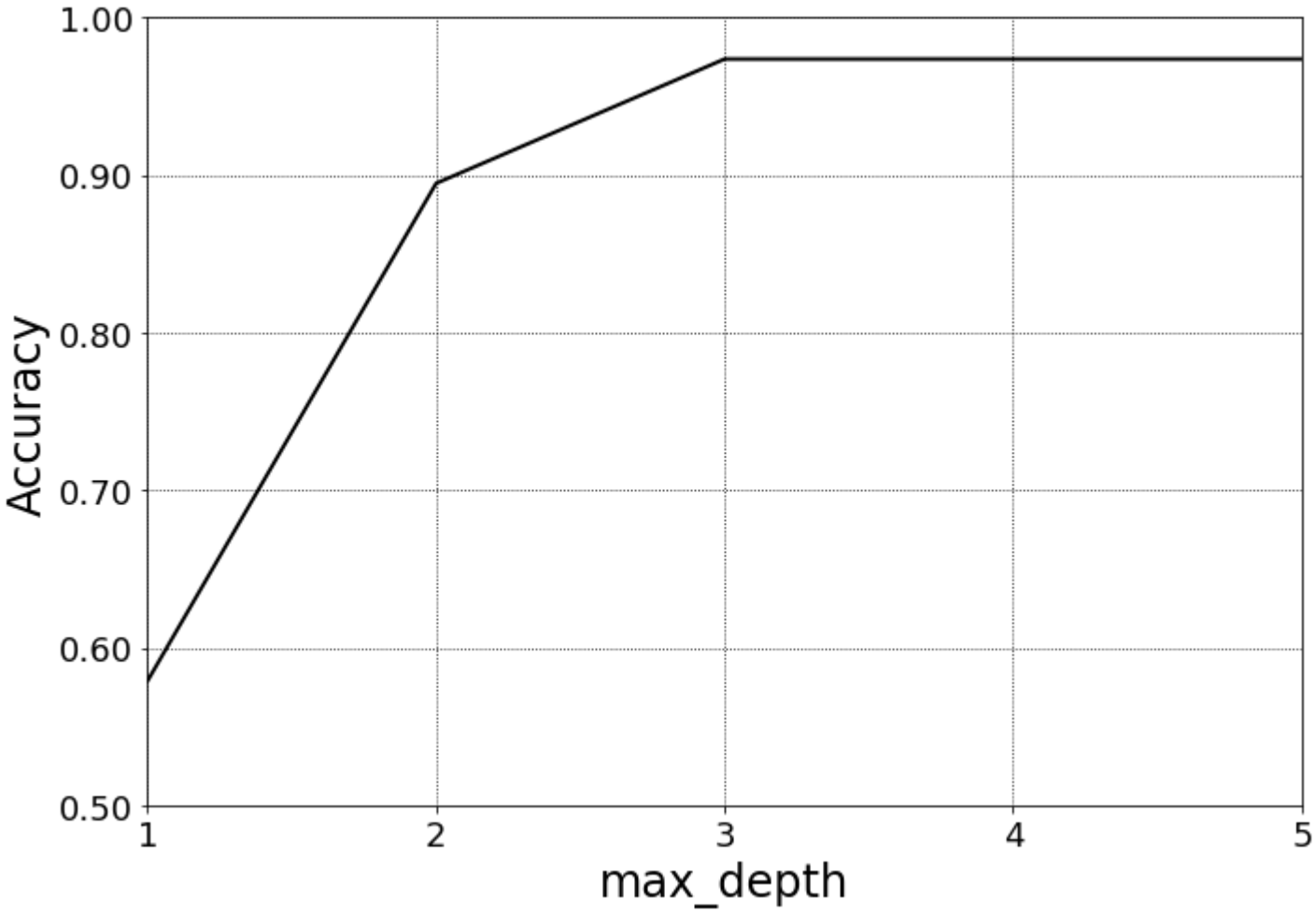
```
# List to store the average RMSE for each value of max_depth:
accuracy = []

for depth in max_depth_range:

    clf = DecisionTreeClassifier(max_depth = depth,
                                 random_state = 0)
    clf.fit(X_train, y_train)

    score = clf.score(X_test, y_test)
    accuracy.append(score)
```

In [80]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));

ax.plot(max_depth_range,
        accuracy,
        lw=2,
        color='k')

ax.set_xlim([1, 5])
ax.set_ylim([.50, 1.00])
ax.grid(True,
        axis = 'both',
        zorder = 0,
        linestyle = ':',
        color = 'k')

yticks = ax.get_yticks()

y_ticklist = []
for tick in yticks:
    y_ticklist.append(str(tick).ljust(4, '0')[0:4])
ax.set_yticklabels(y_ticklist)
ax.tick_params(labelsize = 18)
ax.set_xticks([1,2,3,4,5])
ax.set_xlabel('max_depth', fontsize = 24)
ax.set_ylabel('Accuracy', fontsize = 24)
fig.tight_layout()
#fig.savefig('images/max_depth_vs_accuracy.png', dpi = 300)
```

```
C:\Users\aadar\AppData\Local\Temp/ipykernel_992/1914883130.py:21: UserWarning: FixedFormatter should only be used togethe
r with FixedLocator
```

```
ax.set_yticklabels(y_ticklist)
```



2_8 Modeling pattern

In [81]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn import tree
```

In [83]:
```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[83]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [84]:
```
X_train, X_test, Y_train, Y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [85]:
```
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state = 0)
```

In [86]:
```
clf.fit(X_train, Y_train)
```

Out[86]:
```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

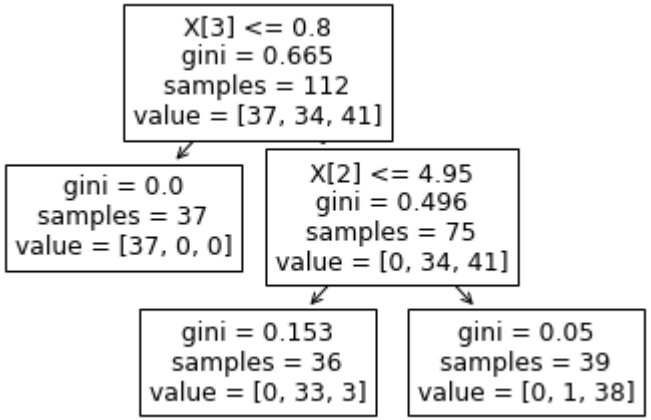
In [87]:
```
clf.predict(X_test[0:10])
```

Out[87]:
```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1])
```

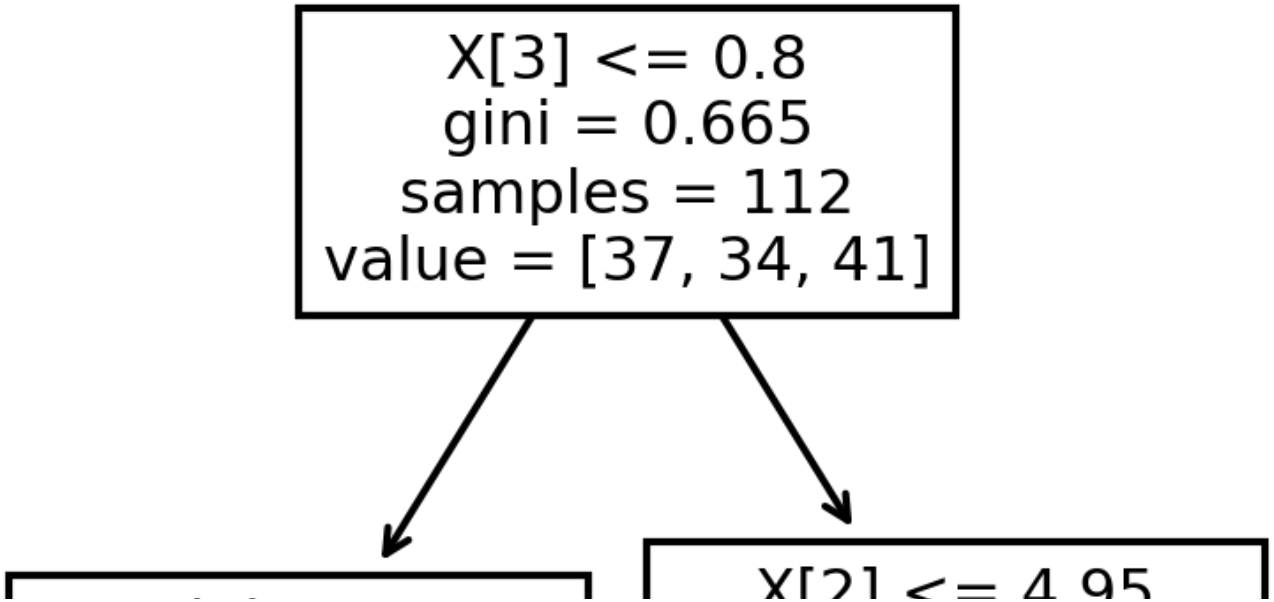
In [88]:
```
score = clf.score(X_test, Y_test)
print(score)
```

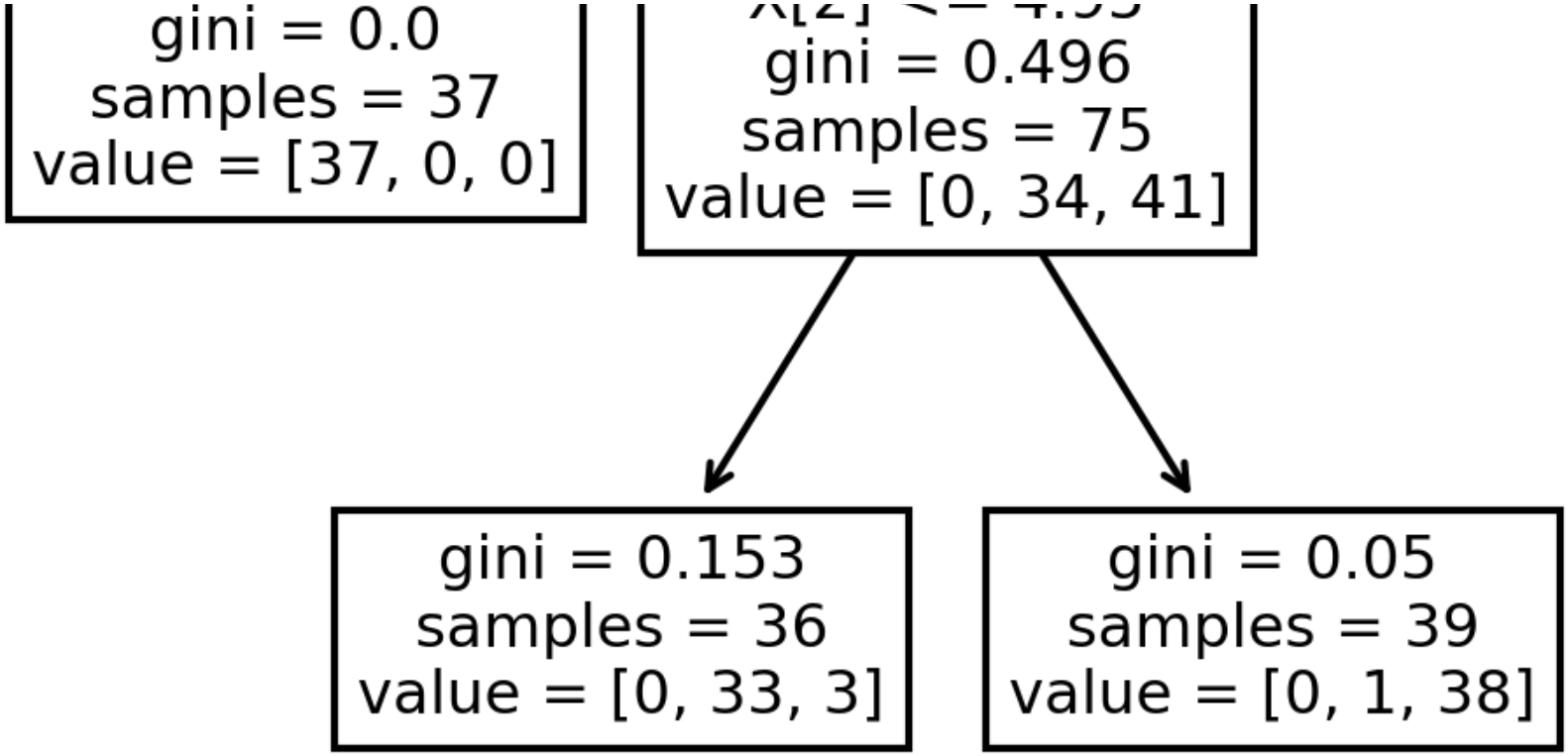
```
0.8947368421052632
```

In [89]:
```
tree.plot_tree(clf);
```



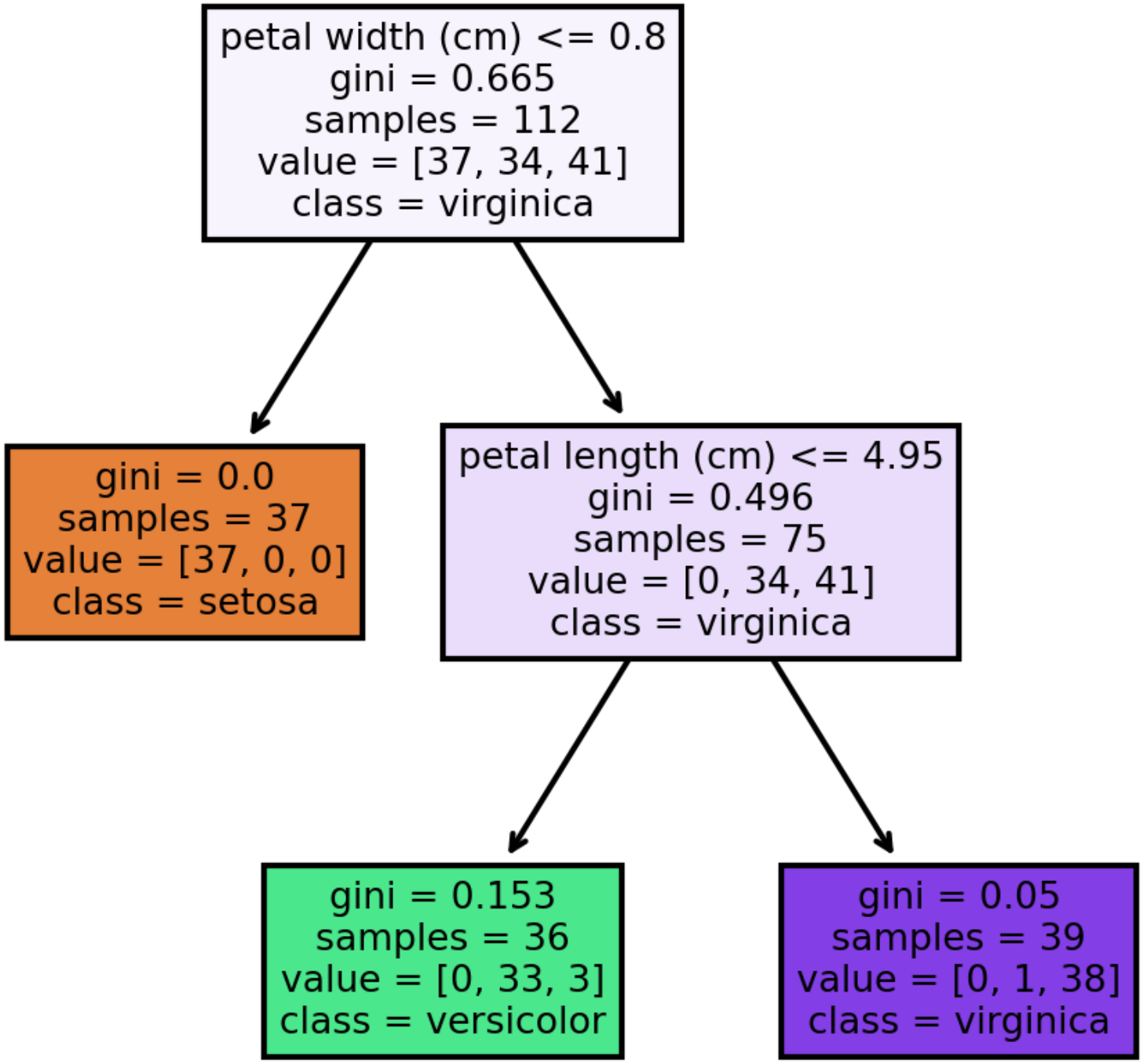
In [91]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi = 300)

tree.plot_tree(clf);
```





In [92]:
```
# Putting the feature names and class names into variables
fn = ['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']
cn = ['setosa', 'versicolor', 'virginica']
```

In [93]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi = 300)

tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('images/plottreefncn.png')
```


petal width (cm) <= 0.8
gini = 0.665
samples = 112
value = [37, 34, 41]
class = virginica
gini = 0.0
samples = 37
value = [37, 0, 0]
class = setosa
petal length (cm) <= 4.95
gini = 0.496
samples = 75
value = [0, 34, 41]
class = virginica
gini = 0.153
samples = 36
value = [0, 33, 3]
class = versicolor
gini = 0.05
samples = 39
value = [0, 1, 38]
class = virginica

Bagged Trees

In [94]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

# Bagged Trees Regressor
from sklearn.ensemble import BaggingRegressor
```

In [95]:
```
df = pd.read_csv('data/kc_house_data.csv')

df.head()
```

Out[95]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

In [96]:
```
# This notebook only selects a couple features for simplicity
# However, I encourage you to play with adding and substracting more features
```

```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors']

X = df.loc[:, features]

y = df.loc[:, 'price'].values
```

In [97]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [98]:
```
reg = BaggingRegressor(n_estimators=100,
                       random_state = 0)
```

In [99]:
```
reg.fit(X_train, y_train)
```

Out[99]:
```
BaggingRegressor(n_estimators=100, random_state=0)
```

In [100…
```
# Returns a NumPy Array
# Predict for One Observation
reg.predict(X_test.iloc[0].values.reshape(1, -1))
```

Out[100…
```
array([353334.6])
```

In [101…
```
reg.predict(X_test[0:10])
```

Out[101…
```
array([ 353334.6 , 1011004.77,  450212.76,  418593.  ,  772871.7 ,
        405436.5 ,  361353.02,  720323.9 ,  580438.82, 1623570.8 ])
```

In [102…
```
score = reg.score(X_test, y_test)
print(score)
```

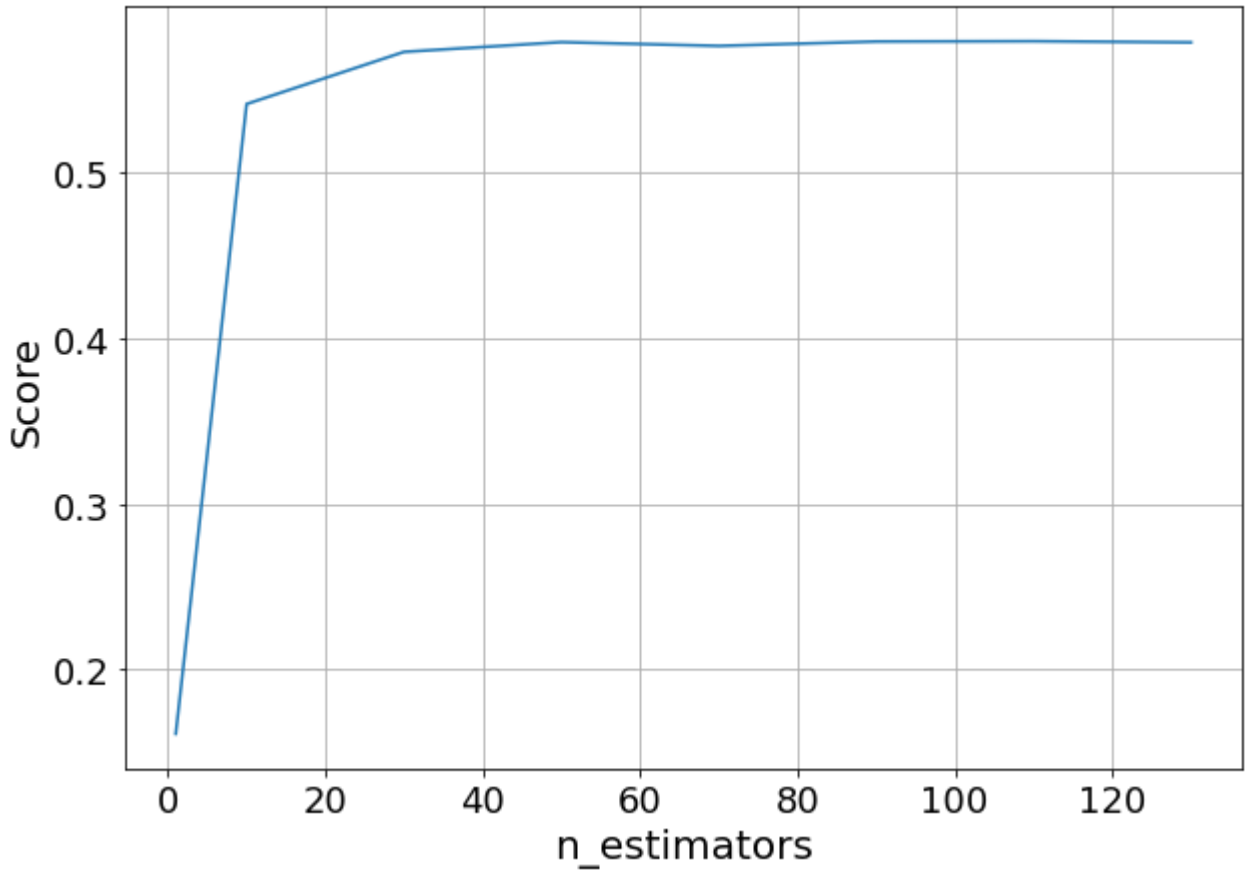
```
0.5786196798753096
```

In [103…
```
# List of values to try for n_estimators:
estimator_range = [1] + list(range(10, 150, 20))

scores = []

for estimator in estimator_range:
```

```
    reg = BaggingRegressor(n_estimators=estimator, random_state=0)
    reg.fit(X_train, y_train)
    scores.append(reg.score(X_test, y_test))
```

In [104…

```
plt.figure(figsize = (10,7))
plt.plot(estimator_range, scores);

plt.xlabel('n_estimators', fontsize =20);
plt.ylabel('Score', fontsize = 20);
plt.tick_params(labelsize = 18)
plt.grid()
```



In [105…

```
### 2_10_Random_Forest
```

In [106…
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Regression Models
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor

# Classifer Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier

# To visualize individual decision trees
from sklearn import tree
from sklearn.tree import export_text
```

In [107…
```
df = pd.read_csv('data/kc_house_data.csv')

df.head()
```

Out[107…

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	…	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	…	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	…	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	…	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	…	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	…	8	1680	

5 rows × 21 columns

In [108…
```
# This notebook only selects a couple features for simplicity
# However, I encourage you to play with adding and substracting features
```

```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors']

X = df.loc[:, features]

y = df.loc[:, 'price'].values
```

In [109…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [110…
```
reg = RandomForestRegressor(n_estimators=100, random_state = 0)
```

In [111…
```
reg.fit(X_train, y_train)
```

Out[111…
```
RandomForestRegressor(random_state=0)
```

In [112…
```
reg.predict(X_test[0:10])
```

Out[112…
```
array([ 354008.56,  999809.  ,  443760.25,  426332.  ,  760570.2 ,
        408775.5 ,  360030.14,  714794.4 ,  585902.14, 1665779.  ])
```

In [113…
```
score = reg.score(X_test, y_test)
print(score)
```

```
0.577684658845681
```

In [114…
```
# Load the Iris Dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [115…
```
# Fir Bagged Tree Model
btc = BaggingClassifier(n_estimators=100,
                        random_state = 1)

btc.fit(X_train, y_train)
```

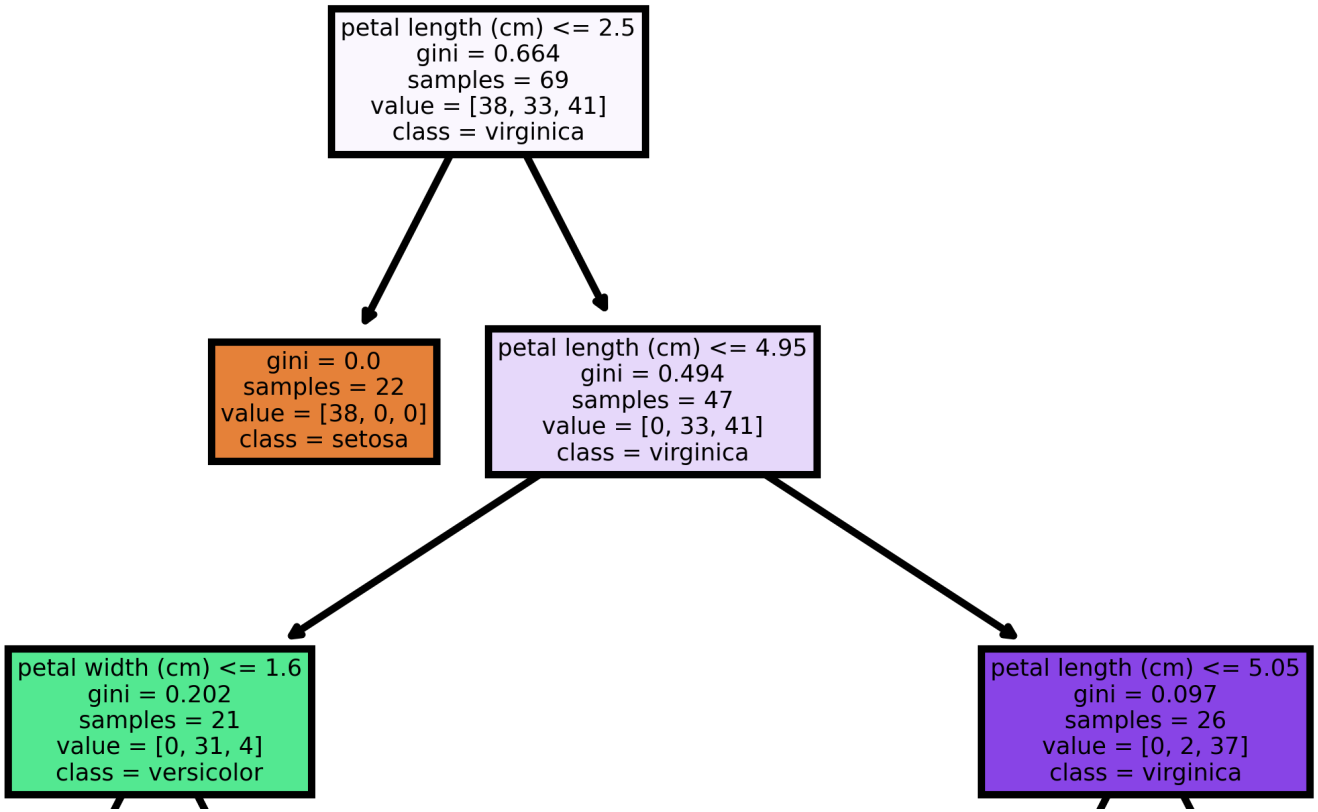
```
# Fit Random Forest Model
rfc = RandomForestClassifier(n_estimators=100, random_state = 1)

rfc.fit(X_train, y_train)
```

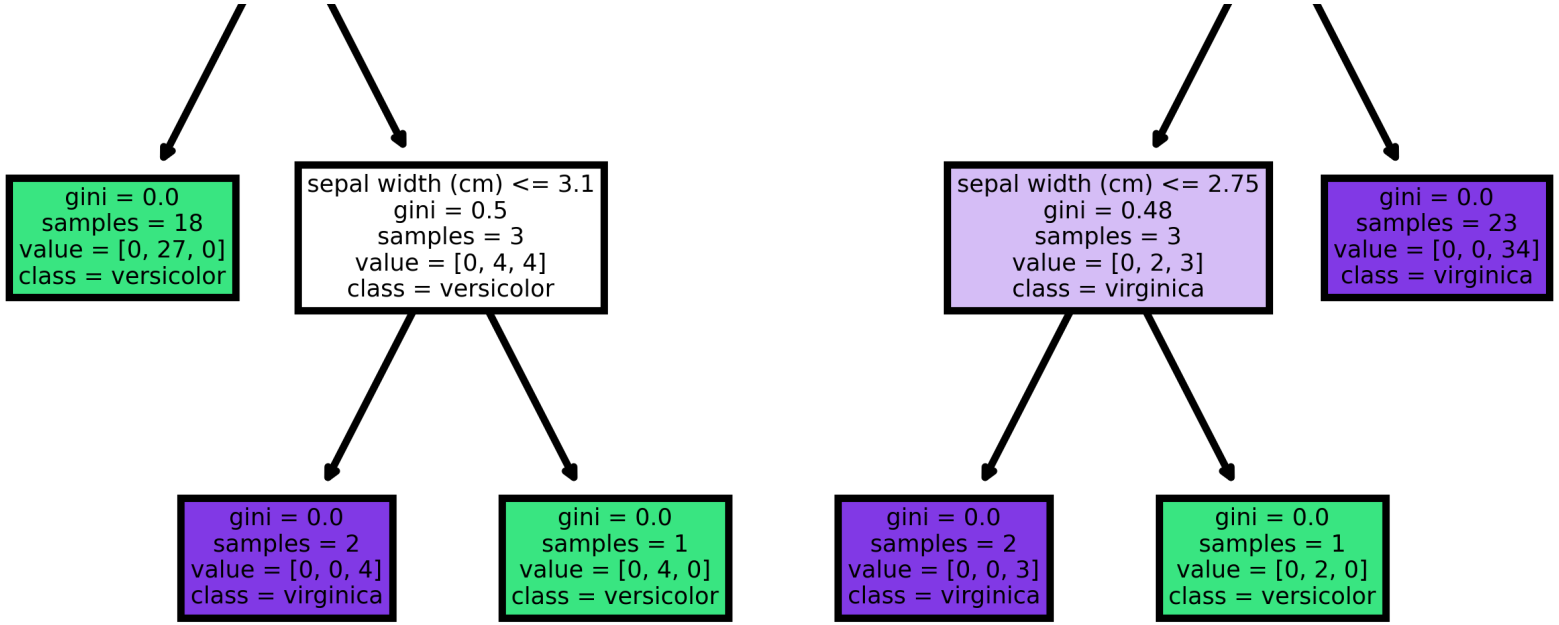
Out[115…

```
RandomForestClassifier(random_state=1)
```

In [116…

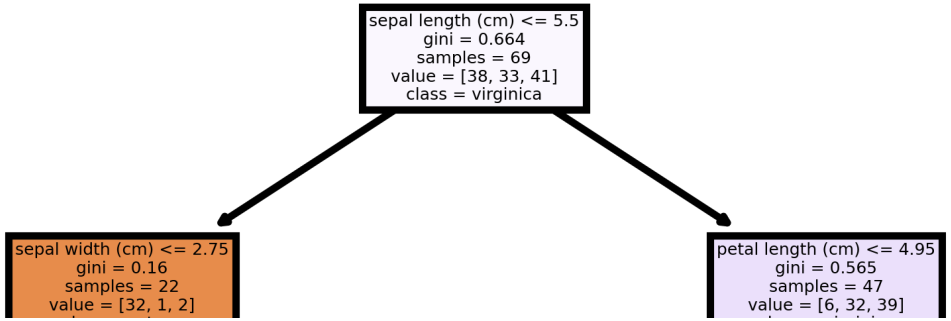
```
# Get the first decision tree for bagged tree model
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(btc.estimators_[0],
               feature_names = data.feature_names,
               class_names=data.target_names,
               filled = True);
```

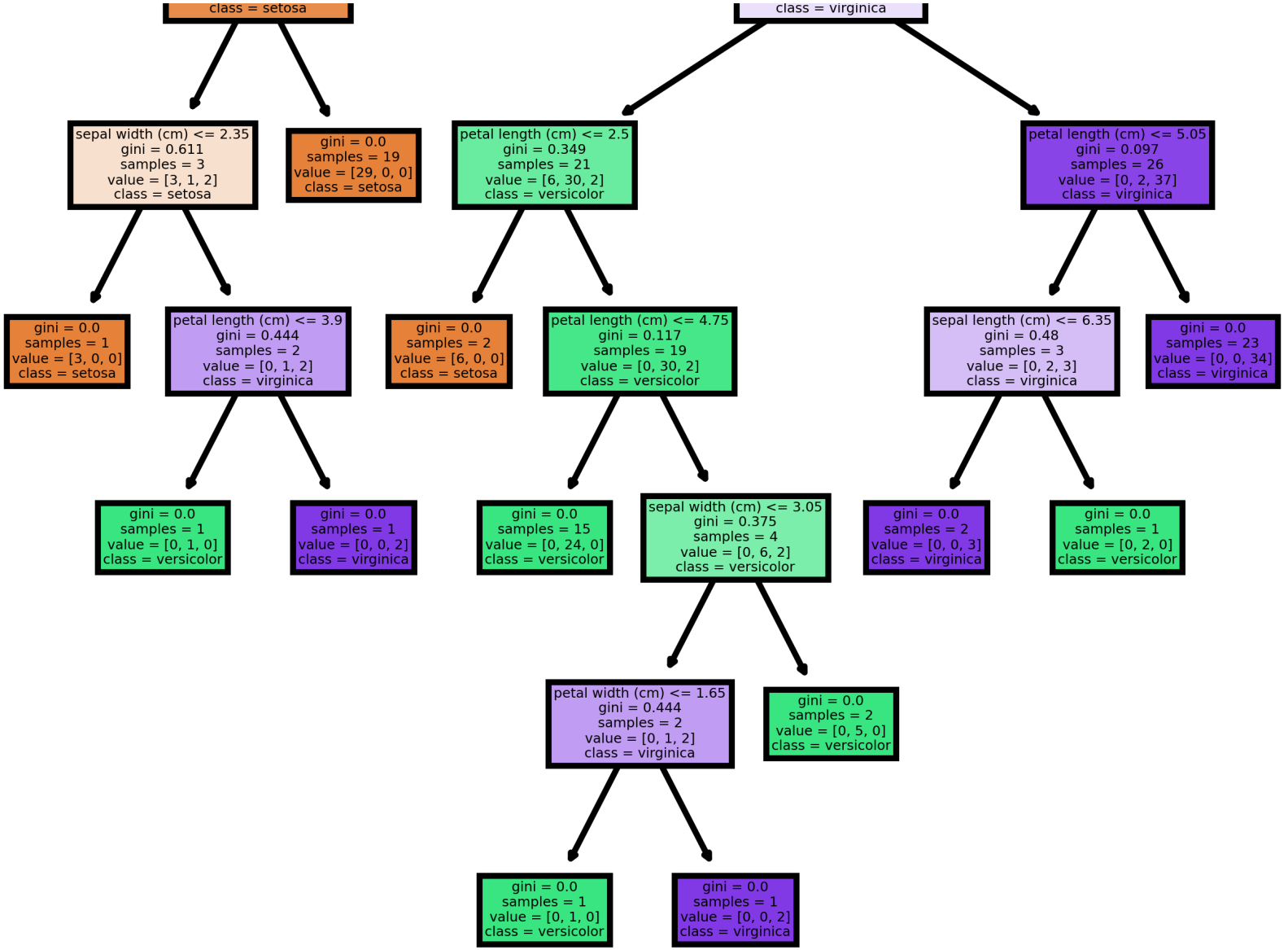




In [117…

```
# Get the first decision tree for a random forest model
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rfc.estimators_[0],
               feature_names = data.feature_names,
               class_names=data.target_names,
               filled = True);
```





In [118…

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(rfc.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False)
```

In [119…
```
importances
```

Out[119…

	feature	importance
2	petal length (cm)	0.453
3	petal width (cm)	0.385
0	sepal length (cm)	0.139
1	sepal width (cm)	0.023

Chapter 3 KMeans

In [120…
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans
```

In [121…
```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df.head()
```

Out[121…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

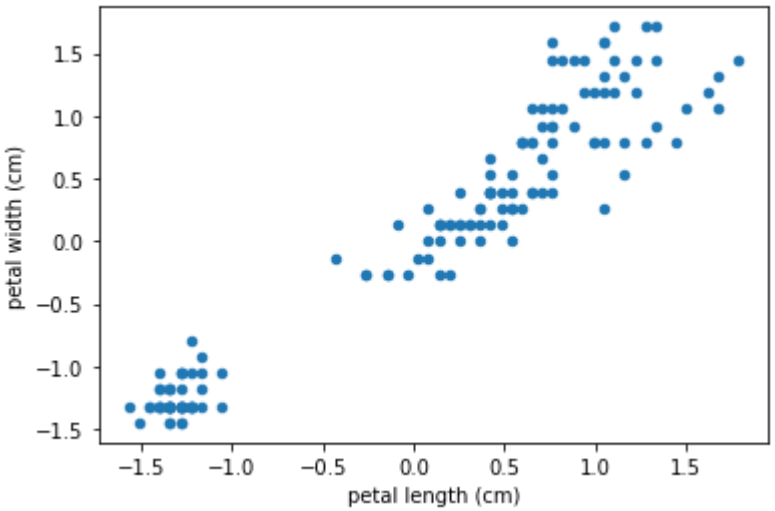
In [122…
```
features = ['petal length (cm)','petal width (cm)']

# Create features matrix
x = df.loc[:, features].values
```

In [123…
```
y = data.target
```

In [124…
```
# Apply Standardization to features matrix X
x = df.loc[:, features].values
```

In [125…
```
x = StandardScaler().fit_transform(x)
```

In [126…
```
# Plot
pd.DataFrame(x, columns = features).plot.scatter('petal length (cm)','petal width (cm)' )

# Add labels
plt.xlabel('petal length (cm)');
plt.ylabel('petal width (cm)');
```



In [127…
```
# Make an instance of KMeans with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=1)
```

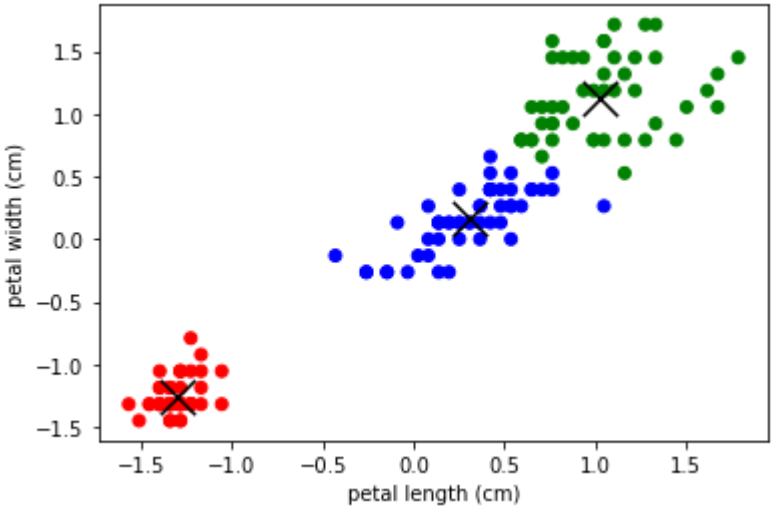
```
# Fit only on a features matrix
kmeans.fit(x)
```

Out[127…
```
KMeans(n_clusters=3, random_state=1)
```

In [128…
```
# Get labels and cluster centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

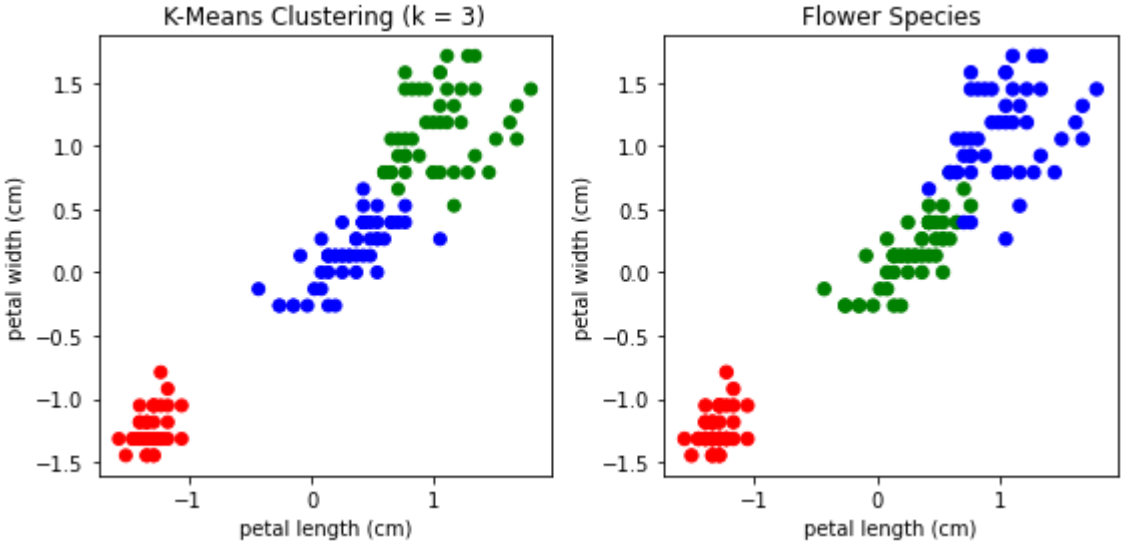
In [129…
```
x = pd.DataFrame(x, columns = features)
```

In [130…
```
colormap = np.array(['r', 'g', 'b'])
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[labels])
plt.scatter(centroids[:,0], centroids[:,1], s = 300, marker = 'x', c = 'k')

plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
```



In [131…
```
plt.figure(figsize=(8,4))

plt.subplot(1, 2, 1)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[labels])
plt.xlabel('petal length (cm)')
```

```
plt.ylabel('petal width (cm)');
plt.title('K-Means Clustering (k = 3)')

plt.subplot(1, 2, 2)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[y], s=40)
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
plt.title('Flower Species')

plt.tight_layout()
```



3_3 PCA

In [132…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
```

In [133…

```
data = load_iris()
```

```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

In [134…

```
speciesDict = {0: 'setosa', 1:'versicolor', 2:'virginica'}

df.loc[:,'target'] = df.loc[:, 'target'].apply(lambda x: speciesDict[x])
```

In [135…

```
df.head()
```

Out[135…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [136…

```
# Apply Standardization to features matrix X
features = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)','petal width (cm)']
x = df.loc[:, features].values
y = df.loc[:,['target']].values
```

In [137…

```
x = StandardScaler().fit_transform(x)
```

In [138…

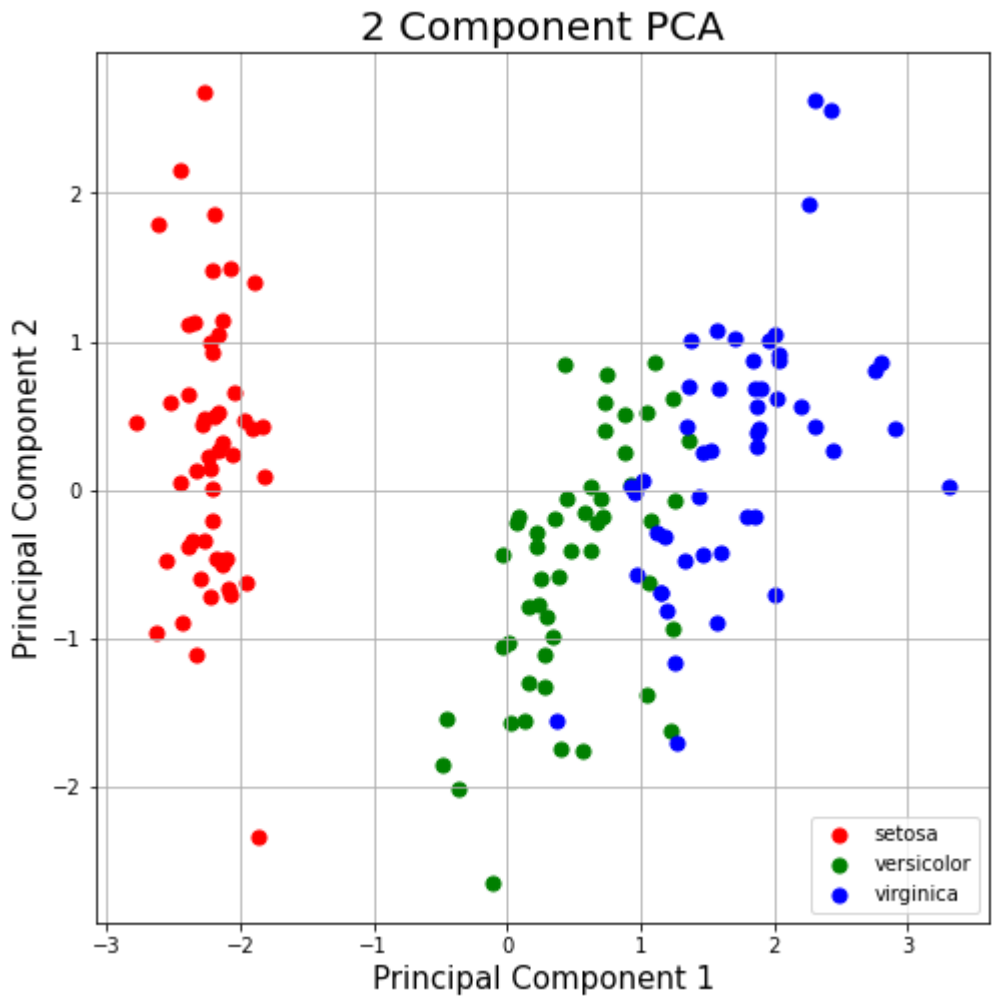
```
# Make an instance of PCA
pca = PCA(n_components=2)

# Fit and transform the data
principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
```

In [139…

```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

In [140…

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (8,8));
targets = df.loc[:, 'target'].unique()
colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)

ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)
ax.legend(targets)
ax.grid()
```



In [141…
```
pca.explained_variance_ratio_
```

Out[141…
```
array([0.72962445, 0.22850762])
```

In [142…
```
sum(pca.explained_variance_ratio_)
```

Out[142…
```
0.9581320720000164
```

#3_04

In [143…
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [144…
```
df = pd.read_csv('data/MNISTonly0_1.csv')
```

In [145…
```
df.head()
```

Out[145…

	0	1	2	3	4	5	6	7	8	9	…	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	1

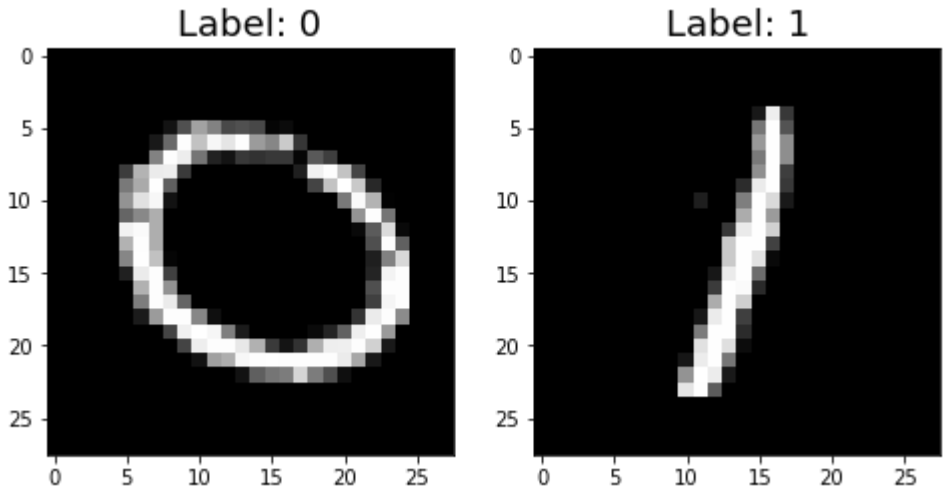
5 rows × 785 columns

In [146…
```
pixel_colnames = df.columns[:-1]
```

In [147…
```
# Get all columns except the label column for the first image
image_values = df.loc[0, pixel_colnames].values
```

In [148…
```
plt.figure(figsize=(8,4))
for index in range(0, 2):
```

```
    plt.subplot(1, 2, 1 + index )
    image_values = df.loc[index, pixel_colnames].values
    image_label = df.loc[index, 'label']
    plt.imshow(image_values.reshape(28,28), cmap ='gray')
    plt.title('Label: ' + str(image_label), fontsize = 18)
```



In [149…

```
X_train, X_test, y_train, y_test = train_test_split(df[pixel_colnames], df['label'], random_state=0)
```

In [150…

```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [151…

```
# Variable created for demonstational purposes in the notebook
scaledTrainImages = X_train.copy()
```

In [152…

```
"""
n_components = .90 means that scikit-learn will choose the minimum number
of principal components such that 90% of the variance is retained.
"""
```

```
pca = PCA(n_components = .90)

# Fit PCA on training set only
pca.fit(X_train)

# Apply the mapping (transform) to both the training set and the test set.
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Logistic Regression
clf = LogisticRegression()
clf.fit(X_train, y_train)

print('Number of dimensions before PCA: ' + str(len(pixel_colnames)))
print('Number of dimensions after PCA: ' + str(pca.n_components_))
print('Classification accuracy: ' + str(clf.score(X_test, y_test)))
```

```
Number of dimensions before PCA: 784
Number of dimensions after PCA: 104
Classification accuracy: 0.997
```

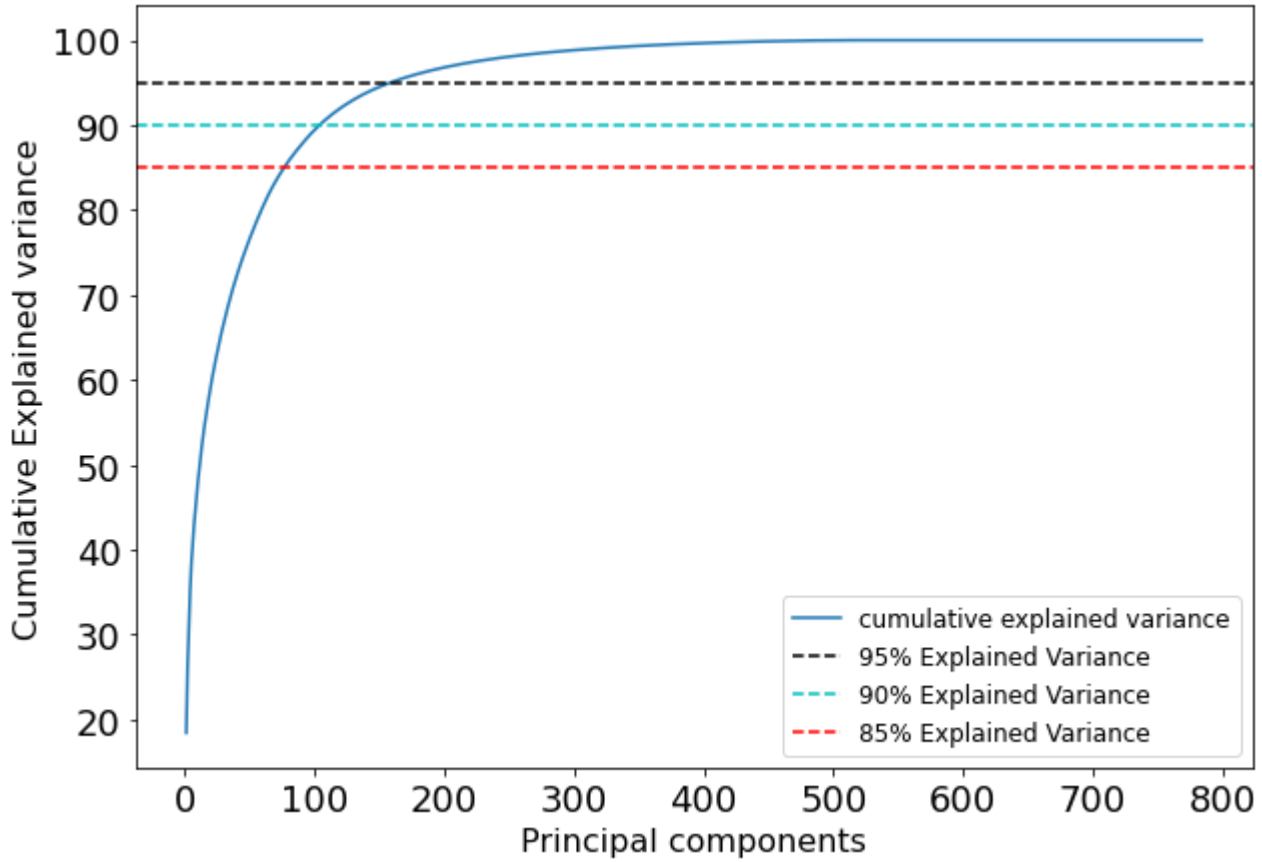
In [153…

```
# if n_components is not set, all components are kept (784 in this case)
pca = PCA()

pca.fit(scaledTrainImages)

# Summing explained variance
tot = sum(pca.explained_variance_)

var_exp = [(i/tot)*100 for i in sorted(pca.explained_variance_, reverse=True)]

# Cumulative explained variance
cum_var_exp = np.cumsum(var_exp)

# PLOT OUT THE EXPLAINED VARIANCES SUPERIMPOSED
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));
ax.tick_params(labelsize = 18)
ax.plot(range(1, 785), cum_var_exp, label='cumulative explained variance')
ax.set_ylabel('Cumulative Explained variance', fontsize = 16)
ax.set_xlabel('Principal components', fontsize = 16)
ax.axhline(y = 95, color='k', linestyle='--', label = '95% Explained Variance')
ax.axhline(y = 90, color='c', linestyle='--', label = '90% Explained Variance')
```

```
ax.axhline(y = 85, color='r', linestyle='--', label = '85% Explained Variance')
ax.legend(loc='best', markerscale = 1.0, fontsize = 12)
```

Out[153…

```
<matplotlib.legend.Legend at 0x1a0693ba4f0>
```



Chapter 3_05 pipelines

In [154…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.pipeline import Pipeline
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [155…

```
df = pd.read_csv('data/MNISTonly0_1.csv')
```

In [156…

```
df.head()
```

Out[156…

	0	1	2	3	4	5	6	7	8	9	…	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	1

5 rows × 785 columns

In [157…

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(df[df.columns[:-1]], df['label'], random_state=0)

# Standardize Data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components = .90, random_state=0)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Apply Logistic Regression
clf = LogisticRegression()
```

```
clf.fit(X_train, y_train)

# Get Model Performance
print(clf.score(X_test, y_test))
```

```
0.997
```

In [158…

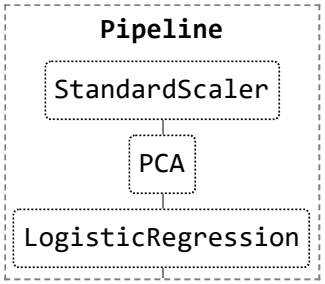
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(df[df.columns[:-1]], df['label'], random_state=0)

# Create a pipeline
pipe = Pipeline([('scaler', StandardScaler()),
                 ('pca', PCA(n_components = .90, random_state=0)),
                 ('logistic', LogisticRegression())])

pipe.fit(X_train, y_train)

# Get Model Performance
print(pipe.score(X_test, y_test))
```

```
0.997
```

In [159…

```
from sklearn import set_config

set_config(display='diagram')
pipe
```

Out[159…



In [ ]: