

▼ Project 3: Machine Learning

STARTER CODE

Goals

- To learn how to implement a Data Science / Machine Learning workflow in Python (using Pandas, Scikit-learn, Matplotlib, and Numpy)
- To learn how to use perform linear regression by least squares using Python and scikit-learn.
- To appreciate that the same linear regression coefficients may be the best fit for dramatically different data distributions -- as illustrated by the Anscombe's quartet.
- To practice with different types of regularization (*lasso* and *ridge*) and understand when to use them.
- To learn how to implement several different machine learning classification models in Python
- To learn how to evaluate and fine-tune the performance of a model using cross-validation
- To learn how to test a model and produce a set of plots and performance measures
- To expand upon the prior experience of manipulating, summarizing, and visualizing representative datasets in data science and machine learning

Instructions

- This assignment is structured in 3 parts, each using their own dataset(s).
- As usual, there will be some Python code to be written and questions to be answered.
- At the end, you should export your notebook to PDF format; it will "automagically" become your report.
- Submit the report (PDF), notebook (.ipynb file), and the link to the "live" version of your solution on Google Colaboratory via Canvas.
- **The number of points is indicated next to each part. They add up to 100.**
- **There are additional (10 points worth of) bonus items**, which are, of course optional.

Important

- For the sake of reproducibility, use `random_state=0` (or equivalent) in all functions that use random number generation.
- It is OK to attempt the bonus points, but please **do not overdo it!**

▼ Imports + Google Drive

```
! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
  Using cached https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
Requirement already satisfied: joblib~=1.1.0 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0) (
Requirement already satisfied: pandas!=1.0.0,!1.0.1,!1.0.2,!1.1.0,>=0.25.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: matplotlib>=3.2.0 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2
Requirement already satisfied: pydantic>=1.8.1 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0
Requirement already satisfied: PyYAML>=5.0.0 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0)
Requirement already satisfied: jinja2>=2.11.1 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0)
Requirement already satisfied: visions[type_image_path]==0.7.5 in /usr/local/lib/python3.7/dist-packages (from pandas-
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0)
Requirement already satisfied: htmlmin>=0.1.12 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0)
Requirement already satisfied: missingno>=0.4.2 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.
Requirement already satisfied: phik>=0.11.1 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0) (
Requirement already satisfied: tangled-up-in-unicode==0.2.0 in /usr/local/lib/python3.7/dist-packages (from pandas-pro
Requirement already satisfied: requests>=2.24.0 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.
Requirement already satisfied: tqdm>=4.48.2 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0) (
Requirement already satisfied: seaborn>=0.10.1 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.0
Requirement already satisfied: multimethod>=1.4 in /usr/local/lib/python3.7/dist-packages (from pandas-profiling==3.2.
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.7/dist-packages (from visions[type_image_path]=
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.7/dist-packages (from visions[type_image_path]=
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from visions[type_image_path]==0.7.5-
Requirement already satisfied: imagehash in /usr/local/lib/python3.7/dist-packages (from visions[type_image_path]==0.7
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.11.1->pandas
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.0-
```

Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.0->pa
Requirement already satisfied: cyclor>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.0->pandas-
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->ma
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas!=1.0.0,!=1.0.1,!=1.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotl
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.24.0->pa
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.24
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.24.0->pandas-p
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.24.0-
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.7/dist-packages (from imagehash->visions[type_imag



```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import seaborn as sns; sns.set()
import scipy.stats as ss
from pandas_profiling import ProfileReport

from tensorflow import keras
from tensorflow.keras import layers
```

▼ Part 1: Regression Analysis

▼ 1a. Linear regression by least squares

In this part, we will look at the correlation between female literacy and fertility (defined as the average number of children born per woman) throughout the world. For ease of analysis and interpretation, we will work with the *illiteracy* rate.

The Python code below plots the fertility versus illiteracy and computes the Pearson correlation coefficient. The Numpy array `illiteracy` has the illiteracy rate among females for most of the world's nations. The array `fertility` has the corresponding fertility data.

```
df = pd.read_csv('https://github.com/ogemarques/data-files/raw/main/female_literacy_fertility.csv')
illiteracy = 100 - df['female literacy']
fertility = df['fertility']

def pearson_r(x, y):
    """Compute Pearson correlation coefficient between two arrays."""
    # Compute correlation matrix: corr_mat
    corr_mat = np.corrcoef(x, y)

    # Return entry [0,1]
    return corr_mat[0,1]

# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility, marker='.', linestyle='none')

# Set the margins and label axes
plt.margins(0.02)
_ = plt.xlabel('% illiterate')
_ = plt.ylabel('fertility')

# Show the plot
plt.show()

# Show the Pearson correlation coefficient
print('Pearson correlation coefficient between illiteracy and fertility: {:.5f}'.format(pearson_r(illiteracy, fertility)))
```

1.1 Your turn! (5 points)

We will assume that fertility is a linear function of the female illiteracy rate: $f=ai+b$, where a is the slope and b is the intercept.

We can think of the intercept as the minimal fertility rate, probably somewhere between one and two.

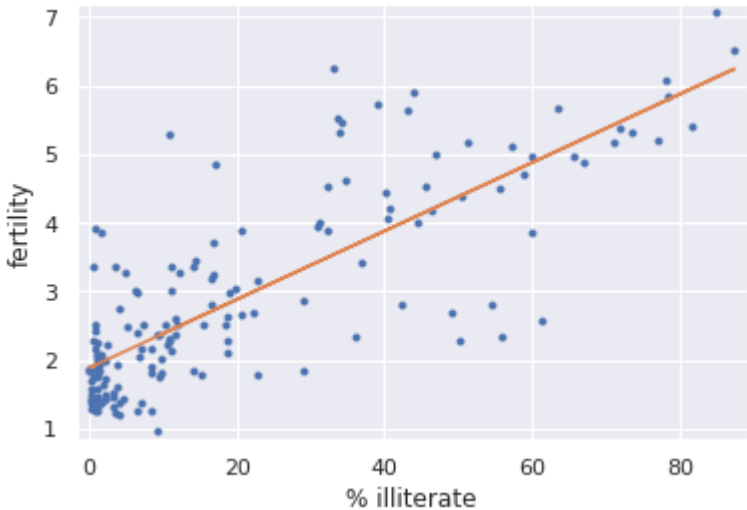
The slope tells us how the fertility rate varies with illiteracy. We can find the best fit line .

Write code to plot the data and the best fit line (using `np.polyfit()`) and print out the slope and intercept.

```
best_fit_line = np.polyfit(illiteracy, fertility, 1)

_ = plt.plot(illiteracy, fertility, '.')
line = np.poly1d(best_fit_line)
_ = plt.plot(illiteracy, line(illiteracy))
plt.margins(0.02)
_ = plt.xlabel('% illiterate')
_ = plt.ylabel('fertility')
plt.show()

print("")
print("Slope: " + str(best_fit_line[0]) + " | Intercept: " + str(best_fit_line[1]))
print("")
print("Best Fit Line Equation: " + str(line))
```



Slope: 0.04979854809063423 | Intercept: 1.8880506106365567

Best Fit Line Equation:
0.0498 x + 1.888

1b. Anscombe's quartet

The Anscombe's quartet is a collection of four small data sets that have nearly identical simple descriptive statistics, yet have very different distributions. Each dataset consists of 11 (x,y) points. The quartet was created in 1973 by the statistician Francis Anscombe to demonstrate: the importance of visualization and exploratory data analysis (EDA), the effect of outliers and other influential observations on statistical properties, and the limitations of summary statistics (*).

(*) See <https://heap.io/blog/data-stories/anscombes-quartet-and-why-summary-statistics-dont-tell-the-whole-story> if you're interested.

The Python code below performs a linear regression on the data set from Anscombe's quartet that is most reasonably interpreted with linear regression.

```
x1 = [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]
y1 = [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]

x2 = [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]
y2 = [9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74]

x3 = [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]
y3 = [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73]

x4 = [8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 19.0, 8.0, 8.0, 8.0]
y4 = [6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89]

# Perform linear regression: a, b
a, b = np.polyfit(x1, y1, 1)
```

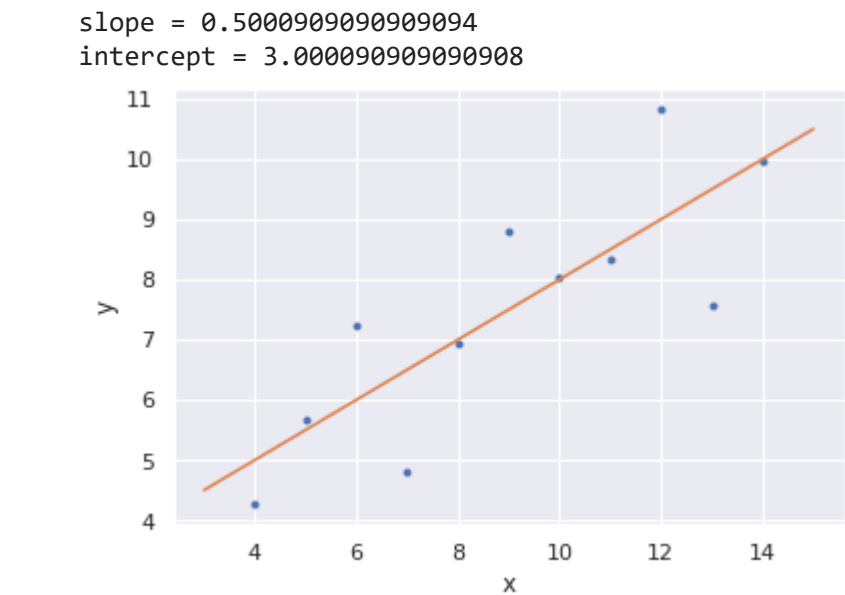
```
# Print the slope and intercept
print('slope =', a)
print('intercept =', b)

# Generate theoretical x and y data: x_theor, y_theor
x_theor = np.array([3, 15])
y_theor = a * x_theor + b

# Plot the Anscombe data and theoretical line
_ = plt.plot(x1, y1, marker='.', linestyle='none')
_ = plt.plot(x_theor, y_theor)

# Label the axes
plt.xlabel('x')
plt.ylabel('y')

# Show the plot
plt.show()
```



▼ 1.2 Your turn! (5 points)

Linear regression on all Anscombe data

Write code to verify that all four of the Anscombe data sets have the same slope and intercept from a linear regression, i.e. compute the slope and intercept for each set.

The data are stored in lists (`anscombe_x = [x1, x2, x3, x4]` and `anscombe_y = [y1, y2, y3, y4]`), corresponding to the x and y values for each Anscombe data set.

```
anscombe_x = [x1, x2, x3, x4]
anscombe_y = [y1, y2, y3, y4]

best_fit_line_an1 = np.polyfit(anscombe_x[0], anscombe_y[0], 1)

_ = plt.plot(anscombe_x[0], anscombe_y[0], '.')
line_an1 = np.poly1d(best_fit_line_an1)
_ = plt.plot(anscombe_x[0], line_an1(anscombe_x[0]))
plt.margins(0.02)
_ = plt.xlabel('x')
_ = plt.ylabel('y')
plt.show()

print("")
print("Slope: " + str(best_fit_line_an1[0]) + " | Intercept: " + str(best_fit_line_an1[1]))
print("")
print("Best Fit Line Equation: " + str(line_an1))
print("")
print("-----")
print("")

best_fit_line_an2 = np.polyfit(anscombe_x[1], anscombe_y[1], 1)

_ = plt.plot(anscombe_x[1], anscombe_y[1], '.')
line_an2 = np.poly1d(best_fit_line_an2)
_ = plt.plot(anscombe_x[1], line_an2(anscombe_x[1]))
plt.margins(0.02)
_ = plt.xlabel('x')
_ = plt.ylabel('y')
plt.show()

print("")
```

```
print("Slope: " + str(best_fit_line_an2[0]) + " | Intercept: " + str(best_fit_line_an2[1]))
print("")
print("Best Fit Line Equation: " + str(line_an2))
print("")
print("-----")
print("")

best_fit_line_an3 = np.polyfit(anscombe_x[2], anscombe_y[2], 1)

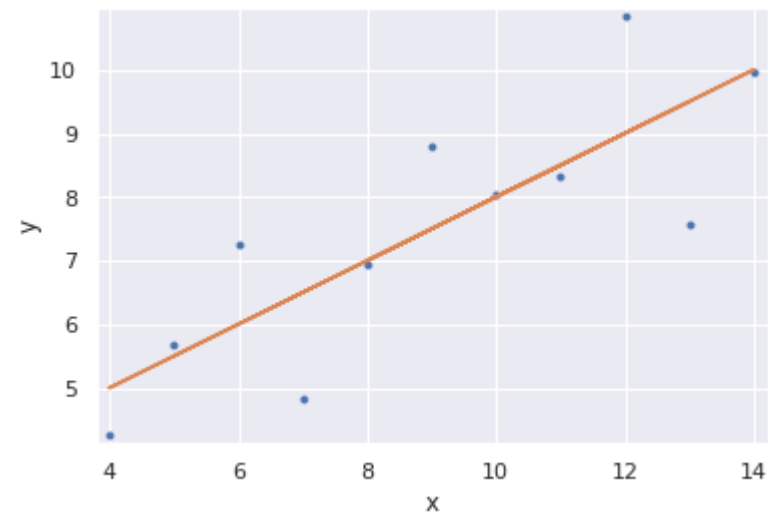
_ = plt.plot(anscombe_x[2], anscombe_y[2], '.')
line_an3 = np.poly1d(best_fit_line_an3)
_ = plt.plot(anscombe_x[2], line_an3(anscombe_x[2]))
plt.margins(0.02)
_ = plt.xlabel('x')
_ = plt.ylabel('y')
plt.show()

print("")
print("Slope: " + str(best_fit_line_an3[0]) + " | Intercept: " + str(best_fit_line_an3[1]))
print("")
print("Best Fit Line Equation: " + str(line_an3))
print("")
print("-----")
print("")

best_fit_line_an4 = np.polyfit(anscombe_x[3], anscombe_y[3], 1)

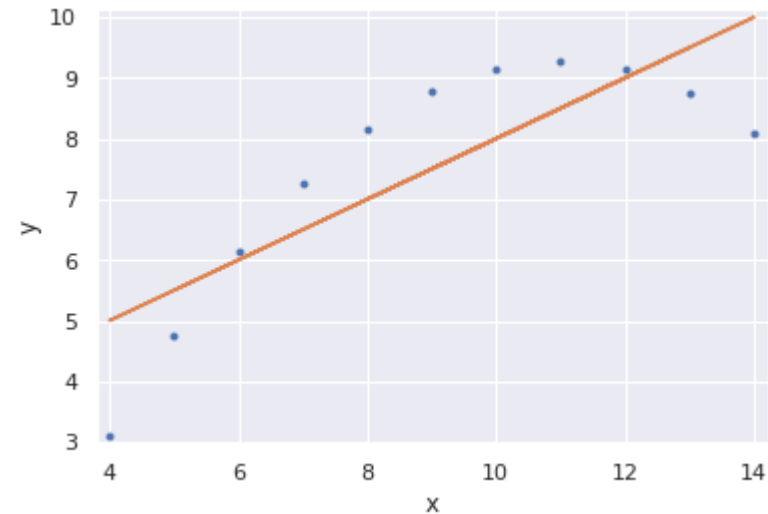
_ = plt.plot(anscombe_x[3], anscombe_y[3], '.')
line_an4 = np.poly1d(best_fit_line_an4)
_ = plt.plot(anscombe_x[3], line_an4(anscombe_x[3]))
plt.margins(0.02)
_ = plt.xlabel('x')
_ = plt.ylabel('y')
plt.show()

print("")
print("Slope: " + str(best_fit_line_an4[0]) + " | Intercept: " + str(best_fit_line_an4[1]))
print("")
print("Best Fit Line Equation: " + str(line_an4))
print("")
print("-----")
print("")
```



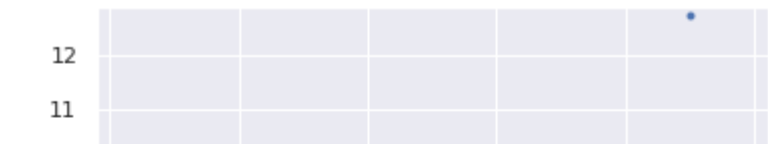
Slope: 0.5000909090909094 | Intercept: 3.000090909090908

Best Fit Line Equation:
0.5001 x + 3



Slope: 0.5000000000000003 | Intercept: 3.00090909090909

Best Fit Line Equation:
0.5 x + 3.001



1c. Regression using scikit-learn

Now that we know the basics of linear regression, we will switch to scikit-learn, a powerful, workflow-oriented library for data science and machine learning.

The Python code below shows a simple linear regression example using scikit-learn. Note the use of the `fit()` and `predict()` methods.

```
import matplotlib.pyplot as plt
import numpy as np

# Generate random data around the y = ax+b line where a=3 and b=-2
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 3 * x - 2 + rng.randn(50)

from sklearn.linear_model import LinearRegression

# Note: If you get a "ModuleNotFoundError: No module named 'sklearn'" error message, don't panic.
# It probably means you'll have to install the module by hand if you're using pip.
# If you're using conda, you should not see any error message.

model = LinearRegression(fit_intercept=True)

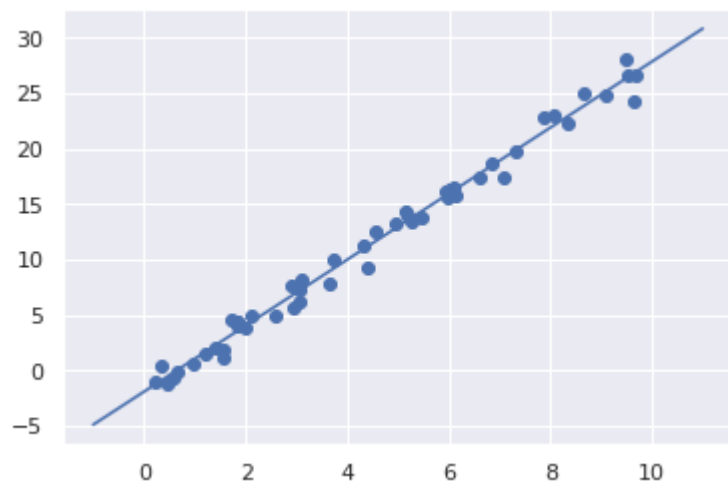
X = x[:, np.newaxis]
X.shape

model.fit(X, y)
print(model.coef_)
print(model.intercept_)
```

```
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

```
[2.9776566]
-1.903310725531119
```



▼ 1d. Polynomial regression

One way to adapt linear regression to nonlinear relationships between variables is to transform the data according to *basis functions*.

The idea is to take the multidimensional linear model:

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \dots$$

and build the x_1, x_2, x_3 , and so on, from our single-dimensional input x . That is, we let $x_n = f_n(x)$, where $f_n()$ is some function that transforms our data.

For example, if $f_n(x) = x^n$, our model becomes a polynomial regression:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots$$

Notice that this is *still a linear model*—the linearity refers to the fact that the coefficients a_n never multiply or divide each other. What we have effectively done is taken our one-dimensional x values and projected them into a higher dimension, so that a linear fit can fit more complicated relationships between x and y .

The code below shows a simple example of polynomial regression using the `PolynomialFeatures` transformer in scikit-learn.

Concretely, it shows how we can use polynomial features with a polynomial of degree seven, i.e.

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_7 x^7$$

It also introduces the notion of a *pipeline* in scikit-learn. "The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters." (<https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>)

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(7),
                           LinearRegression())
```

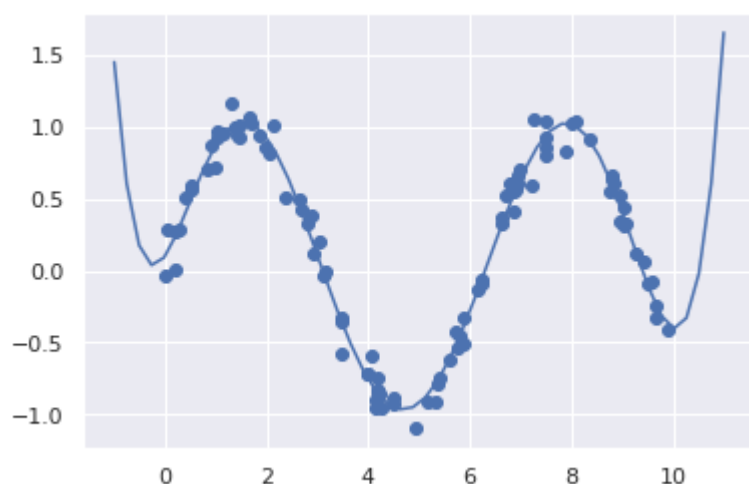
```
rng = np.random.RandomState(1)
x = 10 * rng.rand(100)
y = np.sin(x) + 0.1 * rng.randn(100)
```

```
poly_model.fit(x[:, np.newaxis], y)
yfit = poly_model.predict(xfit[:, np.newaxis])
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

```
print('The R^2 score for the fit is: ', poly_model.score(x[:, np.newaxis], y))
```

The R^2 score for the fit is: 0.9806993128749489



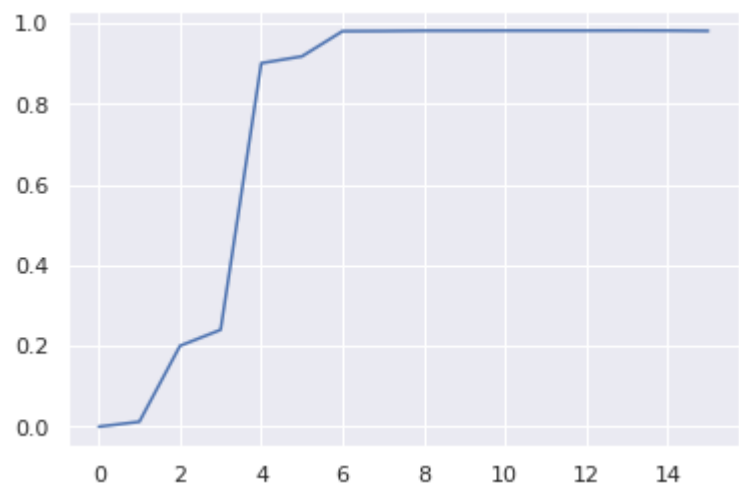
Our linear model, through the use of 7th-order polynomial basis functions, can provide an excellent fit to this non-linear data!

▼ 1.3 Your turn! (10 points)

Write code to find the best degree/order for the polynomial basis functions (between 1 and 15) by computing the quality of the fit using a suitable metric, in this case the R^2 coefficient (which can be computer using the `score()` function).

Remember that **the best possible score is 1.0**. The score can be negative (because the model can be arbitrarily worse). A score of 0 suggests a constant model that always predicts the expected value of y, disregarding the input features.

Hint: If you plot the score against the degree/order of the polynomial, you should see something like this:



```
best_score = -1
best_score_degree = 0
scores = []
degrees = []

for degree in range (15):
    model = make_pipeline(PolynomialFeatures(degree+1),
                          LinearRegression())
    model.fit(x[:, np.newaxis], y)

    plt.scatter(x, y)
    plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

    plt.xlim(0, 10)
    plt.ylim(-1.5, 1.5);

    score = model.score(x[:, np.newaxis], y)
    scores.append(score)
    degrees.append(degree+1)

    if score > best_score:
        best_score = score
        best_score_degree = degree + 1

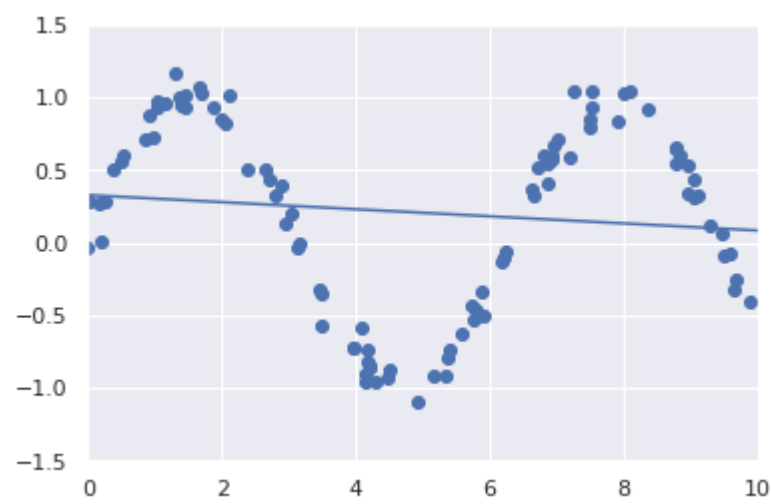
    print("")
    print("Degree: " + str(degree+1) + " Score: " + str(score))
    print("")
    plt.show()

print("")
print("-----")
print("")

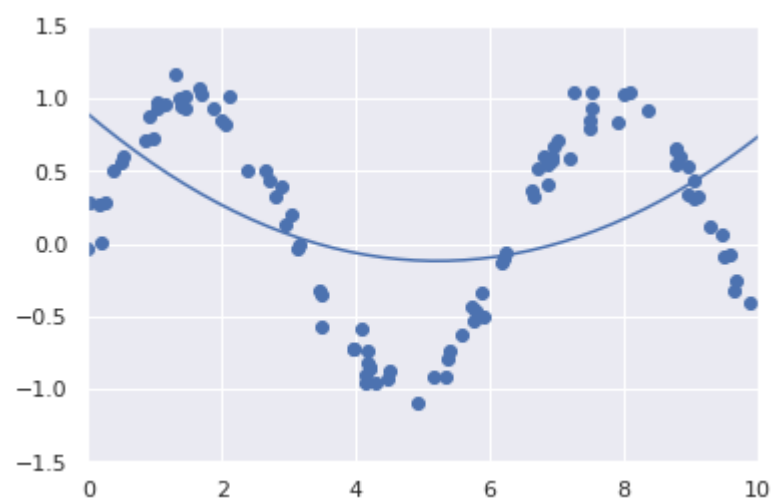
plt.plot(degrees, scores)
_ = plt.xlabel('Degree')
_ = plt.ylabel('Score')
plt.show()

print("")
print("The best degree is " + str(best_score_degree) + " with a score of " + str(best_score))
```

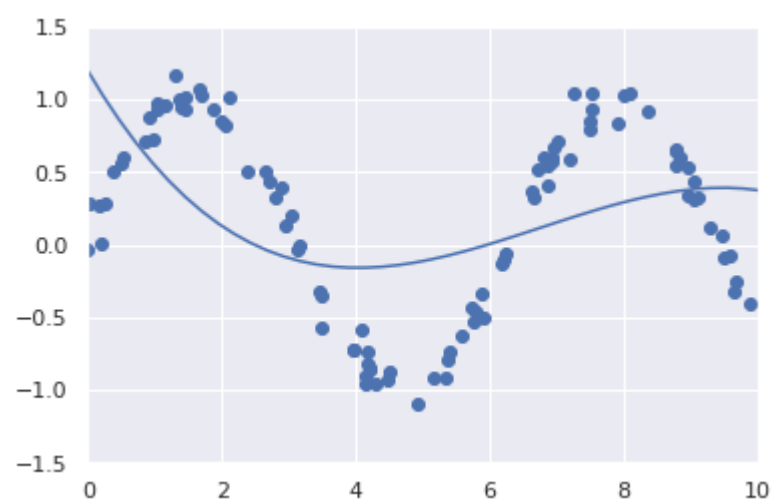

Degree: 1 Score: 0.012172236970290573



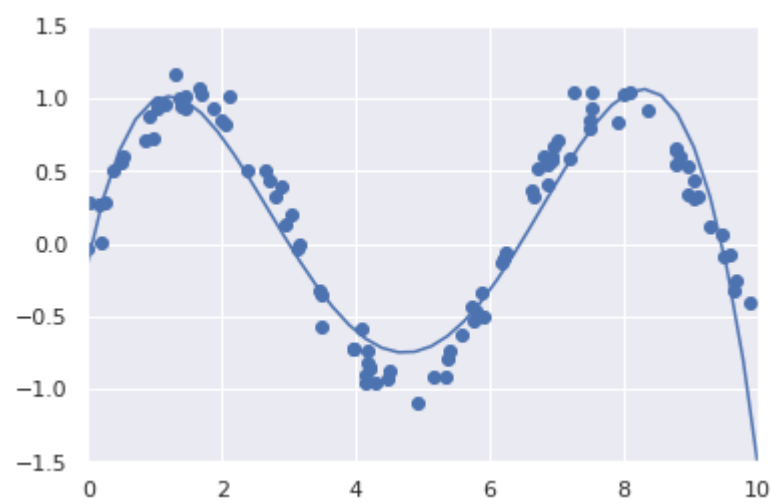
Degree: 2 Score: 0.20041267204540536



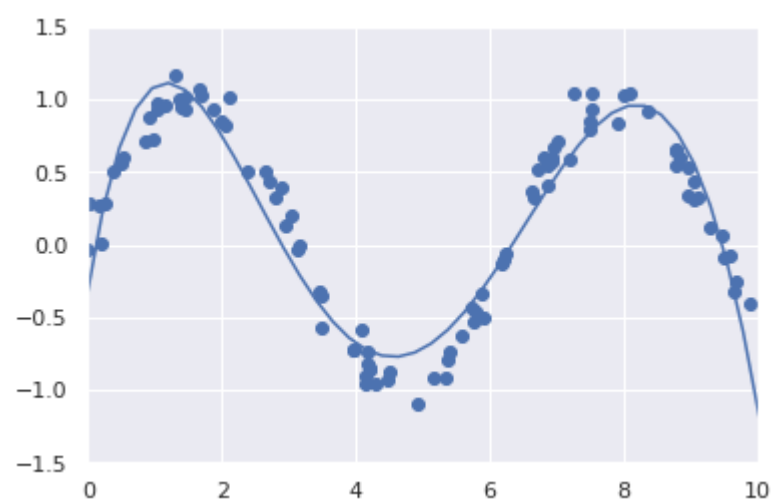
Degree: 3 Score: 0.24010264456793462



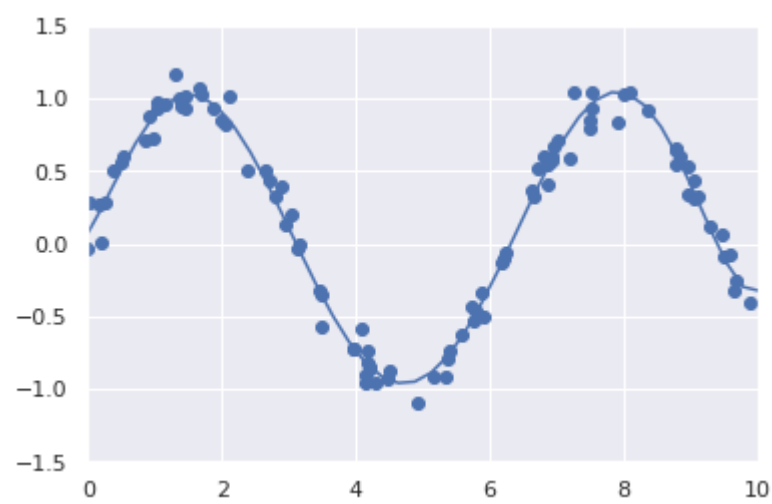
Degree: 4 Score: 0.9010949501596228



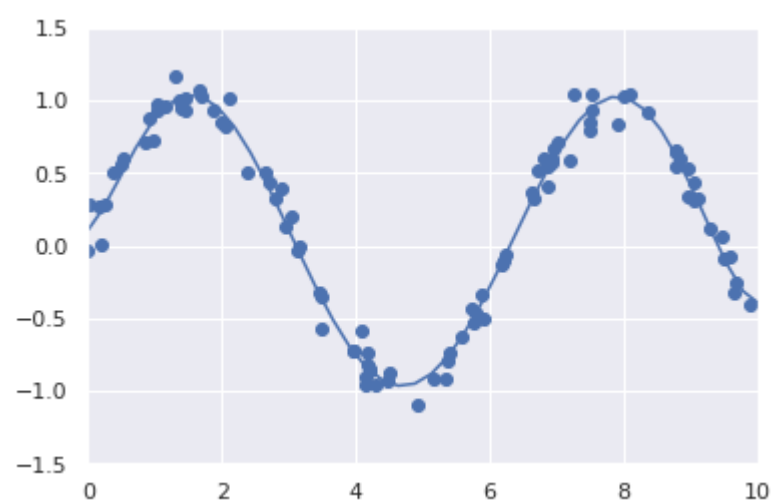
Degree: 5 Score: 0.9178981253300205



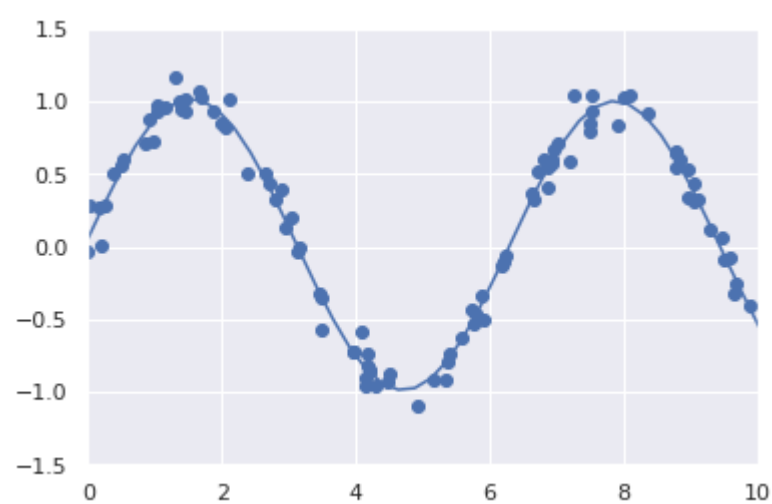
Degree: 6 Score: 0.9803546774621197



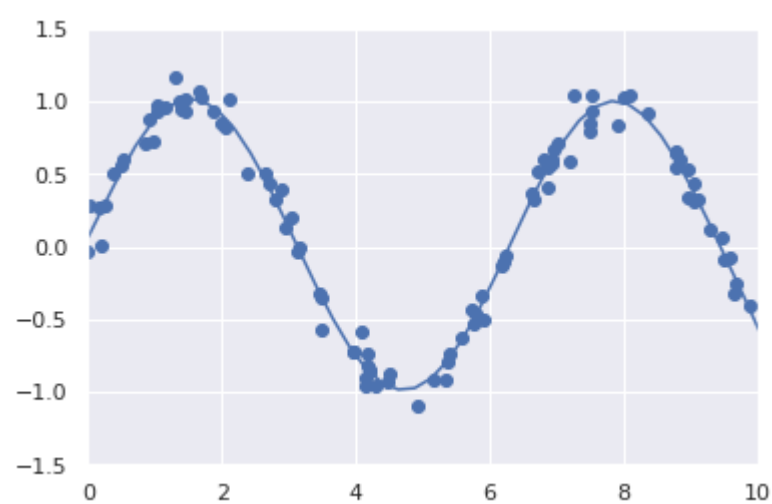
Degree: 7 Score: 0.9806993128749489



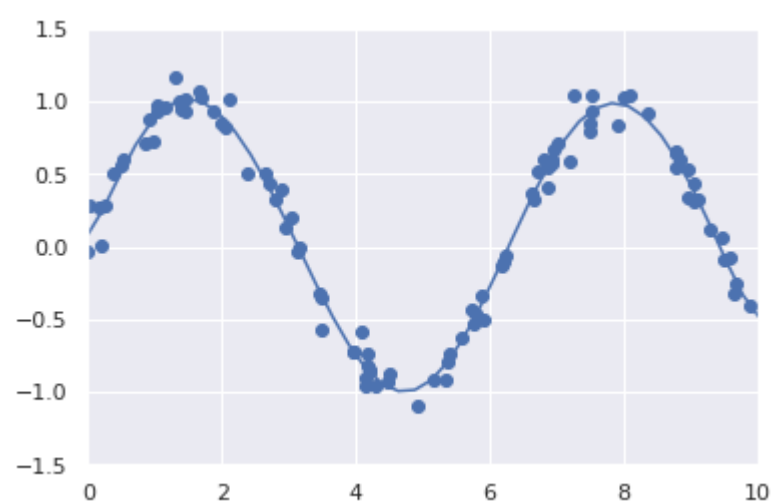
Degree: 8 Score: 0.9815395575076654



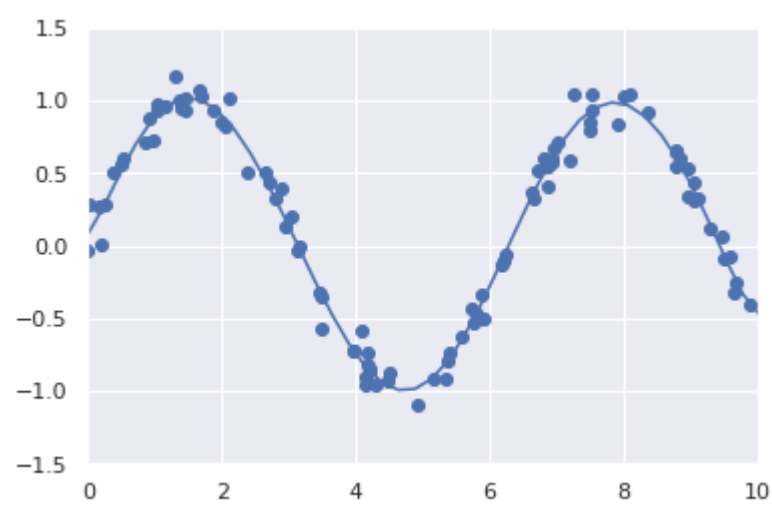
Degree: 9 Score: 0.9815556828827723



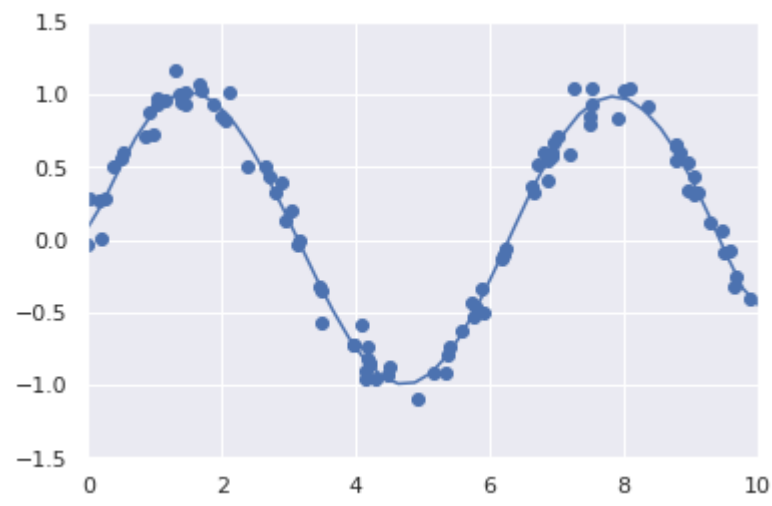
Degree: 10 Score: 0.9817554326412768



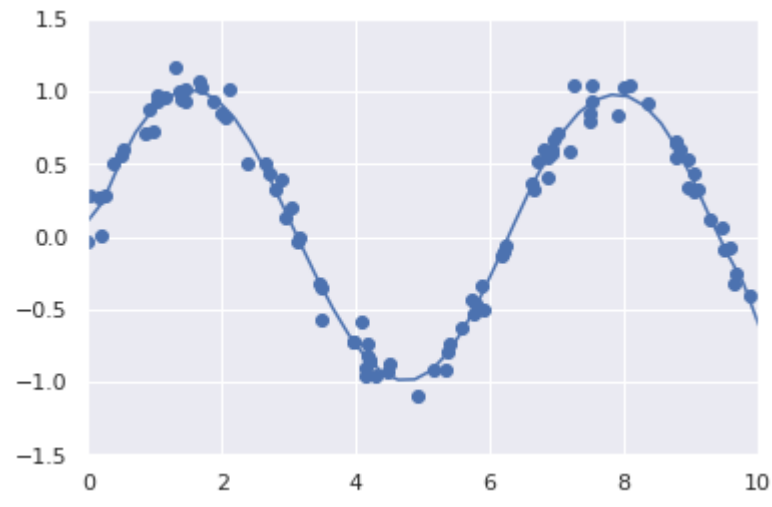
Degree: 11 Score: 0.9817593861429719



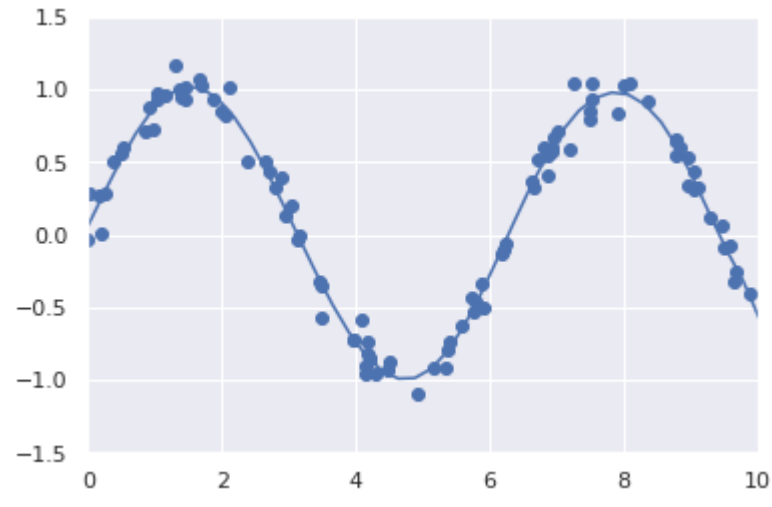
Degree: 12 Score: 0.9817630665937364



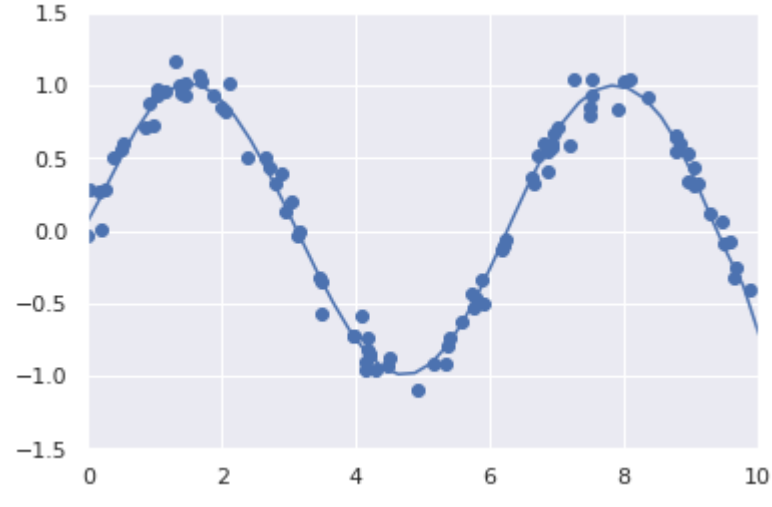
Degree: 13 Score: 0.9819417769966973

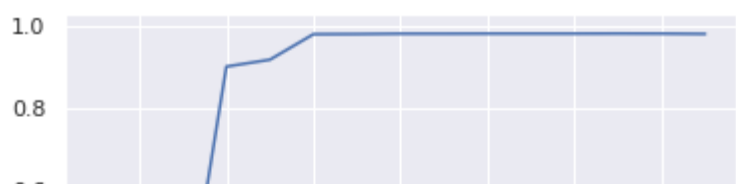


Degree: 14 Score: 0.9817712069108879



Degree: 15 Score: 0.9810106927307879





▼ 1e. Regularization

The use of polynomial regression with high-order polynomials can very quickly lead to over-fitting. In this part, we will look into the use of regularization to address potential overfitting.

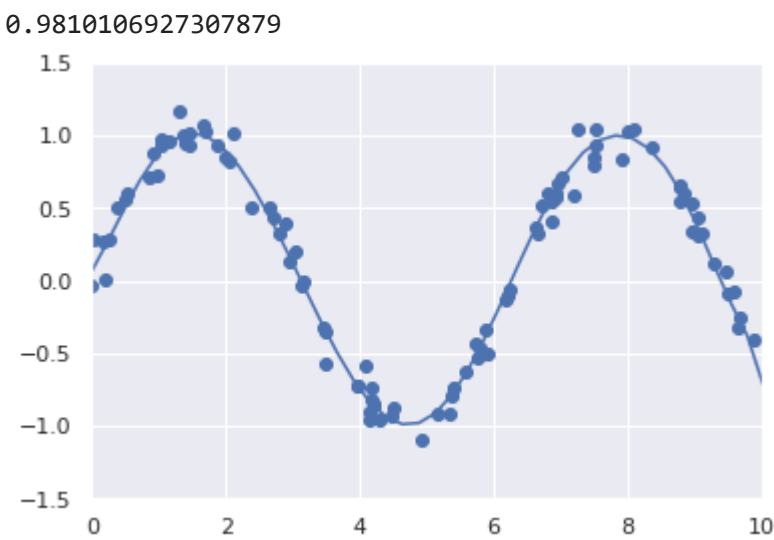
The code below shows an attempt to fit a 15th degree polynomial to a sinusoidal shaped data. The fit is excellent ($R^2 > 0.98$), but might raise suspicions that it will lead to overfitting.

```
model = make_pipeline(PolynomialFeatures(15),
                      LinearRegression())
model.fit(x[:, np.newaxis], y)

plt.scatter(x, y)
plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

plt.xlim(0, 10)
plt.ylim(-1.5, 1.5);

score = model.score(x[:, np.newaxis], y)
print(score)
```



▼ 1.4 Your turn! (5 points)

Write Python code to perform Ridge regression (L_2 Regularization), plot the resulting fit, and compute the R^2 score.

Hints:

1. This type of penalized model is built into Scikit-Learn with the `Ridge` estimator.
2. In the beginning, use all default values for its parameters.
3. After you get your code to work, spend some time trying to fine-tune the model, i.e., experimenting with the regularization parameters.

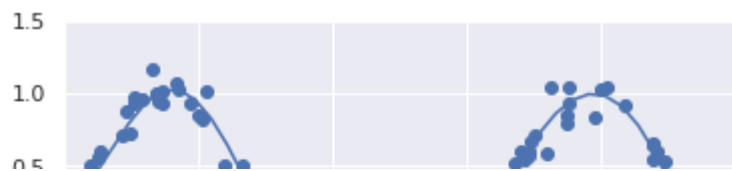
```
from sklearn.linear_model import Ridge
model = make_pipeline(PolynomialFeatures(15),
                      Ridge(alpha=0.5))
model.fit(x[:, np.newaxis], y)

plt.scatter(x, y)
plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

plt.xlim(0, 10)
plt.ylim(-1.5, 1.5);

score = model.score(x[:, np.newaxis], y)
print("R^2: " + str(score))
print("")
```

R^2: 0.9799966212299248



▼ 1.5 Your turn! (5 points)

Write Python code to perform Lasso regression (L_1 Regularization), plot the resulting fit, and compute the R^2 score.

Hints:

1. This type of penalized model is built into Scikit-Learn with the `Lasso` estimator.
2. In the beginning, use `Lasso(alpha=0.1, tol=0.2)`
3. After you get your code to work, spend some time trying to fine-tune the model, i.e., experimenting with the regularization parameters.

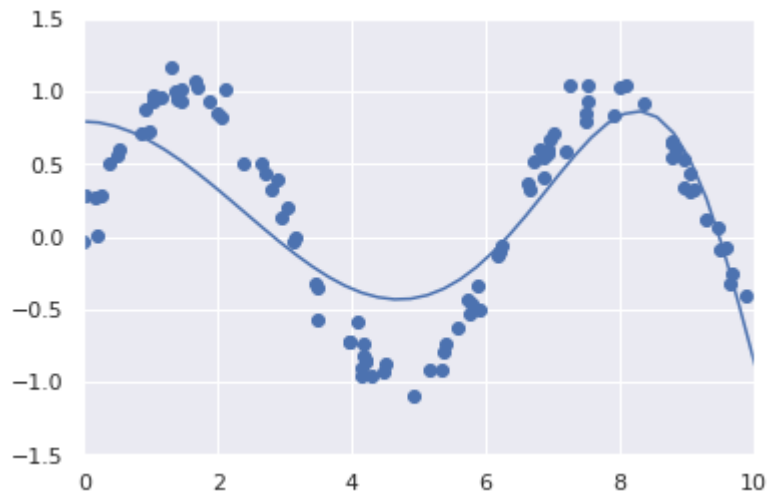
```
from sklearn.linear_model import Lasso
model = make_pipeline(PolynomialFeatures(15),
                      Lasso(alpha = 0.12, tol = 0.2))
model.fit(x[:, np.newaxis], y)

plt.scatter(x, y)
plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

plt.xlim(0, 10)
plt.ylim(-1.5, 1.5);

score = model.score(x[:, np.newaxis], y)
print("R^2: " + str(score))
```

R^2: 0.6981705330840109



▼ 1f. The housing problem

The Boston housing dataset is a classic dataset used in linear regression examples. (See <https://scikit-learn.org/stable/datasets/index.html#boston-dataset> for more)

The Python code below:

- Loads the Boston dataset (using scikit-learn's `load_boston()`) and converts it into a Pandas dataframe
- Selects two features to be used for fitting a model that will then be used to make predictions: LSTAT (% lower status of the population) and RM (average number of rooms per dwelling) (*)
- Splits the data into train and test sets

(*) See <https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155> for details.

```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is depreca

The Boston housing prices dataset has an ethical problem. You can refer to the documentation of this function for further details.

The scikit-learn maintainers therefore strongly discourage the use of this dataset unless the purpose of the code is to study and educate about ethical issues in data science and machine learning.

In this special case, you can fetch the dataset from the original source::

```
import pandas as pd
import numpy as np

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
```

Alternative datasets include the California housing dataset (i.e. :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing dataset. You can load the datasets as follows::


```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

for the California housing dataset and::

```
from sklearn.datasets import fetch_openml
housing = fetch_openml(name="house_prices", as_frame=True)
```

for the Ames housing dataset.

```
warnings.warn(msg, category=FutureWarning)
```

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT 

```
boston['MEDV'] = boston_dataset.target
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT', 'RM'])
y = boston['MEDV']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(404, 2)
(102, 2)
(404,)
(102,)
```

▼ (OPTIONAL EDA)

The (innocent-looking) lines of code below use Pandas Profiling to produce rich reports, plots and insights on the dataset.

Read more about it:

- <https://pypi.org/project/pandas-profiling/>
- <https://www.datacourses.com/pandas-1150/>
- <https://pandas-profiling.github.io/pandas-profiling/docs/master/index.html>
- <https://medium.com/analytics-vidhya/pandas-profiling-5ecd0b977ecd>

```
# fun with pandas_profiling
profile = ProfileReport(boston, title='Pandas Profiling Report for Boston Housing Dataset', explorative=True)
```

```
profile.to_notebook_iframe()
```

Overview

Dataset statistics

Number of variables	14
Number of observations	506
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	55.5 KiB
Average record size in memory	112.3 B

Variable types

Numeric	13
Categorical	1

Alerts

CRIM is highly correlated with ZN and 8 other fields (ZN, INDUS, NOX, AGE, DIS, RAD,

High correlation

▼ 1.6 Bonus! (10 points)

Write Python code to:

1. Fit a linear model to the data.
2. Compute and print the RMSE and R^2 score for both train and test datasets.
3. Fit a polynomial model (of degree 4) to the data.
4. Compute and print the RMSE and R^2 score for both train and test datasets.
5. Apply Ridge regression to the polynomial model.
6. Compute and print the RMSE and R^2 score for both train and test datasets.

```
# Enter Your Code Here
#...
#...
#...
```

▼ Part 2: Classification

▼ 2a. The Iris dataset

The Python code below will load a dataset containing information about three types of Iris flowers that had the size of its petals and sepals carefully measured.


The Fisher’s Iris dataset contains 150 observations with 4 features each:

- sepal length in cm;
- sepal width in cm;
- petal length in cm; and
- petal width in cm.

The class for each instance is stored in a separate column called “species”. In this case, the first 50 instances belong to class Setosa, the following 50 belong to class Versicolor and the last 50 belong to class Virginica.

See: <https://archive.ics.uci.edu/ml/datasets/Iris> for additional information.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
iris = sns.load_dataset("iris")
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	
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	

▼ Histograms, pair plots and summary statistics

The code below:

- 1. Computes and displays relevant summary statistics for the whole dataset.
- 2. Displays the pair plots for all (4) attributes for all (3) categories / species / classes in the Iris dataset.

```
# Display pair plot
sns.pairplot(iris, hue='species', height=2.5);

# Display summary statistics for the whole dataset
iris.describe()
```


	sepal_length	sepal_width	petal_length	petal_width
count	150 0000000	150 0000000	150 0000000	150 0000000



▼ 2.1 Your turn! (15 points)

Write code to:

- 1. Build a decision tree classifier using scikit-learn's `DecisionTreeClassifier` (using the default options). Check documentation at <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
- 2. Plot the resulting decision tree. (Note: if `graphviz` gives you headaches, a text-based 'plot'-- using `export_text` -- should be OK.)
- 3. Perform k-fold cross-validation using `k=3` and display the results.

8

```
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
import graphviz

iris = load_iris()
X, y = iris.data, iris.target
classifier = tree.DecisionTreeClassifier(random_state=0)
classifier = classifier.fit(X, y)

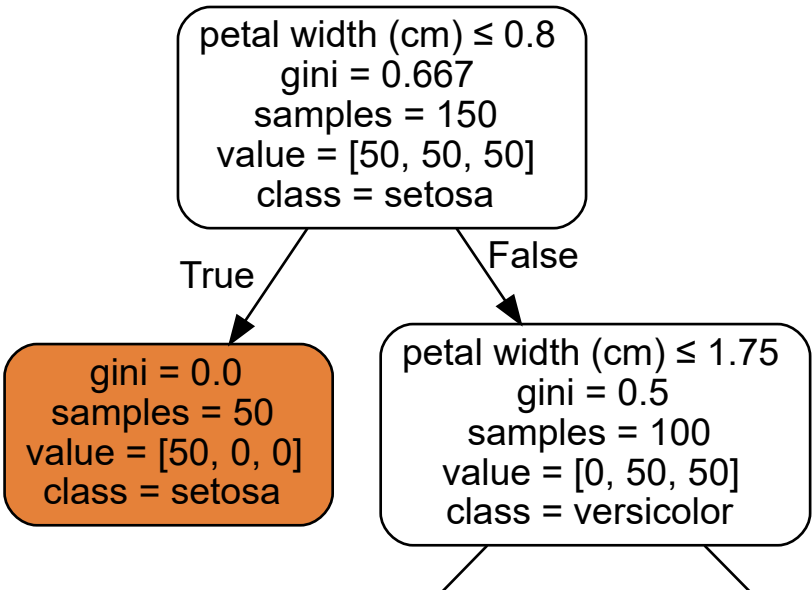
dot_data = tree.export_graphviz(classifier, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")

dot_data = tree.export_graphviz(classifier, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph

scores = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
print("K-fold cross-validation results")
print("-----")
print(scores)
print('Mean Accuracy: %.3f Standard Deviation: %.3f' % (mean(scores), std(scores)))
print("")
graph
```

K-fold cross-validation results

[0.98 0.94 0.98]
Mean Accuracy: 0.967 Standard Deviation: 0.019



2b. Digit classification

The MNIST handwritten digit dataset consists of a training set of 60,000 examples, and a test set of 10,000 examples. Each image in the dataset has 28×28 pixels.

The Python code below loads the images from the MNIST dataset, flattens them, normalizes them (i.e., maps the intensity values from [0..255] to [0..1]), and displays a few images from the training set.

```
from keras.datasets import mnist

# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and validation sets
(X_train, y_train), (X_valid, y_valid) = mnist.load_data()
```

```
X_train.shape

(60000, 28, 28)
```

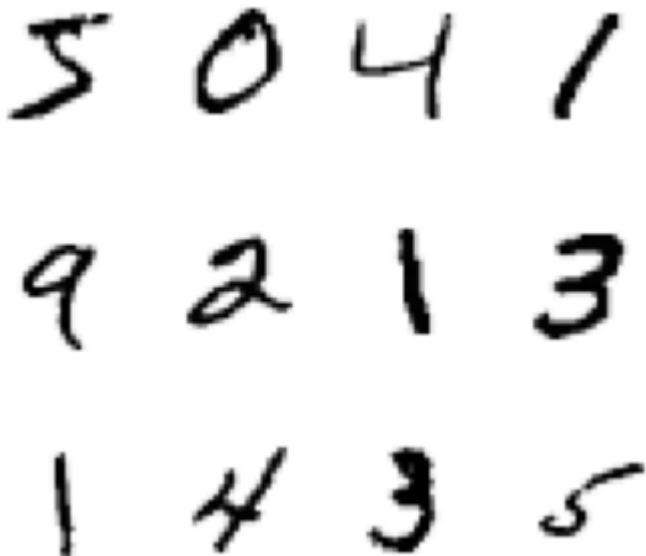
```
y_train.shape

(60000,)
```

```
y_train[0:12]

array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5], dtype=uint8)
```

```
plt.figure(figsize=(5,5))
for k in range(12):
    plt.subplot(3, 4, k+1)
    plt.imshow(X_train[k], cmap='Greys')
    plt.axis('off')
plt.tight_layout()
plt.show()
```



```
X_valid.shape
```

```
(10000, 28, 28)
```

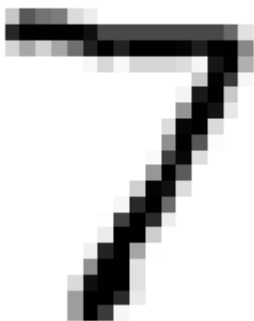
```
y_valid.shape
```

```
(10000,)
```

```
y_valid[0]
```

```
7
```

```
plt.imshow(X_valid[0], cmap='Greys')
plt.axis('off')
plt.show()
```



```
# Reshape (flatten) images
X_train_resaped = X_train.reshape(60000, 784).astype('float32')
X_valid_resaped = X_valid.reshape(10000, 784).astype('float32')

# Scale images to the [0, 1] range
X_train_scaled_resaped = X_train_resaped / 255
X_valid_scaled_resaped = X_valid_resaped / 255

# Renaming for conciseness
X_training = X_train_scaled_resaped
X_validation = X_valid_scaled_resaped

print("X_training shape (after reshaping + scaling):", X_training.shape)
print(X_training.shape[0], "train samples")
print("X_validation shape (after reshaping + scaling):", X_validation.shape)
print(X_validation.shape[0], "validation samples")
```

```
X_training shape (after reshaping + scaling): (60000, 784)
60000 train samples
X_validation shape (after reshaping + scaling): (10000, 784)
10000 validation samples
```

```
# convert class vectors to binary class matrices
y_training = keras.utils.to_categorical(y_train, num_classes)
y_validation = keras.utils.to_categorical(y_valid, num_classes)
```

```
print(y_valid[0])
print(y_validation[0])

7
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

▼ 2.2 Your turn! (10 points)

Write code to:

- 1. Build and fit a 10-class Naive Bayes classifier using scikit-learn's `MultinomialNB()` with default options and using the raw pixel values as features.
- 2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: your accuracy will be around 83.5%

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

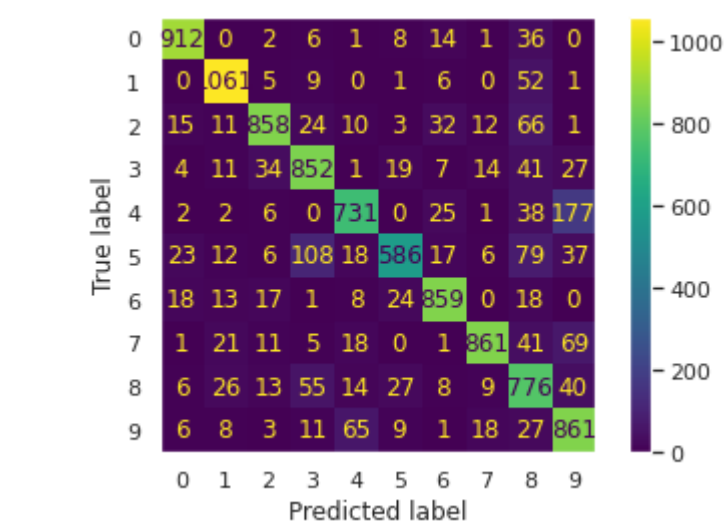
classifier = MultinomialNB()
classifier = classifier.fit(X_training, y_train)
```

```
y_pred = classifier.predict(X_validation)
accuracy = accuracy_score(y_valid, y_pred)

print("The overall accuracy is " + str(accuracy*100) + "%")
print("")

matrix = confusion_matrix(y_valid, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=matrix)
disp.plot()
plt.grid(b=0)
plt.show()
```

The overall accuracy is 83.57%



2.3 Your turn! (10 points)

Write code to:

- 1. Build and fit a 10-class Random Forests classifier using scikit-learn's `RandomForestClassifier()` with default options (don't forget `random_state=0`) and using the raw pixel values as features.
- 2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: your accuracy should be > 90%

```
from sklearn.ensemble import RandomForestClassifier

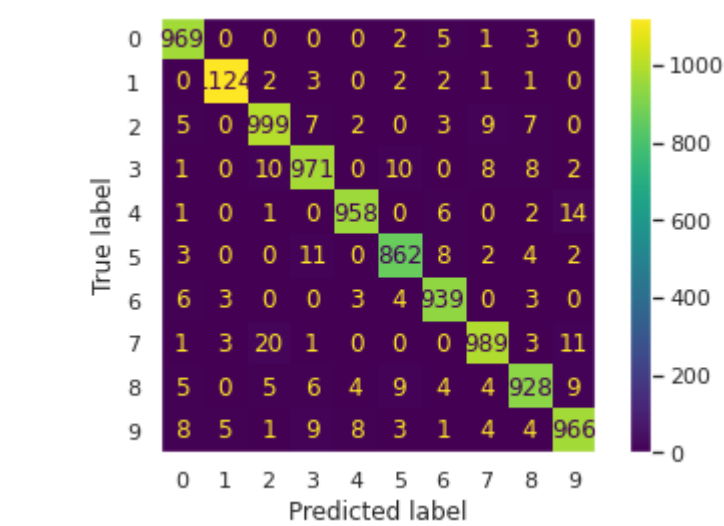
classifier = RandomForestClassifier(random_state=0)
classifier = classifier.fit(X_training, y_train)

y_pred = classifier.predict(X_validation)
accuracy = accuracy_score(y_valid, y_pred)

print("The overall accuracy is " + str(accuracy*100) + "%")
print("")

matrix = confusion_matrix(y_valid, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=matrix)
disp.plot()
plt.grid(b=0)
plt.show()
```

The overall accuracy is 97.05%



2.4 Your turn! (10 points)

Write code to:

1. Build and fit a 10-class classifier of your choice, with sensible initialization options, and using the raw pixel values as features.
2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: A variation of the Random Forests classifier from 2.2 above is acceptable. In that case, document your selection of (hyper)parameters and your rationale for choosing them.

I wanted to try a different classifier than the two above. I used a gradient boosting classifier, and despite being unable to beat the baseline set by the classifier in 2.2, the result got very near the baseline accuracy. Training the gradient boosting classifier using the parameters that I set took more than 20 minutes and had a lower accuracy than either of the previous two classifiers. Given the lower accuracy and the longer training time, the gradient boosting classifier is not a good model for this problem.

```
from sklearn.ensemble import GradientBoostingClassifier

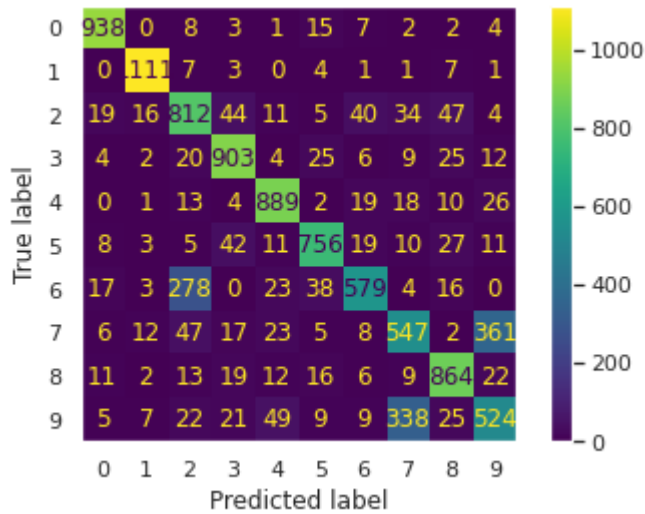
classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
classifier = classifier.fit(X_training, y_train)

y_pred = classifier.predict(X_validation)
accuracy = accuracy_score(y_valid, y_pred)

print("The overall accuracy is " + str(accuracy*100) + "%")
print("")

matrix = confusion_matrix(y_valid, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=matrix)
disp.plot()
plt.grid(b=0)
plt.show()
```

The overall accuracy is 79.23%



▼ Part 3: Face Recognition using PCA (eigenfaces)

In this part you will build a face recognition solution.

We will use a subset of the Labeled Faces in the Wild (LFW) people dataset: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_lfw_people.html

The Python code below loads a dataset of 1867 images (resized to 62 × 47 pixels) from the dataset and displays some of them.

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=40)
print(faces.target_names)
print(faces.images.shape)

['Ariel Sharon' 'Arnold Schwarzenegger' 'Colin Powell' 'Donald Rumsfeld'
 'George W Bush' 'Gerhard Schroeder' 'Gloria Macapagal Arroyo'
 'Hugo Chavez' 'Jacques Chirac' 'Jean Chretien' 'Jennifer Capriati'
 'John Ashcroft' 'Junichiro Koizumi' 'Laura Bush' 'Lleyton Hewitt'
 'Luiz Inacio Lula da Silva' 'Serena Williams' 'Tony Blair'
 'Vladimir Putin']
(1867, 62, 47)

plt.rcParams["figure.figsize"]=15,15
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
```



Tony Blair



Tony Blair



George W Bush



Colin Powell



Donald Rumsfeld



Serena Williams



Colin Powell



Donald Rumsfeld



Colin Powell



Colin Powell



George W Bush



Gerhard Schroeder



Jean Chretien



Colin Powell



Jennifer Capriati

▼ 3.1 Your turn! (15 points)

Write code to:

1. Use Principal Component Analysis (PCA) to reduce the dimensionality of each face to the first 120 components.
2. Build and fit a multi-class SVM classifier, with sensible initialization options, and using the PCA-reduced features.
3. Make predictions on the test data, compute the precision, recall and f1 score for each category, compute the overall accuracy, and plot the resulting confusing matrix.
4. Display examples of correct and incorrect predictions (at least 5 of each).

```

from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.utils.fixes import loguniform

X = faces.data
y = faces.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

pca = PCA(n_components=120, svd_solver="randomized", whiten=True).fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

param_grid = {
    "C": loguniform(1e3, 1e5),

```

```
    "gamma": loguniform(1e-4, 1e-1),
}

classifier = RandomizedSearchCV(
    SVC(kernel="rbf", class_weight="balanced"), param_grid, n_iter=10, random_state=0
)
classifier.fit(X_train_pca, y_train)

y_pred = classifier.predict(X_test_pca)
accuracy = accuracy_score(y_test, y_pred)

print("The overall accuracy is " + str(accuracy*100) + "%")
print("")

disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, display_labels=faces.target_names, xticks_rotation="vertical")
plt.grid(b=0)
plt.show()

print("")
print(classification_report(y_test, y_pred))
```


The overall accuracy is 77.73019271948608%

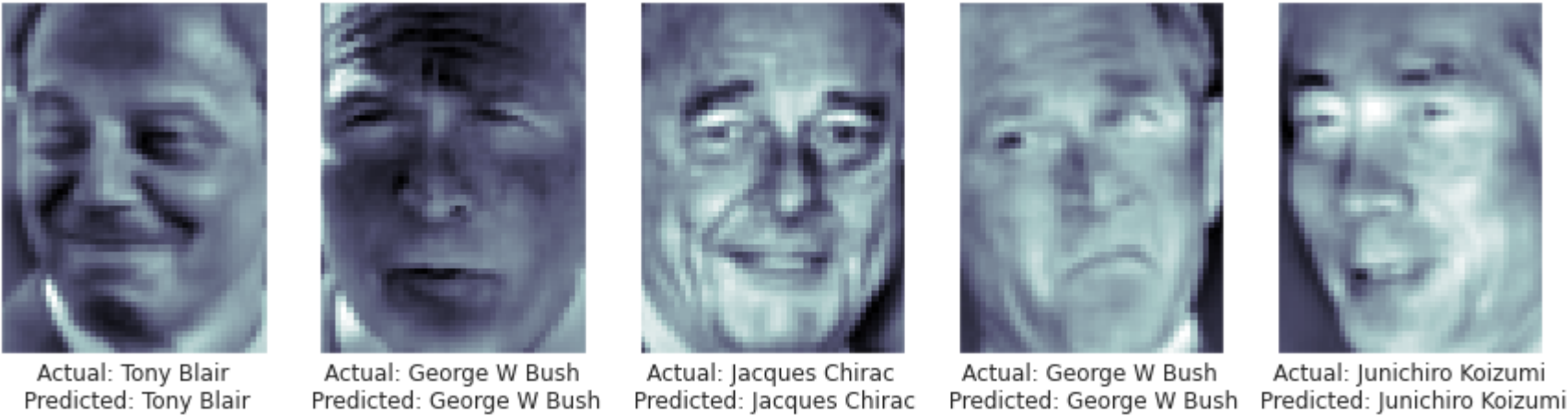
Ariel Sharon	12	1	1	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0
Arnold Schwarzenegger	0	2	0	0	4	3	0	0	0	0	0	0	0	0	0	0	1	2	1
Colin Powell	2	0	48	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Donald Rumsfeld	0	0	0	22	1	0	0	0	0	0	0	0	0	0	1	0	0	1	2
George W Bush	1	0	7	1	121	1	0	0	0	0	0	0	0	0	0	0	0	0	0

```
incorrect_index = []
correct_index = []
correct = []
incorrect = []

for i in range(len(y_pred)):
    if y_pred[i] != y_test[i]:
        incorrect_index.append(i)
    else:
        correct_index.append(i)

print("Correctly Classified: ")
print("")
plt.rcParams["figure.figsize"]=15,15
fig, ax = plt.subplots(1, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[correct_index[i]].reshape((faces.images.shape[1], faces.images.shape[2])), cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel = "Actual: " + str(faces.target_names[y_test[correct_index[i]]]) + "\n Predicted: " + str(faces.target_
```

Correctly Classified:



```
print("Incorrectly Classified: ")
print("")
plt.rcParams["figure.figsize"]=15,15
fig, ax = plt.subplots(1, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[incorrect_index[i]].reshape((faces.images.shape[1], faces.images.shape[2])), cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel = "Actual: " + str(faces.target_names[y_test[incorrect_index[i]]]) + "\n Predicted: " + str(faces.target_
```

Incorrectly Classified:



▼ Conclusions (10 points)

Write your conclusions and make sure to address the issues below:

- What have you learned from this assignment?
- Which parts were the most fun, time-consuming, enlightening, tedious?
- What would you do if you had an additional week to work on this?

This assignment demonstrated the various approaches to solving machine learning problems and helped me understand the scope of available machine learning solutions. Different models were used to solve regression and classification problems. These models included decision trees and optimized variations of decision trees as well as linear and polynomial regression models. I better understood the workflow behind machine learning projects by having to go through the parts of the workflow many different times in this project.

This project also helped me understand the different non-deep learning approaches to machine learning. I do not have much experience with machine learning models other than neural networks, so this project opened my eyes to the different methods available to solve regression and classification problems. These approaches are much more light weight compared to deep learning solutions and can be more effective when considering the overhead of such neural networks. While a neural network would take a few minutes to train, these models took only a few seconds.

The most fun part of this project was working with the polynomial regression in part 1. It was interesting to see how the different degrees impacted the quality of the fit. The model became better and better until it reached a point at which a higher degree did not improve the quality. The most time-consuming portion of this assignment was reading documentation. I spent more than 75% of the time that I worked on this assignment crawling through documentation to understand how to use the different machine learning models. This was especially true for part three of the project. The most tedious part of the assignment was debugging. I had a difficult time with the syntax to print the incorrectly and correctly classified faces in part three. Overall, the most enlightening portion of the project was the second part. The number of models that I had to train helped complete my understanding of the machine learning workflow.

The final portion of this project was the most difficult for me. The face recognition using PCA required me to do a significant amount of documentation reading to understand. Additionally, I had numerous bugs that stumped me, and I had never dealt with the topics covered in part three, so it took an additional amount of time to fully understand. Through reading Sklearn documentation I was able to understand the overall picture of the approach to face recognition. Given more time to do this assignment, I would explore how the different parameters effect the performance of the models that we explored in this assignment. I understand the overall picture of the models, but I would like to further explore how the parameters impact the results. The experimentation that I did with these parameters in this project showed me that these parameters have a great deal to do with the quality of the models.

Finally, this assignment further showed me that the main issue in machine learning is the battle between overfitting and underfitting. Once these models were sufficiently trained, some of them overfit to the training data. These models then needed to be generalized, such as using methods such as regularization for polynomial regression. Overall, the assignment improved my understanding of how machine learning is used as well as why it is used. I better understand how to implement and use machine learning models and I better understand why different methods are used and how they compare to deep learning solutions.