1) Download the sample Python code that comes with the assigned textbook, Python Machine Learning.

class Perceptron(object)

```
#@title class Perceptron(object)
import numpy as np
class Perceptron(object):
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01,
                              size=1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
                self.errors_.append(errors)

        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

class AdalineSGD(object)

```
#@title class AdalineSGD(object)
import numpy as np
class AdalineSGD(object):
    def __init__(self, eta=0.01, n_iter=10, shuffle=True, random_state=None):
        self.eta = eta
        self.n_iter = n_iter
        self.w_initialized = False
        self.shuffle = shuffle
        self.random_state = random_state

    def fit(self, X, y):
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
                cost.append(self._update_weights(xi, target))
            avg_cost = sum(cost) / len(y)
            self.cost_.append(avg_cost)
        return self

    def partial_fit(self, X, y):
        if not self.w_initialized:
            self._initialize_weights(X.shape[1])
        if y.ravel().shape[0] > 1:
            for xi, target in zip(X, y):
                self._update_weights(xi, target)
        else:
            self._update_weights(X, y)
        return self

    def _shuffle(self, X, y):
        r = self.rgen.permutation(len(y))
        return X[r], y[r]

    def _initialize_weights(self, m):
        self.rgen = np.random.RandomState(self.random_state)
        self.w_ = self.rgen.normal(loc=0.0, scale=0.01,
                                   size=1 + m)
        self.w_initialized = True

    def _update_weights(self, xi, target):
        output = self.activation(self.net_input(xi))
        error = (target - output)
        self.w_[1:] += self.eta * xi.dot(error)
        self.w_[0] += self.eta * error
        cost = 0.5 * error**2
        return cost

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return X

    def predict(self, X):
        return np.where(self.activation(self.net_input(X))>= 0.0, 1, -1)
```

plot_decision_regions() function

```
#@title plot_decision_regions() function
#code for showing decision regions:
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')
```

2) Use the Iris dataset that's referenced in the text.

Import Iris dataset

```
#@title Import Iris dataset
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df_iris = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                 header=None)
df_iris.head()
df_iris.tail()
```

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

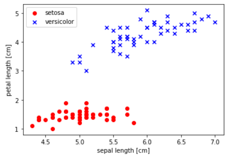
3) You can use the sample code from text as a starting point or you can write your own code from scratch.

4) Pick two classes of data (i.e., two species of Iris) and two features from the four in the dataset, so that the data for two species are linearly separable using the features that you have chosen.

TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE

```
#@title TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE
#two species of iris: setosa, versicolor
#two features from the data set: sepal length, petal length
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df_iris = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                 header=None)
df_iris.tail()

#pull two species of iris: setosa, versicolor
y = df_iris.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

#pull two features from the data set: sepal length, petal length
```

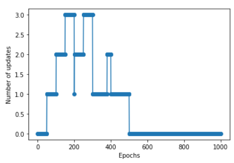
```
X = df_iris.iloc[0:100, [0,2]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')

plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```

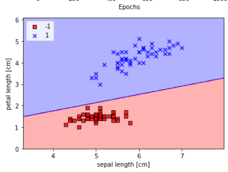


Then Apply the Perceptron and Adaline models to the classes/features that you have chosen and report your results.

TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE: Perceptron

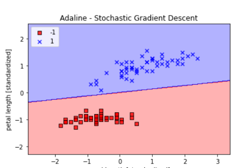
```
#@title TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(setosa, versicolor), two features(sepal length, petal length)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("lin_sep_perceptron_two_class_two_feat.png")

#plot the decision regions:
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```

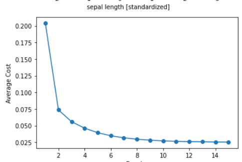




TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / TWO FEATURES - LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(setosa, versicolor), two features(sepal length, petal length)
#standardize X (page 43)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')
plt.show()


plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("lin_sep_adaline_two_class_two_feat.png")
```



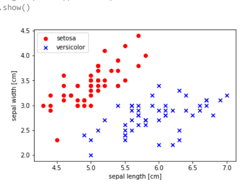



```
<Figure size 432x288 with 0 Axes>
```

5) Repeat Step 4) using three features at a time.

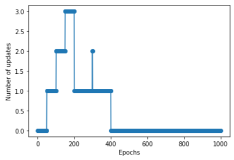
TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE

```
#@title TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE
#pull two species of iris: setosa, versicolor
y = df_iris.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1,1)

#pull three features from the data set: sepal length, sepal width, petal length
X = df_iris.iloc[0:100, [0,1,2]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')

plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('sepal width [cm]')
#plt.zlabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE: Perceptron

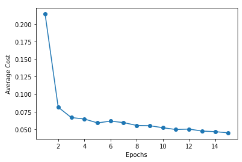
```
#@title TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(setosa, versicolor), three features(sepal length, petal length, sepal width)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("lin_sep_perceptron_two_class_three_feat.png")
```




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

TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / THREE FEATURES - LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(setosa, versicolor), three features(sepal length, petal length, sepal width)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("lin_sep_adaline_two_class_three_feat.png")
```

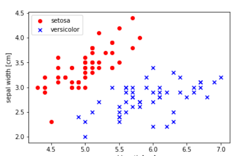



```
<Figure size 432x288 with 0 Axes>
```

6) Repeat Step 4) using all four features at the same time.

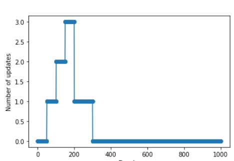
TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE

```
#@title TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE
#pull two species of iris: setosa, versicolor
y = df_iris.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1,1)

#pull four features from the data set: sepal length, sepal width, petal length, petal width
X = df_iris.iloc[0:100, [0,1,2,3]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')

plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('sepal width [cm]')
#plt.zlabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE: Perceptron

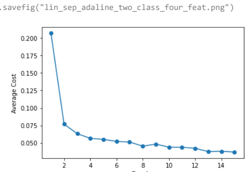
```
#@title TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(setosa, versicolor), four features(sepal length, petal length, sepal width, petal width)

ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("lin_sep_perceptron_two_class_four_feat.png")
```




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

TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / FOUR FEATURES - LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(setosa, versicolor), four features(sepal length, petal length, sepal width, petal width)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("lin_sep_adaline_two_class_four_feat.png")
```

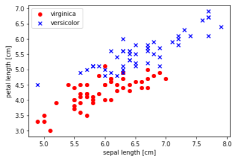



```
<Figure size 432x288 with 0 Axes>
```

7) Pick two classes of data (i.e., two species of Iris) and two features from the four in the dataset, so that the data for two species are NOT linearly separable using the features that you have chosen.

TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE

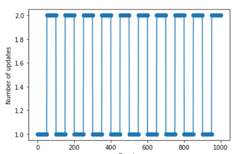
```
#@title TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE
#classes: virginica and versicolor, features: sepal length, petal length
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df_iris = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
                 header=None)
df_iris.tail()

#pull two species of iris: virginica, versicolor
y = df_iris.iloc[50:150, 4].values
y = np.where(y == 'Iris-virginica', -1, 1)

#pull two features from the data set: sepal length, petal length
X = df_iris.iloc[50:150, [0,2]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='virginica')

plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```

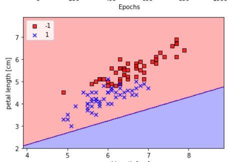


Then Apply the Perceptron and Adaline models to the classes/features that you have chosen and report your results.

TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE: Perceptron

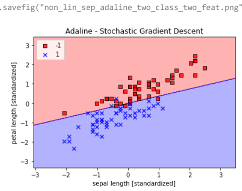
```
#@title TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(virginica, versicolor), two features(sepal length, petal length)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("non_lin_sep_perceptron_two_class_two_feat.png")


#plot the decision regions:
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```

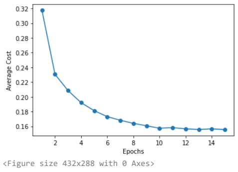




TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / TWO FEATURES - NOT LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(virginica, versicolor), two features(sepal length, petal length)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')
plt.show()


plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("non_lin_sep_adaline_two_class_two_feat.png")
```





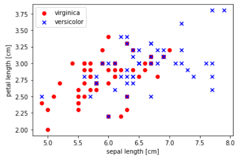
<Figure size 432x288 with 0 Axes>

8) Repeat Step 7) using three features at a time.

TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE

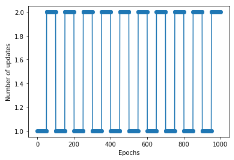
```
#@title TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE
#pull two species of iris: virginica, versicolor
y = df_iris.iloc[50:150, 4].values
y = np.where(y == 'Iris-virginica', -1, 1)

#pull three features from the data set: sepal length, sepal width, petal length
X = df_iris.iloc[50:150, [0,1,2]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='virginica')
```

```
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE: Perceptron

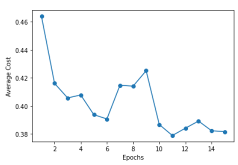
```
#@title TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(virginica, versicolor), three features(sepal length, petal length, sepal width)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("non_lin_sep_perceptron_two_class_three_feat.png")
```




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

TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / THREE FEATURES - NOT LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(virginica, versicolor), three features(sepal length, petal length, sepal width)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)

plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("non_lin_sep_adaline_two_class_three_feat.png")
```

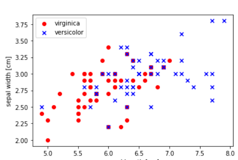



```
<Figure size 432x288 with 0 Axes>
```

9) Repeat Step 7) using all four features at the same time.

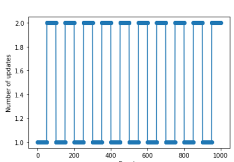
TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE

```
#@title TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE
#pull two species of iris: virginica, versicolor
y = df_iris.iloc[50:150, 4].values
y = np.where(y == 'Iris-virginica', -1,1)

#pull four features from the data set: sepal length, sepal width, petal length, petal width
X = df_iris.iloc[50:150, [0,1,2,3]].values
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='virginica')

plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('sepal width [cm]')
#plt.zlabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE: Perceptron

```
#@title TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE: Perceptron
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

#applying Perceptron to two classes(virginica, versicolor), four features(sepal length, sepal width, petal length, petal width)
ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fit(X, y)
plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
plt.savefig("non_lin_sep_perceptron_two_class_four_feat.png")
```

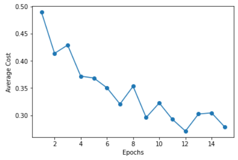



```
<Figure size 432x288 with 0 Axes>
```

TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE: Adaline

```
#@title TWO CLASS / FOUR FEATURES - NOT LINEARLY SEPARABLE: Adaline
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
#applying Adaline to two classes(virginica, versicolor), four features(sepal length, sepal width, petal length, petal width)
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

ada = AdalineSGD(n_iter=15, eta=0.01, random_state=1)
ada.fit(X_std, y)


plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
```

```
plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.show()
plt.savefig("non_lin_sep_adaline_two_class_four_feat.png")
```




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

10) Compare your results for the Perceptron and Adaline models.

All applications of the Adaline models on the non linearly separable data ran with fewer epochs. The perceptron models ran with a high number of epochs on the non linearly separable data.

11) Submit your results as an ipython notebook (i.e. in ipynb format).

✓ 0s completed at 7:51 PM