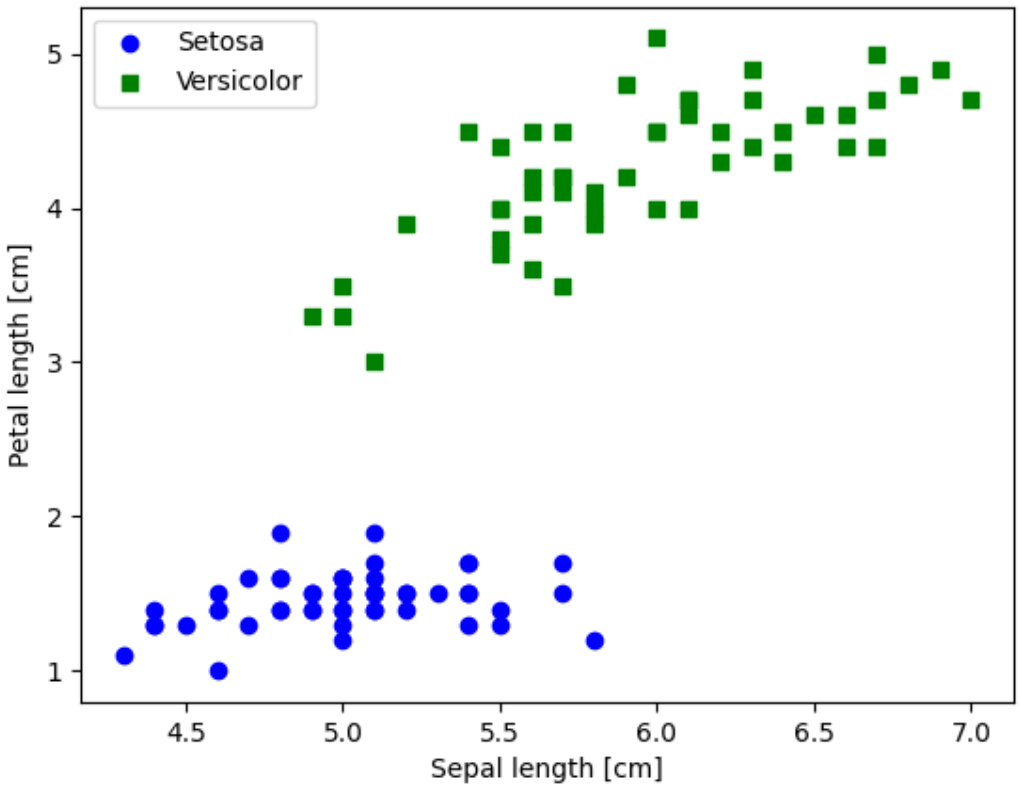
INDEX

1. **Write a python program to Prepare Scatter Plot (Use Forge Dataset Iris Dataset)**

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

from sklearn.datasets import load_iris
iris = load_iris()

df= pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                 columns= iris['feature_names'] + ['target'])

# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', 0, 1)

# extract sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='blue', marker='o', label='Setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='green', marker='s', label='Versicolor')

plt.xlabel('Sepal length [cm]')
plt.ylabel('Petal length [cm]')
plt.legend(loc='upper left')

# plt.savefig('images/02_06.png', dpi=300)
plt.show()
```

Output:


Setosa
Versicolor
Petal length [cm]
Sepal length [cm]
1
2
3
4
5
4.5
5.0
5.5
6.0
6.5
7.0

2. Write a python program to find all null values in a given data set and remove them.

```
import pandas as pd
df = pd.read_csv('data.csv')
null_mask = df.isnull()
null_columns = df.columns[null_mask.any()]
null_rows = df.loc[:, null_columns]
df = df.dropna(axis=0, subset=null_columns)
df.to_csv('clean_data.csv', index=False)
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	0	0	1	3	1	2	4	7	8	3	3	3	10	5
2	0	1	2	1	2	1	3	2	2	6	10	11	5	9
3	0	1	1	3	3	2	6	2	5	9	5	7	4	5
4	0	0	2	0	4	2	2	1	6	7	10	7	9	13
5	0	1	1	3	3	1	3	5	2	4	4	7	6	5

Output:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	0	0.1	1	3	1.1	2	4	7	8	3.1	3.2	3.3	10	5
2	0	1	2	1	2	1	3	2	2	6	10	11	5	9
3	0	1	1	3	3	2	6	2	5	9	5	7	4	5
4	0	0	2	0	4	2	2	1	6	7	10	7	9	13
5	0	1	1	3	3	1	3	5	2	4	4	7	6	5

3. Write a python program the Categorical values in numeric format for a given dataset.

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

def convert_categorical_to_numeric(dataframe, column_name):
  # Create a label encoder object
  encoder = LabelEncoder()

  # Fit the encoder to the categorical column
  encoder.fit(dataframe[column_name])

  # Transform the categorical column to numeric format
  dataframe[column_name] = encoder.transform(dataframe[column_name])

# Example usage
df = pd.DataFrame({'col1': ['cat', 'dog', 'bird', 'cat', 'dog', 'bird'], 'col2': [1, 2, 3, 4, 5, 6]})
convert_categorical_to_numeric(df, 'col1')
print(df)
```

Output:

	col1	col2
0	1	1
1	2	2
2	0	3
3	1	4
4	2	5
5	0	6

4. Write a python program to implement simple Linear Regression for predicting house price.

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Assume that we have a dataset with two columns: 'area' and 'price',
# where 'area' is the size of the house in square feet and 'price' is the price of the house
X = np.array([[1000], [1500], [2000], [2500], [3000]])
y = np.array([300000, 400000, 500000, 600000, 700000])

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X, y)

# Predict the price of a house with an area of 3500 square feet
prediction = model.predict([[3500]])
print(prediction)
```

Output:

[800000.]

5. Write a python program to implement multiple Linear Regression for a given dataset.

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Assume that we have a dataset with three columns: 'area', 'bedrooms', and 'price',
# where 'area' is the size of the house in square feet, 'bedrooms' is the number of bedrooms,
# and 'price' is the price of the house
df = pd.DataFrame({'area': [1000, 1500, 2000, 2500, 3000],
                   'bedrooms': [3, 4, 5, 6, 7],
                   'price': [300000, 400000, 500000, 600000, 700000]})

# Create a Linear Regression model
model = LinearRegression()

# Split the data into training and test sets
X = df[['area', 'bedrooms']]
y = df['price']

model.fit(X, y)

# Predict the price of a house with an area of 3500 square feet and 4 bedrooms
prediction = model.predict([[3500, 4]])
print(prediction)
```

Output:

[799998.4000064]

6. Write a python program to implement Polynomial Regression for given dataset.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Assume that we have a dataset with two columns: 'area' and 'price',
# where 'area' is the size of the house in square feet and 'price' is the price of the house
df = pd.DataFrame({'area': [1000, 1500, 2000, 2500, 3000],
                   'price': [300000, 400000, 500000, 600000, 700000]})

# Create a Linear Regression model
model = LinearRegression()

# Create a polynomial transformer object with a degree of 2
poly_transformer = PolynomialFeatures(degree=2)

# Transform the 'area' column to polynomial features
X_poly = poly_transformer.fit_transform(df[['area']])

# Fit the model to the transformed data
model.fit(X_poly, df['price'])

# Predict the price of a house with an area of 3500 square feet
prediction = model.predict(poly_transformer.transform([[3500]]))
print(prediction)
```

Output:

[799998.4000064]

7. Write a python program to Implement Naïve Bayes.

```
import numpy as np
from sklearn.naive_bayes import GaussianNB

# Assume that we have a dataset with two features: 'age' and 'income',
# and a target variable: 'purchased'
X = np.array([[20, 50000], [30, 60000], [40, 80000], [50, 100000], [60, 120000]])
y = np.array(['yes', 'no', 'yes', 'no', 'yes'])

# Create a Gaussian Naive Bayes model
model = GaussianNB()

# Fit the model to the training data
model.fit(X, y)

# Predict whether a person with an age of 25 and an income of 55000 will purchase a product
prediction = model.predict([[25, 55000]])
print(prediction)
```

Output:

['yes']

8. Write a python program to Implement Decision Tree whether or not to play tennis.

```
#numpy and pandas initialization
import numpy as np
import pandas

PlayTennis = pandas.read_csv('playtennis.csv')
print(PlayTennis)

from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])

print(PlayTennis)

y = PlayTennis['play']
X = PlayTennis.drop(['play'],axis=1)

# Fitting the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(X, y)

# We can visualize the tree using tree.plot_tree
tree.plot_tree(clf)
```

```
# The predictions are stored in X_pred

X_pred = clf.predict(X)

# verifying if the model has predicted it all right.

X_pred == y
```

Output:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

	outlook	temp	humidity	windy	play
0	2	1	0	0	0
1	2	1	0	1	0
2	0	1	0	0	1
3	1	2	0	0	1
4	1	0	1	0	1
5	1	0	1	1	0
6	0	0	1	1	1
7	2	2	0	0	0
8	2	0	1	0	1
9	1	2	1	0	1
10	2	2	1	1	1
11	0	2	0	1	1
12	0	1	1	0	1
13	1	2	0	1	0

9. Write a python program to implement linear SVM.

```
import numpy as np
from sklearn.svm import LinearSVC

# Assume that we have a dataset with two features: 'age' and 'income',
# and a target variable: 'purchased'
X = np.array([[20, 50000], [30, 60000], [40, 80000], [50, 100000], [60, 120000]])
y = np.array(['yes', 'no', 'yes', 'no', 'yes'])

# Create a Linear SVM model
model = LinearSVC()

# Fit the model to the training data
model.fit(X, y)

# Predict whether a person with an age of 25 and an income of 55000 will purchase a product
prediction = model.predict([[25, 55000]])
print(prediction)
```

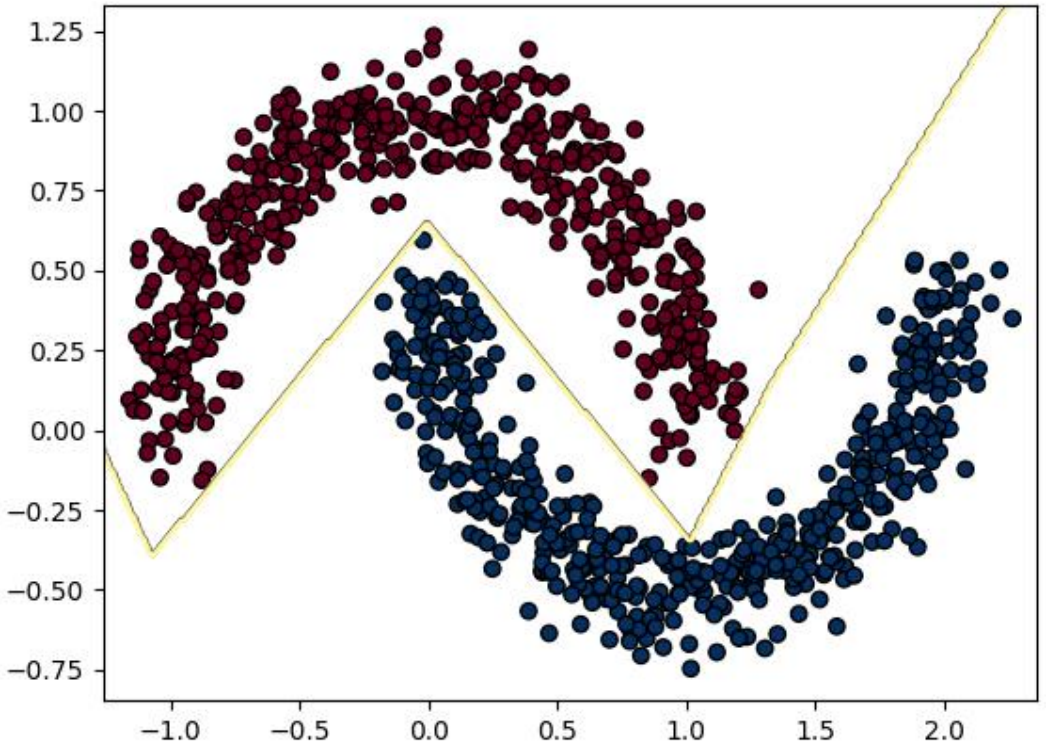
Output:

```
['no']
```

10. Write a python program to find Decision boundary by using a neural network with 10 hidden units on two moons dataset

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.neural_network import MLPClassifier
import numpy as np

# Generate the two moons dataset
X, y = make_moons(n_samples=1000, noise=0.1)

# Create a neural network with 10 hidden units
model = MLPClassifier(hidden_layer_sizes=(10,), solver='lbfgs', random_state=1)

# Fit the model to the training data
model.fit(X, y)

# Plot the decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu, edgecolor='k')

h = 0.01
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired)
plt.show()
```

Output:



11. Write a python program to transform data with Principal Component Analysis (PCA)

```
import numpy as np
from sklearn.decomposition import PCA

# Assume that we have a dataset with three features: 'length', 'width', and 'height'
X = np.array([[2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7]])

# Create a PCA transformer object with a target dimension of 2
pca = PCA(n_components=2)

# Transform the data to the new lower-dimensional space
X_transformed = pca.fit_transform(X)
print(X_transformed)
```

Output:

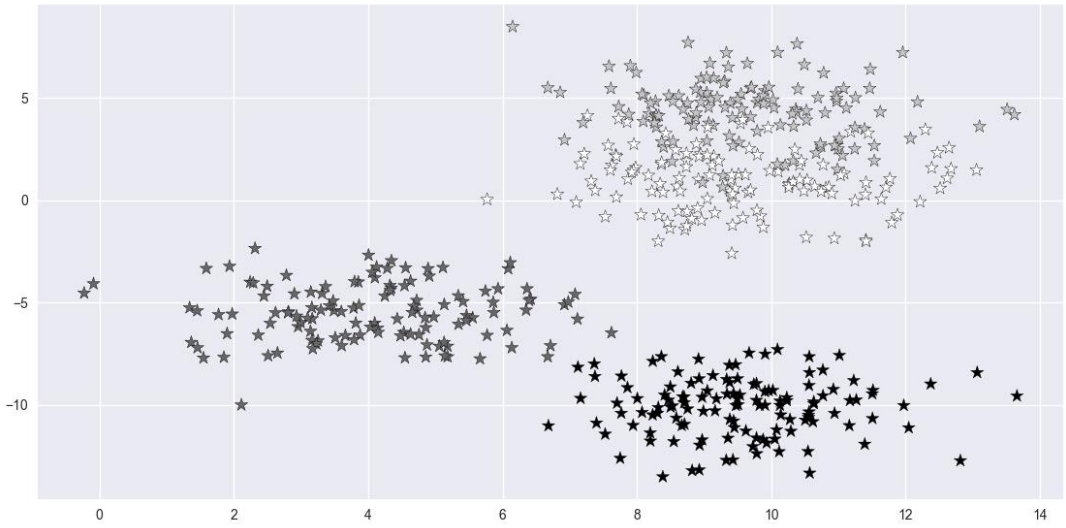
```
[[ 2.59807621  0.        ]
 [ 0.8660254   0.        ]
 [-0.8660254   0.        ]
 [-2.59807621 -0.        ]]
```

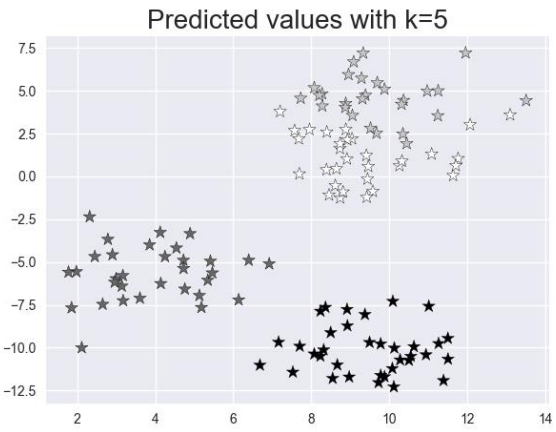
12. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use Forge Dataset)

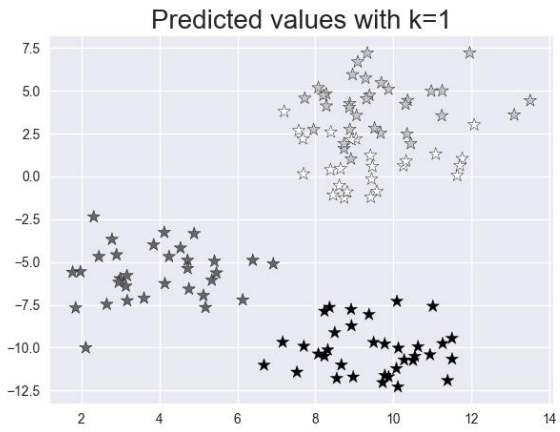
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

X, y = make_blobs(n_samples = 500, n_features = 2, centers = 4,cluster_std = 1.5, random_state = 4)

plt.style.use('seaborn')
plt.figure(figsize = (10,10))
plt.scatter(X[:,0], X[:,1], c=y, marker= '*',s=100,edgecolors='black')
plt.show()

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

knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)

y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)
```

```
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_5, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=5", fontsize=20)

plt.subplot(1,2,2)
plt.scatter(X_test[:,0], X_test[:,1], c=y_pred_1, marker= '*', s=100,edgecolors='black')
plt.title("Predicted values with k=1", fontsize=20)
plt.show()
```

Output:



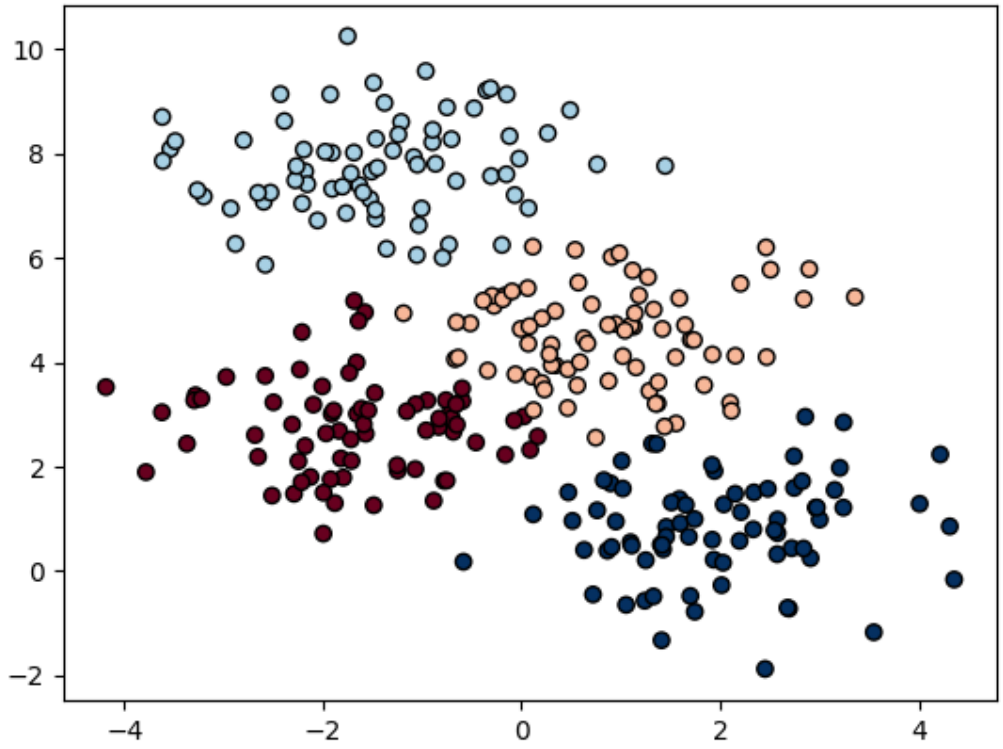

Predicted values with k=5
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
2
4
6
8
10
12
14


Predicted values with k=1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
2
4
6
8
10
12
14

13. Write a python program to implement k-means algorithm on a synthetic dataset.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate the synthetic dataset
X, y = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=1.0)

# Create a KMeans model with 4 clusters
model = KMeans(n_clusters=4)

# Fit the model to the data
model.fit(X)

# Predict the cluster labels for each point
y_pred = model.predict(X)

# Plot the data and the cluster labels
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=plt.cm.RdBu, edgecolor='k')
plt.show()
```

Output:



14. Write a python program to implement Agglomerative clustering on a synthetic dataset.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# Generate the synthetic dataset
X, y = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=1.0)

# Create an Agglomerative Clustering model with 4 clusters
model = AgglomerativeClustering(n_clusters=4)

# Fit the model to the data
model.fit(X)

# Predict the cluster labels for each point
y_pred = model.fit_predict(X)

# Plot the data and the cluster labels
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=plt.cm.RdBu, edgecolor='k')
plt.show()
```

Output: