# K-Means Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

# y = dataset.iloc[:, 3].values

# Splitting the dataset into the Training set and Test set

"""from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)"""

# Feature Scaling

"""from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

sc\_y = StandardScaler()

y\_train = sc\_y.fit\_transform(y\_train)"""

# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

# Fitting K-Means to the dataset

kmeans = KMeans(n\_clusters = 3, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

#plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

#plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

# Naive Bayes

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Naive Bayes to the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn import metrics

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix: ", cm)

print("Accuracy: ", metrics.accuracy\_score(y\_test,y\_pred))

print("Precision: ", metrics.precision\_score(y\_test,y\_pred))

print("Recall: ", metrics.recall\_score(y\_test,y\_pred))

print("F1 Score: ", metrics.f1\_score(y\_test,y\_pred))

print("Report: ", classification\_report(y\_test,y\_pred))

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Polynomial Regression

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:2].values

y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set and Test set

"""from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)"""

# Fitting Linear Regression to the dataset

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

# Fitting Polynomial Regression to the dataset

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 2)

X\_poly = poly\_reg.fit\_transform(X)

poly\_reg.fit(X\_poly, y)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

# Visualising the Linear Regression results

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg.predict(X), color = 'blue')

plt.title('Truth or Bluff (Linear Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

# Visualising the Polynomial Regression results

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue')

plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

# Visualising the Polynomial Regression results (for higher resolution and smoother curve)

X\_grid = np.arange(min(X), max(X), 0.1)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, lin\_reg\_2.predict(poly\_reg.fit\_transform(X\_grid)), color = 'blue')

plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

# Predicting a new result with Linear Regression

lin\_reg.predict([[6.5]])

# Predicting a new result with Polynomial Regression

lin\_reg\_2.predict(poly\_reg.fit\_transform([[6.5]]))

# Multiple Linear Regression

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('50\_Startups.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

# Encoding categorical data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder = LabelEncoder()

X[:, 3] = labelencoder.fit\_transform(X[:, 3])

onehotencoder = OneHotEncoder(categorical\_features = [3])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding the Dummy Variable Trap

X = X[:, 1:]

# Splitting the dataset into the Training set and Test set

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 = 0)

# Fitting Multiple Linear Regression to the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = regressor.predict(X\_test)

plt.scatter3D(X\_train,y\_train)

import seaborn as sn

sn.regplot(X\_train,y\_train)

#Building Optimal Model using Backward Elimination

import statsmodels.formula.api as sm

import statsmodels.regression.linear\_model as smor

X=np.append(arr=np.ones((50,1)), values=X, axis=1)

X\_opt=X[:,[0,1,2,3,4,5]]

regressor\_OLS=sm.OLS(endog=y, exog=X\_opt)

regressor.OLS=regressor\_OLS.fit()

ans=smor.OLSResults(regressor\_OLS,y\_test)

ans.summary()

# Simple Linear Regression

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Salary\_Data.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 1].values

#Handling Missing Data

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

imputer=imputer.fit(X)

X=imputer.transform(X)

y=y.reshape(-1,1)

imputer1 = SimpleImputer(missing\_values=np.nan, strategy='mean')

imputer1=imputer1.fit(y)

y=imputer1.transform(y)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

# Fitting Simple Linear Regression to the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

print("Intercept: ", regressor.intercept\_)

#print("Intercept: %.2f" % regressor.intercept\_)

print("Slope: ",regressor.coef\_)

print("Model is: y = ",regressor.intercept\_," + ",regressor.coef\_," \* x")

# Predicting the Test set results

y\_pred = regressor.predict(X\_test)

from sklearn import metrics

print("R-Squared: ",regressor.score(X,y))

print("Mean Absolute Error: ",metrics.mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared Error: ",metrics.mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared Error: ",np.sqrt(metrics.mean\_absolute\_error(y\_test,y\_pred)))

# Visualising the Training set results

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

# Visualising the Test set results

plt.scatter(X\_test, y\_test, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()