Multiple linear regression

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 4]
states=pd.get_dummies(X['State'],drop_first=True)
X=X.drop('State',axis=1)
X=pd.concat([states,X],axis=1)
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)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
from sklearn.metrics import r2_score
score=r2_score(y_test,y_pred)
print('Accuracy R2 Score',score)
import statsmodels.api as sm
X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
regressor_OLS = sm.OLS( y, X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS( y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3, 4, 5]]
regressor_OLS = sm.OLS( y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3, 5]]
regressor_OLS = sm.OLS(y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
print(regressor_OLS.summary())
```

Apriori Algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Market_Basket_Optimisation.csv', low_memory=False,
header=None)
!pip install apyori
list_of_transactions = []
for i in range(0, 7501):
    list_of_transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
list_of_transactions[0]
from apyori import apriori
rules = apriori(list_of_transactions, min_support = 0.004, min_confidence = 0.2,
min_lift = 3, min_length = 2)
results = list(rules)
def inspect(results):
    lhs         =  [tuple(result [2] [0] [0]) [0] for result in results]
    rhs         =  [tuple(result [2] [0] [1]) [0] for result in results]
    supports = [result [1] for result in results]
    confidences = [result [2] [0] [2]   for result in results]
    lifts = [result [2] [0] [3]   for result in results]
    return list(zip(lhs,rhs,supports,confidences, lifts))
resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Hand Side',
'Right Hand Side', 'Support', 'Confidence', 'Lift'] )
resultsinDataFrame.head(10)
```

K-Means Clustering Algorithm

```
from sklearn.datasets import load_iris
from itertools import cycle
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from numpy.random import RandomState
import pylab as pl
import matplotlib.pyplot as plt
class clustering:
    def __init__(self):
        self.plot(load_iris().data)
    def plot(self, X):
        wcss=[]
        for i in range(1,11):

kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state
=0)
            kmeans.fit(X)
            wcss.append(kmeans.inertia_)
        plt.plot(range(1,11),wcss)
        plt.title('Elbow Method')
        plt.xlabel('Number of Clusters')
        plt.ylabel('WCSS')
        plt.show()
        pca = PCA(n_components=2, whiten=True).fit(X)
        X_pca = pca.transform(X)
        kmeans = KMeans(n_clusters=3, random_state=RandomState(42)).fit(X_pca)
        plot_2D(X_pca, kmeans.labels_, ["c0", "c1", "c2"])
def plot_2D(data, target, target_names):
    colors = cycle('rgbcmykw')
    target_ids = range(len(target_names))
    pl.figure()
    for i, c, label in zip(target_ids, colors, target_names):
        pl.scatter(data[target == i, 0], data[target == i, 1],
                                        c=c, label=label)
    pl.legend()
    pl.show()
if __name__ == '__main__':
        c = clustering()
```

K Nearest Neighbour algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
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)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix of KNN \n",cm)
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('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
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('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
score = classifier.score(X_test, y_test)
print('Accuracy Score is: ',score)
```

Hierarchical Clustering Algorithm

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.axhline(y=200, color='r', linestyle='--')
plt.show()
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Random Forest Classification

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
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)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
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('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
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('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

CART algorithm

```
from sklearn import tree
from sklearn.datasets import load_iris
iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iriscart")
dot_data = tree.export_graphviz(clf, 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.view()
```

Logistic regression using sigmoid function

```python
import numpy as np
import pandas as pd

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
  # Load data
  dataset = pd.read_csv('datasets/Social_Network_Ads.csv')

  # Split data into features and labels
  features = dataset.iloc[:, [2, 3]].values # all rows of columns 2 and 3 (2D array) [age, salary]
  print(features)
  labels = dataset.iloc[:, 4].values # all rows of column 4 (1D array) [purchased] (0, 1)

  # Split data into training and testing sets
  x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.25, random_state=0)

  # Scale features
  scaler = preprocessing.StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Train model
  classifier = LogisticRegression(random_state=0)
  classifier.fit(x_train, y_train)

  # Predict
  y_pred = classifier.predict(x_test)

  # Evaluate
  cm = confusion_matrix(y_test, y_pred) # compare y_test and y_pred
  print(cm)

```

```
39
40      # Visualize
41      def visualize(x_label, y_label, title, features, labels):
42          col_zero = features[:, 0] # all rows, column 0
43          col_one = features[:, 1] # all rows, column 1
44
45          start1 = col_zero.min() - 1
46          stop1 = col_zero.max() + 1
47
48          start2 = col_one.min() - 1
49          stop2 = col_one.max() + 1
50
51          xi = np.arange(start1, stop1, 0.01)
52          yi = np.arange(start2, stop2, 0.01)
53
54          x, y = np.meshgrid(xi, yi) # create meshgrid
55
56
57          predict_data = np.array([x.ravel(), y.ravel()]).transpose() # transpose to get pairs
58          y_pred = classifier.predict(predict_data).reshape(x.shape) # reshape to get matrix
59          plt.contourf(x, y, y_pred, alpha=0.75, cmap=ListedColormap(('red', 'green'))) # plot contour
60
61          plt.xlim(x.min(), x.max()) # set limits
62          plt.ylim(y.min(), y.max()) # set limits
63
64
65          unique_labels = np.unique(labels) # get unique labels
66          for index, value in enumerate(unique_labels): # plot points
67
68              #Two iterations:
69              #first iteration: index = 0, value = 0
70              #second iteration: index = 1, value = 1
71
72              scatter_x = features[labels == value, 0] # get x values - all rows where label is equal to value - col 0
73              scatter_y = features[labels == value, 1] # get y values - all rows where label is equal to value - col 1
74
75              #get color based on index
76              #first iteration: index = 0, color = red
77              #second iteration: index = 1, color = green
78              color = ListedColormap(('orange', 'blue'))(index)
79              plt.scatter(scatter_x, scatter_y, color=color, label=value)
80
81          plt.title(title)
82          plt.xlabel(x_label)
83          plt.ylabel(y_label)
84          plt.legend()
85          plt.show()
86
87
88
89      visualize('Age', 'Estimated Salary', 'Logistic Regression (Training set)', x_train.copy(), y_train.copy())
90      visualize('Age', 'Estimated Salary', 'Logistic Regression (Test set)', x_test.copy(), y_test.copy())
91
92  except Exception as e:
93      print(e)
94
```

Support vector machine algorithm using various kernel functions.

```
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 Kernel SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
#classifier = SVC(kernel = 'linear', random_state = 0)
#from sklearn.svm import LinearSVC
#classifier = LinearSVC( random_state = 0)
#classifier = SVC(kernel = 'poly', random_state = 0)
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
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix of SVM \n",cm)
# 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('Kernel SVM (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('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
score = classifier.score(X_test, y_test)
print('Accuracy Score is: ',score)
```

Logistic regression using sigmoid function:

```
import numpy as np
import pandas as pd

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
  # Load data
  dataset = pd.read_csv('datasets/Social_Network_Ads.csv')

  # Split data into features and labels
  features = dataset.iloc[:, [2, 3]].values # all rows of columns 2 and 3 (2D array) [age, salary]
  print(features)
  labels = dataset.iloc[:, 4].values # all rows of column 4 (1D array) [purchased] (0, 1)

  # Split data into training and testing sets
  x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.25, random_state=0)

  # Scale features
  scaler = preprocessing.StandardScaler()
  x_train = scaler.fit_transform(x_train)
  x_test = scaler.transform(x_test)

  # Train model
  classifier = LogisticRegression(random_state=0)
  classifier.fit(x_train, y_train)

  # Predict
  y_pred = classifier.predict(x_test)

  # Evaluate
  cm = confusion_matrix(y_test, y_pred) # compare y_test and y_pred
  print(cm)

  # Visualize
  def visualize(x_label, y_label, title, features, labels):
    col_zero = features[:, 0] # all rows, column 0
    col_one = features[:, 1] # all rows, column 1

    start1 = col_zero.min() - 1
    stop1 = col_zero.max() + 1

    start2 = col_one.min() - 1
    stop2 = col_one.max() + 1

    xi = np.arange(start1, stop1, 0.01)
    yi = np.arange(start2, stop2, 0.01)

    x, y = np.meshgrid(xi, yi) # create meshgrid
```

```
    predict_data = np.array([x.ravel(), y.ravel()]).transpose() # transpose to get pairs
    y_pred = classifier.predict(predict_data).reshape(x.shape) # reshape to get matrix
    plt.contourf(x, y, y_pred, alpha=0.75, cmap=ListedColormap(('red', 'green'))) # plot contour

    plt.xlim(x.min(), x.max()) # set limits
    plt.ylim(y.min(), y.max()) # set limits


    unique_labels = np.unique(labels) # get unique labels
    for index, value in enumerate(unique_labels): # plot points

      #Two iterations:
      #first iteration: index = 0, value = 0
      #second iteration: index = 1, value = 1

      scatter_x = features[labels == value, 0] # get x values - all rows where label is equal to value - col
0
      scatter_y = features[labels == value, 1] # get y values - all rows where label is equal to value - col
1

      #get color based on index
      #first iteration: index = 0, color = red
      #second iteration: index = 1, color = green
      color = ListedColormap(('orange', 'blue'))(index)
      plt.scatter(scatter_x, scatter_y, color=color, label=value)

    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.legend()
    plt.show()



  visualize('Age', 'Estimated Salary', 'Logistic Regression (Training set)', x_train.copy(), y_train.copy())
  visualize('Age', 'Estimated Salary', 'Logistic Regression (Test set)', x_test.copy(), y_test.copy())

except Exception as e:
  print(e)
```