```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.metrics import accuracy_score

#importing datasets
data_set= pd.read_csv('/content/Social_Network_Ads.csv')
print(data_set.describe())

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
#k=data_set.iloc[0,[2,3]].values
#print(k)
# Splitting the dataset into training 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
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

#Fitting K-NN classifier to the training set
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)
print(y_test==y_pred)
print(accuracy_score(y_test,y_pred))
#print(y_pred)
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print(cm)

newpred = classifier.predict(st_x.transform([[19,18000]]))
print(newpred)

from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
```

```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Naïve bayes:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
y = dataset.iloc[:, 4].values
print(dataset.columns)
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
dataset.describe
dataset.shape
from sklearn.model_selection import cross_val_score
```

```
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)
```

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
 from sklearn.metrics import accuracy_score

print(y_test==y_pred)
print(accuracy_score(y_test,y_pred))
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(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('Naive Bayes (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('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

SVM linear

```
from sklearn.svm import LinearSVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
iris = datasets.load_iris()
features = iris.data[:100,:2]
target = iris.target[:100]
features
```

```
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
features_standardized
```

```
# Create support vector classifier
svc = LinearSVC(C=1.0)
model = svc.fit(features_standardized, target)
```

```
from matplotlib import pyplot as plt
color = ["black" if c == 0 else "red" for c in target]
#print(color)
plt.scatter(features_standardized[:,0],features_standardized[:,1],
c=color)
w = svc.coef_[0]
print(w)
```

```
print(svc.intercept_[0])
a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
print(xx)
yy = a * xx - (svc.intercept_[0]) / w[1]
print(yy)
plt.plot(xx, yy)
plt.axis("off"), plt.show();
```

```
new_observation = [[ -2, 3]]
```

```
svc.predict(new_observation)
```

SVM non linear:

```
# Handling Linearly Inseparable Classes Using Kernels
# Importing Libraries
from sklearn.svm import SVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
```

```
# Set randomization seed
np.random.seed(0)
# Generate two features
features = np.random.randn(200, 2)
print(features)
# Use a XOR gate to generate linearly inseparable classes
target_xor = np.logical_xor(features[:, 0] > 0, features[:, 1] > 0)
print("XOR\n",target_xor)
target = np.where(target_xor, 0, 1)
print(target)
```

```
# Create a support vector machine with a radial basis function kernel
svc = SVC(kernel="rbf", random_state=0, gamma=1, C=1)
# Train the classifier
model = svc.fit(features, target)
```

```
#Plot observations and decision boundary hyperplane
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
```

```
def plot_decision_regions(X, y, classifier):
 cmap = ListedColormap(("red", "blue"))
 xx1, xx2 = np.meshgrid(np.arange(-3, 3, 0.02), np.arange(-3, 3, 0.02))
 Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
 Z = Z.reshape(xx1.shape)
 plt.contourf(xx1, xx2, Z, alpha=0.1, cmap=cmap)
 for idx, cl in enumerate(np.unique(y)):
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
marker="+", label=cl)
```

```
# Create support vector classifier with a linear kernel
svc_linear = SVC(kernel="linear", random_state=0, C=1)
```

```
# Train model
svc_linear.fit(features, target)
```

```
# Plot observations and hyperplane
plot_decision_regions(features, target, classifier=svc_linear)
plt.axis("off")
plt.show()
```

```
# Create a support vector machine with a radial basis function kernel
svc = SVC(kernel="rbf", random_state=0, gamma=1, C=1)
```

```
# Train the classifier
model = svc.fit(features, target)
```

```
plot_decision_regions(features, target, classifier=svc)
plt.axis("off")
plt.show()
```

PCA:

```
from sklearn.decomposition import PCA
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
digits = datasets.load_digits()
data = digits.data
import pandas as pd
df = pd.DataFrame(data)
df.head()
std = StandardScaler()
f_std = std.fit_transform(data)
```

```
pca = PCA(n_components=0.99, whiten=True)
f_reduced = pca.fit_transform(f_std)
```

```
print("Original number of features:", f_std.shape[1])
print("Reduced number of features:", f_reduced.shape[1])
```

```
from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles
```

```
features, _ = make_circles(n_samples=1000, random_state=1, noise=0.1,
factor=0.1)
print(features.shape)
kpca = KernelPCA(kernel="rbf", gamma=15, n_components=1)
features_kpca = kpca.fit_transform(features)
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_kpca.shape[1])
```

using kermel:

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

```
ds = pd.read_csv("/content/Social_Network_Ads.csv")
inp = ds.iloc[:,2:4].values
tgt = ds.iloc[:, 4].values
```

```
from sklearn.model_selection import train_test_split
inp_train, inp_test, tgt_train, tgt_test = train_test_split(inp, tgt,
test_size = 0.25, random_state = 0)
```

```
from sklearn.model_selection import train_test_split
inp_train, inp_test, tgt_train, tgt_test = train_test_split(inp, tgt,
test_size = 0.25, random_state = 0)
```

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 1,kernel = "rbf")
kpca_inp_train = kpca.fit_transform(inp_train)
kpca_inp_test = kpca.transform(inp_test)
print("Original number of features:", inp_train.shape[1])
print("Reduced number of features:", kpca_inp_train.shape[1])

kpca_inp_train
```