```
################################
### general features
################################


# read csv file as dataframe
import pandas as pd
cols = ["fLength", "fWidth", "fSize", "fConc", "fConc1", "fAsym", "fM3Long", "fM3Trans",
"fAlpha", "fDist", "class"]
df = pd.read_csv("magic04.data", names=cols)

# show start of dataframe
df.head()

# drop some columns from dataframe
df = df.drop(["wind", "visibility", "functional"], axis=1)

# change target feature to int
df["class"] = (df["class"] == "g").astype(int)


# draw histogram to all features
for label in cols[:-1]:
  plt.hist(df[df["class"]==1][label], color='blue', label='gamma', alpha=0.7,
density=True)
  plt.hist(df[df["class"]==0][label], color='red', label='hadron', alpha=0.7,
density=True)
  plt.title(label)
  plt.ylabel("Probability")
  plt.xlabel(label)
  plt.legend()
  plt.show()

# show classification report
from sklearn.metrics import classification_report
classification_report(y_test, y_pred)

# split data with np split
train, valid, test = np.split(df.sample(frac=1), [int(0.6*len(df)), int(0.8*len(df))])

# normalize and oversample
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
def scale_dataset(dataframe, oversample=False):
  X = dataframe[dataframe.columns[:-1]].values
  y = dataframe[dataframe.columns[-1]].values

  scaler = StandardScaler()
  X = scaler.fit_transform(X)
```

```
  if oversample:
    ros = RandomOverSampler()
    X, y = ros.fit_resample(X, y)

  data = np.hstack((X, np.reshape(y, (-1, 1))))

  return data, X,

# draw graphs for relations between all features
for i in range(len(cols)-1):
  for j in range(i+1, len(cols)-1):
    x_label = cols[i]
    y_label = cols[j]
    sns.scatterplot(x=x_label, y=y_label, data=df, hue='class')
    plt.show()

# scatter used for datapoints and plot for lines
plt.scatter(transformed_x[:,0], transformed_x[:,1])
plt.show()

################################
### KNN --> KNeighborsClassifier
################################
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)

################################
### Naive Bayes --> GaussianNB
################################
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model = nb_model.fit(X_train, y_train)
y_pred = nb_model.predict(X_test)

################################
### Logistic Regression --> LogisticRegression
################################
from sklearn.linear_model import LogisticRegression
lg_model = LogisticRegression()
lg_model = lg_model.fit(X_train, y_train)
y_pred = lg_model.predict(X_test)
```

```
################################
### SVM --> SVC
################################
from sklearn.svm import SVC
svm_model = SVC()
svm_model = svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)



################################
### Neural Netowrk
################################

import tensorflow as tf

def plot_history(history):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
  ax1.plot(history.history['loss'], label='loss')
  ax1.plot(history.history['val_loss'], label='val_loss')
  ax1.set_xlabel('Epoch')
  ax1.set_ylabel('Binary crossentropy')
  ax1.grid(True)
  ax2.plot(history.history['accuracy'], label='accuracy')
  ax2.plot(history.history['val_accuracy'], label='val_accuracy')
  ax2.set_xlabel('Epoch')
  ax2.set_ylabel('Accuracy')
  ax2.grid(True)

  plt.show()


def train_model(X_train, y_train, num_nodes, dropout_prob, lr, batch_size, epochs):
  nn_model = tf.keras.Sequential([
      tf.keras.layers.Dense(num_nodes, activation='relu', input_shape=(10,)),
      tf.keras.layers.Dropout(dropout_prob),
      tf.keras.layers.Dense(num_nodes, activation='relu'),
      tf.keras.layers.Dropout(dropout_prob),
      tf.keras.layers.Dense(1, activation='sigmoid')
  ])
  nn_model.compile(optimizer=tf.keras.optimizers.Adam(lr), loss='binary_crossentropy',
                   metrics=['accuracy'])
  history = nn_model.fit(
    X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2,
verbose=0
  )

  return nn_model, history
y_pred = least_loss_model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int).reshape(-1,)
```

```
################################
### KMeans --> KMeans
################################
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3).fit(X)

################################
### PCA --> PCA
################################
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
transformed_x = pca.fit_transform(X)


################################
### Linear Regression --> LinearRegression
################################
from sklearn.linear_model import LinearRegression
temp_reg = LinearRegression()
temp_reg.fit(X_train_temp, y_train_temp)
temp_reg.score(X_test_temp, y_test_temp)


# plot points and regression line
plt.scatter(X_train_temp, y_train_temp, label="Data", color="blue")
x = tf.linspace(-20, 40, 100)
plt.plot(x, temp_reg.predict(np.array(x).reshape(-1, 1)), label="Fit", color="red",
linewidth=3)
plt.legend()
plt.title("Bikes vs Temp")
plt.ylabel("Number of bikes")
plt.xlabel("Temp")
plt.show()

################################
###  Regression with Neural Net
################################
temp_normalizer = tf.keras.layers.Normalization(input_shape=(1,), axis=None)
temp_normalizer.adapt(X_train_temp.reshape(-1))
temp_nn_model = tf.keras.Sequential([
    temp_normalizer,
    tf.keras.layers.Dense(1)
])

temp_nn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
loss='mean_squared_error')
history = temp_nn_model.fit(
    X_train_temp.reshape(-1), y_train_temp,
    verbose=0, epochs=1000, validation_data=(X_val_temp, y_val_temp))
plot_loss(history)
```