1. Linear regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model, metrics
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X=california_housing.data # Features (X)
y =california_housing.target # Target (y)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state=1)

reg=linear_model.LinearRegression()
reg.fit(X_train,y_train)
print('Coefficients:',reg.coef_)
print('Variance score: {}'.format(reg.score(X_test,y_test)))

plt.scatter(reg.predict(X_test),reg.predict(X_test)-y_test,color="blue",s = 10,label = "Test data")
plt.scatter(reg.predict(X_train),reg.predict(X_train)-y_train,color="green",s = 10,label = "Train data")

#plt.hlines(y = 0,xmin = 0,xmax = 50,linewidth = 2)
#plt.legend(loc = "upper right")
plt.xlabel('x-axis', fontsize=20)
plt.ylabel('y-axis', fontsize=20)

plt.title("Residual errors")

plt.grid()

plt.show()
```

2. Multiple linear regression

```
# importing modules and packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn import preprocessing
# importing data
df = pd.read_csv('/content/Real-estate1.csv')
df.drop('No', inplace=True, axis=1)
print(df.head())
print(df.columns)
# plotting a scatterplot
sns.scatterplot(x='X4 number of convenience stores',
```

```
y='Y house price of unit area', data=df)
# creating feature variables
X = df.drop('Y house price of unit area', axis=1)
y = df['Y house price of unit area']
print(X)
print(y)
# creating train and test sets
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=101)
# creating a regression model
model = LinearRegression()
# fitting the model
model.fit(X_train, y_train)
# making predictions
predictions = model.predict(X_test)
# model evaluation
print('mean_squared_error : ', mean_squared_error(y_test, predictions))
print('mean_absolute_error : ', mean_absolute_error(y_test, predictions))
```

3. Logistic regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import linear_model, metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Load your dataset
# Replace 'your_dataset.csv' with the path to your CSV
file df = pd.read_csv('/content/home (1).csv')
# Assume 'target' is the column with labels and the rest are
features X = df.drop(columns='Area_Square_Feet')
y = df['Area_Square_Feet']
# Split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)
# Create logistic regression object
reg = linear_model.LogisticRegression(max_iter=200)
# Train the model using the training sets
reg.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = reg.predict(X_test)
# Compare actual response values (y_test) with predicted response
values (y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred) * 100
print("Logistic Regression model accuracy (in %):",
accuracy) # Print confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cf_matrix)
# Print confusion matrix using Seaborn library
sns.heatmap(cf_matrix, annot=True, fmt='d',
cmap='Blues') plt.xlabel('Predicted Label')
```

```
plt.ylabel('True Label')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

4. Bagging

```
from google.colab import files
upload =files.upload()
import pandas as pd
df = pd.read_csv('diabetes.csv')
df.head()


df.isnull().sum()


df.describe()


df.Outcome.value_counts()


x = df.drop("Outcome",axis = "columns")
y = df.Outcome
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled[:3]


from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y,stratify = y,random_state = 10)
X_train.shape
(576, 8)
X_test.shape
(192, 8)
y_train.value_counts()


201/375


y_test.value_counts()
```

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
scores = cross_val_score(DecisionTreeClassifier(),x,y,cv = 5)
Scores
array([0.70779221, 0.67532468, 0.68181818, 0.79738562, 0.73202614])
scores.mean()

from sklearn.ensemble import RandomForestClassifier
scores = cross_val_score(RandomForestClassifier(n_estimators = 40),x,y,cv = 5)
scores.mean()

from sklearn.ensemble import BaggingClassifier
bag_model = BaggingClassifier(
base_estimator = DecisionTreeClassifier(),
n_estimators = 100,
max_samples = 0.8,
oob_score = True,
random_state = 0
)
bag_model.fit(x_train,y_train)
bag_model.oob_score_

bag_model.fit(x_train,y_train)
bag_model.oob_score_

from sklearn import metrics
from sklearn.metrics import accuracy_score
# Assuming 'bag_model' is your trained model
y_pred = bag_model.predict(x_test) # Generate predictions
# Nowcalculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```
print(accuracy)

bag_model.score(x_test,y_test)
bag_model.oob_score_

from sklearn import metrics
from sklearn.metrics import accuracy_score
metrics.accuracy_score(y_test,y_pred)

bag_model = BaggingClassifier(
base_estimator = DecisionTreeClassifier(),
n_estimators = 100,
max_samples = 0.8,
oob_score = True,
random_state = 0
)
scores = cross_val_score(bag_model,x,y,cv = 5)
scores

scores.mean()

from sklearn.metrics import classification_report
#y_pred = bag_model.predict(x_test)
print(classification_report(y_test,y_pred))

from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat,square = True,annot = True,fmt = 'd',cbar = False)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

5. Support vector machine

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
diabetes_data = pd.read_csv( '/content/diabetes (1).csv' )
# Explore the dataset (optional)
print (diabetes_data.head())
print (diabetes_data.info())
# Assuming the last column is the target variable (0 for non-diabetic, 1
for diabetic)
X = diabetes_data.iloc[:, : -1 ].values
y = diabetes_data.iloc[:, -1 ].values
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2 ,
random_state= 42 )
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create an instance of the SVM classifier
clf = SVC(kernel= 'linear' )
# Train the SVM classifier
clf.fit(X_train, y_train)
# Make predictions on the test set
y_pred = clf.predict(X_test)
# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print ( f 'Accuracy: {accuracy :.2f } ' )
# Visualize decision boundary (Modified)
def plot_decision_boundary ( clf , X , y ):
# Note: This visualization is simplified for 2D.
```

```
# For higher dimensions, techniques like PCA or  t-SNE are needed.
# The following line is changed to use all features  of X_test
Z = clf.predict(X)
# The rest of the plotting code remains the same,
# but it will now use predictions based on all  features.
plt.scatter(X[:,  0  ], X[:,  1  ], c=Z, edgecolors=  'k'  ,  marker=  'o'  )
plt.title(  'SVM Decision Regions (First Two Features)'  )
plt.xlabel(  'Feature 1'  )
plt.ylabel(  'Feature 2'  )
plt.show()
# Plot the decision boundary (Using all features of X_test)
plot_decision_boundary(clf, X_test, y_test)
```

6. Principle component Analysis

```
from sklearn.datasets import load_wine
winedata = load_wine()
X, y = winedata['data'], winedata['target']
print(X.shape)
print(y.shape)

import matplotlib.pyplot as plt
plt.scatter(X[:,1], X[:,2], c=y)
plt.show()

from sklearn.decomposition import PCA
pca = PCA()
Xt = pca.fit_transform(X)
plot = plt.scatter(Xt[:,0], Xt[:,1], c=y)
plt.legend(labels=list(winedata['target_names']))
plt.show()

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pca = PCA()
pipe = Pipeline([('scaler', StandardScaler()), ('pca', pca)])
plt.figure(figsize=(8,6))
Xt = pipe.fit_transform(X)
plot = plt.scatter(Xt[:,0], Xt[:,1], c=y)
plt.legend(labels=list(winedata['target_names']))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("First two principal components after scaling")
plt.show()
```

7. Singular value decomposition

```
import numpy as np
# Function to compute SVD and reconstruct the matrix
def svd_decomposition(matrix):
# Compute SVD
U, S, Vt = np.linalg.svd(matrix, full_matrices=False)  # Use reduced SVD
# Create the diagonal matrix of singular values
Sigma = np.diag(S)  # No need to manually create a zero matrix
# Reconstruct the original matrix using U, Sigma, and Vt
A_reconstructed = np.dot(U, np.dot(Sigma, Vt))
return U, S, Vt, A_reconstructed
# Main program
if __name__ == "__main__":
# Input matrix dimensions
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
# Input matrix elements
print(f"Enter the elements of the {rows}x{cols} matrix row by row:")
matrix = []
for i in range(rows):
row = list(map(float, input(f"Enter row {i+1}: ").split()))
matrix.append(row)
# Convert the list to a NumPy array
matrix = np.array(matrix)
# Perform SVD
U, S, Vt, A_reconstructed = svd_decomposition(matrix)
# Output results
print("\nInput Matrix:")
print(matrix)
print("\nU matrix (Left singular vectors):")
print(U)
print("\nSingular values (Diagonal of Σ):")
print(S)
print("\nVt matrix (Transpose of Right singular vectors):")
print(Vt)
print("\nReconstructed Matrix (Using U, Σ, Vt):")
print(A_reconstructed)
```