1

```
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import datasets

cancer= datasets.load_breast_cancer()
X = cancer.data
y = cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues')
```

2

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

def objective_function(x):
    return x**2 + 2*x +1

result = minimize_scalar(objective_function)
optima = result.x

x = np.linspace(-10, 10, 100)
y = objective_function(x)

plt.plot(x, y)
plt.scatter(optima, objective_function(optima), color='red', label='Optima')
plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Optimizing Unconstrained Convex Univariate Function')
plt.show()
```

```
print(f"Optima: ({int(optima)},{int(objective_function(optima))})")
```

3

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def objective(x):
    return x[0]**2 + x[1]**2

def constraint(x):
    return x[0] + x[1] - 1

x0 = np.array([0.5, 0.5])
result = minimize(objective, x0, method='SLSQP', constraints={'type': 'eq', 'fun': constraint})

print("Minimum found at:")
print("x:", result.x[0])
print("y:", result.x[1])
print("Objective value:", result.fun)

x1_grid, x2_grid = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
Z = objective([x1_grid, x2_grid])
x_values = np.linspace(0, 1, 100)
y_values = 1 - x_values

plt.contourf(x1_grid, x2_grid, Z, levels=20, cmap='viridis')
plt.plot(x_values, y_values, label='Constraint: x + y = 1', color='red')
plt.scatter(result.x[0], result.x[1], color='black', marker='x', s=100, label='Minimum')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Optimization with Constraint and Objective Function')
plt.colorbar(label='Objective Function Value')
plt.legend()
plt.grid(True)
plt.show()
```

4

```
import numpy as np
from sklearn.metrics import mean_squared_error

def user_based_cf(user_item_matrix, target_user):
    similarities = np.dot(user_item_matrix, user_item_matrix[target_user]) / (
        np.linalg.norm(user_item_matrix, axis=1) * np.linalg.norm(user_item_matrix[target_user]))
    top_items = np.argsort(similarities)[::-1][:3]
    recommended_items = [item for item in top_items if user_item_matrix[target_user, item] == 0]
    unrated_items = np.where(user_item_matrix[target_user] == 0)[0]
    predicted_ratings = np.sum(similarities[:, np.newaxis] * user_item_matrix[:, unrated_items],
axis=0)
    actual_ratings = user_item_matrix[target_user, unrated_items]
    rmse = mean_squared_error(actual_ratings, predicted_ratings, squared=False)
    return recommended_items, rmse

user_item_matrix = np.array([[5, 4, 0],
                             [4, 0, 5],
                             [0, 2, 3]])
target_user = 0
recommended_items, rmse = user_based_cf(user_item_matrix, target_user)
print("Recommended items for user", target_user, ":", recommended_items)
print("RMSE:", rmse)
```

———

```
import numpy as np
import math

def user_based_collaborative_filtering(data):
    user_means = np.mean(data, axis=1)
    centered_data = data - user_means[:, np.newaxis]
    similarity = np.dot(centered_data, centered_data.T) / (np.linalg.norm(centered_data,
axis=1)[:, np.newaxis] * np.linalg.norm(centered_data.T, axis=0))
    np.fill_diagonal(similarity, 0)
    predicted_ratings = user_means[:, np.newaxis] + np.dot(similarity, centered_data) /
np.sum(np.abs(similarity), axis=1)[:, np.newaxis]
    mse = np.mean((data - predicted_ratings) ** 2)
    return predicted_ratings, mse

data = np.array([[5, 3, 0, 1],
                 [4, 0, 0, 1],
                 [1, 1, 0, 5],
```

```
            [1, 0, 0, 4],
            [0, 1, 5, 4]])

predicted_ratings, mse = user_based_collaborative_filtering(data)
pred=list(predicted_ratings)

new=[]
for i in range(data.shape[0]):
    new.append([])
    for j in range(data.shape[1]):
        if data[i,j] == 0:
            replace_idx = i % len(pred)
            f=pred[replace_idx]
            new[i].append(abs(round(f[j],2)))
        else:
            new[i].append(data[i][j])

print("Predicted Ratings:")
for i in new:
    print(i)

print("Mean Squared Error (MSE):", mse)
```

5

```
!pip install gensim
!pip install nltk
import nltk
nltk.download('punkt')

from gensim.models import Word2Vec, Doc2Vec
from gensim.models.doc2vec import TaggedDocument
from sklearn.metrics.pairwise import cosine_similarity
from nltk import word_tokenize

document1 = "This is the first document."
document4 = "Is this the first document?"

tokenized_doc1 = word_tokenize(document1.lower())
tokenized_doc4 = word_tokenize(document4.lower())

word2vec_model = Word2Vec([tokenized_doc1,tokenized_doc4], min_count=1)
```

```
tagged_documents = [TaggedDocument(words=word_tokenize(doc.lower()), tags=[str(i)]) for i,
doc in enumerate([document1, document4])]
doc2vec_model = Doc2Vec(tagged_documents, vector_size=100, min_count=1, epochs=40)

word_embedding1 = word2vec_model.wv[tokenized_doc1]
word_embedding4 = word2vec_model.wv[tokenized_doc4]

doc_embedding1 = doc2vec_model.infer_vector(tokenized_doc1)
doc_embedding4 = doc2vec_model.infer_vector(tokenized_doc4)

cosine_similarity_word = cosine_similarity([word_embedding1[:,0]], [word_embedding4[:,0]])
cosine_similarity_doc = cosine_similarity([doc_embedding1], [doc_embedding4])
jaccard_similarity = len(set(tokenized_doc1).intersection(tokenized_doc4)) /
len(set(tokenized_doc1).union(tokenized_doc4))

print("Cosine Similarity (Word Embedding):", cosine_similarity_word[0][0])
print("Cosine Similarity (Doc Embedding):", cosine_similarity_doc[0][0])
print("Jaccard Similarity:", jaccard_similarity)
```