Assignment -1 - 23B2233 - Dev Suthar

June 10, 2025

0.1 Week 1 - 23B2233

Q.1.

[1]:
```
import numpy as np

wizzle = np.random.randint(1, 51, size=(5, 4))
print(wizzle)

anti_diag = [wizzle[i, -i-1] for i in range(min(wizzle.shape))]
print(anti_diag)

max_row_vals = np.max(wizzle, axis=1)
print(max_row_vals)

mean_val = np.mean(wizzle)
less_equal_mean = wizzle[wizzle <= mean_val]
print(less_equal_mean)

def numpy_boundary_traversal(matrix):
    top = matrix[0, :].tolist()
    right = matrix[1:-1, -1].tolist()
    bottom = matrix[-1, ::-1].tolist()
    left = matrix[-2:0:-1, 0].tolist()
    return top + right + bottom + left

boundary = numpy_boundary_traversal(wizzle)
print(boundary)
```

```
[[42 16 30 23]
 [47  8 23 13]
 [18 41 19  1]
 [49 10  9 19]
 [17 13 18 21]]
[23, 23, 41, 49]
[42 47 41 49 21]
[16  8 13 18 19  1 10  9 19 17 13 18 21]
[42, 16, 30, 23, 13, 1, 19, 21, 18, 13, 17, 49, 18, 47]
```

0.1.1 Problem 1 Solution

We started by creating a 2D NumPy array filled with random integers between 1 and 50. The anti-diagonal elements were extracted by selecting elements that run from the top-right to bottom-left of the matrix.

Next, for each row in the matrix, we computed the maximum value.

Then, we constructed a new array containing only those elements that are less than or equal to the overall mean of the matrix.

Lastly, we defined a function `numpy_boundary_traversal(matrix)` that returns the boundary elements of the matrix in a clockwise fashion. This was done by combining slices from each side of the matrix to achieve the correct traversal order.

Q.2.

[2]:
```python
import numpy as np

grumpy = np.random.uniform(0, 10, size=20)
print(grumpy)

rounded = np.round(grumpy, 2)
print(rounded)

min_val = np.min(grumpy)
max_val = np.max(grumpy)
median_val = np.median(grumpy)
print(min_val, max_val, median_val)

grumpy[grumpy < 5] = grumpy[grumpy < 5] ** 2
print(grumpy)

def numpy_alternate_sort(array):
    sorted_array = np.sort(array)
    result = []
    left, right = 0, len(sorted_array) - 1
    while left <= right:
        result.append(sorted_array[left])
        left += 1
        if left <= right:
            result.append(sorted_array[right])
            right -= 1
    return np.array(result)

alt_sorted = numpy_alternate_sort(grumpy)
print(alt_sorted)
```

```
[8.82034238 8.478123   9.20707936 8.42563718 7.81784376 4.82901459
 1.17422161 0.15715916 4.26040852 1.35682094 9.78391306 5.54613363
 7.60028536 7.03950595 7.71524312 1.79499242 5.74315917 0.35046256
```

```
 1.48202714 9.53354629]
[8.82 8.48 9.21 8.43 7.82 4.83 1.17 0.16 4.26 1.36 9.78 5.55 7.6  7.04
 7.72 1.79 5.74 0.35 1.48 9.53]
0.15715915518557777 9.783913060868036 6.3913325590334935
[ 8.82034238  8.478123    9.20707936  8.42563718  7.81784376 23.31938189
  1.37879639  0.024699   18.15108073  1.84096306  9.78391306  5.54613363
  7.60028536  7.03950595  7.71524312  3.22199779  5.74315917  0.122824
  2.19640445  9.53354629]
[ 0.024699   23.31938189  0.122824   18.15108073  1.37879639  9.78391306
  1.84096306  9.53354629  2.19640445  9.20707936  3.22199779  8.82034238
  5.54613363  8.478123    5.74315917  8.42563718  7.03950595  7.81784376
  7.60028536  7.71524312]
```

0.1.2 Problem 2 Solution

A 1D NumPy array of random floats between 0 and 10 was generated. The elements were rounded to two decimal places for a clearer display.

We calculated the minimum, maximum, and median of this array to understand its distribution.

To add an interesting transformation, we squared all elements that were less than 5.

Lastly, the function `numpy_alternate_sort(array)` was implemented to sort the array in an alternating pattern: smallest, largest, second smallest, second largest, and so on. This was done using a two-pointer approach on the sorted array.

Q.3.

[3]:
```
import pandas as pd
import random

names = ['Alice', 'Bob', 'Charlie', 'Daisy', 'Evan', 'Fay', 'George', 'Hilda',␣
 ↪'Ivan', 'Julia']
subjects = ['Math', 'Physics', 'Chemistry', 'Biology', 'English']
subject_choices = [random.choice(subjects) for _ in range(10)]
scores = np.random.randint(50, 101, size=10)

df_buddy = pd.DataFrame({
    'Name': names,
    'Subject': subject_choices,
    'Score': scores,
    'Grade': [''] * 10
})

def assign_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
```

```
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'

df_buddy['Grade'] = df_buddy['Score'].apply(assign_grade)
print(df_buddy)

sorted_df = df_buddy.sort_values(by='Score', ascending=False)
print(sorted_df)

avg_scores = df_buddy.groupby('Subject')['Score'].mean()
print(avg_scores)

def pandas_filter_pass(dataframe):
    return dataframe[dataframe['Grade'].isin(['A', 'B'])]

df_pass = pandas_filter_pass(df_buddy)
print(df_pass)
```

```
      Name    Subject  Score Grade
0    Alice    Biology     72     C
1      Bob    Biology     57     F
2  Charlie  Chemistry     55     F
3    Daisy    Biology     56     F
4     Evan    Physics     53     F
5      Fay  Chemistry     93     A
6   George  Chemistry     52     F
7    Hilda  Chemistry     86     B
8     Ivan    Biology     95     A
9    Julia       Math     77     C
      Name    Subject  Score Grade
8     Ivan    Biology     95     A
5      Fay  Chemistry     93     A
7    Hilda  Chemistry     86     B
9    Julia       Math     77     C
0    Alice    Biology     72     C
1      Bob    Biology     57     F
3    Daisy    Biology     56     F
2  Charlie  Chemistry     55     F
4     Evan    Physics     53     F
6   George  Chemistry     52     F
Subject
Biology      70.0
Chemistry    71.5
Math         77.0
Physics      53.0
```

```
Name: Score, dtype: float64
    Name    Subject  Score Grade
5    Fay  Chemistry     93     A
7  Hilda  Chemistry     86     B
8   Ivan    Biology     95     A
```

0.1.3 Problem 3 Solution

We created a Pandas DataFrame representing 10 student records with columns: Name, Subject, Score, and Grade.

The grades were assigned based on the following criteria: - A: 90–100 - B: 80–89 - C: 70–79 - D: 60–69 - F: below 60

The DataFrame was then sorted in descending order based on Score.

We also computed the average score per Subject using groupby aggregation.

Finally, the function `pandas_filter_pass(dataframe)` was implemented to return only the records of students who received grades A or B.

Q.4.

[4]:
```python
import pandas as pd
import random
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

positive_reviews = ['Great movie!' for _ in range(50)]
negative_reviews = ['Terrible movie.' for _ in range(50)]
reviews = positive_reviews + negative_reviews
sentiments = ['positive'] * 50 + ['negative'] * 50

df_reviews = pd.DataFrame({'Review': reviews, 'Sentiment': sentiments})

vectorizer_fizz = CountVectorizer(max_features=500, stop_words='english')
X_fizz = vectorizer_fizz.fit_transform(df_reviews['Review'])

X_train_fizz, X_test_fizz, y_train_fizz, y_test_fizz = train_test_split(
    X_fizz, df_reviews['Sentiment'], test_size=0.2, random_state=42
)

model_fizz = MultinomialNB()
model_fizz.fit(X_train_fizz, y_train_fizz)

y_pred_fizz = model_fizz.predict(X_test_fizz)
accuracy_fizz = accuracy_score(y_test_fizz, y_pred_fizz)
print(accuracy_fizz)
```

```
def predict_review_sentiment(model, vectorizer, review):
    X_new = vectorizer.transform([review])
    return model.predict(X_new)[0]

sample_sentiment = predict_review_sentiment(model_fizz, vectorizer_fizz, 'I␣
↪loved this movie!')
print(sample_sentiment)
```

```
1.0
negative
```

0.1.4 Problem 4 Solution

We created a synthetic dataset of 100 short movie reviews — 50 positive and 50 negative.

Using `CountVectorizer`, we tokenized the reviews into a feature matrix with a maximum of 500 features and removed stop words.

The data was split into training (80%) and testing (20%) sets.

A Multinomial Naive Bayes classifier was trained on the training data and its accuracy was reported on the test set.

Finally, the function `predict_review_sentiment(model, vectorizer, review)` was created to predict the sentiment of a new review based on the trained model and vectorizer.

Q.5.

[5]:
```
import pandas as pd
import random
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score

good_feedback = ['Excellent product.' for _ in range(50)]
bad_feedback = ['Poor quality item.' for _ in range(50)]
feedbacks = good_feedback + bad_feedback
labels = ['good'] * 50 + ['bad'] * 50

df_feedback = pd.DataFrame({'Feedback': feedbacks, 'Label': labels})

vectorizer_glow = TfidfVectorizer(max_features=300, lowercase=True,␣
↪stop_words='english')
X_glow = vectorizer_glow.fit_transform(df_feedback['Feedback'])

X_train_glow, X_test_glow, y_train_glow, y_test_glow = train_test_split(
    X_glow, df_feedback['Label'], test_size=0.25, random_state=42
)
```

```
model_glow = LogisticRegression()
model_glow.fit(X_train_glow, y_train_glow)

y_pred_glow = model_glow.predict(X_test_glow)

precision_glow = precision_score(y_test_glow, y_pred_glow, pos_label='good')
recall_glow = recall_score(y_test_glow, y_pred_glow, pos_label='good')
f1_glow = f1_score(y_test_glow, y_pred_glow, pos_label='good')

print(precision_glow, recall_glow, f1_glow)

def text_preprocess_vectorize(texts, vectorizer):
    return vectorizer.transform(texts)

sample_vectorized = text_preprocess_vectorize(['Amazing experience.'],␣
 ↪vectorizer_glow)
print(sample_vectorized.shape)
```

```
1.0 1.0 1.0
(1, 5)
```

0.1.5 Problem 5 Solution

A synthetic dataset of 100 product feedback entries (50 good, 50 bad) was created.

Text preprocessing was performed using `TfidfVectorizer`, with a maximum of 300 features, lowercasing, and stop word removal.

We split the data into training (75%) and testing (25%) sets.

A Logistic Regression model was trained on the vectorized training data. We then reported precision, recall, and F1-score for the model on the test set.

Lastly, the function `text_preprocess_vectorize(texts, vectorizer)` was written to preprocess and vectorize any list of text samples using a fitted `TfidfVectorizer`.