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

problem 1

```
arr = np.random.randint(1, 51, size=(5, 4))
print("Array:\n", arr)
anti_diag = [arr[i, -i - 1] for i in range(min(arr.shape))]
print("Anti-diagonal:", anti_diag)
```

```
Array:
 [[48  1 34 13]
 [16 47 27 17]
 [30 32 29 37]
 [25 30 38 18]
 [50 16 45 25]]
Anti-diagonal: [np.int64(13), np.int64(27), np.int64(32), np.int64(25)]
```

```
row_max = np.max(arr, axis=1)
print("Row-wise max:", row_max)
```

```
Row-wise max: [48 47 37 38 50]
```

```
mean_val = np.mean(arr)
filtered_elements = arr[arr <= mean_val]
print("Elements ≤ mean (", mean_val, "):", filtered_elements)
```

```
Elements ≤ mean ( 28.9 ): [ 1 13 16 27 17 25 18 16 25]
```

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

print("Boundary traversal:", numpy_boundary_traversal(arr))
```

```
Boundary traversal: [np.int64(48), np.int64(1), np.int64(34), np.int64(13), np.in
```

problem 2

```
arr1d = np.random.uniform(0, 10, 20)
arr1d = np.round(arr1d, 2)
print("Rounded Array:", arr1d)
```

```
Rounded Array: [8.86 1.69 4.65 5.82 4.08 3.15 1.77 6.04 4.45 2.89 2.8  1.73 1.87
 5.53 8.45 8.18 6.25 3.7  3.52]
```

```
print("Min:", np.min(arr1d))
print("Max:", np.max(arr1d))
print("Median:", np.median(arr1d))
```

```
Min: 1.69
Max: 8.86
Median: 3.89
```

```
arr1d_mod = np.array([x**2 if x < 5 else x for x in arr1d])
print("Modified Array:", arr1d_mod)
```

```
Modified Array: [ 8.86    2.8561 21.6225  5.82   16.6464  9.9225  3.1329  6.04
  8.3521  7.84    2.9929  3.4969 13.5424  5.53    8.45    8.18    6.25
 13.69   12.3904]
```

```
def numpy_alternate_sort(array):
    sorted_arr = np.sort(array)
    result = []
    i, j = 0, len(sorted_arr) - 1
    while i <= j:
        if i == j:
            result.append(sorted_arr[i])
        else:
            result.extend([sorted_arr[i], sorted_arr[j]])
        i += 1
        j -= 1
    return np.array(result)

print("Alternating Sorted Array:", numpy_alternate_sort(arr1d))
```

```
Alternating Sorted Array: [1.69 8.86 1.73 8.45 1.77 8.18 1.87 6.25 2.8  6.04 2.89
 3.52 4.65 3.68 4.45 3.7  4.08]
```

problem 3

```
names = [f"Student{i}" for i in range(1, 11)]
subjects = ['Math', 'Physics', 'Chem', 'Math', 'Chem', 'Physics', 'Math', 'Chem', 'Ph
scores = np.random.randint(50, 101, size=10)

df = pd.DataFrame({'Name': names, 'Subject': subjects, 'Score': scores})
df['Grade'] = pd.cut(df['Score'], bins=[0, 59, 69, 79, 89, 100],
                     labels=['F', 'D', 'C', 'B', 'A'], right=True)
print("Graded DataFrame:\n", df)
```

```
Graded DataFrame:
          Name  Subject  Score Grade
    0   Student1     Math     69     D
    1   Student2  Physics     89     B
    2   Student3     Chem     73     C
    3   Student4     Math     80     B
    4   Student5     Chem     83     B
    5   Student6  Physics     70     C
    6   Student7     Math     83     B
    7   Student8     Chem     78     C
    8   Student9  Physics     75     C
```

```
9  Student10     Math     82     B
```

```
print("Sorted by Score:\n", df.sort_values(by='Score', ascending=False))
```

```
Sorted by Score:
          Name  Subject  Score Grade
1   Student2  Physics     89     B
4   Student5     Chem     83     B
6   Student7     Math     83     B
9  Student10     Math     82     B
3   Student4     Math     80     B
7   Student8     Chem     78     C
8   Student9  Physics     75     C
2   Student3     Chem     73     C
5   Student6  Physics     70     C
0   Student1     Math     69     D
```

```
print("Average Score by Subject:\n", df.groupby('Subject')['Score'].mean())
```

```
Average Score by Subject:
 Subject
Chem       78.0
Math       78.5
Physics    78.0
Name: Score, dtype: float64
```

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

print("Students with Grade A or B:\n", pandas_filter_pass(df))
```

```
Students with Grade A or B:
          Name  Subject  Score Grade
1   Student2  Physics     89     B
3   Student4     Math     80     B
4   Student5     Chem     83     B
6   Student7     Math     83     B
9  Student10     Math     82     B
```

problem 4

```
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! Loved it." for _ in range(50)]
negative_reviews = ["Terrible movie. Waste of time." for _ in range(50)]
reviews = positive_reviews + negative_reviews
sentiments = ['positive'] * 50 + ['negative'] * 50

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


vectorizer = CountVectorizer(max_features=500, stop_words='english')
X = vectorizer.fit_transform(df_reviews['Review'])
```

```
y = df_reviews['Sentiment']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state


model = MultinomialNB()
model.fit(X_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
preds = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, preds))
```

```
Accuracy: 1.0
```

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

print("Prediction:", predict_review_sentiment(model, vectorizer, "Awesome plot and ac
```

```
Prediction: negative
```

problem 5

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score


good_feedback = ["I love this product!" for _ in range(50)]
bad_feedback = ["Very bad experience." for _ in range(50)]
feedback = good_feedback + bad_feedback
labels = ['good'] * 50 + ['bad'] * 50

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


vectorizer_tfidf = TfidfVectorizer(max_features=300, lowercase=True, stop_words='engl
X = vectorizer_tfidf.fit_transform(df_feedback['Feedback'])
y = df_feedback['Label']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_stat


clf = LogisticRegression()
clf.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred = clf.predict(X_test)
print("Precision:", precision_score(y_test, y_pred, pos_label='good'))
print("Recall:", recall_score(y_test, y_pred, pos_label='good'))
print("F1-Score:", f1_score(y_test, y_pred, pos_label='good'))
```

```
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

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

print("Vectorized:", text_preprocess_vectorize(["Product is okay."], vectorizer_tfidf
```

```
Vectorized: [[0. 0. 0. 1.]]
```