Nithin S
221IT085

IT464 Practice Assignment

Q1. 1. Generate a matrix with random entries of a size 4*4 and nd and show i. The transpose of the matrix ii. Inverse iii. Trace iv. Eigenvalues v. Eigenvectors

```python
import numpy as np
def generate_random_matrix(n, low=0, high=100):
    return np.random.randint(low, high, size=(n, n))

n = 4
matrix = generate_random_matrix(n)
print(matrix)
```

```
[[75 93 99 63]
 [84 56 78 42]
 [ 6 72 84 64]
 [48 33 60 87]]
```

```python
matrix.T
```

```
array([[75, 84,  6, 48],
       [93, 56, 72, 33],
       [99, 78, 84, 60],
       [63, 42, 64, 87]])
```

```python
np.linalg.inv(matrix)
```

```
array([[ 0.01057316,  0.00147231, -0.01654584,  0.00380445],
       [ 0.05496434, -0.04600385, -0.02000872, -0.00287395],
       [-0.05803068,  0.05615986,  0.03687552, -0.01221626],
       [ 0.01333914, -0.02209351, -0.00871314,  0.01891037]])
```

```python
int(np.trace(matrix))
```

```
302
```

```python
eigen_values,eigen_vectors=np.linalg.eig(matrix)
```

```python
print(eigen_values)
```

```
[254.77761222 -21.73004059  25.28505106  43.66737732]
```

```python
print(eigen_vectors)
```

```
[[-0.6323862  -0.21242012 -0.46342224 -0.53958833]
 [-0.51261618  0.7312642  -0.46795235 -0.57766986]
 [-0.39714946 -0.61314973  0.74401777  0.44691069]
 [-0.42377429  0.21018507 -0.11268535  0.41882312]]
```

Q2. Verify and show the adherence of the following vector operations to vector algebra laws (like commutative, associative, and distributive laws) by testing each law. Formulate a table to depict the results of this test. (i) Vector addition (ii) Vector subtraction (iii) Scalar-Vector multiplication (iv) Vector-Vector multiplication (Inner product)

```python
import numpy as np

def test_vector_laws():
    # Test vectors
    a = np.array([1, 2, 3])
    b = np.array([4, 5, 6])
    c = np.array([7, 8, 9])
    k = 2  # scalar

    # Results dictionary to store all test results
    results = {
        "Vector Addition": {},
        "Vector Subtraction": {},
        "Scalar-Vector Multiplication": {},
        "Inner Product": {}
    }

    # 1. Vector Addition Tests
    # Commutative: a + b = b + a
    results["Vector Addition"]["Commutative"] = {
        "Law": "a + b = b + a",
        "Left Side": a + b,
        "Right Side": b + a,
        "Result": np.array_equal(a + b, b + a)
    }

    # Associative: (a + b) + c = a + (b + c)
    results["Vector Addition"]["Associative"] = {
        "Law": "(a + b) + c = a + (b + c)",
        "Left Side": (a + b) + c,
        "Right Side": a + (b + c),
        "Result": np.array_equal((a + b) + c, a + (b + c))
    }

    # 2. Vector Subtraction Tests
    # Non-commutative: a - b ≠ b - a
    results["Vector Subtraction"]["Non-Commutative"] = {
        "Law": "a - b ≠ b - a",
        "Left Side": a - b,
        "Right Side": b - a,
        "Result": not np.array_equal(a - b, b - a)
    }
```

```python
    # 3. Scalar-Vector Multiplication Tests
    # Associative: k(a + b) = ka + kb
    results["Scalar-Vector Multiplication"]["Distributive"] = {
        "Law": "k(a + b) = ka + kb",
        "Left Side": k * (a + b),
        "Right Side": k * a + k * b,
        "Result": np.array_equal(k * (a + b), k * a + k * b)
    }

    # 4. Inner Product Tests
    # Commutative: a·b = b·a
    results["Inner Product"]["Commutative"] = {
        "Law": "a·b = b·a",
        "Left Side": np.dot(a, b),
        "Right Side": np.dot(b, a),
        "Result": np.dot(a, b) == np.dot(b, a)
    }

    # Distributive: a·(b + c) = a·b + a·c
    results["Inner Product"]["Distributive"] = {
        "Law": "a·(b + c) = a·b + a·c",
        "Left Side": np.dot(a, (b + c)),
        "Right Side": np.dot(a, b) + np.dot(a, c),
        "Result": np.dot(a, (b + c)) == (np.dot(a, b) + np.dot(a, c))
    }

    # Print results in table format
    print("Vector Algebra Laws Verification Table")
    print("="*50)

    for operation, laws in results.items():
        print(f"\n{operation}:")
        print("-"*50)
        print(f"{'Law':<30} | {'Result':<10}")
        print("-"*50)
        for law_name, details in laws.items():
            print(f"{details['Law']:<30} | {'✓' if details['Result'] else '✗':<10}")

    return results

# Run the tests
test_vector_laws()
```

```
Vector Algebra Laws Verification Table
==================================================

Vector Addition:
--------------------------------------------------
Law                            | Result
--------------------------------------------------
a + b = b + a                  | ✓
(a + b) + c = a + (b + c)      | ✓

Vector Subtraction:
--------------------------------------------------
Law                            | Result
--------------------------------------------------
a - b ≠ b - a                  | ✓

Scalar-Vector Multiplication:
--------------------------------------------------
Law                            | Result
--------------------------------------------------
k(a + b) = ka + kb             | ✓

Inner Product:
--------------------------------------------------
Law                            | Result
--------------------------------------------------
a·b = b·a                      | ✓
a·(b + c) = a·b + a·c          | ✓
```

Q3. Consider marks obtained by a class of Engg. students for three assignments, a mid-sem, one mini-project work and an end-sem are mentioned in .xlxs les that can be downloaded from the moodle portal to make the nal report for grading the students of the class. Course evaluation plan is made by giving a weightage of 20% each to the assignments (for all the three), the mid-sem exam and the mini-project work. Remaining 40% weightage is given to the end-sem exam. Now using the vector operations calculate and show the nal consolidated marks (normalized to hundred) of the class using vector operations. Statics to show/display: i. Final scores of each assignment, mid, end-sem and project after normalization of nal score to 100. (ii). Find mean, variance and standard deviation along with the maximum and minimum of the normalization nal marks. (iii) Apply barchart on #students obtaining scores in dierent ranges like 0-10, 11-20,...,91-100. (iv) Apply barchart for the score ranges 100-81, 80-74, 73-63, 62-50, 49-38, 36-27 and 26-20 to nd the number of students obtained the scores in those ranges. (v) Plot the marks distribution of the students using Normal distribution

```
import pandas as pd
```

```
df=pd.read_excel("StudentsMarks.xlsx",header=3)
```

```
df.columns
```

```
Index(['Sl. No.', 'Mid-Semester (50 -> 20.0)', 'End_Semester (100 -> 40.0)',
       'Project (100 -> 20.0)', 'Lab Assigment-1 (72 -> 6.66)',
       'Lab Assigment-2 (100 -> 6.67)', 'Lab Assigment-3 (100 -> 6.67)'],
      dtype='object')
```

```
df.rename(columns={"Mid-Semester (50 -> 20.0)":"MidSem","End_Semester (100 -> 40.0)":"EndSem","Project (100 -> 20.0)":"Project","Lab Assigmer
```

```
df
```

	Sl. No.	MidSem	EndSem	Project	Lab1	Lab2	Lab3
0	1	12.2	31.6	19.65	6.20	6.40	6.67
1	2	6.0	21.6	17.45	6.20	6.27	6.34
2	3	9.4	28.4	17.95	6.66	6.67	6.34
3	4	14.4	26.4	18.95	6.43	6.67	6.60
4	5	10.4	19.8	18.00	6.20	6.67	6.67
...	...	...	...	...	...	...	...
78	79	8.6	24.4	17.80	3.33	6.47	6.67
79	80	12.2	21.4	18.78	5.74	6.67	6.67
80	81	9.2	18.8	17.35	2.87	6.64	6.67
81	82	6.4	24.4	17.68	2.87	6.54	6.67
82	83	10.8	25.2	17.68	6.66	6.47	6.67

83 rows × 7 columns

```
df["Total"]=df["MidSem"]+df["EndSem"]+df["Project"]+df["Lab1"]+df["Lab2"]+df["Lab3"]
```

```
df
```

[191]:
```python
df
```

[191]:

	Sl. No.	MidSem	EndSem	Project	Lab1	Lab2	Lab3	Total
0	1	12.2	31.6	19.65	6.20	6.40	6.67	82.72
1	2	6.0	21.6	17.45	6.20	6.27	6.34	63.86
2	3	9.4	28.4	17.95	6.66	6.67	6.34	75.42
3	4	14.4	26.4	18.95	6.43	6.67	6.60	79.45
4	5	10.4	19.8	18.00	6.20	6.67	6.67	67.74
...	...	...	...	...	...	...	...	...
78	79	8.6	24.4	17.80	3.33	6.47	6.67	67.27
79	80	12.2	21.4	18.78	5.74	6.67	6.67	71.46
80	81	9.2	18.8	17.35	2.87	6.64	6.67	61.53
81	82	6.4	24.4	17.68	2.87	6.54	6.67	64.56
82	83	10.8	25.2	17.68	6.66	6.47	6.67	73.48

83 rows × 8 columns

[192]:
```python
df["Total"].var()
```

[192]: 138.5255366735233

[193]:
```python
df["Total"].std()
```

[193]: 11.769687195228398

[194]:
```python
float(df["Total"].mean())
```

[194]: 70.64590361445782

[195]:
```python
df["Total"].max()
```

[195]: 90.95

[196]:
```python
df["Total"].min()
```

[196]: 33.85

[197]:
```python
df["Total"].describe()
```

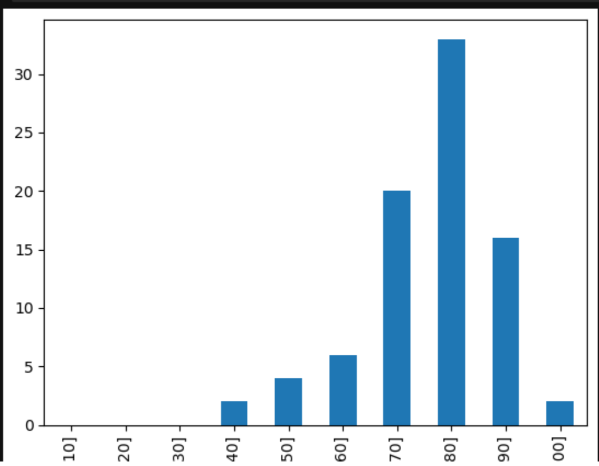
[197]:
```
count    83.000000
mean     70.645904
std      11.769687
min      33.850000
25%      64.275000
50%      72.020000
75%      79.285000
max      90.950000
Name: Total, dtype: float64
```

[198]:
```python
import matplotlib.pyplot as plt
```

[199]:
```python
bins = [0,10,20,30,40,50,60,70,80,90,100]
data = df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
data.plot(kind='bar')
```

```
/tmp/ipykernel_2321252/1132862309.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
ersion of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  data = df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
```

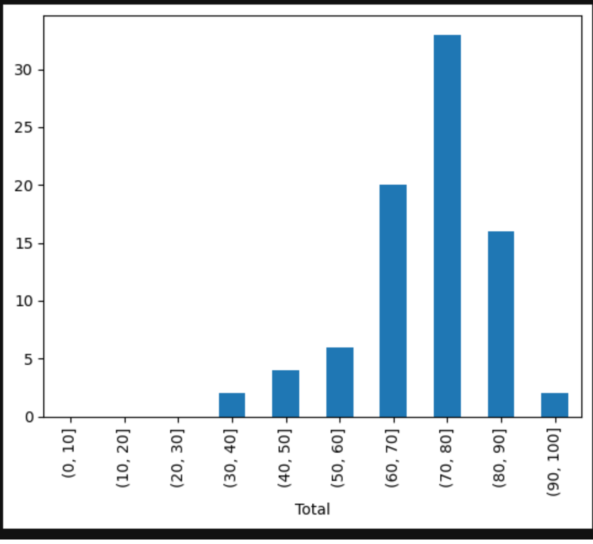
[199]: <Axes: xlabel='Total'>



[199]:
```python
bins = [0,10,20,30,40,50,60,70,80,90,100]
data = df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
data.plot(kind='bar')
```

```
/tmp/ipykernel_2321252/1132862309.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
ersion of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  data = df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
```

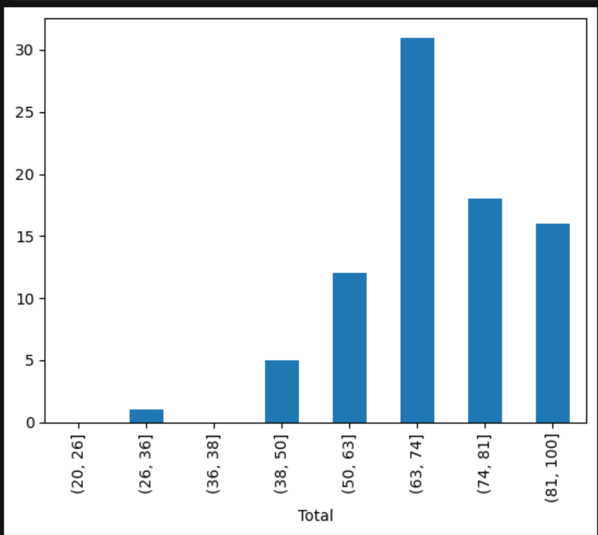
[199]:
```
<Axes: xlabel='Total'>
```



[201]:
```python
bins = [20,26,36,38,50,63,74,81,100]
data= df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
data.plot(kind='bar')
```

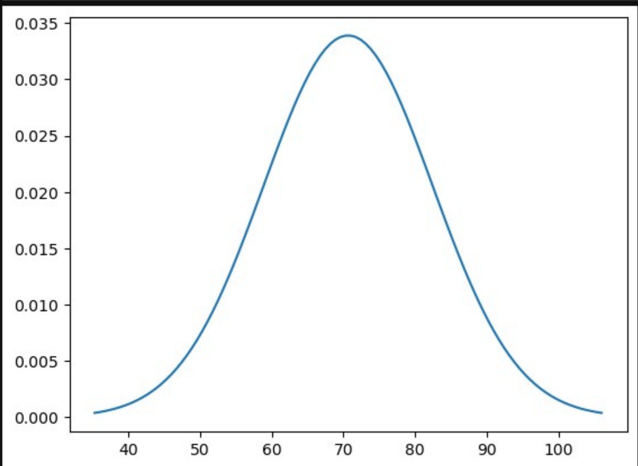
```
/tmp/ipykernel_2321252/2419661360.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future v
ersion of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  data= df.groupby(pd.cut(df['Total'], bins=bins)).Total.count()
```

[201]:
```
<Axes: xlabel='Total'>
```

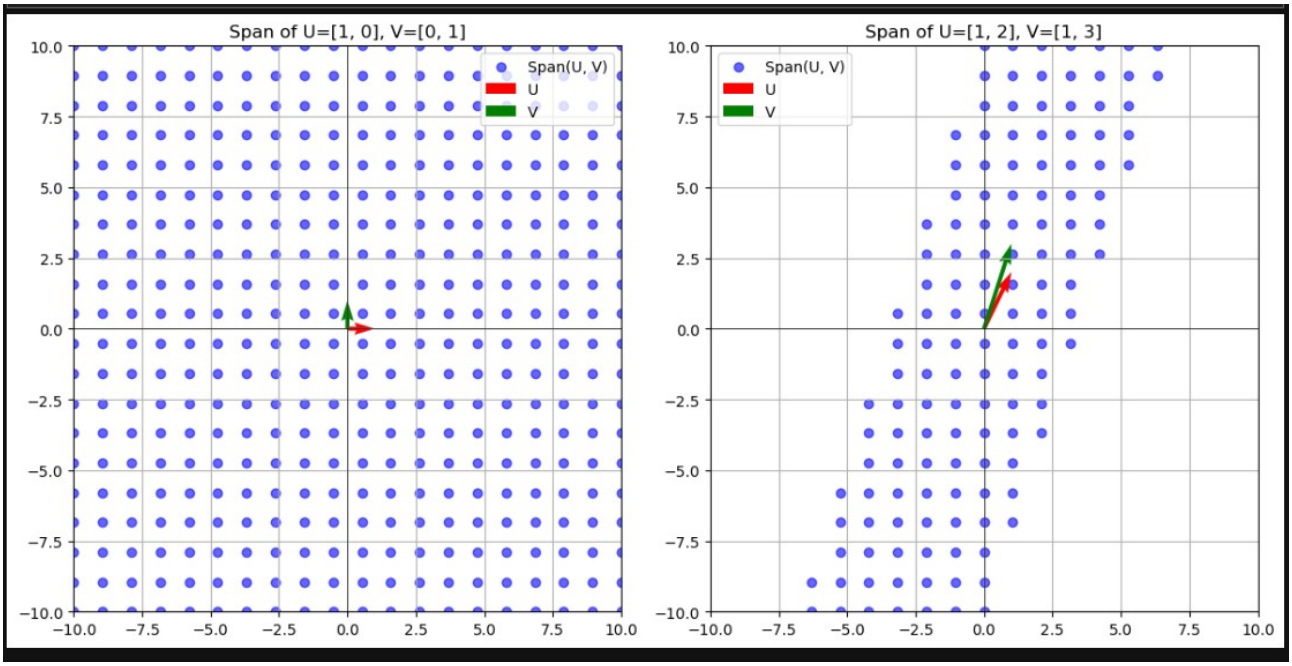


```
[204]: import matplotlib.pyplot as plt
       import numpy as np
       import scipy.stats as stats
       import math

       mu = df["Total"].mean()
       variance = df["Total"].var()
       sigma = math.sqrt(variance)
       x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
       plt.plot(x, stats.norm.pdf(x, mu, sigma))
       plt.show()
```



Q4. Compute linear combinations of the following vectors to nd the span and visualize the vector span in a higher dimensional plane. (i) U = [1 0]; V=[0 1]; (ii) U = [1 2]; V=[1 3];



```python
U1, V1 = np.array([1, 0]), np.array([0, 1])
U2, V2 = np.array([1, 2]), np.array([1, 3])


def generate_combinations(U, V, scalar_range=(-10, 10), steps=20):
    scalars = np.linspace(*scalar_range, steps)
    combinations = [a * U + b * V for a in scalars for b in scalars]
    return np.array(combinations)


span1 = generate_combinations(U1, V1)
span2 = generate_combinations(U2, V2)


fig, axes = plt.subplots(1, 2, figsize=(12, 6))


axes[0].scatter(span1[:, 0], span1[:, 1], alpha=0.6, c='blue', label="Span(U, V)")
axes[0].quiver(0, 0, U1[0], U1[1], angles='xy', scale_units='xy', scale=1, color='red', label="U")
axes[0].quiver(0, 0, V1[0], V1[1], angles='xy', scale_units='xy', scale=1, color='green', label="V")
axes[0].set_title("Span of U=[1, 0], V=[0, 1]")
axes[0].set_xlim(-10, 10)
axes[0].set_ylim(-10, 10)
axes[0].axhline(0, color='black', linewidth=0.5)
axes[0].axvline(0, color='black', linewidth=0.5)
axes[0].legend()
axes[0].grid()


axes[1].scatter(span2[:, 0], span2[:, 1], alpha=0.6, c='blue', label="Span(U, V)")
axes[1].quiver(0, 0, U2[0], U2[1], angles='xy', scale_units='xy', scale=1, color='red', label="U")
axes[1].quiver(0, 0, V2[0], V2[1], angles='xy', scale_units='xy', scale=1, color='green', label="V")
axes[1].set_title("Span of U=[1, 2], V=[1, 3]")
axes[1].set_xlim(-10, 10)
axes[1].set_ylim(-10, 10)
axes[1].axhline(0, color='black', linewidth=0.5)
axes[1].axvline(0, color='black', linewidth=0.5)
axes[1].legend()
axes[1].grid()

plt.tight_layout()
plt.show()
```

Q5. Generate a random matrix of size 30*50 and write a Python code to nd the no.of independent vectors or columns of the matrix. Find the rank of the matrix using Python code. Make your comment.

```python
matrix=np.random.randint(1, 100, size=(30, 50))
matrix
```

```
array([[94, 43, 33, ..., 77,  1, 31],
       [61, 73, 74, ..., 86, 59, 60],
       [17, 35,  1, ..., 27,  2, 48],
       ...,
       [31, 65, 76, ..., 29, 13, 87],
       [32, 41, 80, ..., 74, 33, 32],
       [58, 40, 31, ..., 97, 88,  1]])
```

```python
int(np.linalg.matrix_rank(matrix))
```

```
30
```

The rank represents the maximum number of linearly independent rows or columns.
The number of independent columns is equal to the rank of the matrix.
The rank is the same for rows and columns in a matrix, so independent rows = independent columns.