Start coding or generate with AI.

Consider a list datatype (1D) then reshape it into2D, 3D matrix using numpy

```
import numpy as np
list = [10,20,30,40,50,60,70,80]
re_list = np.array(list)
re_list = re_list.reshape(2,4)
print(re_list)
re_list = re_list.reshape(2,2,2)
print(re_list)
```

```
[[10 20 30 40]
 [50 60 70 80]]
[[[10 20]
  [30 40]]

 [[50 60]
  [70 80]]]
```

Generate random matrices using numpy

```
mat = np.random.rand(3,3)

print(mat)
```

```
[[0.51120706 0.71597105 0.37998171]
 [0.08174204 0.24277228 0.70202174]
 [0.01095817 0.14745475 0.46160723]]
```

Find the determinant of a matrix using scipy

```
from scipy import linalg

det_mat = linalg.det(mat)
```

Find eigen value and eigen vector of a matrix using scipy

```
import numpy as np
eigen_val, eigen_vec = linalg.eig(mat)
print(eigen_val)
print(eigen_vec)
```

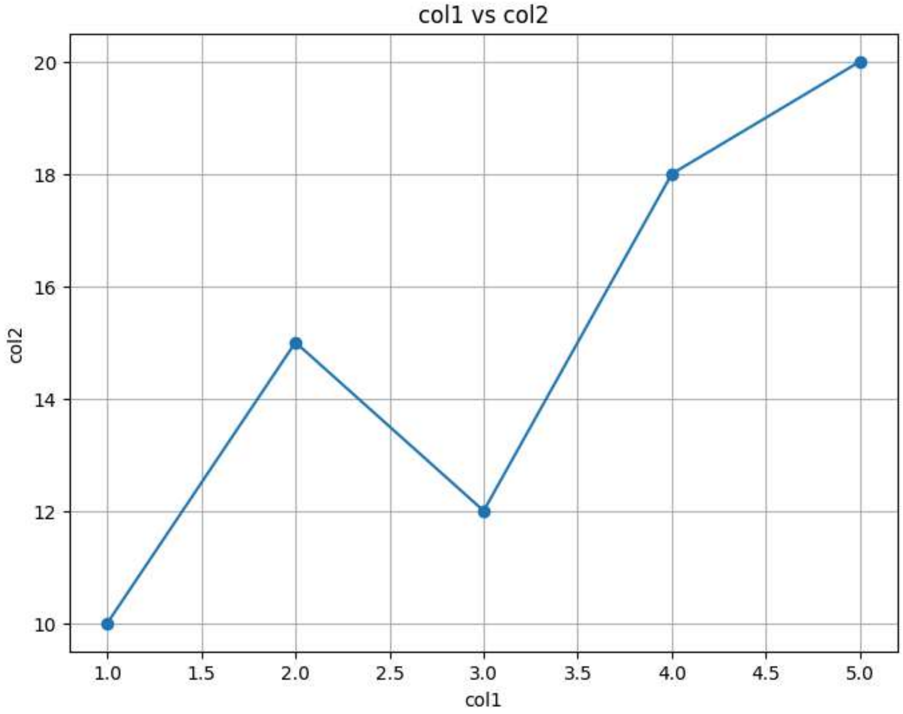
```
[ 0.82462246+0.j  0.42929199+0.j -0.03832788+0.j]
[[-0.93264926 -0.99078524  0.73285744]
 [-0.32357024  0.04568776 -0.6567798 ]
 [-0.15958587  0.12750385  0.17765211]]
```

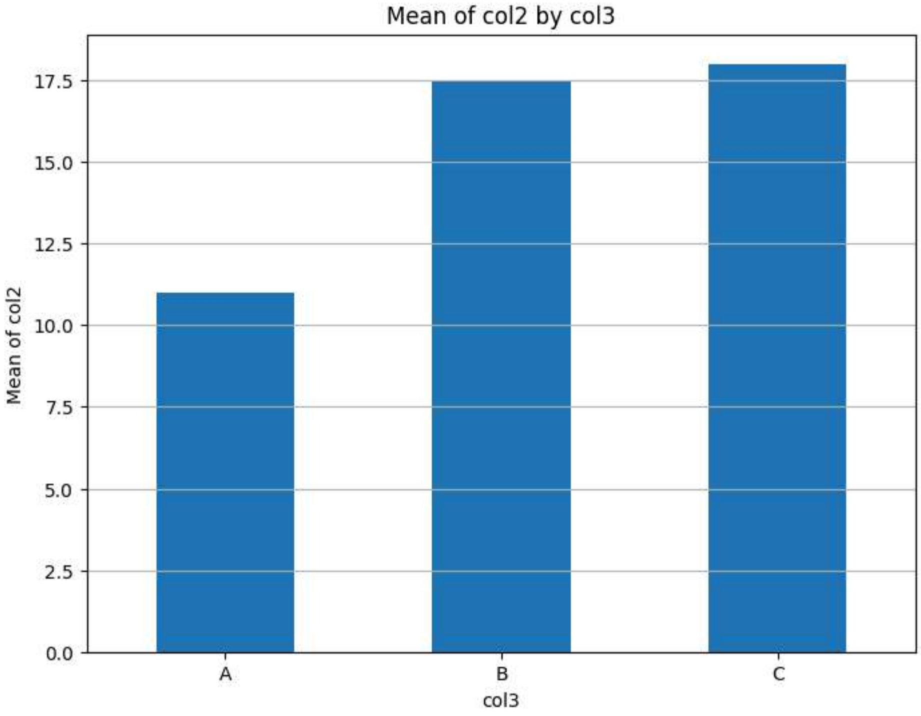
Implementation of Python Libraries for ML application such as Pandas and Matplotlib

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'col1': [1, 2, 3, 4, 5],
        'col2': [10, 15, 12, 18, 20],
        'col3': ['A', 'B', 'A', 'C', 'B']}
df = pd.DataFrame(data)

print("\nSample Pandas DataFrame:")
print(df)
```

```
filtered_df = df[df['col2'] > 15]
print("\nFiltered DataFrame (col2 > 15):")
print(filtered_df)

grouped_data = df.groupby('col3')['col2'].mean()
print("\nMean of col2 grouped by col3:")
print(grouped_data)

plt.figure(figsize=(8, 6))
plt.plot(df['col1'], df['col2'], marker='o', linestyle='-')
plt.title('col1 vs col2')
plt.xlabel('col1')
plt.ylabel('col2')
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))
grouped_data.plot(kind='bar')
plt.title('Mean of col2 by col3')
plt.xlabel('col3')
plt.ylabel('Mean of col2')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

```
Sample Pandas DataFrame:
   col1  col2 col3
0     1    10    A
1     2    15    B
2     3    12    A
3     4    18    C
4     5    20    B

Filtered DataFrame (col2 > 15):
   col1  col2 col3
3     4    18    C
4     5    20    B

Mean of col2 grouped by col3:
col3
A    11.0
B    17.5
C    18.0
Name: col2, dtype: float64
```





Create a pandas Series, access its index and values, compare it with a NumPy array, create a Series with custom indices, access a single value in a Series, load a dataset into a pandas DataFrame, and demonstrate different Matplotlib plotting methods.

Create a pandas series

Create a simple pandas Series and display it.

```
my_series = pd.Series([1, 2, 3, 4, 5])
print(my_series)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Access series elements

Demonstrate how to access the index and values of the created Series.

```
print("Index of the Series:")
print(my_series.index)

print("\nValues of the Series:")
print(my_series.values)
```

```
Index of the Series:
RangeIndex(start=0, stop=5, step=1)

Values of the Series:
[1 2 3 4 5]
```

Compare numpy arrays and pandas series

Explain the differences between NumPy arrays and pandas Series.

```
numpy_array = np.array([100, 200, 300,400])

print("\nNumPy Array:")
print(numpy_array)
print("NumPy Array Index (implicit):", range(len(numpy_array)))
print("NumPy Array Values:", numpy_array)
print("NumPy Array Data Type:", numpy_array.dtype)

pandas_series_default = pd.Series([10, 20, 30, 40])
print("\nPandas Series with Default Index:")
print(pandas_series_default)
print("Pandas Series Default Index (implicit):", pandas_series_default.index)
print("Pandas Series Values:", pandas_series_default.values)
print("Pandas Series Data Type:", pandas_series_default.dtype)

custom_index = ['a', 'b', 'c', 'd']
pandas_series_custom = pd.Series([10, 20, 30, 40], index=custom_index)
print("\nPandas Series with Custom Index:")
print(pandas_series_custom)
print("Pandas Series Custom Index (explicit):", pandas_series_custom.index)
print("Pandas Series Values:", pandas_series_custom.values)
print("Pandas Series Data Type:", pandas_series_custom.dtype)

print("\nAccessing element at index 'b' in custom Series:", pandas_series_custom['b'])

numpy_with_nan = np.array([1, 2, np.nan, 4])
print("\nNumPy Array with NaN:", numpy_with_nan)

print("Is NaN in NumPy array?", np.isnan(numpy_with_nan))

pandas_with_nan = pd.Series([1, 2, np.nan, 4])
```

```
print("\nPandas Series with NaN:", pandas_with_nan)

print("Is NaN in Pandas Series?", pandas_with_nan.isnull())

print("Pandas Series after dropping NaN:", pandas_with_nan.dropna())

print("Pandas Series after filling NaN with 0:", pandas_with_nan.fillna(0))
```

```
NumPy Array:
[100 200 300 400]
NumPy Array Index (implicit): range(0, 4)
NumPy Array Values: [100 200 300 400]
NumPy Array Data Type: int64

Pandas Series with Default Index:
0    10
1    20
2    30
3    40
dtype: int64
Pandas Series Default Index (implicit): RangeIndex(start=0, stop=4, step=1)
Pandas Series Values: [10 20 30 40]
Pandas Series Data Type: int64

Pandas Series with Custom Index:
a    10
b    20
c    30
d    40
dtype: int64
Pandas Series Custom Index (explicit): Index(['a', 'b', 'c', 'd'], dtype='object')
Pandas Series Values: [10 20 30 40]
Pandas Series Data Type: int64

Accessing element at index 'b' in custom Series: 20

NumPy Array with NaN: [ 1.  2. nan  4.]
Is NaN in NumPy array? [False False  True False]

Pandas Series with NaN: 0    1.0
1    2.0
2    NaN
3    4.0
dtype: float64
Is NaN in Pandas Series? 0    False
1    False
2     True
3    False
dtype: bool
Pandas Series after dropping NaN: 0    1.0
1    2.0
3    4.0
dtype: float64
Pandas Series after filling NaN with 0: 0    1.0
1    2.0
2    0.0
3    4.0
dtype: float64
```

```
import pandas as pd
import numpy as np

print("\n--- Demonstrating Differences ---")

print("\n1. Indexing:")
numpy_array_indexing = np.array([100, 200, 300])
print("NumPy Array:", numpy_array_indexing)
print("NumPy indexing is implicit integer (0, 1, 2)")

pandas_series_indexing = pd.Series([100, 200, 300], index=['apple', 'banana', 'cherry'])
print("Pandas Series:", pandas_series_indexing, sep='\n')
print("Pandas indexing is explicit (apple, banana, cherry)")

print("Accessing element at index 1 in NumPy:", numpy_array_indexing[1])
print("Accessing element at index 'banana' in Pandas:", pandas_series_indexing['banana'])

print("\n2. Data Types (Homogeneity):")
numpy_homogeneous = np.array([1, 2, 3, 4.5])
print("NumPy Array (homogeneous):", numpy_homogeneous, "Data Type:", numpy_homogeneous.dtype)
print("Pandas Series (underlying NumPy array is homogeneous):", pandas_series_indexing, "Data Type:", pandas_series_indexing.dtype)

print("\n3. Handling of Missing Data:")
numpy_missing = np.array([1, 2, np.nan, 4])
print("NumPy Array with NaN:", numpy_missing)
print("Checking for NaN in NumPy:", np.isnan(numpy_missing))


pandas_missing = pd.Series([1, 2, np.nan, 4])
print("Pandas Series with NaN:", pandas_missing, sep='\n')
print("Checking for NaN in Pandas:", pandas_missing.isnull(), sep='\n')
print("Dropping NaN in Pandas:", pandas_missing.dropna(), sep='\n')
print("Filling NaN with 0 in Pandas:", pandas_missing.fillna(0), sep='\n')

print("\n4. Functionality:")
print("NumPy is primarily for numerical operations and array manipulation.")
print("Pandas adds higher-level data analysis tools.")

print("Mean of NumPy array:", np.mean(numpy_homogeneous))
print("Mean of Pandas Series:", pandas_series_indexing.mean())

s1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s2 = pd.Series([4, 5, 6], index=['b', 'c', 'd'])
print("\nPandas Series 1:\n", s1)
print("Pandas Series 2:\n", s2)
print("Adding Pandas Series (aligns by index, introduces NaN):")
print(s1 + s2)

n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
print("\nNumPy Array 1:", n1)
print("NumPy Array 2:", n2)
print("Adding NumPy Arrays (element-wise based on position):")
print(n1 + n2)

print("\n--- Summary ---")
print("NumPy: Fast numerical operations, homogeneous data, implicit integer index.")
print("Pandas Series: Built on NumPy, explicit flexible index, better missing data handling, integrated with DataFrame, higher-level data ma
```

```
NumPy Array (homogeneous): [1.  2.  3.  4.5] Data Type: float64
Pandas Series (underlying NumPy array is homogeneous): apple     100
banana    200
cherry    300
dtype: int64 Data Type: int64

3. Handling of Missing Data:
NumPy Array with NaN: [ 1.  2. nan  4.]
Checking for NaN in NumPy: [False False  True False]
Pandas Series with NaN:
0    1.0
1    2.0
2    NaN
3    4.0
dtype: float64
Checking for NaN in Pandas:
0    False
1    False
2     True
3    False
dtype: bool
Dropping NaN in Pandas:
0    1.0
1    2.0
3    4.0
```

```
b    2
```

```
c    3
dtype: int64
Pandas Series 2:
 b    4
c    5
d    6
dtype: int64
Adding Pandas Series (aligns by index, introduces NaN):
a    NaN
b    6.0
c    8.0
d    NaN
dtype: float64

NumPy Array 1: [1 2 3]
NumPy Array 2: [4 5 6]
```

```
import pandas as pd
import numpy as np

my_series = pd.Series([1, 2, 3, 4, 5])
print(my_series)

print("Index of the Series:")
print(my_series.index)

print("\nValues of the Series:")
print(my_series.values)


numpy_array = np.array([10, 20, 30, 40])

print("\nNumPy Array:")
print(numpy_array)
print("NumPy Array Index (implicit):", list(range(len(numpy_array))))
print("NumPy Array Values:", numpy_array)
print("NumPy Array Data Type:", numpy_array.dtype)

pandas_series_default = pd.Series([10, 20, 30, 40])
print("\nPandas Series with Default Index:")
print(pandas_series_default)
print("Pandas Series Default Index (implicit):", pandas_series_default.index)
print("Pandas Series Values:", pandas_series_default.values)
print("Pandas Series Data Type:", pandas_series_default.dtype)

custom_index = ['a', 'b', 'c', 'd']
pandas_series_custom = pd.Series([10, 20, 30, 40], index=custom_index)
print("\nPandas Series with Custom Index:")
print(pandas_series_custom)
print("Pandas Series Custom Index (explicit):", pandas_series_custom.index)
print("Pandas Series Values:", pandas_series_custom.values)
print("Pandas Series Data Type:", pandas_series_custom.dtype)

print("\nAccessing element at index 'b' in custom Series:", pandas_series_custom['b'])

numpy_with_nan = np.array([1, 2, np.nan, 4])
print("\nNumPy Array with NaN:", numpy_with_nan)

print("Is NaN in NumPy array?", np.isnan(numpy_with_nan))

pandas_with_nan = pd.Series([1, 2, np.nan, 4])
print("\nPandas Series with NaN:", pandas_with_nan)

print("Is NaN in Pandas Series?", pandas_with_nan.isnull())

print("Pandas Series after dropping NaN:", pandas_with_nan.dropna())

print("Pandas Series after filling NaN with 0:", pandas_with_nan.fillna(0))


print("\n--- Demonstrating Differences ---")

print("\n1. Indexing:")
numpy_array_indexing = np.array([100, 200, 300])
print("NumPy Array:", numpy_array_indexing)
print("NumPy indexing is implicit integer (0, 1, 2)")
```

```
pandas_series_indexing = pd.Series([100, 200, 300], index=['apple', 'banana', 'cherry'])
print("Pandas Series:", pandas_series_indexing, sep='\n')
print("Pandas indexing is explicit (apple, banana, cherry)")

print("Accessing element at index 1 in NumPy:", numpy_array_indexing[1])
print("Accessing element at index 'banana' in Pandas:", pandas_series_indexing['banana'])

print("\n2. Data Types (Homogeneity):")
numpy_homogeneous = np.array([1, 2, 3, 4.5])
print("NumPy Array (homogeneous):", numpy_homogeneous, "Data Type:", numpy_homogeneous.dtype)

print("Pandas Series (underlying NumPy array is homogeneous):", pandas_series_indexing, "Data Type:", pandas_series_indexing.dtype)


print("\n3. Handling of Missing Data:")
numpy_missing = np.array([1, 2, np.nan, 4])
print("NumPy Array with NaN:", numpy_missing)
print("Checking for NaN in NumPy:", np.isnan(numpy_missing))


pandas_missing = pd.Series([1, 2, np.nan, 4])
print("Pandas Series with NaN:", pandas_missing, sep='\n')
print("Checking for NaN in Pandas:", pandas_missing.isnull(), sep='\n')
print("Dropping NaN in Pandas:", pandas_missing.dropna(), sep='\n')
print("Filling NaN with 0 in Pandas:", pandas_missing.fillna(0), sep='\n')

print("\n4. Functionality:")
print("NumPy is primarily for numerical operations and array manipulation.")
print("Pandas adds higher-level data analysis tools.")

print("Mean of NumPy array:", np.mean(numpy_homogeneous))
print("Mean of Pandas Series:", pandas_series_indexing.mean())

s1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s2 = pd.Series([4, 5, 6], index=['b', 'c', 'd'])
print("\nPandas Series 1:\n", s1)
print("Pandas Series 2:\n", s2)
print("Adding Pandas Series (aligns by index, introduces NaN):")
print(s1 + s2)

n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
print("\nNumPy Array 1:", n1)
print("NumPy Array 2:", n2)
print("Adding NumPy Arrays (element-wise based on position):")
print(n1 + n2)

print("\n--- Summary ---")
print("NumPy: Fast numerical operations, homogeneous data, implicit integer index.")
print("Pandas Series: Built on NumPy, explicit flexible index, better missing data handling, integrated with DataFrame, higher-level data man
2    NaN
3    4.0
dtype: float64
Checking for NaN in Pandas:
0    False
1    False
2     True
3    False
dtype: bool
Dropping NaN in Pandas:
0    1.0
1    2.0
3    4.0
dtype: float64
Filling NaN with 0 in Pandas:
0    1.0
1    2.0
2    0.0
3    4.0
dtype: float64

4. Functionality:
NumPy is primarily for numerical operations and array manipulation.
Pandas adds higher-level data analysis tools.
Mean of NumPy array: 2.625
Mean of Pandas Series: 200.0

Pandas Series 1:
 a    1
b    2
```

Create a series with custom indices

Create a pandas Series with user-defined indices and display it.

```
values = [10, 20, 30, 40, 50]
custom_indices = ['a', 'b', 'c', 'd', 'e']
my_series_with_index = pd.Series(values, index=custom_indices)
print(my_series_with_index)
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Access a single value in a series

Show how to access a specific value in the Series using its index.

```
single_value = my_series_with_index['c']
print("Value at index 'c':")
print(single_value)
```

```
Value at index 'c':
30
```

Load data into a dataframe

Load a dataset into a pandas DataFrame (using a sample dataset or a file if available).

```
data = {'col1': [1, 2, 3, 4, 5],
        'col2': [10, 15, 12, 18, 20],
        'col3': ['A', 'B', 'A', 'C', 'B']}
df = pd.DataFrame(data)
print(df)
```

```
   col1  col2 col3
0     1    10    A
1     2    15    B
2     3    12    A
3     4    18    C
```

```
        4     5    20    B
```

Demonstrate matplotlib methods

Showcase different plotting methods available in Matplotlib using the loaded DataFrame (e.g., scatter plot, histogram).

```
plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
plt.scatter(df['col1'], df['col2'])
plt.title('Scatter plot of Column 1 vs Column 2')
plt.xlabel('Column 1')
plt.ylabel('Column 2')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.hist(df['col2'], bins=5, edgecolor='black')
plt.title('Histogram of Column 2')
plt.xlabel('Column 2 Values')
```