50 PYTHON INTERVIEW QUESTIONS

FOCUSED ON PANDAS AND NUMPY

FOR DATA ANALYSIS

With Examples and Python Code

1. What is the difference between a Pandas Series and a NumPy array?

- **Pandas Series**: A one-dimensional labelled array capable of holding any data type. It includes an index that makes it easier to label and access individual elements. It is similar to a NumPy array but with additional functionality like handling missing data.
- **NumPy Array**: A homogeneous, multi-dimensional array. Unlike Pandas Series, it does not support labels and is limited to numerical data types, making it faster for numerical operations.

Example:

```
import numpy as np

import pandas as pd

# NumPy Array

np_array = np.array([1, 2, 3, 4])

# Pandas Series

pd_series = pd.Series([1, 2, 3, 4])

print(type(np_array)) # <class 'numpy.ndarray'>

print(type(pd_series)) # <class 'pandas.core.series.Series'>
```

2. How do you create a DataFrame in Pandas from a dictionary?

You can create a Pandas DataFrame by passing a dictionary, where keys represent column names and values are lists (or other iterable types) representing the data.

Example:

```
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}

df = pd.DataFrame(data)

print(df)
```

Output:

```
 Name Age

0 Alice 25

1 Bob 30

2 Charlie 35
```

3. How do you convert a Pandas DataFrame into a NumPy array?

You can use the .values attribute or .to_numpy() method to convert a Pandas DataFrame into a NumPy array.

Example:

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

np_array = df.to_numpy()

print(np_array)
```

Output:

```
[[1 4]
 [2 5]
 [3 6]]
```

4. Explain the concept of broadcasting in NumPy and how it works.

Broadcasting refers to the ability of NumPy to perform element-wise operations on arrays of different shapes. NumPy automatically adjusts the smaller array to match the shape of the larger array during arithmetic operations.

Example:

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([10])

# Broadcasting allows addition

result = arr1 + arr2

print(result)
```

Output:

```
[11 12 13]
```

In this example, arr2 is broadcasted across arr1.

5. How can you check for missing values in a Pandas DataFrame?

You can use the .isnull() or .isna() method to check for missing values. These methods return a DataFrame of the same shape with True for missing values and False for non-missing values.

Example:

```
df = pd.DataFrame({'A': [1, 2, np.nan], 'B': [4, np.nan, 6]})

print(df.isnull())
```

Output:

```
   A      B
0  False  False
1  False  True
2  True   False
```

6. What are some ways to handle missing data in Pandas?

- **Fill missing values**: Use .fillna() to replace missing values with a specific value or method (e.g., forward fill or backward fill).
- **Drop missing values**: Use .dropna() to remove rows or columns with missing values.

Example:

```
df = pd.DataFrame({'A': [1, 2, np.nan], 'B': [4, np.nan, 6]})

# Fill missing values with a specific value

df_filled = df.fillna(0)

print(df_filled)

# Drop rows with any missing values

df_dropped = df.dropna()

print(df_dropped)
```

7. What is the use of the groupby() function in Pandas?

The groupby() function in Pandas is used to group data by one or more columns and then apply an aggregation function (such as sum, mean, etc.) on the grouped data.

Example:

```
df = pd.DataFrame({'Category': ['A', 'B', 'A', 'B'], 'Value': [10, 20, 30, 40]})

grouped = df.groupby('Category').sum()

print(grouped)
```

Output:

```
 Value

Category

A 40

B 60
```

8. How would you join two Pandas DataFrames on a specific column?

You can use the .merge() function to join two DataFrames on a common column. This is similar to SQL joins (inner, left, right, outer).

Example:

```
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})

df2 = pd.DataFrame({'ID': [1, 2, 4], 'Age': [25, 30, 40]})

merged = pd.merge(df1, df2, on='ID', how='inner')

print(merged)
```

Output:

```
 ID Name Age

0 1 Alice 25

1 2 Bob 30
```

9. How do you filter a Pandas DataFrame based on multiple conditions?

You can use boolean indexing with multiple conditions combined using & (AND) or | (OR) operators.

Example:

```
df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35], 'Score': [85, 90, 95]})

filtered = df[(df['Age'] > 25) & (df['Score'] > 90)]

print(filtered)
```

Output:

Name Age Score

2 Charlie 35 95

10. Explain the difference between .loc[] and .iloc[] in Pandas.

- .loc[]: Accesses rows and columns by label (index and column names).
- .iloc[]: Accesses rows and columns by integer-location-based indexing (positions).

Example:

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['x', 'y', 'z'])

 # Using .loc[] (by label)

print(df.loc['y', 'A'])

# Using .iloc[] (by position)

print(df.iloc[1, 0])
```

Output:

2

2

11. How can you apply a custom function to a Pandas DataFrame using apply()?

The apply() function in Pandas allows you to apply a custom function along an axis (rows or columns) of a DataFrame.

Example:

```
import pandas as pd

# Sample DataFrame

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Define a custom function

def add_ten(x):

 return x + 10

# Apply custom function to each element of a column

df['A'] = df['A'].apply(add_ten)

print(df)
```

Output:

A B

0 11 4

1 12 5

2 13 6

12. How do you handle categorical data in Pandas?

Categorical data in Pandas can be handled using the Categorical type or the astype() method to convert columns into categorical data types. Categorical data is efficient for memory and speed when working with data that has a limited number of distinct values.

Example:

df = pd.DataFrame({'Category': ['A', 'B', 'A', 'C']})

df['Category'] = df['Category'].astype('category')

print(df.dtypes)

Output:

Category category

dtype: object

You can also use the get_dummies() function to convert categorical data into one-hot encoding.

Example:

df = pd.DataFrame({'Category': ['A', 'B', 'A', 'C']})

df_encoded = pd.get_dummies(df['Category'])

print(df_encoded)

Output:

A B C

0 1 0 0

1 0 1 0

2 1 0 0

3 0 0 1

13. What is the difference between dropna() and fillna() in Pandas?

- **dropna()**: Removes missing values (NaN) from the DataFrame.
- **fillna()**: Replaces missing values (NaN) with a specified value or method (e.g., forward fill, backward fill).

Example:

df = pd.DataFrame({'A': [1, 2, None], 'B': [4, None, 6]})

Using dropna()

df_dropped = df.dropna()

print(df_dropped)

Using fillna()

df_filled = df.fillna(0)

print(df_filled)

Output:

dropna()

A B

0 1.0 4.0

fillna()

A B

0 1.0 4.0

1 2.0 0.0

2 0.0 6.0

14. How can you create a pivot table in Pandas?

You can use the pivot_table() function in Pandas to create a pivot table. It allows summarization of data based on one or more categorical columns.

Example:

df = pd.DataFrame({'Category': ['A', 'B', 'A', 'B'], 'Value': [10, 20, 30, 40]})

pivot_table = df.pivot_table(values='Value', index='Category', aggfunc='sum')

```
print(pivot_table)
```

Output:

```
 Value

Category

A 40

B 60
```

15. How would you remove duplicates in a Pandas DataFrame?

You can use the drop_duplicates() function in Pandas to remove duplicate rows from a DataFrame.

Example:

```
df = pd.DataFrame({'A': [1, 2, 2, 3], 'B': [4, 5, 5, 6]})

df_unique = df.drop_duplicates()

print(df_unique)
```

Output:

```
 A B

0 1 4

1 2 5

3 3 6
```

16. How do you merge two NumPy arrays element-wise?

You can use the numpy.add(), numpy.subtract(), or any other arithmetic operator to perform element-wise operations on two NumPy arrays.

Example:

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

# Element-wise addition

result = np.add(arr1, arr2)

print(result)
```

Output:

[5 7 9]

17. What is the role of numpy.reshape() in manipulating arrays?

The numpy.reshape() function is used to change the shape of a NumPy array without changing its data. It allows you to reorganize the elements into a new shape.

Example:

```
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = np.reshape(arr, (2, 3))
print(reshaped_arr)
```

Output:

```
[[1 2 3]
 [4 5 6]]
```

18. Explain how to compute the mean, median, and standard deviation using Pandas and NumPy.

- **Pandas**: Use .mean(), .median(), and .std() for Series or DataFrame.
- **NumPy**: Use numpy.mean(), numpy.median(), and numpy.std() for arrays.

Example:

```
import pandas as pd
import numpy as np
# Pandas
df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})
mean_pandas = df['A'].mean()
median_pandas = df['A'].median()
std_pandas = df['A'].std()
print(mean_pandas, median_pandas, std_pandas)
# NumPy
arr = np.array([1, 2, 3, 4, 5])
mean_numpy = np.mean(arr)
median_numpy = np.median(arr)
std_numpy = np.std(arr)
print(mean_numpy, median_numpy, std_numpy)
```

```
3.0 3.0 1.5811388300841898
3.0 3.0 1.4142135623730951
```

19. How do you perform element-wise mathematical operations on a Pandas DataFrame?

You can perform element-wise mathematical operations on a Pandas DataFrame directly using operators like +, -, *, /, etc.

Example:

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df_result = df + 10 # Add 10 to each element
print(df_result)
```

Output:

```
 A B
0 11 14
1 12 15
2 13 16
```

20. How can you perform data normalization using NumPy?

Data normalization can be done by scaling the data to a specific range, typically [0, 1], using the formula:

Example:

```
arr = np.array([10, 20, 30, 40, 50])
normalized_arr = (arr - arr.min()) / (arr.max() - arr.min())
print(normalized_arr)
```

Output:

```
[0. 0.25 0.5 0.75 1. ]
```

21. What is the purpose of the crosstab() function in Pandas?

The crosstab() function computes a cross-tabulation of two (or more) factors. It is useful for summarizing data by creating contingency tables that show the frequency of different combinations of categories.

Example:

```
import pandas as pd

# Sample DataFrame

df = pd.DataFrame({'Category': ['A', 'B', 'A', 'B', 'C', 'A'],

 'Value': [10, 20, 10, 20, 30, 30]})

# Crosstab function

crosstab_result = pd.crosstab(df['Category'], df['Value'])

print(crosstab_result)
```

Output:

```
Value 10 20 30

Category

A 2 0 1

B 0 2 0

C 0 0 1
```

22. How can you handle time-series data in Pandas?

Pandas provide powerful tools for handling time-series data, such as to_datetime() to convert a column to DateTime format, and resampling methods like .resample() to perform operations such as aggregation.

Example:

```
import pandas as pd

# Sample time-series data

data = {'Date': ['2024-01-01', '2024-01-02', '2024-01-03'],

 'Value': [10, 20, 30]}

df = pd.DataFrame(data)
```

```
# Convert the 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])
# Set 'Date' as the index
df.set_index('Date', inplace=True)
# Resample by day and calculate the sum
resampled_df = df.resample('D').sum()
print(resampled_df)
```

Output:

```
            Value
Date
2024-01-01     10
2024-01-02     20
2024-01-03     30
```

23. How would you calculate the correlation between two columns in a Pandas DataFrame?

You can calculate the correlation between two columns using the .corr() method, which computes the Pearson correlation coefficient by default.

Example:

```
df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [4, 3, 2, 1]})
correlation = df['A'].corr(df['B'])
print(correlation)
```

Output:

```
-1.0
```

24. How can you filter rows in a Pandas DataFrame where a specific column's value is greater than a certain threshold?

You can filter rows by using boolean indexing where the column's values meet the condition.

Example:

```
df = pd.DataFrame({'A': [10, 20, 30, 40], 'B': [5, 15, 25, 35]})

filtered_df = df[df['A'] > 20]

print(filtered_df)
```

Output:

```
 A B

2 30 25

3 40 35
```

25. What is the difference between numpy.concatenate() and numpy.vstack()?

- **numpy.concatenate()**: Joins two or more arrays along an existing axis (specified by the axis parameter).
- **numpy.vstack()**: Stacks arrays vertically (along rows), i.e., add new rows to the existing array.

Example:

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

# Using concatenate

concatenated = np.concatenate([arr1, arr2])

print(concatenated)

# Using vstack

stacked = np.vstack([arr1, arr2])

print(stacked)
```

Output:

```
# concatenate

[1 2 3 4 5 6]

# vstack

[[1 2 3]

 [4 5 6]]
```

26. How do you create a random NumPy array of integers between two values?

You can use numpy.random.randint() to generate random integers within a specified range.

Example:

random_array = np.random.randint(1, 10, size=5) # Generates 5 random integers between 1 and 10

print(random_array)

Output::

[5 2 7 4 8]

27. How do you calculate a rolling mean in Pandas?

You can calculate a rolling mean using the .rolling() method followed by .mean() to compute the moving average.

Example:

df = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6]})

rolling_mean = df['A'].rolling(window=3).mean()

print(rolling_mean)

Output:

0 NaN

1 NaN

2 2.0

3 3.0

4 4.0

5 5.0

Name: A, dtype: float64

28. What is the difference between np.mean() and np.median() in NumPy?

- **np.mean()**: Calculates the arithmetic mean (average) of the values in an array.
- **np.median()**: Calculates the median value (middle value) of the values in an array.

Example:

arr = np.array([1, 2, 3, 4, 5])

```
mean_value = np.mean(arr)

median_value = np.median(arr)

print(mean_value, median_value)
```

Output:

3.0 3.0

29. How can you concatenate two DataFrames along rows and columns in Pandas?

- **Along rows**: Use pd.concat() with axis=0.
- **Along columns**: Use pd.concat() with axis=1.

Example:

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

# Concatenate along rows

df_rows = pd.concat([df1, df2], axis=0)

print(df_rows)

# Concatenate along columns

df_columns = pd.concat([df1, df2], axis=1)

print(df_columns)
```

Output:

```
# Along rows

 A B

0 1 3

1 2 4

0 5 7

1 6 8

# Along columns

 A B A B

0 1 3 5 7
```

1 2 4 6 8

30. Explain how to handle outliers in a dataset using Pandas and NumPy.

Outliers can be handled by removing or replacing them. You can identify outliers by calculating statistical measures like Z-scores or using percentile-based thresholds (e.g., using IQR method). Once identified, you can either replace them or remove them.

Example:

```
# Z-score method
from scipy import stats
df = pd.DataFrame({'A': [1, 2, 3, 100, 5]})
z_scores = stats.zscore(df['A'])
df_no_outliers = df[(z_scores < 3)] # Removing rows with outliers
print(df_no_outliers)
```

Output:

A

0 1

1 2

2 3

4 5

31. How do you sort a Pandas DataFrame based on multiple columns?

You can use the .sort_values() function to sort a DataFrame by multiple columns by passing a list of column names.

Example:

```
df = pd.DataFrame({'A': [3, 1, 2], 'B': [4, 5, 6]})
sorted_df = df.sort_values(by=['A', 'B'])
print(sorted_df)
```

Output::

A B

1 1 5

2 2 6

0 3 4

32. How can you apply vectorized operations in NumPy to improve performance?

Vectorized operations in NumPy allow you to perform operations on arrays without explicit for-loops, which is computationally efficient. These operations use underlying C code for faster execution.

Example:

arr1 = np.array([1, 2, 3, 4])

arr2 = np.array([5, 6, 7, 8])

Vectorized addition (avoiding loops)

result = arr1 + arr2

print(result)

Output:

[6 8 10 12]

33. How would you merge two Pandas DataFrames with different column names?

You can merge DataFrames with different column names using the left_on and right_on parameters in the merge() function.

Example:

df1 = pd.DataFrame({'A': [1, 2, 3], 'B': ['X', 'Y', 'Z']})

df2 = pd.DataFrame({'C': [4, 5, 6], 'D': ['P', 'Q', 'R']})

merged_df = pd.merge(df1, df2, left_on='A', right_on='C')

print(merged_df)

Output:

A B C D

0 1 X 4 P

1 2 Y 5 Q

2 3 Z 6 R

34. How can you perform the element-wise comparison between two NumPy arrays?

Element-wise comparison between two arrays can be performed using comparison operators like ==, !=, <, >, etc.

Example:

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([3, 2, 1])

result = arr1 == arr2

print(result)
```

Output:

```
[False True False]
```

35. How would you handle missing values in NumPy arrays?

You can handle missing values in NumPy arrays by using np.nan for missing values and using functions like np.isnan(), np.nanmean(), np.nanstd() to handle these values.

Example:

```
arr = np.array([1, 2, np.nan, 4, 5])

# Checking for NaN values

nan_values = np.isnan(arr)

print(nan_values)

# Replacing NaN with mean

mean_value = np.nanmean(arr)

arr[np.isnan(arr)] = mean_value

print(arr)
```

Output:

```
[False False True False False]

[1. 2. 3. 4. 5.]
```

36. What is the purpose of numpy.nanmean() and numpy.nansum()?

- **numpy.nanmean()**: Computes the mean of an array, ignoring NaN values.

- **numpy.nansum()**: Computes the sum of an array, ignoring NaN values.

Example:

```
import numpy as np

arr = np.array([1, 2, np.nan, 4, 5])

# Calculate mean ignoring NaN

mean_value = np.nanmean(arr)

print(mean_value) # Output: 3.0

# Calculate sum ignoring NaN

sum_value = np.nansum(arr)

print(sum_value) # Output: 12
```

37. How can you generate a sequence of numbers in NumPy using numpy.arange()?

numpy.arange() generates an array of evenly spaced values within a specified range.

Example:

```
arr = np.arange(0, 10, 2)

print(arr) # Output: [0 2 4 6 8]
```

38. How do you create a Pandas DataFrame from a CSV file?

You can use the pd.read_csv() function to read a CSV file and create a Pandas DataFrame.

Example:

```
import pandas as pd

# Reading a CSV file

df = pd.read_csv('data.csv')

print(df.head())
```

39. What is the difference between np.dot() and np.matmul()?

- **np.dot()**: Computes the dot product of two arrays. It works for both 1D and 2D arrays.
- **np.matmul()**: Performs matrix multiplication. It is equivalent to the @ operator and works for 2D arrays or higher-dimensional arrays (like tensors).

Example:

```
# Dot product
a = np.array([1, 2])
b = np.array([3, 4])
dot_product = np.dot(a, b)
print(dot_product) # Output: 11
# Matrix multiplication
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
matmul_result = np.matmul(A, B)
print(matmul_result)
```

Output:

```
# [[19 22]
# [43 50]]
```

40. How do you apply a lambda function to a Pandas DataFrame column?

You can use the apply() function to apply a lambda function to a column in a Pandas DataFrame.

Example:

```
df = pd.DataFrame({'A': [1, 2, 3, 4]})
df['B'] = df['A'].apply(lambda x: x * 2)
print(df)
```

Output:

```
 A B
0 1 2
1 2 4
2 3 6
3 4 8
```

41. How do you handle and work with dates and times in Pandas?

Pandas provides the pd.to_datetime() function to convert strings to datetime objects. You can also manipulate date and time with methods like .dt.

Example:

```
# Convert string to datetime
df = pd.DataFrame({'Date': ['2024-01-01', '2024-01-02']})
df['Date'] = pd.to_datetime(df['Date'])
# Extract year and month
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
print(df)
```

Output:

```
 Date Year Month
0 2024-01-01 2024 1
1 2024-01-02 2024 1
```

42. What is the purpose of numpy.random.rand() in generating random values?

numpy.random.rand() generates random values from a uniform distribution over [0, 1).

Example:

```
rand_values = np.random.rand(3, 2)
print(rand_values)
```

Output:

A 3x2 array of random values between 0 and 1

43. How would you use NumPy to compute the dot product of two vectors?

You can use np.dot() to compute the dot product of two vectors.

Example:

```
a = np.array([1, 2])
b = np.array([3, 4])
```

```
dot_product = np.dot(a, b)

print(dot_product) # Output: 11
```

44. Explain the concept of "vectorization" in NumPy and why it's beneficial.

Vectorization refers to performing operations on entire arrays (vectors) without explicit loops. It is more efficient because it leverages low-level optimizations in NumPy for faster computation.

Example:

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

# Without vectorization (using a loop)

result = [a * b for a, b in zip(arr1, arr2)]

print(result) # Output: [4, 10, 18]

 # With vectorization

result_vectorized = arr1 * arr2

print(result_vectorized) # Output: [4 10 18]
```

Vectorization is beneficial because it speeds up operations by avoiding loops and taking advantage of optimized libraries.

45. How would you calculate the standard deviation of a column in a Pandas DataFrame?

You can use the .std() method to calculate the standard deviation of a column.

Example:

```
df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})

std_dev = df['A'].std()

print(std_dev) # Output: 1.5811388300841898
```

46. How can you count the occurrences of unique values in a Pandas DataFrame column?

You can use the .value_counts() method to count unique values in a column.

Example:

```
df = pd.DataFrame({'A': ['cat', 'dog', 'cat', 'cat', 'dog']})

value_counts = df['A'].value_counts()

print(value_counts)
```

Output:

```
cat 3

dog 2

Name: A, dtype: int64
```

47. How can you perform matrix multiplication using NumPy arrays?

You can use np.matmul() or np.dot() to perform matrix multiplication.

Example:

```
A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

 matrix_product = np.matmul(A, B)

print(matrix_product)
```

Output:

```
# [[19 22]

# [43 50]]
```

48. What is the use of numpy.unique() in NumPy?

numpy.unique() returns the sorted unique elements of an array.

Example:

```
arr = np.array([1, 2, 2, 3, 3, 3, 4])

unique_values = np.unique(arr)

print(unique_values) # Output: [1 2 3 4]
```

49. How do you handle duplicates in a Pandas DataFrame based on multiple columns?

You can use the .drop_duplicates() method and specify the subset of columns to check for duplicates.

Example:

```
df = pd.DataFrame({'A': [1, 2, 2, 3], 'B': [4, 5, 5, 6]})

df_no_duplicates = df.drop_duplicates(subset=['A', 'B'])

print(df_no_duplicates)
```

Output:

```
 A B

0 1 4

1 2 5

2 3 6
```

50. What are some ways to optimize the performance of large datasets in Pandas?

To optimize the performance of large datasets:

1. **Use categorical data types** for columns with repeating values.
2. **Use efficient file formats** like Parquet or Feather instead of CSV for large datasets.
3. **Load data in chunks** with pd.read_csv(chunk_size=...) when working with large files.
4. **Vectorize operations** instead of using loops.
5. **Reduce memory usage** by selecting appropriate data types for columns (e.g., using float32 instead of float64).

Example:

```
# Use categorical data type to optimize memory usage

df['Category'] = df['Category'].astype('category')
```