Data Science libraries content

Interview Questions

(Practice Project)

PW SKILLS

Theoretical Questions

1. What is the difference between Series and DataFrame in Pandas?

Answer:

A Series is a one-dimensional array-like object in Pandas that can hold data of any type (integers, strings, floating point numbers, etc.) and has an associated index. It is similar to a list or a dictionary in Python, but it provides additional functionality for data manipulation.

A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). It can be thought of as a collection of Series objects sharing the same index. DataFrames are more versatile and are used for handling structured data, making them suitable for data analysis tasks.

2. What are NumPy arrays, and why are they preferred over Python lists for data operations?

Answer:

NumPy arrays are powerful, multi-dimensional arrays provided by the NumPy library in Python. They are preferred over Python lists for several reasons:

- **Performance:** NumPy arrays are implemented in C and optimized for performance, making them faster for numerical computations.
- **Memory Efficiency:** NumPy arrays consume less memory compared to Python lists due to their fixed data type.
- **Functionality:** NumPy provides a wide range of mathematical functions that can be applied directly to arrays, enabling vectorized operations that are not possible with lists.

3. What is broadcasting in NumPy?

Answer:

Broadcasting in NumPy refers to the ability of NumPy to perform arithmetic operations on arrays of different shapes and sizes. When performing operations, NumPy automatically expands the smaller array to match the shape of the larger array without making copies of the data. This allows for efficient computation and simplifies code. For example, adding a scalar value to a NumPy array will add that value to each element of the array.

4. What is the difference between .loc and .iloc in Pandas?

Answer:

- **.loc:** This indexer is used for label-based indexing. It allows you to access a group of rows and columns by labels or a boolean array. For example, df.loc[0:5, 'column_name'] selects rows with labels from 0 to 5 and the specified column.
- **.iloc:** This indexer is used for integer-based indexing. It allows you to access rows and columns by their integer position. For example, df.iloc[0:5, 1] selects the first five rows and the second column based on integer indices.

5. Explain the concept of "axis" in NumPy and Pandas operations.

Answer:

In NumPy and Pandas, the concept of "axis" refers to the direction along which operations are performed:
Axis 0 refers to the vertical direction (rows).
Axis 1 refers to the horizontal direction (columns).
When using methods like DataFrame.sum(), specifying axis=0 sums across rows (for each column), while axis=1 sums across columns (for each row).

5. Explain the concept of "axis" in NumPy and Pandas operations.

Answer:

In NumPy and Pandas, the concept of "axis" refers to the direction along which operations are performed:
Axis 0 refers to the vertical direction (rows).
Axis 1 refers to the horizontal direction (columns).
When using methods like DataFrame.sum(), specifying axis=0 sums across rows (for each column), while axis=1 sums across columns (for each row).

6. What are the different types of joins available in Pandas?

Answer:

Pandas supports several types of joins when merging DataFrames:

- **Inner Join:** Returns only the rows with keys that are present in both DataFrames.
- **Outer Join:** Returns all rows from both DataFrames, filling in NaN for missing values.
- **Left Join:** Returns all rows from the left DataFrame and matched rows from the right DataFrame, filling in NaN for unmatched rows.
- **Right Join:** Returns all rows from the right DataFrame and matched rows from the left DataFrame, filling in NaN for unmatched rows.

7. What is the significance of the reshape function in NumPy?

Answer:

The reshape function in NumPy is used to change the shape of an existing array without changing its data. This is significant because it allows for flexibility in manipulating data structures to fit the requirements of various operations or algorithms. For example, you can convert a 1D array into a 2D array, making it easier to perform matrix operations.

8. Explain the importance of data aggregation in Pandas and give examples.

Answer:

Data aggregation in Pandas is the process of summarizing data by grouping and applying functions to obtain insights. It is important for:

- **Data Analysis:** Aggregation helps in understanding trends and patterns in the data.
- **Performance Improvement:** It reduces the size of the dataset for easier handling.

Examples include:

- Using groupby() to calculate the average sales per region: df.groupby('region')['sales'].mean().
- Aggregating multiple statistics at once: df.groupby('category').agg({'sales': ['sum', 'mean'], 'profit': 'max'}).

9. Explain the concept of vectorization in NumPy and its benefits.

Answer:

Vectorization in NumPy refers to the ability to perform operations on entire arrays rather than element-by-element. This is achieved through the use of NumPy's optimized C code, which allows for faster computations. Benefits include:

- **Performance:** Vectorized operations are significantly faster than using loops in Python.
- **Simplicity:** Code becomes more concise and easier to read, as operations can be applied directly to arrays without explicit iteration.

10. What are categorical plots in Seaborn, and when would you use them?

Answer:

Categorical plots in Seaborn are visualizations that display the relationship between categorical variables. They include plots like bar plots, box plots, and violin plots. We would use them :

- When we want to compare distributions of categorical data.
- When we want to visualize the relationship between a categorical variable and a continuous variable.
- When we aim to summarize data across categories to identify patterns or trends.

11. How does the 'groupby' function work in Pandas, and what are its common use cases?

Answer:

The groupby function in Pandas is used to split the data into groups based on some criteria, apply a function to each group, and combine the results. Common use cases include:

- **Aggregating Data:** Summarizing data by calculating statistics like mean, sum, or count for each group.
- **Transforming Data:** Applying transformations to each group, such as normalizing or scaling.
- **Filtering Data:** Selecting groups that meet certain criteria.

Example: df.groupby('category')['sales'].sum() calculates total sales for each category.

12. Describe the concept of linear algebra operations in NumPy.

Answer:

Linear algebra operations in NumPy involve mathematical computations with vectors and matrices, including:

- **Matrix Multiplication:** Using np.dot() or the @ operator to multiply matrices.
- Determinants and Inverses: Using functions like np.linalg.det() and np.linalg.inv() to compute determinants and inverses of matrices.
- **Eigenvalues and Eigenvectors:** Using np.linalg.eig() to compute eigenvalues and eigenvectors of a matrix.

These operations are fundamental in many scientific and engineering applications, including machine learning algorithms.

13. How can you perform time series analysis using Pandas?

Answer:

Time series analysis in Pandas can be performed using the following steps:

- **Date Parsing:** Convert date strings to datetime objects using pd.to_datetime().
- **Setting Date as Index:** Set the datetime column as the index of the DataFrame for easier slicing and manipulation.
- **Resampling:** Use the resample() method to aggregate data over different time periods (e.g., daily, monthly).
- **Rolling Windows:** Apply rolling functions like rolling().mean() to compute moving averages.
- **Visualization:** Use plotting libraries to visualize trends and seasonal patterns.

Example: df.set_index('date').resample('M').mean() computes monthly averages.

14. What are some ways to optimize Seaborn plots for large datasets?

Answer:

To optimize Seaborn plots for large datasets, you can:

- **Downsample the Data:** Reduce the number of data points by sampling or aggregating.
- **Use the scatterplot with size Parameter:** Instead of plotting all points, use the size parameter to represent the density of points.
- **Use hue for Categorical Variables:** Instead of plotting every point, use color to represent categories, reducing clutter.
- **Limit the Number of Unique Values:** For categorical variables, limit the number of unique values displayed to avoid overcrowding.

Coding Questions

1. **Write a Pandas code snippet to create a DataFrame from a dictionary.**

Ans:

```
import pandas as pd

# Create a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)
```

2. Generate a random array of 10 numbers between 0 and 1 using NumPy.

Ans:

```
import numpy as np

# Generate a random array of 10 numbers between 0 and 1
random_array = np.random.rand(10)
print(random_array)
```

3. Create a bar plot using Seaborn to visualize the counts of categories in a dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

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

# Create a bar plot
sns.countplot(x='Category', data=df)
plt.title('Counts of Categories')
plt.show()
```

4. Slice the first three rows and columns from a NumPy array.

```
# Create a sample NumPy array
array = np.random.rand(5, 5)

# Slice the first three rows and columns
sliced_array = array[:3, :3]
print(sliced_array)
```

5. Plot a simple line graph using Matplotlib.

```
# Sample data for plotting
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a line plot
plt.plot(x, y)
plt.title('Simple Line Graph')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid()
plt.show()
```

6. Create a Pandas DataFrame from a list of lists and display the first 5 rows.

```
# Create a list of lists
data = [
    [1, 'Alice', 25],
    [2, 'Bob', 30],
    [3, 'Charlie', 35],
    [4, 'David', 40],
    [5, 'Eve', 45],
    [6, 'Frank', 50]
]

# Create a DataFrame from the list of lists
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
print(df.head(5))
```

7. Create a scatter plot using Matplotlib with custom labels and a title.

```
# Sample data for scatter plot
x = np.random.rand(50)
y = np.random.rand(50)

# Create a scatter plot
plt.scatter(x, y, color='blue', marker='o')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.grid()
plt.show()
```

8. Convert a Pandas DataFrame column to a NumPy array.

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

# Convert column 'A' to a NumPy array
array_A = df['A'].to_numpy()
print(array_A)
```

9. Create a subplot with 2 rows and 2 columns using Matplotlib and plot different types of graphs.

```
# Create a 2x2 subplot
fig, axs = plt.subplots(2, 2)

# Line plot
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Line Plot')

# Scatter plot
axs[0, 1].scatter(x, y)
axs[0, 1].set_title('Scatter Plot')

# Bar plot
axs[1, 0].bar(['A', 'B', 'C'], [10, 20, 15])
axs[1, 0].set_title('Bar Plot')

# Histogram
axs[1, 1].hist(np.random.randn(100), bins=10)
axs[1, 1].set_title('Histogram')

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

10. Use Seaborn to visualize the distribution of data with a kernel density plot (KDE).

```
# Sample data
data = np.random.normal(size=100)

# Create a KDE plot
sns.kdeplot(data, fill=True)
plt.title('Kernel Density Estimate')
plt.xlabel('Value')
plt.ylabel('Density')
plt.show()
```

11. Given a DataFrame, select the rows where 'Category' is 'A' and only the 'Value' and 'Date' columns. Write the code for this.

```
# Sample DataFrame
data = {
    'Category': ['A', 'B', 'A', 'C'],
    'Value': [10, 20, 30, 40],
    'Date': pd.to_datetime(['2021-01-01', '2021-01-02',
'2021-01-03', '2021-01-04'])
}
df = pd.DataFrame(data)

# Select rows where 'Category' is 'A' and only 'Value' and
'Date' columns
result = df.loc[df['Category'] == 'A', ['Value', 'Date']]
print(result)
```

12. Implement a function in Pandas to calculate the rolling average with a window size of 7 days.

```
# Sample DataFrame with a date range
dates = pd.date_range(start='2021-01-01', periods=10)
data = {'Values': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
df = pd.DataFrame(data, index=dates)

# Calculate the rolling average with a window size of 7
days
df['Rolling_Avg'] = df['Values'].rolling(window=7).mean()
print(df)
```

13. Implement a Pandas code to perform a complex groupby operation followed by an aggregation and transformation.Make your own data

```
# Sample DataFrame
data = {
    'Category': ['A', 'A', 'B', 'B', 'C', 'C', 'A', 'B'],
    'Value': [10, 20, 30, 40, 50, 60, 70, 80]
}
df = pd.DataFrame(data)

# Group by 'Category' and calculate the sum and mean
grouped = df.groupby('Category').agg({'Value': ['sum',
'mean']})

# Transform: Add a new column with the difference from the
mean
grouped['Difference_From_Mean'] = grouped['Value']['mean']
- grouped['Value']['sum']
print(grouped)
```