    - Custom roles can be created to fit specific needs.
- **Data Encryption**:
    - Enable TLS/SSL for secure data in transit.
    - Use MongoDB's built-in encryption for data at rest.
- **Network Security**:
    - Limit network access to the MongoDB instance and use firewalls to control traffic.

11. MongoDB Administration and Monitoring

- **Backup and Restore**:
    - Use `mongodump` and `mongorestore` for data backup and restoration.
    - For production, consider incremental backups and backup replicas.
- **Performance Monitoring**:
    - Use `mongostat` and `mongotop` for real-time statistics.
    - Leverage the MongoDB Monitoring Service (MMS) for advanced metrics.
- **Scaling**:
    - For scaling, consider horizontal scaling (sharding) or vertical scaling (adding resources to existing instances).
    - Optimize indexes, queries, and schema design to improve performance.

12. MongoDB Atlas

- **What is MongoDB Atlas?**
  MongoDB's fully managed cloud database service, offering automated backups, scaling, and security

Give pandas notes for freshers interview in detail

Here's a detailed set of notes on **Pandas** that covers fundamental concepts for a fresher's interview:

---

1. Introduction to Pandas

- **What is Pandas?**
  Pandas is an open-source data analysis and manipulation library in Python, built on top of NumPy. It provides data structures and operations to work with structured data seamlessly.
- **Key Features**:
  - Supports data alignment, merging, joining, and reshaping.
  - Provides data cleaning, filtering, and aggregation tools.
  - Allows easy handling of time-series data.
- **Data Structures in Pandas**:
  - **Series**: One-dimensional labeled array capable of holding data of any type (integers, strings, etc.).
  - **DataFrame**: Two-dimensional labeled data structure, similar to a table or spreadsheet in Excel, with rows and columns.

---

2. Pandas Data Structures

- **Series**:
  - Created using `pd.Series(data, index)`.
  - Can hold any data type, indexed by a default or custom index.
  - Basic Operations:
    - Access elements by index using `.loc` (label-based) and `.iloc` (integer-based) indexing.
    - Perform arithmetic operations, with or without broadcasting.

```python
import pandas as pd
s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print(s['a'])  # Output: 1
```

- **DataFrame**:
  - Created using `pd.DataFrame(data, columns, index)`.

- Columns represent features (or variables), and rows represent records (or observations).
- Supports different types of data (numerical, categorical, etc.) in different columns.
- Basic Operations:
  - Column selection: `df['column_name']`
  - Row selection: `.loc[]` for label-based, `.iloc[]` for index-based.

```python
data = {'Name': ['Alice', 'Bob'], 'Age': [24, 27]}
df = pd.DataFrame(data)
print(df['Name'])  # Output: Alice, Bob
```

---

3. Data Loading and Exporting

- **Reading Data**:
  - **CSV**: `pd.read_csv('file.csv')`
  - **Excel**: `pd.read_excel('file.xlsx')`
  - **SQL**: `pd.read_sql(query, connection)`
  - **JSON**: `pd.read_json('file.json')`
- **Writing Data**:
  - **CSV**: `df.to_csv('file.csv')`
  - **Excel**: `df.to_excel('file.xlsx')`
  - **SQL**: `df.to_sql('table', connection)`
- **Additional Parameters**:
  - `sep`: Delimiter for `read_csv()`.
  - `index_col`: Column to use as index.
  - `header`: Row to use as column names.

---

4. DataFrame Manipulation

- **Viewing Data**:
  - `.head(n)`: Returns the first `n` rows.
  - `.tail(n)`: Returns the last `n` rows.
  - `.info()`: Provides summary of data (type, null values).
  - `.describe()`: Gives summary statistics for numerical columns.
- **Selecting Data**:
  - **Single Column**: `df['column_name']` or `df.column_name`.
  - **Multiple Columns**: `df[['col1', 'col2']]`.
  - **Rows by Index**:
    - **By Label**: `df.loc[row_index]`.
    - **By Position**: `df.iloc[row_position]`.
- **Filtering Data**:
  - **Condition-Based Filtering**: `df[df['column'] > value]`
  - **Multiple Conditions**: Use `&` (AND) or `|` (OR) with parentheses.

```python
# Rows where Age > 25 and Gender == 'Male'
df[(df['Age'] > 25) & (df['Gender'] == 'Male')]
```

---

5. Data Cleaning

- **Handling Missing Data**:
  - **Detecting Missing Values**: `df.isnull()` or `df.notnull()`
  - **Filling Missing Values**:
    - `df.fillna(value)`: Replaces `NaN` values with `value`.
    - `df.fillna(method='ffill' or 'bfill')`: Forward or backward fill.
  - **Dropping Missing Values**:

    - `df.dropna()`: Drops rows with `NaN` values.
    - `df.dropna(axis=1)`: Drops columns with `NaN` values.
- **Removing Duplicates**:
    - `df.duplicated()`: Returns a boolean Series marking duplicates.
    - `df.drop_duplicates()`: Removes duplicate rows.
- **Data Type Conversion**:
    - **Change Data Type**: `df['column'] = df['column'].astype(new_type)`.
    - Common conversions: `str`, `int`, `float`, `datetime`.
- **Renaming Columns**:
    - `df.rename(columns={'old_name': 'new_name'})`: Rename specific columns.

---

6. Data Transformation and Aggregation

- **Applying Functions**:
    - `apply()`: Apply a function along an axis.
        - `df['column'].apply(lambda x: x + 1)`
    - `map()`: Element-wise function mapping for Series.
        - `df['column'].map({old_val: new_val})`
    - `applymap()`: Apply function element-wise to the entire DataFrame.
- **Grouping and Aggregating Data**:
    - `df.groupby('column')`: Groups data by column.
    - Common aggregations: `sum()`, `mean()`, `count()`, `min()`, `max()`.
    - **Example**:

```python
# Average age by gender
df.groupby('Gender')['Age'].mean()
```

- **Pivot Tables**:

- `df.pivot_table(values, index, columns, aggfunc)`: Aggregates data and reshapes it for multi-dimensional analysis.
- Example:

```python
# Pivot table with average salary by gender and department
df.pivot_table(values='Salary', index='Department', columns='Gender', aggfunc='mean')
```

---

7. Merging and Concatenation

- **Concatenating DataFrames**:
  - `pd.concat([df1, df2], axis=0)`: Combines DataFrames along rows (vertical).
  - `pd.concat([df1, df2], axis=1)`: Combines DataFrames along columns (horizontal).
- **Merging DataFrames**:
  - `pd.merge(df1, df2, on='key')`: Combines DataFrames based on a common key column.
  - **Types of Joins**:
    - **Inner**: Only matching rows (default).
    - **Outer**: All rows from both DataFrames.
    - **Left**: All rows from the left DataFrame, with matching rows from the right.
    - **Right**: All rows from the right DataFrame, with matching rows from the left.

```python
# Example of merging
merged_df = pd.merge(df1, df2, on='ID', how='inner')
```

---

8. Working with Dates and Times

- **Datetime Conversion**:
  - Convert to datetime: `pd.to_datetime(df['date_column'])`
- **Extracting Date Components**:
  - `df['date_column'].dt.year`: Extracts year.
  - `df['date_column'].dt.month`: Extracts month.
  - `df['date_column'].dt.day`: Extracts day.
- **Date Range and Frequency**:
  - **Generating Date Range**: `pd.date_range(start, end, freq)`.
  - Common frequency aliases: `D` (daily), `W` (weekly), `M` (month-end), `Q` (quarter-end).

---

9. Data Visualization with Pandas

- **Basic Plotting**:
  - **Line Plot**: `df.plot()` or `df.plot.line()`.
  - **Bar Plot**: `df.plot.bar()`.
  - **Histogram**: `df['column'].plot.hist()`.
  - **Box Plot**: `df.plot.box()`.

```python

# Line plot of 'Sales' over time
df['Sales'].plot(title='Sales Over Time')
```

- **Customization**:
  - `title`, `xlabel`, `ylabel`: For chart labels.
  - `color`: Define color for each plot.
  - `grid=True`: Adds gridlines for better readability.

---

10. Advanced Pandas Operations

- **Reshaping Data**:
  - **Melt**: Unpivots DataFrame from wide to long format. `pd.melt(df, id_vars, value_vars)`.
  - **Stack and Unstack**: Rearranges data into hierarchical indexes.

```python
# Example of melting
pd.melt(df, id_vars=['Name'], value_vars=['Math', 'Science'])
```

- **Window Functions**:
  - **Rolling**: Computes a moving statistic (e.g., rolling average).
    - `df['

Give numpy notes for freshers interview in detail

Here's a detailed set of notes on **NumPy** for freshers preparing for an interview. These notes cover foundational concepts, functions, and operations in **NumPy**.

---

1. Introduction to NumPy

- **What is NumPy?**
  NumPy (Numerical Python) is a powerful library for numerical and scientific computing in Python. It provides support for arrays, matrices, and a range of mathematical functions to operate on these arrays.
- **Key Features**:
  - Efficiently handles large datasets and high-dimensional data.
  - Offers mathematical, logical, and statistical functions.
  - Integrates well with other libraries like Pandas, Matplotlib, and SciPy.
