Basic Level Questions

Load Data: Load a CSV file into a Pandas DataFrame.
Display Data: Display the first 5 and last 5 rows of the DataFrame.
Check Data Types: Check the data types of each column.
Handle Missing Values: Identify missing values and handle them using appropriate methods (e.g., imputation, deletion).
Remove Duplicates: Identify and remove duplicate rows.
Filter Data: Filter rows based on specific conditions.
Sort Data: Sort the DataFrame by one or more columns.
Group and Aggregate: Group data by a specific column and calculate summary statistics.
Rename Columns: Rename columns to more descriptive names.
Create New Columns: Create new columns based on existing columns using calculations or transformations.
Convert Data Types: Convert data types of columns (e.g., string to numeric).
Handle Outliers: Identify and handle outliers using techniques like capping or winsorization.
Text Cleaning: Clean text data by removing stop words, punctuation, and extra whitespace.
Date and Time Manipulation: Extract specific information from date and time columns (e.g., year, month, day).
Concatenate DataFrames: Concatenate multiple DataFrames vertically or horizontally.
Merge DataFrames: Merge DataFrames based on a common column.
Pivot Tables: Create pivot tables to summarize and analyze data.
Time Series Analysis: Handle time series data, calculate rolling averages, and perform time-based aggregations.
Visualize Data: Create basic visualizations using libraries like Matplotlib or Seaborn.
Export Data: Export the cleaned and processed data to a CSV or Excel file.

Medium Level Questions

Complex Filtering: Filter data based on multiple conditions and logical operations.
Text Analysis: Perform text analysis tasks like tokenization, stemming, and sentiment analysis.
Categorical Data Encoding: Encode categorical variables using techniques like one-hot encoding or label encoding.
Feature Engineering: Create new features from existing ones to improve model performance.
Data Normalization: Normalize numerical data to a common scale.
Data Standardization: Standardize numerical data to have a mean of 0 and a standard deviation of 1.
Time Series Forecasting: Use time series models to predict future values.
Statistical Tests: Conduct statistical tests to assess the significance of relationships between variables.
Machine Learning Pipelines: Build machine learning pipelines using libraries like scikit-learn.
Data Profiling: Analyze data quality, completeness, and consistency.
Data Validation: Validate data against business rules and constraints.
Data Security and Privacy: Implement techniques to protect sensitive data.
Data Integration: Combine data from multiple sources into a unified dataset.
Data Quality Assessment: Evaluate data quality and identify potential issues.
Data Reconciliation: Identify and resolve discrepancies between data sources.
Data Transformation: Transform data into a suitable format for analysis.
Data Enrichment: Add additional information to the dataset.
Data Visualization: Create advanced visualizations to explore data patterns.
Data Storytelling: Communicate data insights effectively through visualizations and narratives.
Interactive Data Visualization: Create interactive visualizations using libraries like Plotly or Dash.

Advanced Level Questions

Handling Imbalanced Data: Address class imbalance using techniques like oversampling, undersampling, or SMOTE.
Anomaly Detection: Identify outliers and anomalies using statistical methods or machine learning algorithms.
Natural Language Processing (NLP): Apply NLP techniques to unstructured text data.
Time Series Decomposition: Decompose time series data into trend, seasonal, and residual components.
Hierarchical Time Series Analysis: Analyze time series data with hierarchical structures.
Geospatial Data Analysis: Analyze geospatial data using libraries like GeoPandas.
Big Data Wrangling: Handle large datasets using tools like PySpark or Dask.
Data Governance: Implement data governance policies and procedures.
Data Ethics: Consider ethical implications of data collection, analysis, and usage.
Data Privacy Regulations: Adhere to data privacy regulations like GDPR and CCPA.
Data Security Best Practices: Implement security measures to protect data.
Cloud Data Warehousing: Design and implement cloud-based data warehouses.
Data Lake and Data Swamp: Understand the concepts and best practices for data lakes and data swamps.
Data Lineage: Track the origin and transformation of data.
Data Catalog: Create and maintain a data catalog to document data assets.
Data Monetization: Identify opportunities to monetize data assets.
Data Product Development: Develop data products that deliver value to customers.
Data-Driven Decision Making: Use data to inform strategic decisions.
A/B Testing: Design and analyze A/B tests to evaluate the impact of changes.
Machine Learning Model Deployment: Deploy machine learning models into production environments.

Note: To practice these questions, you can use a sample dataset like the Iris dataset or a larger dataset from Kaggle or other sources. Remember to explore different techniques and libraries to solve these problems effectively.

	Basic Level Questions
	Use pd.read_csv() to load the CSV file.
	Use df.head() and df.tail() to display the first and last 5 rows.
	Use df.dtypes to check data types.
	Use df.isnull().sum() to identify missing values and df.fillna(), df.dropna(), or imputation techniques to handle them.
	Use df.drop_duplicates() to remove duplicates.
	Use boolean indexing or df.query() to filter rows.
	Use df.sort_values() to sort data.
	Use df.groupby() and aggregation functions like mean, sum, count, etc.
	Use df.rename() to rename columns.
	Use arithmetic operations, string manipulation, or custom functions to create new columns.
	Use df.astype() to convert data types.
	Use techniques like capping, flooring, or IQR-based outlier detection and removal.
	Use regular expressions or libraries like NLTK to clean text data.
	Use pd.to_datetime() to convert to datetime format and extract information using dt accessor.
	Use pd.concat() to concatenate DataFrames.
	Use pd.merge() to merge DataFrames.
	Use pd.pivot_table() to create pivot tables.
	Use df.resample() for time-based aggregations and rolling() for rolling averages.
	Use matplotlib.pyplot or seaborn for visualizations.
	Use df.to_csv() or df.to_excel() to export data.
	Medium Level Questions
	Combine multiple conditions using logical operators like &, |, and ~.
	Use NLTK or spaCy for tokenization, stemming, and sentiment analysis.
	Use pd.get_dummies() for one-hot encoding or LabelEncoder for label encoding.
	Create new features based on domain knowledge or statistical techniques.
	Use MinMaxScaler, StandardScaler, or RobustScaler for normalization and standardization.
	Use time series forecasting models like ARIMA, Prophet, or LSTM.

	Use statistical tests like t-tests, ANOVA, or chi-squared tests.
	Use Pipeline to create machine learning pipelines.
	Use pandas_profiling or custom functions for data profiling.
	Use validation rules and assertions to validate data.
	Implement encryption, access controls, and data masking.
	Integrate data from various sources using ETL tools or APIs.
	Use data quality metrics like accuracy, completeness, and consistency.
	Identify and resolve data inconsistencies and anomalies.
	Use data reconciliation techniques like fuzzy matching and probabilistic matching.
	Use data transformation techniques like aggregation, filtering, and sorting.
	Use external data sources like APIs or databases to enrich data.
	Use advanced visualization techniques like interactive dashboards and geospatial maps.
	Use storytelling techniques to communicate insights effectively.
	Use Plotly or Dash to create interactive visualizations.
	Advanced Level Questions
	Use techniques like SMOTE, ADASYN, or class weighting.
	Use statistical methods like Z-score or IQR-based outlier detection or machine learning algorithms like Isolation Forest or One-Class SVM.
	Use NLP techniques like text classification, sentiment analysis, and topic modeling.
	Use time series decomposition techniques like STL decomposition.
	Use hierarchical time series models like hierarchical ARIMA or hierarchical LSTM.
	Use GeoPandas for geospatial data analysis.
	Use PySpark or Dask for big data processing.
	Implement data governance policies and procedures.
	Consider ethical implications of data usage and potential biases.
	Adhere to data privacy regulations and best practices.
	Implement security measures like encryption, access controls, and regular security audits.
	Design and implement cloud-based data warehouses on platforms like AWS Redshift or GCP BigQuery.
	Understand the differences between data lakes and data swamps and their appropriate use cases.
	Track data lineage using tools like Apache Airflow or Luigi.

	Create data catalogs using tools like Amundsen or DataHub.				
	Identify monetization opportunities, such as data licensing or data products.				
	Develop data products like dashboards, APIs, or machine learning models.				
	Use data-driven insights to inform strategic decisions.				
	Design A/B tests, collect data, and analyze results using statistical methods.				
	Use tools like MLflow or Kubeflow to deploy machine learning model				

I'll put together 60 data-wrangling questions for you, covering easy, medium, and hard levels, using Python's built-in datasets and popular public ones like the Titanic dataset. Let's start with questions for each difficulty level, and I'll provide answers for each one afterward.

Dataset: Titanic (readily available via `seaborn` or `kaggle`)

Easy (20 Questions)

1. **View the first few rows of the dataset.**
2. **Find the number of null values in each column.**
3. **Count the number of unique values in the `Embarked` column.**
4. **Filter rows where passengers are below the age of 10.**
5. **Replace all null values in the `Age` column with the median age.**
6. **Convert the `Sex` column to numeric values (0 for male, 1 for female).**
7. **Create a new column that calculates the family size by adding `SibSp` and `Parch`.**
8. **Sort the data by `Fare` in descending order.**
9. **Group data by `Pclass` and calculate the average fare.**
10. **Find the count of male and female passengers in each `Pclass`.**
11. **Rename the `SibSp` column to `Siblings_Spouse`.**
12. **Extract the titles (like Mr., Mrs., Miss) from the `Name` column.**
13. **Convert the `Fare` column to integer type.**
14. **Filter rows where `Pclass` is 1 and `Age` is above 30.**
15. **Count the number of survivors (1) and non-survivors (0) in the `Survived` column.**
16. **Drop duplicate rows based on `Name` and `Age`.**
17. **Get the minimum and maximum values in the `Age` column.**
18. **Identify if there are any duplicate rows in the dataset.**
19. **Calculate the mean age of passengers grouped by `Embarked`.**
20. **Generate a simple summary of the data using `.describe()`.**

Medium (20 Questions)

1. **Fill missing values in the `Embarked` column with the most common port.**
2. **Extract the deck level from the `Cabin` column (e.g., A, B, C).**
3. **Create a new binary column `Is_Alone` (1 if `SibSp` and `Parch` are 0, else 0).**
4. **Drop columns `Ticket` and `Cabin`.**
5. **Bin the `Age` column into age groups (e.g., 0–10, 11–20, etc.).**
6. **Create a new column `Fare_Per_Person` by dividing `Fare` by the number of family members (SibSp + Parch + 1).**
7. **Check for outliers in the `Fare` column using the IQR method.**
8. **Standardize the `Fare` column (subtract mean and divide by standard deviation).**
9. **Group by `Sex` and `Pclass` and calculate the survival rate for each group.**
10. **Calculate the correlation between `Fare`, `Age`, and `Survived`.**
11. **Pivot the dataset to show the average `Age` of passengers grouped by `Pclass` and `Sex`.**
12. **Sort the dataset first by `Pclass` and then by `Age`.**
13. **Find the age group with the highest survival rate.**
14. **Count the number of passengers in each age group.**
15. **Plot the survival count based on the `Pclass` column.**
16. **Calculate the median fare per `Embarked` port and fill missing `Fare` values with the median fare.**
17. **Filter out rows where passengers are below 18 and in the third class.**
18. **Group by `Embarked` and calculate both the mean and standard deviation of `Fare`.**
19. **Convert the `Name` column to lowercase.**
20. **Replace any numeric values in the `Name` column with an empty string.**

Hard (20 Questions)

1. **Create a function to extract the surname from the `Name` column.**
2. **Implement a custom function to categorize passengers into "Child", "Adult", and "Senior" based on `Age`.**
3. **Encode the `Embarked` column using one-hot encoding.**
4. **Standardize all numeric columns in the dataset.**
5. **Use groupby and apply a lambda function to find the mean age of survivors and non-survivors in each `Pclass`.**
6. **Add a column that assigns a unique identifier for each passenger with the format "Passenger_X" (X = row index).**
7. **Implement a rolling mean on the `Fare` column with a window size of 5.**
8. **Identify the top 5 surnames with the most survivors.**
9. **Perform a cross-tabulation of `Survived` and `Pclass`.**
10. **Calculate the z-score of `Fare` and flag outliers as True or False.**
11. **Replace `Age` null values using predictive imputation with a basic regression model on `Pclass` and `Fare`.**
12. **Identify the most common fare price by rounding and grouping `Fare` values.**
13. **Create a boolean column marking if a passenger paid above the average `Fare` for their class.**
14. **Bin `Fare` into quartiles and count the number of survivors in each bin.**
15. **Group by `Embarked`, then get the minimum and maximum `Age` for each port.**
16. **Create a column that assigns a unique ID for each family group based on `Surname` and `Pclass`.**
17. **Determine which cabin deck (A, B, C, etc.) had the highest survival rate.**
18. **Write a function that inputs a cabin string and outputs deck level and room number. Apply it to `Cabin`.**
19. **Use `.apply()` to find each passenger's family group and mark if anyone in that family survived.**
20. **Combine `Pclass`, `Embarked`, and `Sex` to create a feature encoding all three columns in one.**

I'll now start compiling answers. Are you looking to focus on any specific question level for immediate prep, or would you like all the answers at once?

I'll compile answers to each of these questions based on the Titanic dataset. You can use the following code to load the dataset:

```python
import seaborn as sns

# Load Titanic dataset
titanic = sns.load_dataset('titanic')
```

I'll structure answers by difficulty level below.

Easy Answers

1. **View the first few rows of the dataset.**

```python
titanic.head()
```

2. **Find the number of null values in each column.**

```python
titanic.isnull().sum()
```

3. **Count the number of unique values in the `Embarked` column.**

```python
```

```python
titanic['embarked'].nunique()
```

4. **Filter rows where passengers are below the age of 10.**

```python
titanic[titanic['age'] < 10]
```

5. **Replace all null values in the `Age` column with the median age.**

```python
titanic['age'].fillna(titanic['age'].median(), inplace=True)
```

6. **Convert the `Sex` column to numeric values (0 for male, 1 for female).**

```python
titanic['sex'] = titanic['sex'].map({'male': 0, 'female': 1})
```

7. **Create a new column that calculates the family size by adding `SibSp` and `Parch`.**

```python
titanic['family_size'] = titanic['sibsp'] + titanic['parch']
```

8. **Sort the data by `Fare` in descending order.**

```python
titanic.sort_values(by='fare', ascending=False)
```

```

9. **Group data by `Pclass` and calculate the average fare.**

```python
titanic.groupby('pclass')['fare'].mean()
```

10. **Find the count of male and female passengers in each `Pclass`.**

```python
titanic.groupby(['pclass', 'sex']).size()
```

11. **Rename the `SibSp` column to `Siblings_Spouse`.**

```python
titanic.rename(columns={'sibsp': 'siblings_spouse'}, inplace=True)
```

12. **Extract the titles (like Mr., Mrs., Miss) from the `Name` column.**

```python
titanic['title'] = titanic['name'].str.extract('([A-Za-z]+)\.', expand=False)
```

13. **Convert the `Fare` column to integer type.**

```python
titanic['fare'] = titanic['fare'].astype(int)
```
```

14. **Filter rows where `Pclass` is 1 and `Age` is above 30.**

```python
titanic[(titanic['pclass'] == 1) & (titanic['age'] > 30)]
```

15. **Count the number of survivors (1) and non-survivors (0) in the `Survived` column.**

```python
titanic['survived'].value_counts()
```

16. **Drop duplicate rows based on `Name` and `Age`.**

```python
titanic.drop_duplicates(subset=['name', 'age'], inplace=True)
```

17. **Get the minimum and maximum values in the `Age` column.**

```python
titanic['age'].min(), titanic['age'].max()
```

18. **Identify if there are any duplicate rows in the dataset.**

```python
titanic.duplicated().any()
```

19. **Calculate the mean age of passengers grouped by `Embarked`.**

```python
titanic.groupby('embarked')['age'].mean()
```

20. **Generate a simple summary of the data using `.describe()`.**

```python
titanic.describe()
```

Medium Answers

I'll continue compiling the answers for the medium and hard levels right after this! Stay tuned for the next sets.

Here are the answers for the **Medium** level questions:

Medium Answers

1. **Fill missing values in the `Embarked` column with the most common port.**

```python
most_common_port = titanic['embarked'].mode()[0]
titanic['embarked'].fillna(most_common_port, inplace=True)
```

2. **Extract the deck level from the `Cabin` column (e.g., A, B, C).**

```python
titanic['deck'] = titanic['cabin'].str[0]
```

3. **Create a new binary column `Is_Alone` (1 if `SibSp` and `Parch` are 0, else 0).**

```python
titanic['is_alone'] = ((titanic['sibsp'] == 0) & (titanic['parch'] == 0)).astype(int)
```

4. **Drop columns `Ticket` and `Cabin`.**

```python
titanic.drop(['ticket', 'cabin'], axis=1, inplace=True)
```

5. **Bin the `Age` column into age groups (e.g., 0–10, 11–20, etc.).**

```python
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80]
labels = ['0-10', '11-20', '21-30', '31-40', '41-50', '51-60', '61-70', '71-80']
titanic['age_group'] = pd.cut(titanic['age'], bins=bins, labels=labels)
```

6. **Create a new column `Fare_Per_Person` by dividing `Fare` by the number of family members (SibSp + Parch + 1).**

```python
titanic['fare_per_person'] = titanic['fare'] / (titanic['sibsp'] + titanic['parch'] + 1)
```

7. **Check for outliers in the `Fare` column using the IQR method.**

```python
Q1 = titanic['fare'].quantile(0.25)
Q3 = titanic['fare'].quantile(0.75)
IQR = Q3 - Q1
outliers = titanic[(titanic['fare'] < (Q1 - 1.5 * IQR)) | (titanic['fare'] > (Q3 + 1.5 * IQR))]
```

8. **Standardize the `Fare` column (subtract mean and divide by standard deviation).**

```python
titanic['fare_standardized'] = (titanic['fare'] - titanic['fare'].mean()) / titanic['fare'].std()
```

9. **Group by `Sex` and `Pclass` and calculate the survival rate for each group.**

```python
titanic.groupby(['sex', 'pclass'])['survived'].mean()
```

10. **Calculate the correlation between `Fare`, `Age`, and `Survived`.**

```python
titanic[['fare', 'age', 'survived']].corr()
```

11. **Pivot the dataset to show the average `Age` of passengers grouped by `Pclass` and `Sex`.**

```python
titanic.pivot_table(values='age', index='pclass', columns='sex', aggfunc='mean')
```

12. **Sort the dataset first by `Pclass` and then by `Age`.**

```python
titanic.sort_values(by=['pclass', 'age'], inplace=True)
```

13. **Find the age group with the highest survival rate.**

```python
titanic.groupby('age_group')['survived'].mean().idxmax()
```

14. **Count the number of passengers in each age group.**

```python
titanic['age_group'].value_counts()
```

15. **Plot the survival count based on the `Pclass` column.**

```python
import matplotlib.pyplot as plt
titanic.groupby('pclass')['survived'].sum().plot(kind='bar')
plt.xlabel('Pclass')
plt.ylabel('Survival Count')
plt.title('Survival Count by Passenger Class')
plt.show()
```

16. **Calculate the median fare per `Embarked` port and fill missing `Fare` values with the median fare.**

```python
median_fares = titanic.groupby('embarked')['fare'].median()
titanic['fare'] = titanic.apply(lambda row: median_fares[row['embarked']] if pd.isnull(row['fare']) else row['fare'], axis=1)
```

17. **Filter out rows where passengers are below 18 and in the third class.**

```python
filtered_data = titanic[(titanic['age'] >= 18) | (titanic['pclass'] != 3)]
```

18. **Group by `Embarked` and calculate both the mean and standard deviation of `Fare`.**

```python
titanic.groupby('embarked')['fare'].agg(['mean', 'std'])
```

19. **Convert the `Name` column to lowercase.**

```python
titanic['name'] = titanic['name'].str.lower()
```

20. **Replace any numeric values in the `Name` column with an empty string.**

```python
titanic['name'] = titanic['name'].str.replace('\d+', '', regex=True)
```

These are the answers for the medium-level questions. I'll continue with the hard-level answers next. Let me know if there are any specific questions where you'd like more details!

Here are the answers to the **Hard** level questions:

Hard Answers

1. **Impute missing values in the `Age` column using median age grouped by `Pclass` and `Sex`.**

```python
titanic['age'] = titanic.groupby(['pclass', 'sex'])['age'].apply(lambda x: x.fillna(x.median()))
```

2. **Create a new column `Age_Category` with categories 'Child', 'Adult', and 'Senior' based on age (e.g., < 18 for Child, 18–60 for Adult, and > 60 for Senior).**

```python
def age_category(age):
if age < 18:
return 'Child'
elif age <= 60:
return 'Adult'
else:
return 'Senior'

titanic['age_category'] = titanic['age'].apply(age_category)
```

3. **Find the top 5 most common titles extracted from the `Name` column and replace all other titles with 'Other'.**

```python
title_counts = titanic['title'].value_counts()
top_titles = title_counts.nlargest(5).index
titanic['title'] = titanic['title'].apply(lambda x: x if x in top_titles else 'Other')
```

4. **Calculate the survival rate for each `Age_Category` within each `Pclass`.**

```python
titanic.groupby(['pclass', 'age_category'])['survived'].mean()
```

5. **Create a custom function to fill in missing values in the `Embarked` column based on the most common `Embarked` value within each `Pclass`.**

```python
def fill_embarked(row):
if pd.isnull(row['embarked']):
return titanic[titanic['pclass'] == row['pclass']]['embarked'].mode()[0]
else:
return row['embarked']

titanic['embarked'] = titanic.apply(fill_embarked, axis=1)
```

6. **One-hot encode the `Sex`, `Embarked`, and `Pclass` columns and concatenate the resulting columns back to the dataset.**

```python
titanic = pd.get_dummies(titanic, columns=['sex', 'embarked', 'pclass'], drop_first=True)
```

7. **Scale the `Fare` column using min-max scaling to range between 0 and 1.**

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
titanic['fare_scaled'] = scaler.fit_transform(titanic[['fare']])
```

8. **Calculate the z-score for the `Age` and `Fare` columns and filter out rows where either column has a z-score above 3 or below -3 (i.e., outliers).**

```python
from scipy.stats import zscore

titanic['age_zscore'] = zscore(titanic['age'].fillna(titanic['age'].mean()))
titanic['fare_zscore'] = zscore(titanic['fare'])
```

filtered_titanic = titanic[(titanic['age_zscore'].abs() <= 3) & (titanic['fare_zscore'].abs() <= 3)]
```

9. **Create a pivot table showing survival rate for each `Sex` and `Embarked` category within each `Pclass`.**

```python
survival_pivot = titanic.pivot_table(values='survived', index=['pclass', 'sex'], columns='embarked', aggfunc='mean')
```

10. **Generate a cross-tabulation of `Pclass` and `Survived` and add margins to get totals.**

```python
pd.crosstab(titanic['pclass'], titanic['survived'], margins=True)
```

11. **Replace outliers in the `Fare` column (values beyond 1.5 * IQR) with the median fare value.**

```python
Q1 = titanic['fare'].quantile(0.25)
Q3 = titanic['fare'].quantile(0.75)
IQR = Q3 - Q1

outlier_filter = (titanic['fare'] < (Q1 - 1.5 * IQR)) | (titanic['fare'] > (Q3 + 1.5 * IQR))
titanic.loc[outlier_filter, 'fare'] = titanic['fare'].median()
```

12. **Create a new column `Is_Minor` where values are 1 if the passenger is under 18 and 0 otherwise.**

```python
titanic['is_minor'] = (titanic['age'] < 18).astype(int)
```

13. **Calculate the survival rate of passengers who paid more than the median fare.**

```python
median_fare = titanic['fare'].median()
survival_rate_above_median = titanic[titanic['fare'] > median_fare]['survived'].mean()

```

14. **Extract only numeric columns from the dataset and display their correlation matrix.**

```python
numeric_columns = titanic.select_dtypes(include='number')
correlation_matrix = numeric_columns.corr()
```

15. **Impute missing values in the `Age` column by fitting a simple linear regression model using available columns (e.g., `Fare`, `Pclass`, `Sex`).**

```python
from sklearn.linear_model import LinearRegression
import numpy as np

# Select rows without missing age for training
train_data = titanic.dropna(subset=['age'])
X_train = train_data[['fare', 'pclass', 'sex']]
y_train = train_data['age']

# Linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict missing age values
missing_age = titanic['age'].isnull()
titanic.loc[missing_age, 'age'] = lr.predict(titanic.loc[missing_age, ['fare', 'pclass', 'sex']])
```

16. **Generate a random sample of 20% of the data.**

```python
titanic_sample = titanic.sample(frac=0.2, random_state=42)
```

17. **Normalize the numeric columns to have a mean of 0 and a standard deviation of 1.**

```python

```python
from sklearn.preprocessing import StandardScaler

numeric_cols = titanic.select_dtypes(include='number').columns
scaler = StandardScaler()
titanic[numeric_cols] = scaler.fit_transform(titanic[numeric_cols])
```

18. **Count the number of unique combinations of `Pclass`, `Sex`, and `Embarked`.**

```python
unique_combinations = titanic.groupby(['pclass', 'sex', 'embarked']).ngroups
```

19. **Calculate the rolling average of the `Fare` column over a window of 5 rows.**

```python
titanic['fare_rolling_avg'] = titanic['fare'].rolling(window=5).mean()
```

20. **Create a column `Family_Survived` showing the average survival rate of family members (based on `Name`).**

```python
titanic['family_survived'] = titanic.groupby('name')['survived'].transform('mean')
```

These answers cover each of the hard-level questions for data wrangling tasks using the Titanic dataset. Let me know if you need further clarification on any of these solutions!

Certainly! Here’s a set of plotting questions segmented by difficulty, focusing on **Seaborn** and **Matplotlib**.

Easy Level Plotting Questions

1. **Create a bar plot of the count of passengers in each class (`Pclass`).**
2. **Plot a histogram of passenger ages using Matplotlib with 10 bins.**
3. **Create a count plot for the `Embarked` column using Seaborn.**
4. **Use a boxplot to show the distribution of `Fare` across different `Pclass` values.**
5. **Plot the distribution of `Fare` using Seaborn’s `distplot` or `histplot` function.**
6. **Display the survival counts (`Survived`) as a pie chart.**
7. **Generate a KDE (Kernel Density Estimate) plot for `Age`.**
8. **Create a line plot of `Fare` vs. index to show the fare distribution over entries.**
9. **Display a violin plot showing the distribution of `Age` across different `Sex` categories.**
10. **Plot a scatter plot of `Age` vs `Fare` to see the relationship between the two variables.**

Medium Level Plotting Questions

1. **Plot a bar plot showing the average `Fare` paid by each `Pclass` using Seaborn.**
2. **Use a `FacetGrid` to create separate histograms of `Age` for each `Sex`.**
3. **Create a boxplot for `Fare` grouped by `Pclass` and `Embarked`.**
4. **Generate a heatmap of the correlation matrix for all numeric columns in the dataset.**
5. **Plot a stacked bar chart showing the survival count for each `Pclass`.**
6. **Use Seaborn to create a joint plot of `Age` and `Fare` with `kind='hex'`.**
7. **Create a line plot to show the cumulative sum of `Fare` values across the dataset entries.**
8. **Plot a scatter plot of `Age` vs `Fare` and color the points by `Pclass`.**
9. **Display a violin plot showing the distribution of `Fare` across `Sex` for each `Pclass` in separate subplots.**
10. **Create a boxen plot of `Age` by `Survived` status for each `Pclass`.**

Hard Level Plotting Questions

1. **Plot a pairplot for the dataset showing relationships among `Age`, `Fare`, `Pclass`, and `Survived`, and color the plots by `Survived`.**
2. **Create a seaborn `heatmap` of survival rates for each combination of `Pclass` and `Embarked` using a pivot table.**
3. **Plot the survival rates across `Age` groups using a line plot, where each `Pclass` is a different line.**
4. **Display a `FacetGrid` with KDE plots for `Fare` by `Pclass`, with `hue` set to `Survived`.**
5. **Generate a swarm plot of `Age` by `Fare`, where each point is colored by `Pclass` and arranged by `Survived`.**
6. **Create an overlaid histogram to compare the distribution of `Age` for survived and non-survived passengers.**
7. **Make a multi-panel boxplot showing `Fare` distribution across `Pclass` and `Embarked`, each panel representing a different `Sex`.**
8. **Use a violin plot to show the distribution of `Fare` across `Pclass`, with split colors based on `Survived`.**
9. **Plot a stacked area chart showing the cumulative survival rate by `Pclass` over `Age` groups.**
10. **Create a seaborn `FacetGrid` with scatter plots of `Fare` vs `Age` for each combination of `Pclass` and `Embarked`.**

Let me know if you'd like any of these questions expanded with hints or example answers!

Here's a curated list of 20 interview questions for data science, divided into three segments: Easy, Medium, and Hard, focusing on regression, classification, decision trees, ensemble methods, SVM, and unsupervised learning (clustering techniques like DBSCAN and hierarchical clustering).

Easy Questions
1. **What is regression?**
- Regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables, often used for prediction.

2. **What is the difference between linear regression and logistic regression?**
- Linear regression predicts a continuous output, while logistic regression predicts a binary outcome by estimating probabilities using a logistic function.

3. **What is a decision tree?**
- A decision tree is a flowchart-like structure used for both classification and regression tasks, where decisions are made based on the values of input features.

4. **What is overfitting in the context of decision trees?**
- Overfitting occurs when a decision tree becomes too complex and captures noise in the training data, leading to poor generalization on unseen data.

5. **What are ensemble methods?**
- Ensemble methods combine multiple models to improve overall performance and robustness, often leading to better predictions than individual models.

6. **What is the purpose of cross-validation in model evaluation?**
- Cross-validation is a technique used to assess how well a model generalizes to an independent data set, helping to prevent overfitting.

7. **What is Support Vector Machine (SVM)?**
- SVM is a supervised learning algorithm used for classification and regression that finds the optimal hyperplane to separate different classes in the feature space.

8. **What is the difference between classification and regression tasks?**
- Classification tasks predict discrete labels, while regression tasks predict continuous numerical values.

9. **What is clustering in unsupervised learning?**
- Clustering is the process of grouping similar data points together based on certain features, allowing for the discovery of inherent patterns in the data.

10. **What is the K-means algorithm?**
- K-means is a popular clustering algorithm that partitions data into K clusters by minimizing the variance within each cluster.

Medium Questions
1. **Explain the concept of bias-variance tradeoff.**
- The bias-variance tradeoff is the balance between a model's simplicity (bias) and complexity (variance). High bias can lead to underfitting, while high variance can lead to overfitting.

2. **What are the advantages of using ensemble methods like Random Forest?**
- Random Forest reduces overfitting, improves accuracy, and can handle large datasets with high dimensionality by aggregating predictions from multiple decision trees.

3. **What is the Gini index, and how is it used in decision trees?**
- The Gini index is a measure of impurity or purity used to determine the best split at each node in a decision tree. A lower Gini index indicates a better split.

4. **What is DBSCAN, and how does it differ from K-means clustering?**
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups together points that are closely packed together, while K-means requires a predefined number of clusters and can be sensitive to outliers.

5. **What is hierarchical clustering?**
- Hierarchical clustering is an unsupervised learning technique that builds a hierarchy of clusters using either a bottom-up (agglomerative) or top-down (divisive) approach.

6. **What is feature importance in the context of decision trees?**
- Feature importance indicates how much each feature contributes to the model's predictive power. It helps in feature selection and understanding the model.

7. **How does the SVM algorithm handle non-linearly separable data?**
- SVM uses kernel functions to transform the input space into a higher-dimensional space, allowing it to find a linear hyperplane in that space.

8. **What is the purpose of hyperparameter tuning?**
- Hyperparameter tuning optimizes the model's performance by adjusting settings like learning rate, number of trees in Random Forest, or number of clusters in K-means.

9. **What is the silhouette score?**
- The silhouette score measures how similar a data point is to its own cluster compared to other clusters, providing insight into the quality of the clustering.

10. **What is the elbow method in clustering?**
- The elbow method is a heuristic used to determine the optimal number of clusters by plotting the explained variance against the number of clusters and looking for an "elbow" point.

Hard Questions
1. **What are the assumptions of linear regression?**
- The assumptions include linearity, independence, homoscedasticity (constant variance of errors), normality of errors, and no multicollinearity.

2. **How do you evaluate the performance of a classification model?**
- Performance can be evaluated using metrics like accuracy, precision, recall, F1 score, ROC-AUC, and confusion matrix.

3. **What are the advantages and disadvantages of decision trees?**
- Advantages include interpretability and ease of use; disadvantages include susceptibility to overfitting and instability with small changes in data.

4. **Explain how Random Forest reduces overfitting.**
- Random Forest reduces overfitting by averaging the predictions of multiple decision trees, which stabilizes the output and mitigates the impact of individual trees' errors.

5. **What is feature scaling, and why is it important for SVM?**
- Feature scaling transforms features to a similar scale, improving convergence during optimization. It is crucial for SVM since it relies on the distances between data points.

6. **Describe the differences between agglomerative and divisive hierarchical clustering.**
- Agglomerative clustering starts with individual points and merges them into larger clusters, while divisive clustering starts with one cluster and recursively splits it into smaller clusters.

7. **What is cross-entropy loss, and how is it used in classification?**
- Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. It quantifies the difference between the true distribution and the predicted distribution.

8. **How can you handle class imbalance in classification problems?**
- Techniques include resampling methods (oversampling the minority class or undersampling the majority class), using different evaluation metrics, or applying cost-sensitive learning.

9. **What is the role of the kernel trick in SVM?**
- The kernel trick allows SVM to operate in a higher-dimensional feature space without explicitly computing the coordinates of the data in that space, enabling non-linear classification.

10. **How do you interpret the results of clustering algorithms?**
- Interpretation involves analyzing the characteristics of each cluster, visualizing clusters, and understanding the business or scientific implications of the groupings.

Feel free to use these questions to prepare for your interviews or discussions on machine learning!

Building a model to predict the survival of passengers on the Titanic is a classic project in data science. Below are detailed steps to follow, along with explanations for each step, focusing on data exploration, preprocessing, modeling, and evaluation using various machine learning algorithms.

Project Overview

- **Goal**: Predict whether a passenger survived the Titanic disaster based on various features (e.g., age, gender, class).
- **Dataset**: The Titanic dataset can be easily obtained from Kaggle or any other source. Here, we will use `pandas` for data manipulation and `sklearn` for modeling.

1. Import Required Libraries
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, roc_auc_score, roc_curve)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")
```

2. Load and Explore the Dataset
```python
# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Display the first few rows of the dataset
print(data.head())

# Check the shape of the dataset
print("Data Shape:", data.shape)

# Basic statistics and data types
print(data.info())

# Check for null values
print(data.isnull().sum())
```
Explanation: We load the data and perform initial inspections to understand its structure and identify missing values.

3. Exploratory Data Analysis (EDA)
```python
# Visualize the distribution of the target variable
sns.countplot(data['Survived'])
plt.title('Count of Survival')
plt.show()

# Check survival rate based on gender
sns.barplot(x='Sex', y='Survived', data=data)
plt.title('Survival Rate by Gender')
plt.show()

# Correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f")
plt.show()
```
Explanation: EDA helps us understand the relationships between features and the target variable, identify trends, and visualize distributions.

4. Data Preprocessing
```python
# Drop irrelevant features
data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Fill missing values
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Convert categorical variables into dummy/indicator variables
data = pd.get_dummies(data, columns=['Sex', 'Embarked'], drop_first=True)

# Check the cleaned data
print(data.head())
```
Explanation: We remove irrelevant features that won't contribute to the model, fill missing values, and convert categorical variables into numerical format for modeling.

5. Split the Data
```python
# Define features and target variable
X = data.drop('Survived', axis=1)
y = data['Survived']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Explanation: We define the features and target variable, then split the dataset into training and test sets, ensuring stratification for balanced classes.

6. Handle Class Imbalance (If Any)
```python
# Check for class imbalance
print(y_train.value_counts())

# Use SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print(y_train_resampled.value_counts())
```
Explanation: We check for class imbalance and apply SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes in the training set.

7. Standardization / Normalization
```python
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)
```
Explanation: Standardization helps bring all features onto the same scale, which is particularly important for algorithms sensitive to the scale of input data, such as SVM.

8. Model Training and Evaluation
Logistic Regression
```python
# Train Logistic Regression model
log_model = LogisticRegression()
log_model.fit(X_train_scaled, y_train_resampled)

# Predictions
y_pred_log = log_model.predict(X_test_scaled)

# Evaluation Metrics
print("Logistic Regression Metrics")
print("Accuracy:", accuracy_score(y_test, y_pred_log))
print("Precision:", precision_score(y_test, y_pred_log))
print("Recall:", recall_score(y_test, y_pred_log))
print("F1 Score:", f1_score(y_test, y_pred_log))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_log))

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_log), annot=True, fmt='d')
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```
Explanation: We train the Logistic Regression model, make predictions, and evaluate its performance using metrics such as accuracy, precision, recall, F1 score, and ROC AUC.

Random Forest Classifier
```python
# Train Random Forest Classifier model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_scaled, y_train_resampled)

# Predictions
y_pred_rf = rf_model.predict(X_test_scaled)

# Evaluation Metrics
print("Random Forest Metrics")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1 Score:", f1_score(y_test, y_pred_rf))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_rf))

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d')
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```
Explanation: Similar to Logistic Regression, we train and evaluate the Random Forest Classifier.

Support Vector Machine (SVM)
```python
# Train SVM model
svm_model = SVC(probability=True)
svm_model.fit(X_train_scaled, y_train_resampled)

# Predictions
y_pred_svm = svm_model.predict(X_test_scaled)

# Evaluation Metrics
print("SVM Metrics")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm))
print("Recall:", recall_score(y_test, y_pred_svm))
print("F1 Score:", f1_score(y_test, y_pred_svm))
```

print("ROC AUC Score:", roc_auc_score(y_test, y_pred_svm))

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d')
plt.title('Confusion Matrix - SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```
**Explanation**: We train the SVM model, make predictions, and evaluate it similarly to the previous models.

#### XGBoost Classifier
```python
# Train XGBoost model
xgb_model = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
xgb_model.fit(X_train_scaled, y_train_resampled)

# Predictions
y_pred_xgb = xgb_model.predict(X_test_scaled)

# Evaluation Metrics
print("XGBoost Metrics")
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Precision:", precision_score(y_test, y_pred_xgb))
print("Recall:", recall_score(y_test, y_pred_xgb))
print("F1 Score:", f1_score(y_test, y_pred_xgb))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_xgb))

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred_xgb), annot=True, fmt='d')
plt.title('Confusion Matrix - XGBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```
Explanation: We follow the same training and evaluation process for the XGBoost model, leveraging its advanced boosting algorithms for improved performance.

9. Model Comparison
```python
# Store all models and their accuracy scores
models = {
Logistic Regression: log_model,
Random Forest: rf_model,
SVM: svm_model,
XGBoost: xgb_model
}

for model_name, model in models.items():
y_pred = model.predict(X_test_scaled)
acc = accuracy_score(y_test, y_pred)
print(f"{model_name} Accuracy: {acc:.2f}")
```
Explanation: We compare the accuracy of all models, allowing us to select the best-performing model based on the evaluation metrics.

10. Conclusion
In this project, we've gone through the entire machine learning pipeline, from data loading and preprocessing to model training and evaluation using various algorithms. This not only provides a comprehensive understanding of the Titanic dataset but also solidifies knowledge of machine learning practices.

Notes:
- Hyperparameter tuning can be performed for each model to

potentially improve accuracy further.
- Cross-validation can be applied to get a better estimate of model performance.
- Consider using techniques like feature engineering to improve the model's performance further.

Feel free to ask if you need any specific parts elaborated on or any additional steps!

You're right! Creating a pipeline is essential for maintaining a clean and efficient workflow in machine learning projects, especially when dealing with preprocessing and model evaluation. Pipelines help automate the process and ensure that the same transformations are applied consistently to both training and testing datasets. Below is an updated version of the Titanic survival prediction project that includes the creation of a pipeline, mod

Updated Titanic Survival Prediction Project with Pipeline

1. Import Required Libraries
(Use the same libraries as mentioned previously.)

2. Load and Explore the Dataset
(Same as before.)

3. Exploratory Data Analysis (EDA)
(Same as before.)

4. Data Preprocessing and Pipeline Creation
```python
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Define preprocessing for numerical features
num_features = ['Age', 'Fare']

```
num_transformer = Pipeline(steps=[
('imputer', SimpleImputer(strategy='median')), # Handle missing values
('scaler', StandardScaler()) # Standardization
])

# Define preprocessing for categorical features
cat_features = ['Sex', 'Embarked', 'Pclass']
cat_transformer = Pipeline(steps=[
('imputer', SimpleImputer(strategy='most_frequent')), # Handle missing values
('onehot', OneHotEncoder(handle_unknown='ignore')) # One-hot encoding
])

# Combine preprocessing for numerical and categorical features
preprocessor = ColumnTransformer(
transformers=[
('num', num_transformer, num_features),
('cat', cat_transformer, cat_features)
])

# Define the model pipeline
model_pipeline = Pipeline(steps=[
('preprocessor', preprocessor),
('classifier', LogisticRegression())
])
```
**Explanation**: Here, we create a pipeline that first preprocesses the numerical and categorical features and then applies a classifier. This ensures that any transformations applied during training are also applied during testing.

### 5. Split the Data
(Same as before.)

### 6. Handle Class Imbalance (If Any)
(Same as before.)

### 7. Model Training with Pipeline
```python
# Fit the model using the pipeline
model_pipeline.fit(X_train_resampled, y_train_resampled)

# Predictions on the test set
y_pred = model_pipeline.predict(X_test)

# Evaluation Metrics
print("Logistic Regression with Pipeline Metrics")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred))

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
plt.title('Confusion Matrix - Logistic Regression with Pipeline')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```
Explanation: The model is trained and evaluated using the pipeline, ensuring that preprocessing steps are automatically applied.

8. Saving the Model
```python
import joblib

# Save the model
joblib.dump(model_pipeline, 'titanic_model_pipeline.joblib')
```
Explanation: We use `joblib` to save the trained pipeline to a file, allowing for easy loading and use in the future.

9. Load the Model for Inference
```python
# Load the model
loaded_model = joblib.load('titanic_model_pipeline.joblib')

# Predictions on new data
new_predictions = loaded_model.predict(X_test)

# Display predictions
print("New Predictions:", new_predictions)
```
Explanation: This code demonstrates how to load a saved model and use it to make predictions on new data.

10. Model Comparison (Using Pipeline)
You can follow the same steps to train and evaluate other models (Random Forest, SVM, XGBoost) using similar pipelines.

Conclusion
Using a pipeline not only streamlines the process of building a machine learning model but also minimizes the risk of data leakage and ensures consistent preprocessing across different datasets. This structured approach is crucial for real-world applications.

Let me know if you have any further questions or if you'd like to delve deeper into specific aspects!