Data Science Lab Viva Preparation Guide

Practical 1: Data Wrangling I

Basic Questions and Answers

Q1: What is data wrangling and why is it important?

A1: Data wrangling is the process of cleaning, structuring, and enriching raw data into a desired format for better decision-making. It involves data collection, cleaning, and transformation to make it suitable for analysis. It's important because real-world data is often messy, incomplete, and needs preparation before it can be used for analysis or modeling.

Q2: What Python libraries are commonly used for data wrangling?

A2: The most common libraries are:

- Pandas: For data manipulation and analysis
- NumPy: For numerical operations
- Matplotlib/Seaborn: For data visualization
- Scikit-learn: For data preprocessing and machine learning

Q3: How do you check for missing values in a dataset using pandas?

A3: We can use several methods:

python

```python
# Check for missing values
df.isnull().sum()  # Returns count of missing values by column
df.isna().sum()    # Alternative syntax
df.info()          # Shows datatype and non-null counts
```

Q4: How would you handle categorical variables in Python?

A4: Categorical variables can be handled in multiple ways:

1. Label Encoding: Convert categories to numerical values (0, 1, 2, etc.)
2. One-Hot Encoding: Create binary columns for each category
3. Ordinal Encoding: For ordered categories

```python
# Label Encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['category_encoded'] = le.fit_transform(df['category'])

# One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['category'])
```

Q5: What is data normalization and why is it used?

A5: Data normalization is the process of scaling numeric variables to a standard range (typically 0-1 or -1 to 1). It's used to prevent features with larger scales from dominating the model training process. Common methods include Min-Max scaling and Z-score normalization.

```python
# Min-Max scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['normalized'] = scaler.fit_transform(df[['feature']])
```

Advanced/Related Questions

Q6: What's the difference between .loc and .iloc in pandas?

A6: `.loc` is label-based indexing, used to access data by row/column labels, while `.iloc` is integer position-based indexing, used to access data by their integer positions.

Q7: How would you identify the data types of all columns in a pandas DataFrame?

A7: We can use `df.dtypes` to see the data types of all columns, or `df.info()` for a more comprehensive overview including non-null counts.

Q8: What is the purpose of the describe() function in pandas?

A8: The `describe()` function provides descriptive statistics including count, mean, std deviation, min, 25th percentile, median, 75th percentile, and max for numerical columns. For categorical columns, it shows count, unique values, top value, and its frequency.

Practical 2: Data Wrangling II

Basic Questions and Answers

Q1: What are outliers and why are they important to identify?

A1: Outliers are data points that significantly differ from other observations in a dataset. They're important to identify because they can disproportionately influence statistical analyses and machine learning models, potentially leading to misleading results or poor model performance.

Q2: What techniques can be used to detect outliers?

A2: Common techniques include:

1. Visual methods: Box plots, scatter plots, histograms
2. Statistical methods: Z-score, IQR (Interquartile Range)
3. Model-based methods: Isolation Forest, DBSCAN, LOF

```python
# IQR method
Q1 = df['column'].quantile(0.25)
Q3 = df['column'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['column'] < lower_bound) | (df['column'] > upper_bound)]
```

Q3: How do you handle missing values in a dataset?

A3: Methods for handling missing values include:

1. Deletion: Remove rows or columns with missing values
2. Imputation: Replace with mean, median, mode, or predicted values
3. Flagging: Add a binary indicator for missingness
4. Advanced methods: KNN imputation, MICE (Multiple Imputation by Chained Equations)

```python
# Simple imputation examples
df['column'].fillna(df['column'].mean())  # Mean imputation
df['column'].fillna(df['column'].median())  # Median imputation
df['column'].fillna(df['column'].mode()[0])  # Mode imputation
```

Q4: What are common data transformations and why are they used?

A4: Common transformations include:

1. Log transformation: Reduces skewness, useful for right-skewed data
2. Square root transformation: Less aggressive than log for right-skewed data
3. Box-Cox transformation: Finds the best transformation to normalize data
4. Power transformations: Raising data to a power to adjust distribution
5. Min-Max scaling and Z-score normalization: For feature scaling

They're used to make data more normally distributed, linearize relationships, or standardize features for modeling.

Q5: What is data skewness and how can it be addressed?

A5: Skewness is a measure of asymmetry in data distribution. A right-skewed (positive) distribution has a long tail to the right, while a left-skewed (negative) distribution has a long tail to the left. Skewness can be addressed through transformations like log, square root, or Box-Cox transformations.

Advanced/Related Questions

Q6: What is the difference between Z-score and IQR method for outlier detection?

A6: The Z-score method identifies outliers based on how many standard deviations a point is from the mean (typically $|Z| > 3$). The IQR method uses the interquartile range and identifies outliers as points below $Q1 - 1.5\times IQR$ or above $Q3 + 1.5\times IQR$. Z-score assumes normal distribution while IQR is more robust to non-normal distributions.

Q7: When would you use imputation versus deletion for missing values?

A7: Use imputation when:

- The dataset is small and losing observations would significantly reduce statistical power
- The missingness is at random or completely at random
- The variables with missing values are important for analysis

Use deletion when:

- Missing data percentage is very small relative to dataset size
- Missingness is not at random and imputation might introduce bias
- You need a quick, simple solution with minimal computational overhead

Q8: How would you choose the appropriate transformation for skewed data?

A8: The choice depends on:

1. Degree of skewness: More severe skewness might need stronger transformations

2. Data characteristics: Non-negative data might use log, while data including zeros may need log(x+1)
3. Domain knowledge: Some fields have conventional transformations
4. Empirical testing: Try multiple transformations and select based on resulting distributions or model performance

Practical 3: Descriptive Statistics

Basic Questions and Answers

Q1: What are measures of central tendency?

A1: Measures of central tendency are values that represent the center or middle point of a data distribution. The three main measures are:

1. Mean: The arithmetic average of all values
2. Median: The middle value when data is arranged in order
3. Mode: The most frequently occurring value

Q2: What are measures of variability/dispersion?

A2: Measures of variability quantify the spread or dispersion of data points. Common measures include:

1. Range: Difference between maximum and minimum values
2. Variance: Average of squared deviations from the mean
3. Standard Deviation: Square root of the variance
4. Interquartile Range (IQR): Difference between the 75th and 25th percentiles

Q3: When would you use median instead of mean?

A3: The median is preferred over the mean when:

- The data contains outliers that would skew the mean
- The distribution is significantly skewed (non-symmetric)
- Working with ordinal data where averages don't make sense
- Dealing with variables like income or house prices that tend to be right-skewed

Q4: How do you calculate percentiles in Python?

A4: Percentiles can be calculated using NumPy or Pandas:

python

```python
import numpy as np
import pandas as pd

# Using NumPy
percentile_75 = np.percentile(data, 75)

# Using Pandas
percentile_75 = df['column'].quantile(0.75)
```

Q5: How would you compare the variation between different groups in a dataset?

A5: To compare variation between groups:

1. Calculate descriptive statistics (mean, median, std, etc.) for each group separately
2. Use box plots to visually compare distributions
3. Apply statistical tests (F-test, Levene's test) to formally test for differences in variance
4. Use coefficient of variation (CV = std/mean) to compare dispersion relative to the mean

python

```python
# Group by categorical variable and calculate summary statistics
grouped_stats = df.groupby('category')['numeric_var'].agg(['mean', 'median', 'std', 'min', 'max
```

Advanced/Related Questions

Q6: What is the relationship between standard deviation and variance?

A6: Variance is the average of squared deviations from the mean, while standard deviation is the square root of the variance. Standard deviation is often preferred because it's in the same units as the original data, making interpretation easier.

Q7: What is a coefficient of variation and when is it useful?

A7: The coefficient of variation (CV) is the ratio of the standard deviation to the mean (CV = σ/μ), usually expressed as a percentage. It's useful for comparing the relative variability of different variables that have different units or vastly different means, as it's a unitless measure.

Q8: How would you detect if your data follows a normal distribution?

A8: Methods to detect normality include:

1. Visual methods: Histogram, Q-Q plot

2. Statistical tests: Shapiro-Wilk test, Kolmogorov-Smirnov test, Anderson-Darling test
3. Skewness and kurtosis measurements

python

```python
from scipy import stats

# Shapiro-Wilk test
stat, p_value = stats.shapiro(data)
if p_value > 0.05:
    print("Data appears normally distributed")
else:
    print("Data does not appear normally distributed")
```

Practical 4: Data Analytics I - Linear Regression

Basic Questions and Answers

Q1: What is linear regression and when is it used?

A1: Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation. It's used when we want to predict a continuous numerical outcome based on one or more predictor variables, assuming a linear relationship exists.

Q2: What are the assumptions of linear regression?

A2: The key assumptions are:

1. Linearity: The relationship between X and Y is linear
2. Independence: Observations are independent of each other
3. Homoscedasticity: Constant variance of errors
4. Normality: The residuals (errors) follow a normal distribution
5. No or little multicollinearity: Predictor variables aren't strongly correlated with each other

Q3: How do you evaluate a linear regression model?

A3: Common evaluation metrics include:

1. R-squared (coefficient of determination): Proportion of variance explained by the model
2. Adjusted R-squared: R-squared adjusted for the number of predictors
3. RMSE (Root Mean Squared Error): Square root of the average squared differences between predicted and actual values

4. MAE (Mean Absolute Error): Average of absolute differences between predicted and actual values
5. Residual analysis: Plotting residuals to check assumptions

python

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Calculate metrics
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
```

Q4: What is the difference between simple and multiple linear regression?

A4: Simple linear regression has only one independent variable (X) predicting the dependent variable (Y), represented as $Y = \beta_0 + \beta_1 X + \varepsilon$. Multiple linear regression has two or more independent variables, represented as $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$.

Q5: How do you interpret the coefficients in a linear regression model?

A5: The coefficients (β values) represent the change in the dependent variable for a one-unit change in the corresponding independent variable, holding all other variables constant. The intercept (β_0) represents the predicted value of Y when all X variables equal zero.

Advanced/Related Questions

Q6: What is multicollinearity and how can it affect regression models?

A6: Multicollinearity occurs when independent variables in a regression model are highly correlated with each other. It can:

- Make coefficients unstable and hard to interpret
- Increase the variance of coefficient estimates
- Make it difficult to determine which variables are truly important

Detection methods include correlation matrices and Variance Inflation Factor (VIF). Solutions include removing one of the correlated variables or using regularization techniques.

Q7: What is regularization in linear regression?

A7: Regularization adds a penalty term to the regression's cost function to reduce model complexity and prevent overfitting. Common methods include:

1. Ridge Regression (L2): Adds the sum of squared coefficients as a penalty
2. Lasso Regression (L1): Adds the sum of absolute coefficients as a penalty
3. Elastic Net: Combines both L1 and L2 penalties

```python
from sklearn.linear_model import Ridge, Lasso, ElasticNet

# Ridge regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

# Lasso regression
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```

Q8: How do you handle categorical variables in linear regression?

A8: Categorical variables must be converted to numerical format before using them in regression. Common techniques include:

1. One-hot encoding: Create binary columns for each category
2. Label encoding: Convert categories to numerical values (use carefully as it implies ordering)
3. Effect coding: Similar to one-hot but uses -1, 0, 1 coding
4. Target encoding: Replace categories with their target mean value

Practical 5: Data Analytics II - Logistic Regression

Basic Questions and Answers

Q1: What is logistic regression and when is it used?

A1: Logistic regression is a classification algorithm used to predict the probability of a categorical dependent variable. Despite its name, it's a classification algorithm, not a regression algorithm. It's used when the dependent variable is binary (binary logistic regression) or has multiple classes (multinomial logistic regression).

Q2: What is the sigmoid function and why is it used in logistic regression?

A2: The sigmoid function (also called the logistic function) transforms any real-valued number into a value between 0 and 1. It's defined as:

$\sigma(z) = 1/(1 + e^{(-z)})$

It's used in logistic regression to model the probability that a given input belongs to a certain class, constraining the output to be between 0 and 1, which is appropriate for probability values.

Q3: What is a confusion matrix and what does it tell us?

A3: A confusion matrix is a table that visualizes the performance of a classification algorithm. For binary classification, it shows:

- True Positives (TP): Correctly predicted positive class
- False Positives (FP): Incorrectly predicted positive class (Type I error)
- True Negatives (TN): Correctly predicted negative class
- False Negatives (FN): Incorrectly predicted negative class (Type II error)

It helps us understand not just overall accuracy, but specific types of errors the model makes.

Q4: How do you calculate accuracy, precision, recall, and F1-score?

A4:

- Accuracy = (TP + TN) / (TP + TN + FP + FN) The proportion of correct predictions among the total predictions
- Precision = TP / (TP + FP) The proportion of true positive predictions among all positive predictions
- Recall (Sensitivity) = TP / (TP + FN) The proportion of actual positives correctly identified
- F1-score = 2 * (Precision * Recall) / (Precision + Recall) The harmonic mean of precision and recall

python

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

Q5: What is the difference between a ROC curve and precision-recall curve?

A5:

- ROC (Receiver Operating Characteristic) curve plots True Positive Rate (Sensitivity) against False Positive Rate (1-Specificity) at various threshold settings. The area under the ROC curve (AUC-ROC) measures the model's ability to distinguish between classes.
- Precision-Recall curve plots Precision against Recall at various threshold settings. The area under this curve (AUC-PR) is useful when dealing with imbalanced datasets where negative examples are more

common.

ROC curves are better when classes are balanced, while Precision-Recall curves are more informative for imbalanced datasets.

Advanced/Related Questions

Q6: What is the cost function in logistic regression?

A6: The cost function in logistic regression is the log loss (logarithmic loss) or cross-entropy loss:

$J(\theta) = -1/m * \Sigma[y^{(i)} * \log(h\_\theta(x^{(i)})) + (1-y^{(i)}) * \log(1-h\_\theta(x^{(i)}))]$

Where:

- m is the number of training examples
- $y^{(i)}$ is the actual class (0 or 1)
- $h\_\theta(x^{(i)})$ is the predicted probability

Unlike linear regression's mean squared error, this cost function penalizes confident incorrect predictions more severely.

Q7: How do you handle class imbalance in logistic regression?

A7: Methods to handle class imbalance include:

1. Resampling techniques:
    - Oversampling the minority class (e.g., SMOTE)
    - Undersampling the majority class
    - Combination of both
2. Using class weights in the model
3. Using different evaluation metrics (precision, recall, F1 instead of accuracy)
4. Adjusting the classification threshold
5. Use of cost-sensitive learning algorithms

python

```python
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE

# Using class weights
model = LogisticRegression(class_weight='balanced')

# Using SMOTE for oversampling
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Q8: What is the difference between L1 and L2 regularization in logistic regression?

A8: Both L1 and L2 regularization help prevent overfitting by penalizing large coefficients:

- L1 regularization (Lasso) adds the sum of the absolute values of the coefficients to the cost function. It can lead to sparse models by driving some coefficients to exactly zero, effectively performing feature selection.
- L2 regularization (Ridge) adds the sum of squared coefficients to the cost function. It tends to shrink coefficients toward zero but rarely sets them exactly to zero.

L1 is useful when you suspect many features are irrelevant, while L2 works better when most features contribute to the outcome.

Practical 6: Data Analytics III - Naive Bayes

Basic Questions and Answers

Q1: What is Naive Bayes classification and how does it work?

A1: Naive Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of independence between features. It calculates the probability of each class given the input features, and then selects the class with the highest probability. The "naive" part refers to the assumption that features are conditionally independent given the class label, which simplifies calculations but is often not true in real-world data.

Q2: What is Bayes' theorem and how is it used in Naive Bayes?

A2: Bayes' theorem states:

$P(Y|X) = P(X|Y) * P(Y) / P(X)$

Where:

- $P(Y|X)$ is the posterior probability of class Y given features X

- P(X|Y) is the likelihood of features X given class Y
- P(Y) is the prior probability of class Y
- P(X) is the probability of features X

In Naive Bayes, we calculate P(Y|X) for each class Y and choose the class with the highest probability.

Q3: What are the different types of Naive Bayes classifiers?

A3: The main types are:

1. Gaussian Naive Bayes: Assumes features follow a normal distribution; used for continuous data
2. Multinomial Naive Bayes: Used for discrete data, commonly with text classification (word counts)
3. Bernoulli Naive Bayes: Used for binary/boolean features
4. Complement Naive Bayes: An adaptation of Multinomial NB for imbalanced datasets

python

```python
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# Gaussian NB for continuous features
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Multinomial NB for discrete features like word counts
mnb = MultinomialNB()
mnb.fit(X_train, y_train)
```

Q4: What are the advantages and disadvantages of Naive Bayes?

A4: Advantages:

- Simple and easy to implement
- Works well with high-dimensional data
- Efficient with small training datasets
- Fast training and prediction
- Less sensitive to irrelevant features

Disadvantages:

- The "naive" independence assumption is often unrealistic
- Performs poorly when features are highly correlated
- May be outperformed by more sophisticated models

- Can be sensitive to how the input data is prepared
- Poor estimator (the probability estimates may not be accurate)

Q5: How is performance evaluated for a Naive Bayes classifier?

A5: Performance evaluation uses the same metrics as other classification algorithms:

- Confusion matrix
- Accuracy: (TP + TN) / (TP + TN + FP + FN)
- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)
- F1-score: Harmonic mean of precision and recall
- ROC curve and AUC for binary classification
- Cross-validation to ensure robustness

Advanced/Related Questions

Q6: What is the "zero frequency problem" in Naive Bayes and how is it addressed?

A6: The zero frequency problem occurs when a class and feature value never occur together in the training data, resulting in a zero probability that wipes out information from all other probabilities when they're multiplied. This is addressed using smoothing techniques:

1. Laplace (add-one) smoothing: Add 1 to all counts
2. Lidstone smoothing: Add α (0 < α < 1) to all counts
3. Dirichlet priors: More general form of smoothing

python

```python
# Laplace smoothing in scikit-learn
mnb = MultinomialNB(alpha=1.0)  # Default alpha=1.0 for Laplace smoothing
```

Q7: How does Naive Bayes handle continuous features?

A7: Gaussian Naive Bayes handles continuous features by assuming they follow a normal (Gaussian) distribution. For each class, the mean and variance of each feature are calculated from the training data. Then, for new instances, the probability density function of the normal distribution is used to estimate P(X|Y).

Alternatives include:

1. Discretizing continuous features into bins

2. Kernel density estimation for non-normal distributions
3. Using mixed Naive Bayes variants

Q8: When would you choose Naive Bayes over other classification algorithms?

A8: Naive Bayes is particularly well-suited for:

1. Text classification and document categorization
2. Spam filtering
3. Sentiment analysis
4. When training data is limited
5. When features are relatively independent
6. When very fast training and prediction are required
7. As a baseline classifier to compare against more complex models
8. High-dimensional data where dimensionality reduction might lose information

Practical 7: Text Analytics

Basic Questions and Answers

Q1: What is text preprocessing and why is it important in text analytics?

A1: Text preprocessing is the process of cleaning and transforming raw text data into a format suitable for analysis. It's important because raw text is unstructured and contains various inconsistencies, noise, and irrelevant information that can negatively impact the performance of text analytics algorithms.

Q2: What is tokenization and how is it implemented in Python?

A2: Tokenization is the process of breaking down text into smaller units called tokens, typically words, phrases, or sentences. In Python, it can be implemented using various libraries:

```python
# Using NLTK
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize, sent_tokenize

words = word_tokenize("This is a sample sentence.")
sentences = sent_tokenize("This is sentence one. This is sentence two.")

# Using spaCy
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("This is a sample sentence.")
tokens = [token.text for token in doc]
```

Q3: What is POS (Part-of-Speech) tagging and why is it useful?

A3: POS tagging is the process of marking up words in text with their corresponding part of speech (noun, verb, adjective, etc.) based on their definition and context. It's useful for:

- Word sense disambiguation
- Named entity recognition
- Syntactic parsing
- Feature extraction for machine learning models
- Information extraction and relationship identification

```python
# Using NLTK
import nltk
nltk.download('averaged_perceptron_tagger')
tokens = word_tokenize("NLTK is a powerful Python library for NLP.")
pos_tags = nltk.pos_tag(tokens)
# [('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('powerful', 'JJ'), ...]

# Using spaCy
doc = nlp("spaCy is another powerful Python library for NLP.")
pos_tags = [(token.text, token.pos_) for token in doc]
```

Q4: What are stop words and why are they often removed in text preprocessing?

A4: Stop words are common words that typically don't carry significant meaning in text analysis (e.g., "the", "is", "at", "which"). They're often removed because:

- They occur frequently but add little value to understanding the content
- They can increase dimensionality unnecessarily in vector representations
- Removing them focuses the analysis on the more meaningful content words
- They can improve the performance of models like TF-IDF

```python
# Using NLTK
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word.lower() not in stop_words]

# Using spaCy
tokens = [token.text for token in doc if not token.is_stop]
```

Q5: What is the difference between stemming and lemmatization?

A5: Both techniques reduce words to their base or root form, but with different approaches:

Stemming:

- Uses heuristic rules to chop off word endings
- Faster but less accurate
- May produce non-existent words
- Examples: "running", "runner", "ran" → "run"

Lemmatization:

- Uses vocabulary and morphological analysis
- Considers the context and part of speech
- Returns actual dictionary words
- More accurate but computationally more expensive
- Examples: "better" → "good", "was" → "be"

```python
# Stemming with NLTK
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed = [stemmer.stem(word) for word in tokens]

# Lemmatization with NLTK
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(word) for word in tokens]

# Lemmatization with spaCy
lemmatized = [token.lemma_ for token in doc]
```

Advanced/Related Questions

Q6: What is TF-IDF and how is it calculated?

A6: TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents (corpus).

- Term Frequency (TF): The frequency of a term in a document TF(t, d) = (Number of times term t appears in document d) / (Total number of terms in document d)
- Inverse Document Frequency (IDF): Measures how common or rare a word is across all documents IDF(t) = log(Total number of documents / Number of documents containing term t)
- TF-IDF: The product of TF and IDF TF-IDF(t, d) = TF(t, d) * IDF(t)

High TF-IDF values indicate terms that are distinctive to specific documents.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(documents)
feature_names = vectorizer.get_feature_names_out()
```

Q7: What are word embeddings and how do they differ from TF-IDF?

A7: Word embeddings are dense vector representations of words in a continuous vector space where semantically similar words are mapped close to each other. Unlike TF-IDF:

1. Word embeddings capture semantic relationships and word similarities

2. They have fixed dimensions regardless of vocabulary size
3. They preserve context and word order information
4. They can be pre-trained on large corpora (Word2Vec, GloVe, FastText)
5. They're better for deep learning models and semantic tasks

TF-IDF creates sparse, high-dimensional vectors based on word frequencies and doesn't capture semantic relationships.

Q8: What is named entity recognition (NER) and how is it implemented?

A8: Named Entity Recognition is the task of identifying and classifying named entities in text into predefined categories such as person names, organizations, locations, dates, etc.

```python
# Using spaCy for NER
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is planning to open a new store in New York next month.")
for ent in doc.ents:
    print(f"Entity: {ent.text}, Type: {ent.label_}")
# Entity: Apple, Type: ORG
# Entity: New York, Type: GPE
# Entity: next month, Type: DATE

# Using NLTK
from nltk import ne_chunk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

nltk.download('maxent_ne_chunker')
nltk.download('words')
tokens = word_tokenize("Tim Cook is the CEO of Apple Inc.")
pos_tags = pos_tag(tokens)
named_entities = ne_chunk(pos_tags)
print(named_entities)
```

Practical 8: Data Visualization I

Basic Questions and Answers

Q1: What is data visualization and why is it important in data science?

A1: Data visualization is the graphical representation of data and information using visual elements like charts, graphs, and maps. It's important because:

- It helps in understanding patterns, trends, and outliers in data that might be difficult to identify in raw form
- It communicates insights effectively to both technical and non-technical audiences
- It facilitates exploratory data analysis and hypothesis generation
- It helps in making data-driven decisions more quickly and effectively
- It can reveal relationships and correlations between variables

Q2: What are the different types of visualizations used for different types of data?

A2: Different visualizations are suitable for different data types:

For categorical data:

- Bar charts/Column charts
- Pie charts
- Treemaps
- Heatmaps

For numerical/continuous data:

- Histograms
- Box plots
- Violin plots
- Line charts
- Scatter plots
- Density plots

For time series data:

- Line charts
- Area charts
- Candlestick charts

For multivariate data:

- Scatter plot matrices
- Parallel coordinates
- Radar charts
- Bubble charts

Q3: What Python libraries are commonly used for data visualization?

A3: Common Python visualization libraries include:

- Matplotlib: Foundation library for creating static visualizations
- Seaborn: Statistical visualization library built on Matplotlib with enhanced aesthetics
- Plotly: Interactive visualization library supporting web-based graphics
- Bokeh: Interactive visualization library targeting web browsers
- Altair: Declarative visualization library based on Vega and Vega-Lite
- Pandas: Built-in plotting functionality for quick visualizations
- Folium: Specialized for geographic data visualization

Q4: How would you create a histogram in Python using Seaborn?

A4: A histogram shows the distribution of a continuous variable by dividing it into bins and counting observations in each bin.

python

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

# Using the titanic dataset
df = sns.load_dataset('titanic')

# Create a histogram of fare prices
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='fare', bins=30, kde=True)
plt.title('Distribution of Titanic Fare Prices')
plt.xlabel('Fare Price')
plt.ylabel('Count')
plt.show()
```

Q5: What information can you extract from a histogram?

A5: A histogram can reveal:

- The shape of the data distribution (normal, skewed, bimodal, etc.)
- The central tendency (where most values fall)
- The spread or variability of the data
- The presence of outliers
- Gaps or unusual patterns in the data

- Whether the distribution is symmetric or skewed
- Potential maximum and minimum values
- Natural groupings or clusters in the data

Advanced/Related Questions

Q6: What is the difference between a histogram and a bar chart?

A6: While they may look similar, histograms and bar charts represent different types of data:

Histogram:

- Represents the distribution of a continuous variable
- Bars touch each other (no gaps) because they represent continuous intervals
- The area of each bar is proportional to frequency
- The x-axis represents the variable's value ranges (bins)
- Used to show the shape of a distribution

Bar chart:

- Represents categorical data
- Bars are separated by gaps to emphasize distinct categories
- The height/length of each bar represents the frequency or value
- The x-axis represents distinct categories
- Used to compare values across different categories

Q7: How would you choose the optimal bin size for a histogram?

A7: Choosing the optimal bin size for a histogram involves balancing detail and clarity:

1. Rules of thumb:
   - Sturges' rule: $k = 1 + \log_2(n)$ where n is sample size
   - Rice rule: k = 2n^(1/3)
   - Freedman-Diaconis rule: bin width = 2 * IQR * n^(-1/3)
2. Visual approach:
   - Try different bin sizes and select one that reveals the underlying structure without over-smoothing or showing too much noise
   - Look for a bin size that reveals important features of the distribution
3. Using library defaults:
   - Libraries like Seaborn often implement automated bin size selection algorithms

```python
# Let seaborn choose bin size automatically
sns.histplot(data=df, x='fare')

# Specify number of bins
sns.histplot(data=df, x='fare', bins=20)

# Specify bin width
sns.histplot(data=df, x='fare', binwidth=10)
```

Q8: How can you overlay multiple histograms for comparison?

A8: To compare distributions using multiple histograms:

```python
# Method 1: Using transparency (alpha)
plt.figure(figsize=(10, 6))
sns.histplot(data=df[df['survived']==1], x='age', alpha=0.5, label='Survived')
sns.histplot(data=df[df['survived']==0], x='age', alpha=0.5, label='Did not survive')
plt.legend()
plt.title('Age Distribution by Survival Status')
plt.show()

# Method 2: Using KDE (Kernel Density Estimation) plots
plt.figure(figsize=(10, 6))
sns.kdeplot(data=df, x='age', hue='survived', fill=True, common_norm=False)
plt.title('Age Distribution Density by Survival Status')
plt.show()

# Method 3: Using multiple subplot panels
g = sns.FacetGrid(df, col='survived', height=5, aspect=1.5)
g.map(sns.histplot, 'age', kde=True)
g.set_axis_labels('Age', 'Count')
g.set_titles('Survived: {col_name}')
plt.show()
```

Practical 9: Data Visualization II

Basic Questions and Answers

Q1: What is a box plot and what information does it provide?

A1: A box plot (also called a box-and-whisker plot) is a standardized way of displaying the distribution of data based on a five-number summary:

1. Minimum (excluding outliers)
2. First quartile (Q1, 25th percentile)
3. Median (Q2, 50th percentile)
4. Third quartile (Q3, 75th percentile)
5. Maximum (excluding outliers)

The box spans from Q1 to Q3, with a line at the median. The "whiskers" extend to the minimum and maximum values within 1.5 * IQR (Interquartile Range = Q3-Q1) from the box. Points beyond the whiskers are plotted individually as outliers.

Box plots provide information about:

- Central tendency
- Dispersion/spread of data
- Skewness
- Outliers
- Quartiles and interquartile range

Q2: How would you create a box plot in Seaborn to compare distributions by category?

A2: You can create box plots to compare distributions across categories using Seaborn:

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

# Load the titanic dataset
titanic = sns.load_dataset('titanic')

# Create box plot of age distribution by gender and survival status
plt.figure(figsize=(12, 6))
sns.boxplot(x='sex', y='age', hue='survived', data=titanic)
plt.title('Age Distribution by Gender and Survival Status')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.legend(title='Survived', loc='upper right')
plt.show()
```

Q3: What is a violin plot and how does it differ from a box plot?

A3: A violin plot is similar to a box plot but adds a rotated kernel density plot on each side, showing the distribution shape. Differences include:

- Box plots show only summary statistics (quartiles, median, outliers)
- Violin plots show the full distribution of data
- Violin plots are better for visualizing multimodal distributions
- The width of a violin plot at different points represents the density of data
- Box plots are better for precise comparisons of medians and quartiles
- Violin plots provide richer visualization of distribution characteristics

python

```python
plt.figure(figsize=(12, 6))
sns.violinplot(x='sex', y='age', hue='survived', data=titanic, split=True)
plt.title('Age Distribution by Gender and Survival Status')
plt.show()
```

Q4: How would you interpret a box plot showing outliers?

A4: When interpreting box plots with outliers:

1. Identify outliers: Points beyond the whiskers (typically 1.5 * IQR beyond Q1 or Q3)
2. Assess their distance: How far they are from the central distribution
3. Investigate their nature: Are they errors, special cases, or valid extreme values?
4. Consider their impact: How they might affect statistical analyses
5. Compare across groups: Are outliers more prevalent in certain categories?
6. Look for patterns: Multiple outliers on one side suggests skewness
7. Make decisions: Whether to investigate, remove, transform, or keep outliers

Q5: What conclusions might you draw from comparing box plots of age by gender and survival in the Titanic dataset?

A5: From box plots of age by gender and survival in the Titanic dataset, you might observe:

1. Age distributions between survivors and non-survivors
   - Younger passengers may have higher survival rates
   - Median age differences between those who survived and those who didn't
2. Gender differences in survival patterns
   - Women might show higher survival rates across age groups
   - Men's survival might be more age-dependent
3. Outliers in specific demographic groups
   - Very young or old passengers in specific gender groups

4. Evidence of "women and children first" policy
    - Higher survival rates for younger age groups
    - Higher survival rates for females across ages
5. Variability within groups
    - More consistent age patterns in some groups than others

These insights would be preliminary and should be confirmed with statistical tests.

Advanced/Related Questions

Q6: What are the advantages and disadvantages of box plots compared to other visualization methods?

A6: Advantages of box plots:

- Efficiently summarize distributions with minimal space
- Allow easy comparison across multiple groups
- Clearly identify outliers
- Show measures of central tendency and spread
- Are not affected by bin size choices (unlike histograms)
- Work well with large datasets
- Standardized format makes them easy to interpret

Disadvantages of box plots:

- Hide the actual distribution shape (can't see bimodality)
- Don't show the actual sample size
- Can obscure the precise distribution of values
- May oversimplify complex distributions
- Don't show the underlying data points (unless modified)
- Can be less intuitive for non-technical audiences

Q7: How can you enhance a basic box plot to show more information?

A7: Enhanced box plot variations include:

1. Box plot with individual data points (strip plot overlay):

python

```python
plt.figure(figsize=(12, 6))
sns.boxplot(x='sex', y='age', hue='survived', data=titanic)
sns.stripplot(x='sex', y='age', hue='survived', data=titanic,
              dodge=True, alpha=0.3, jitter=True)
plt.show()
```

2. Box plot with violin plot overlay:

python

```python
plt.figure(figsize=(12, 6))
sns.violinplot(x='sex', y='age', hue='survived', data=titanic,
               inner=None, alpha=0.4)
sns.boxplot(x='sex', y='age', hue='survived', data=titanic,
            width=0.3, palette="Set3")
plt.show()
```

3. Box plot with swarm plot (no overlapping points):

python

```python
plt.figure(figsize=(12, 6))
sns.boxplot(x='sex', y='age', hue='survived', data=titanic)
sns.swarmplot(x='sex', y='age', hue='survived', data=titanic,
              dodge=True, alpha=0.6)
plt.show()
```

4. Notched box plot (showing confidence interval around median):

python

```python
plt.figure(figsize=(12, 6))
sns.boxplot(x='sex', y='age', hue='survived', data=titanic, notch=True)
plt.show()
```

Q8: When would you use a box plot versus a violin plot?

A8: Use box plots when:

- You need precise statistical summaries (quartiles, median)
- Comparing multiple groups with emphasis on central tendency
- Identifying outliers is a primary concern
- Working with smaller datasets where distribution shape is less reliable

- Presenting to audiences familiar with box plots
- Space is limited and you need compact visualization

Use violin plots when:

- The shape of the distribution is important (e.g., identifying bimodality)
- You want to show the full probability density of the data
- Comparing distribution shapes across categories
- Working with larger datasets where density estimation is reliable
- Presenting richer visual information is valuable
- You're more interested in the overall distribution than specific quartiles

Practical 10: Data Visualization III

Basic Questions and Answers

Q1: How would you identify the different types of features in a dataset?

A1: To identify feature types in a dataset:

1. Check data types using DataFrame methods:

```python
# Display data types of each column
df.dtypes
# Get a summary of data types
df.info()
```

2. Distinguish between:
   - Numeric features:
     - Continuous (floating-point, representing measurements)
     - Discrete (integers, representing counts)
   - Categorical features:
     - Nominal (unordered categories, like colors)
     - Ordinal (ordered categories, like size: S/M/L)
   - Temporal features (dates, times)
   - Text features
   - Binary features (True/False, 0/1)
3. For ambiguous cases, examine unique values:

```python
# Check if a numeric column might actually be categorical
df['column'].nunique()  # Small number suggests categorical
df['column'].value_counts()  # See distribution of values
```

Q2: How would you create histograms for all numeric features in a dataset?

A2: You can create histograms for all numeric features using various approaches:

```python
# Method 1: Using pandas built-in hist method
import pandas as pd
import matplotlib.pyplot as plt

# Assuming iris dataset is loaded
import seaborn as sns
iris = sns.load_dataset('iris')

# Create histograms for all numeric columns
iris.hist(figsize=(12, 10))
plt.tight_layout()
plt.show()

# Method 2: Using seaborn's displot for more customization
numeric_features = iris.select_dtypes(include=['float64', 'int64']).columns
for feature in numeric_features:
    plt.figure(figsize=(8, 5))
    sns.histplot(data=iris, x=feature, kde=True, hue='species')
    plt.title(f'Distribution of {feature}')
    plt.show()

# Method 3: Using seaborn's FacetGrid for grid layout
g = sns.FacetGrid(iris.melt(id_vars=['species'],
                          value_vars=numeric_features,
                          var_name='feature',
                          value_name='value'),
                col='feature', col_wrap=2, height=4, aspect=1.5)
g.map(sns.histplot, 'value', kde=True)
g.set_titles('{col_name}')
plt.show()
```

Q3: What is a box plot and how would you create box plots for all features in a dataset?

A3: A box plot shows the distribution of a dataset through quartiles, with outliers plotted as individual points. To create box plots for all features:

python

```python
# Method 1: Individual box plots
for feature in numeric_features:
    plt.figure(figsize=(8, 5))
    sns.boxplot(data=iris, y=feature, x='species')
    plt.title(f'Box Plot of {feature} by Species')
    plt.show()

# Method 2: Combined box plots using melt for long-format data
melted_iris = iris.melt(id_vars=['species'],
                        value_vars=numeric_features,
                        var_name='feature',
                        value_name='value')
plt.figure(figsize=(12, 8))
sns.boxplot(data=melted_iris, x='feature', y='value', hue='species')
plt.title('Box Plots for All Features by Species')
plt.xticks(rotation=45)
plt.show()

# Method 3: Using catplot for faceted box plots
sns.catplot(data=melted_iris, x='species', y='value', col='feature',
           kind='box', col_wrap=2, height=4, aspect=1.2)
plt.show()
```

Q4: How would you identify outliers using visualization techniques?

A4: Outliers can be identified using several visualization techniques:

1. Box plots:

python

```python
plt.figure(figsize=(12, 6))
sns.boxplot(data=iris, orient='h')
plt.title('Box Plots Showing Outliers')
plt.show()
```

2. Scatter plots:

python

```python
plt.figure(figsize=(10, 8))
sns.scatterplot(data=iris, x='sepal_length', y='sepal_width', hue='species')
plt.title('Scatter Plot to Identify Outliers')
plt.show()
```

3. Z-score visualization:

python

```python
from scipy import stats
import numpy as np

# Calculate z-scores
z_scores = stats.zscore(iris.select_dtypes(include=[np.number]))
z_scores_df = pd.DataFrame(z_scores, columns=iris.select_dtypes(include=[np.number]).columns)

# Plot z-scores
plt.figure(figsize=(12, 10))
sns.heatmap(z_scores_df.abs() > 3, cmap='viridis', cbar=False)
plt.title('Outliers by Z-score > 3')
plt.show()
```

4. Histograms with density plots:

python

```python
plt.figure(figsize=(10, 6))
sns.histplot(data=iris, x='petal_length', kde=True)
plt.title('Histogram with Density Plot Showing Potential Outliers')
plt.show()
```

Q5: How do you compare distributions across different groups or categories?

A5: To compare distributions across different groups:

1. Side-by-side box plots:

python

```python
plt.figure(figsize=(12, 6))
sns.boxplot(data=iris, x='species', y='sepal_length')
plt.title('Sepal Length Distribution by Species')
plt.show()
```

2. Violin plots:

```python
plt.figure(figsize=(12, 6))
sns.violinplot(data=iris, x='species', y='sepal_length')
plt.title('Violin Plot of Sepal Length by Species')
plt.show()
```

3. Grouped histograms:

```python
plt.figure(figsize=(12, 6))
sns.histplot(data=iris, x='sepal_length', hue='species', kde=True, element='step')
plt.title('Histogram of Sepal Length by Species')
plt.show()
```

4. Ridge plots:

```python
import joypy
from matplotlib import cm

plt.figure(figsize=(10, 8))
fig, axes = joypy.joyplot(iris, by='species', column=['sepal_length', 'sepal_width',
                                                      'petal_length', 'petal_width'],
                         colormap=cm.viridis)
plt.title('Ridge Plots of Features by Species', fontsize=12, pad=10)
plt.show()
```

5. Faceted KDE plots:

```python
g = sns.FacetGrid(iris, col="species", height=4, aspect=1)
g.map(sns.kdeplot, "sepal_length", fill=True)
g.set_titles("{col_name}")
plt.show()
```

Advanced/Related Questions

Q6: How do you handle visualization of highly skewed data?

A6: For visualizing highly skewed data:

1. Apply transformations before plotting:

python

```python
# Log transformation for right-skewed data
import numpy as np
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(data=df, x='right_skewed_feature')
plt.title('Original Distribution')

plt.subplot(1, 2, 2)
sns.histplot(data=df, x=np.log1p(df['right_skewed_feature']))
plt.title('Log-Transformed Distribution')
plt.tight_layout()
plt.show()

# Other transformations based on skewness
# Square root: np.sqrt(x)
# Box-Cox: from scipy.stats import boxcox
```

2. Use visualization methods less sensitive to skewness:

python

```python
# Q-Q plot to compare with normal distribution
import scipy.stats as stats
plt.figure(figsize=(10, 6))
stats.probplot(df['skewed_feature'], plot=plt)
plt.title('Q-Q Plot')
plt.show()

# ECDFs (Empirical Cumulative Distribution Functions)
plt.figure(figsize=(10, 6))
sns.ecdfplot(data=df, x='skewed_feature')
plt.title('ECDF Plot')
plt.show()
```

3. Adjust axis scales:

python

```python
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(data=df, x='right_skewed_feature')
plt.title('Linear Scale')

plt.subplot(1, 2, 2)
sns.histplot(data=df, x='right_skewed_feature')
plt.xscale('log')
plt.title('Log Scale')
plt.tight_layout()
plt.show()
```

Q7: How would you visualize relationships between multiple numeric features simultaneously?

A7: To visualize relationships between multiple numeric features:

1. Correlation matrix heatmap:

python

```python
plt.figure(figsize=(10, 8))
correlation_matrix = iris.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

2. Pairplot (scatterplot matrix):

python

```python
plt.figure(figsize=(12, 10))
sns.pairplot(iris, hue='species', corner=True)
plt.suptitle('Pairplot of Iris Features', y=1.02)
plt.show()
```

3. Parallel coordinates plot:

python

```python
from pandas.plotting import parallel_coordinates

plt.figure(figsize=(12, 6))
parallel_coordinates(iris, 'species', colormap=plt.cm.viridis)
plt.title('Parallel Coordinates Plot')
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```

4. Andrews curves:

python

```python
from pandas.plotting import andrews_curves

plt.figure(figsize=(12, 6))
andrews_curves(iris, 'species', colormap=plt.cm.viridis)
plt.title('Andrews Curves')
plt.show()
```

5. Radar/Spider plot:

python

```python
import matplotlib.pyplot as plt
import numpy as np

# Group by species and get mean values
grouped_means = iris.groupby('species').mean().reset_index()

# Create radar plot
features = numeric_features
num_features = len(features)

# Create angle for each feature
angles = np.linspace(0, 2*np.pi, num_features, endpoint=False).tolist()
angles += angles[:1]  # Close the loop

fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(polar=True))

for i, species in enumerate(grouped_means['species']):
    values = grouped_means.loc[i, features].values.tolist()
    values += values[:1]  # Close the loop

    # Plot values
    ax.plot(angles, values, linewidth=2, label=species)
    ax.fill(angles, values, alpha=0.1)

# Set labels
ax.set_xticks(angles[:-1])
ax.set_xticklabels(features)
plt.title('Radar Chart of Mean Values by Species')
plt.legend(loc='upper right')
plt.show()
```

Q8: What visualization techniques would you use to compare the distributions of features in a dataset to identify which ones can best separate different classes?

A8: To identify features that best separate classes:

1. Box plots with overlaid strip plots by class:

python

```python
for feature in numeric_features:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='species', y=feature, data=iris)
    sns.stripplot(x='species', y=feature, data=iris, jitter=True, alpha=0.3)
    plt.title(f'Distribution of {feature} by Species')
    plt.show()
```

2. Violin plots with box plots inside:

python

```python
for feature in numeric_features:
    plt.figure(figsize=(10, 6))
    sns.violinplot(x='species', y=feature, data=iris, inner='box')
    plt.title(f'Violin Plot of {feature} by Species')
    plt.show()
```

3. Swarm plots:

python

```python
for feature in numeric_features:
    plt.figure(figsize=(10, 6))
    sns.swarmplot(x='species', y=feature, data=iris)
    plt.title(f'Swarm Plot of {feature} by Species')
    plt.show()
```

4. Class separation visualization:

python

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# LDA for visualization
lda = LDA(n_components=2)
X = iris.drop('species', axis=1)
y = iris['species']
X_lda = lda.fit_transform(X, y)

# Plot LDA results
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_lda[:, 0], X_lda[:, 1], c=iris['species'].astype('category').cat.codes)
plt.legend(handles=scatter.legend_elements()[0], labels=iris['species'].unique())
plt.title('LDA Projection Showing Class Separation')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.tight_layout()
plt.show()
```

5. Feature importance using decision trees:

python

```python
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

# Train a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Plot feature importances
plt.figure(figsize=(10, 6))
importances = pd.Series(rf.feature_importances_, index=X.columns)
importances.sort_values().plot(kind='barh')
plt.title('Feature Importance for Class Separation')
plt.show()
```

Group B: Big Data Analytics - JAVA/SCALA

Practical 1: WordCount with Hadoop MapReduce

Basic Questions and Answers

Q1: What is Hadoop MapReduce and what problem does it solve?

A1: Hadoop MapReduce is a programming model and software framework for distributed processing of large datasets across clusters of computers. It solves the problem of processing massive amounts of data in parallel by breaking the task into smaller sub-tasks that can be processed independently and then combining the results. The key benefits include:

- Scalability to handle petabytes of data
- Fault tolerance through data replication
- Distributed processing to reduce computation time
- Data locality (moving computation to data rather than vice versa)
- Simple programming model (Map and Reduce functions)

Q2: Explain the MapReduce paradigm and its key components.

A2: The MapReduce paradigm consists of two main phases:

1. Map phase:
    - Input data is divided into chunks
    - Each chunk is processed independently by a mapper
    - Mapper processes input records and emits key-value pairs
    - The output is sorted and partitioned based on keys
2. Reduce phase:
    - Reducers receive grouped key-value pairs from mappers
    - Each reducer processes values associated with a specific key
    - Reducer outputs final results as key-value pairs

Key components:

- JobTracker/ResourceManager: Coordinates jobs and resource allocation
- TaskTracker/NodeManager: Manages task execution on individual nodes
- HDFS (Hadoop Distributed File System): Stores input and output data
- InputFormat: Defines how input is split and read
- OutputFormat: Defines how output is written
- Partitioner: Determines which reducer receives which keys

Q3: What are the main functions in a typical MapReduce WordCount program?

A3: A typical WordCount program in Hadoop MapReduce includes:

1. Mapper class:
    - Takes text input and splits it into words

- Emits each word with a count of 1
- Key function: `map(LongWritable key, Text value, Context context)`

java

```java
public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Interrup
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

2. Reducer class:
   - Aggregates the counts for each word
   - Key function: `reduce(Text key, Iterable<IntWritable> values, Context context)`

java

```java
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExcept
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

3. Driver class (main method):
   - Configures and submits the MapReduce job
   - Sets input/output paths, mapper/reducer classes, and other configurations

java

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1
```