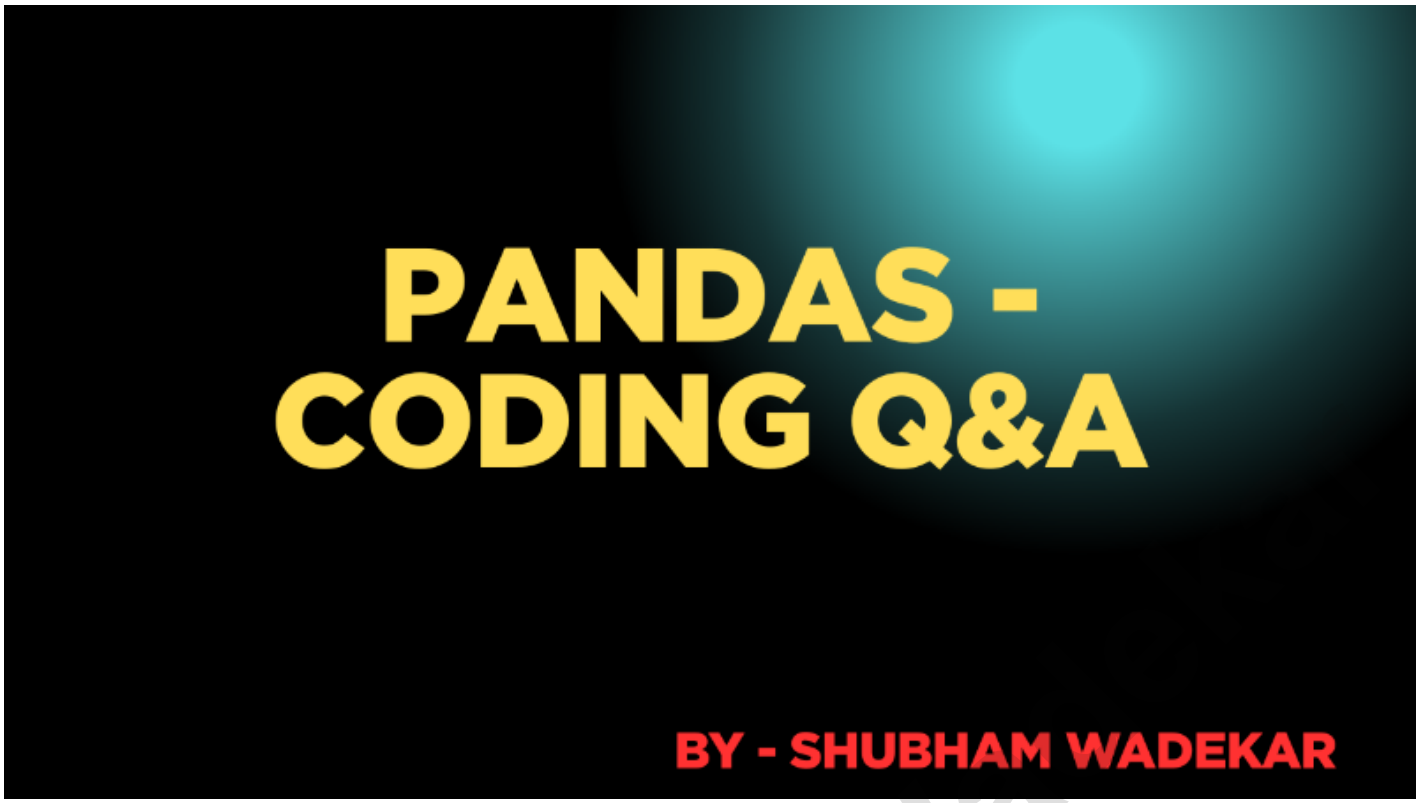

70 PANDAS CODING QUESTIONS FOR DATA ENGINEERS

1. Load a CSV file into a DataFrame and display the first 5 rows.

Explanation:
This program loads a CSV file into a Pandas DataFrame and displays the first 5 rows.

Logic:

- Use pd.read_csv() to load the CSV file.
- Use .head() to display the first 5 rows.

Program:

```
import pandas as pd

# Load CSV file
df = pd.read_csv("data.csv")

# Display first 5 rows
print(df.head())
```

2. Filter rows where a specific column value is greater than a threshold (e.g., age > 30).

Explanation:
This program filters rows where the value in the age column is greater than 30.

Logic:

- Use a conditional expression to filter rows.

Program:

```
filtered_df = df[df['age'] > 30]
print(filtered_df)
```

3. Select specific columns from a DataFrame and display them.

Explanation:
This program selects specific columns (e.g., name and age) from a DataFrame.

Logic:

- Pass a list of column names to the DataFrame.

Program:

```
selected_columns = df[['name', 'age']]
print(selected_columns)
```

4. Add a new column to a DataFrame that is a function of existing columns (e.g., total = price * quantity).

Explanation:
This program adds a new column total calculated as the product of price and quantity.

Logic:

- Use arithmetic operations to create the new column.

Program:

```
df['total'] = df['price'] * df['quantity']
print(df)
```

5. Rename columns in a DataFrame to more meaningful names.

Explanation:
This program renames columns in a DataFrame.

Logic:

- Use the .rename() method with a dictionary mapping old names to new names.

Program:

```
df = df.rename(columns={'old_name1': 'new_name1', 'old_name2': 'new_name2'})
print(df)
```

6. Drop a column from a DataFrame.

Explanation:
This program drops a specific column (e.g., unwanted_column) from a DataFrame.

Logic:

- Use the .drop() method with the columns parameter.

Program:

```
df = df.drop(columns=['unwanted_column'])
print(df)
```

7. Drop rows with missing values in any column.

Explanation:
This program drops rows where any column has missing values.

Logic:

- Use the .dropna() method.

Program:

```
df = df.dropna()
print(df)
```

8. Fill missing values in a column with the mean of that column.

Explanation:
This program fills missing values in the age column with the mean of the column.

Logic:

- Use the .fillna() method with the mean value.

Program:

```
df['age'] = df['age'].fillna(df['age'].mean())
print(df)
```

9. Sort a DataFrame by a specific column in ascending order.

Explanation:
This program sorts a DataFrame by the age column in ascending order.

Logic:

- Use the .sort_values() method.

Program:

```
df = df.sort_values(by='age', ascending=True)
print(df)
```

10. Reset the index of a DataFrame after sorting or filtering.

Explanation:
This program resets the index of a DataFrame after sorting or filtering.

Logic:

- Use the .reset_index() method with drop=True to avoid adding the old index as a column.

Program:

```
df = df.reset_index(drop=True)
print(df)
```

11. Merge two DataFrames on a common column (e.g., customer ID).

Explanation:
This program merges two DataFrames (df1 and df2) on the customer_id column.

Logic:

- Use the .merge() method.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id')
print(merged_df)
```

12. Concatenate two DataFrames vertically (row-wise).

Explanation:
This program concatenates two DataFrames (df1 and df2) vertically.

Logic:

- Use the pd.concat() function with axis=0.

Program:

```
concatenated_df = pd.concat([df1, df2], axis=0)
print(concatenated_df)
```

13. Concatenate two DataFrames horizontally (column-wise).

Explanation:
This program concatenates two DataFrames (df1 and df2) horizontally.

Logic:

- Use the pd.concat() function with axis=1.

Program:

```
concatenated_df = pd.concat([df1, df2], axis=1)
print(concatenated_df)
```

14. Group data by a categorical column and calculate the mean of a numerical column.

Explanation:
This program groups data by the category column and calculates the mean of the price column.

Logic:

- Use the .groupby() method and .mean() function.

Program:

```
grouped_df = df.groupby('category')['price'].mean()
print(grouped_df)
```

15. Group data by a categorical column and calculate the sum of a numerical column.

Explanation:
This program groups data by the category column and calculates the sum of the price column.

Logic:

- Use the .groupby() method and .sum() function.

Program:

```
grouped_df = df.groupby('category')['price'].sum()
print(grouped_df)
```

16. Apply a function to a column (e.g., convert all values to uppercase).

Explanation:
This program converts all values in the name column to uppercase.

Logic:

- Use the .apply() method with a lambda function.

Program:

```
df['name'] = df['name'].apply(lambda x: x.upper())
print(df)
```

17. Create a pivot table from a DataFrame to summarize data.

Explanation:
This program creates a pivot table summarizing the price column by category and region.

Logic:

- Use the .pivot_table() method.

```
pivot_table = df.pivot_table(values='price', index='category', columns='region', aggfunc='mean')
print(pivot_table)
```

18. Melt a DataFrame from wide to long format.

Explanation:
This program melts a DataFrame from wide to long format.

Logic:

- Use the .melt() method.

```
melted_df = df.melt(id_vars=['id'], value_vars=['col1', 'col2'], var_name='variable', value_name
='value')
print(melted_df)
```

19. Perform a cross-tabulation between two columns.

Explanation:
This program performs a cross-tabulation between the category and region columns.

Logic:

- Use the pd.crosstab() function.

Program:

```
cross_tab = pd.crosstab(df['category'], df['region'])
print(cross_tab)
```

20. Calculate the correlation between numerical columns in a DataFrame.

Explanation:
This program calculates the correlation between numerical columns in a DataFrame.

Logic:

- Use the .corr() method.

Program:

```
correlation_matrix = df.corr()
print(correlation_matrix)
```

21. Handle duplicate rows in a DataFrame by keeping the first occurrence.

Explanation:
This program removes duplicate rows, keeping the first occurrence.

Logic:

- Use the .drop_duplicates() method.

Program:

```
df = df.drop_duplicates(keep='first')
print(df)
```

22. Handle outliers in a column by replacing values above a threshold with the threshold value.

Explanation:
This program replaces values in the price column above 100 with 100.

Logic:

- Use the .clip() method.

Program:

```
df['price'] = df['price'].clip(upper=100)
print(df)
```

23. Perform a SQL-like join on two DataFrames based on a common column.

Explanation:
This program performs an inner join on two DataFrames (df1 and df2) based on the customer_id column.

Logic:

- Use the .merge() method.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id', how='inner')
print(merged_df)
```

24. Perform a SQL-like left join on two DataFrames.

Explanation:
This program performs a left join on two DataFrames (df1 and df2) based on the customer_id column.

Logic:

- Use the .merge() method with how='left'.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id', how='left')
print(merged_df)
```

25. Perform a SQL-like right join on two DataFrames.

Explanation:
This program performs a right join on two DataFrames (df1 and df2) based on the customer_id column.

Logic:

- Use the .merge() method with how='right'.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id', how='right')
print(merged_df)
```

26. Perform a SQL-like outer join on two DataFrames.

Explanation:
This program performs an outer join on two DataFrames (df1 and df2) based on the customer_id column.

Logic:

- Use the .merge() method with how='outer'.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id', how='outer')
print(merged_df)
```

27. Perform a SQL-like inner join on two DataFrames.

Explanation:
This program performs an inner join on two DataFrames (df1 and df2) based on the customer_id column.

Logic:

- Use the .merge() method with how='inner'.

Program:

```
merged_df = pd.merge(df1, df2, on='customer_id', how='inner')
print(merged_df)
```

28. Use the apply() function to create a new column based on existing columns (e.g., calculate BMI from height and weight).

Explanation:
This program calculates BMI using the formula:

BMI = $$\text{BMI} = \frac{\text{weight (kg)}}{\text{height (m)}^2}$$

Logic:

- Use the .apply() method with a lambda function.

Program:

```
df['bmi'] = df.apply(lambda row: row['weight'] / (row['height'] ** 2), axis=1)
print(df)
```

29. Use the map() function to replace values in a column based on a dictionary mapping.

Explanation:
This program replaces values in the gender column using a dictionary mapping.

Logic:

- Use the .map() method.

Program:

```
gender_map = {'M': 'Male', 'F': 'Female'}
df['gender'] = df['gender'].map(gender_map)
print(df)
```

30. Use the groupby() function to calculate multiple aggregations (e.g., mean, sum, count) for a column.

Explanation:
This program calculates the mean, sum, and count of the price column grouped by category.

Logic:

- Use the .groupby() method with .agg().

Program:

```
grouped_df = df.groupby('category')['price'].agg(['mean', 'sum', 'count'])
print(grouped_df)
```

31. You have a dataset with customer information. Remove all rows where the customer's age is missing.

Explanation:
This program removes rows where the age column has missing values.

Logic:

- Use the .dropna() method with the subset parameter.

Program:

```
df = df.dropna(subset=['age'])
print(df)
```

32. You have sales data. Calculate the total sales for each product category.

Explanation:
This program calculates the total sales for each category.

Logic:

- Use the .groupby() method and .sum() function.

Program:

```
total_sales = df.groupby('category')['sales'].sum()
print(total_sales)
```

33. You have employee data. Find the average salary for each department.

Explanation:
This program calculates the average salary for each department.

Logic:

- Use the .groupby() method and .mean() function.

Program:

```
avg_salary = df.groupby('department')['salary'].mean()
print(avg_salary)
```

34. You have a dataset with timestamps. Extract all rows where the date is between two specific dates.

Explanation:
This program extracts rows where the date column is between start_date and end_date.

Logic:

- Use a conditional expression with .loc[].

Program:

```
start_date = '2023-01-01'
end_date = '2023-12-31'
filtered_df = df.loc[(df['date'] >= start_date) & (df['date'] <= end_date)]
print(filtered_df)
```

35. You have a dataset with product prices. Adjust the prices by applying a 10% discount.

Explanation:
This program applies a 10% discount to the price column.

Logic:

- Multiply the price column by 0.9.

Program:

```
df['price'] = df['price'] * 0.9
print(df)
```

36. You have a dataset with customer reviews. Count the number of positive and negative reviews.

Explanation:
This program counts the number of positive and negative reviews in the review column.

Logic:

- Use the .value_counts() method.

Program:

```
review_counts = df['review'].value_counts()
print(review_counts)
```

37. You have a dataset with website traffic data. Calculate the total number of unique visitors per day.

Explanation:
This program calculates the total number of unique visitors per day.

Logic:

- Use the .groupby() method and .nunique() function.

Program:

```
unique_visitors = df.groupby('date')['visitor_id'].nunique()
print(unique_visitors)
```

38. You have a dataset with stock prices. Calculate the daily percentage change in stock prices.

Explanation:
This program calculates the daily percentage change in the price column.

Logic:

- Use the .pct_change() method.

Program:

```
df['daily_change'] = df['price'].pct_change()
print(df)
```

39. You have a dataset with student grades. Find the top 5 students with the highest grades.

Explanation:
This program finds the top 5 students with the highest grades.

Logic:

- Use the .nlargest() method.

Program:

```
top_students = df.nlargest(5, 'grade')
print(top_students)
```

40. You have a dataset with employee attendance. Find the employees with the most absences.

Explanation:
This program finds the employees with the most absences.

Logic:

- Use the .nlargest() method.

Program:

```
most_absences = df.nlargest(5, 'absences')
print(most_absences)
```

41. You have a dataset with customer transactions. Identify customers who made purchases above a certain amount.

Explanation:
This program identifies customers who made purchases above $100.

Logic:

- Use a conditional expression to filter rows.

Program:

```
high_spenders = df[df['purchase_amount'] > 100]
print(high_spenders)
```

42. You have a dataset with sensor data. Detect anomalies where the sensor reading is above a threshold.

Explanation:
This program detects anomalies where the sensor_reading is above 100.

Logic:

- Use a conditional expression to filter rows.

Program:

```
anomalies = df[df['sensor_reading'] > 100]
print(anomalies)
```

43. You have a dataset with product sales. Identify the best-selling product in each category.

Explanation:
This program identifies the best-selling product in each category.

Logic:

- Use the .groupby() method and .idxmax() function.

Program:

```
best_selling = df.loc[df.groupby('category')['sales'].idxmax()]
print(best_selling)
```

44. You have a dataset with employee performance. Rank employees based on their performance scores.

Explanation:
This program ranks employees based on their performance_score.

Logic:

- Use the .rank() method.

Program:

```
df['rank'] = df['performance_score'].rank(ascending=False)
print(df)
```

45. You have a dataset with financial data. Calculate the moving average of stock prices over a 7-day window.

Explanation:
This program calculates the 7-day moving average of the price column.

Logic:

- Use the .rolling() method.

Program:

```
df['7_day_ma'] = df['price'].rolling(window=7).mean()
print(df)
```

46. You have a dataset with customer churn. Predict which customers are likely to churn based on their activity.

Explanation:
This program identifies customers with low activity (e.g., fewer than 5 logins).

Logic:

- Use a conditional expression to filter rows.

Program:

```
likely_churn = df[df['logins'] < 5]
print(likely_churn)
```

47. You have a dataset with social media posts. Analyze the sentiment of the posts using a sentiment analysis library.

Explanation:
This program uses the TextBlob library to analyze the sentiment of the post column.

Logic:

- Use TextBlob to calculate sentiment polarity.

Program:

```
from textblob import TextBlob

df['sentiment'] = df['post'].apply(lambda x: TextBlob(x).sentiment.polarity)
print(df)
```

48. You have a dataset with weather data. Calculate the average temperature for each month.

Explanation:
This program calculates the average temperature for each month.

Logic:

- Use the .groupby() method and .mean() function.

Program:

```
df['month'] = pd.to_datetime(df['date']).dt.month
avg_temp = df.groupby('month')['temperature'].mean()
print(avg_temp)
```

49. You have a dataset with e-commerce orders. Identify orders that were delivered late.

Explanation:
This program identifies orders where the delivery_date is later than the expected_delivery_date.

Logic:

- Use a conditional expression to filter rows.

Program:

```
late_orders = df[df['delivery_date'] > df['expected_delivery_date']]
print(late_orders)
```

50. You have a dataset with website clickstream data. Analyze the most common user paths on the website.

Explanation:
This program analyzes the most common user paths by counting the frequency of each path.

Logic:

- Use the .value_counts() method.

Program:

```
common_paths = df['path'].value_counts()
print(common_paths)
```

51. Merge Multiple DataFrames and Calculate Aggregations

Scenario:
You have three DataFrames: orders, customers, and products. Merge them and calculate the total revenue per customer.

Program:

```
import pandas as pd

# Merge DataFrames
merged_df = pd.merge(pd.merge(orders, customers, on='customer_id'), products, on='product_id')

# Calculate total revenue per customer
revenue_per_customer = merged_df.groupby('customer_name')['price'].sum().reset_index()
print(revenue_per_customer)
```

52. Resample Time Series Data and Calculate Rolling Metrics

Scenario:
You have a DataFrame with daily stock prices. Resample the data to monthly frequency and calculate the 3-month rolling average.

Program:

```
import pandas as pd

# Convert date column to datetime and set as index
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# Resample to monthly frequency and calculate rolling average
monthly_avg = df['price'].resample('M').mean()
rolling_avg = monthly_avg.rolling(window=3).mean()
print(rolling_avg)
```

53. Handle Missing Data and Impute Values

Scenario:
You have a DataFrame with missing values in multiple columns. Impute missing values with the median for numerical columns and the mode for categorical columns.

Program:

```
import pandas as pd

# Impute missing values
df.fillna({col: df[col].median() for col in df.select_dtypes(include='number')}, inplace=True)
df.fillna({col: df[col].mode()[0] for col in df.select_dtypes(include='object')}, inplace=True)
```

54. Flatten Nested JSON Data

Scenario:
You have a nested JSON dataset. Flatten it into a DataFrame and extract specific fields.

Program:

```
import pandas as pd
import json

# Load nested JSON and normalize
with open('data.json') as f:
    data = json.load(f)
df = pd.json_normalize(data, record_path=['orders'], meta=['customer_id', 'order_date'])
print(df)
```

55. Detect and Remove Outliers Using Z-Score

Scenario:
You have a DataFrame with numerical data. Detect and remove outliers using the Z-score method.

Program:

```
import pandas as pd
from scipy.stats import zscore

# Calculate Z-scores and filter outliers
df['z_score'] = zscore(df['value'])
df = df[(df['z_score'].abs() <= 3)]
print(df)
```

56. Perform Advanced GroupBy Operations

Scenario:
You have a DataFrame with sales data. Group by region and product_category, and calculate the total sales and average discount.

Program:

```
import pandas as pd

# Group by multiple columns and calculate aggregations
grouped_df = df.groupby(['region', 'product_category']).agg(total_sales=('sales', 'sum'), avg_di
scount=('discount', 'mean'))
print(grouped_df)
```

57. Create a Pivot Table with Multiple Aggregations

Scenario:
You have a DataFrame with sales data. Create a pivot table showing total sales and average price per region and product category.

Program:

```
import pandas as pd

# Create pivot table with multiple aggregations
pivot_table = df.pivot_table(values=['sales', 'price'], index='region', columns='product_categor
y', aggfunc={'sales': 'sum', 'price': 'mean'})
print(pivot_table)
```

58. Merge DataFrames and Handle Duplicates

Scenario:
You have two DataFrames: df1 and df2. Merge them on a common column and remove duplicate rows.

Program:

```
import pandas as pd

# Merge DataFrames and remove duplicates
merged_df = pd.merge(df1, df2, on='key_column').drop_duplicates()
print(merged_df)
```

59. Calculate Cumulative Sum and Percentage Contribution

Scenario:
You have a DataFrame with monthly sales data. Calculate the cumulative sum and percentage contribution of each month to the total sales.

Program:

```
import pandas as pd

# Calculate cumulative sum and percentage contribution
df['cumulative_sum'] = df['sales'].cumsum()
```

60. Perform Advanced Filtering with Multiple Conditions

Scenario:
You have a DataFrame with customer data. Filter rows where the customer is from USA, has made more than 5 purchases, and the total spend is above $1000.

Program:

```
import pandas as pd

# Filter rows with multiple conditions
filtered_df = df[(df['country'] == 'USA') & (df['purchases'] > 5) & (df['total_spend'] > 1000)]
print(filtered_df)
```

61. Calculate Moving Averages and Detect Trends

Scenario:
You have a DataFrame with daily stock prices. Calculate the 7-day and 30-day moving averages and detect trends.

Program:

```
import pandas as pd

# Calculate moving averages
df['7_day_ma'] = df['price'].rolling(window=7).mean()
df['30_day_ma'] = df['price'].rolling(window=30).mean()
print(df)
```

62. Perform Advanced String Operations

Scenario:
You have a DataFrame with text data. Extract all email addresses from a column and count their occurrences.

Program:

```
import pandas as pd
import re

# Extract email addresses and count occurrences
df['emails'] = df['text'].apply(lambda x: re.findall(r'[\w\.-]+@[\w\.-]+', x))
email_counts = df['emails'].explode().value_counts()
print(email_counts)
```

63. Handle Time Zones in Time Series Data

Scenario:
You have a DataFrame with timestamps in UTC. Convert the timestamps to a specific time zone and extract the hour.

Program:

```
import pandas as pd

# Convert timestamps to a specific time zone and extract hour
df['timestamp'] = pd.to_datetime(df['timestamp']).dt.tz_localize('UTC').dt.tz_convert
ew_York')
df['hour'] = df['timestamp'].dt.hour
```

64. Perform Advanced Data Cleaning

Scenario:
You have a DataFrame with messy data. Clean the data by removing special characters, converting text to lowercase, and stripping whitespace.

Program:

```
import pandas as pd

# Clean text data
df['text'] = df['text'].str.replace(r'[^\w\s]', '', regex=True).str.lower().str.strip()
print(df)
```

65. Calculate Percentiles and Bin Data

Scenario:
You have a DataFrame with numerical data. Calculate percentiles and bin the data into quartiles.

Program:

```
import pandas as pd

# Calculate percentiles and bin data
df['percentile'] = df['value'].rank(pct=True)
df['quartile'] = pd.qcut(df['value'], q=4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

66. Perform Advanced Data Transformation

Scenario:
You have a DataFrame with hierarchical data. Flatten the hierarchical structure and pivot the data.

Program:

```
import pandas as pd

# Flatten hierarchical data and pivot
flattened_df = pd.json_normalize(df['hierarchical_column'])
pivot_df = flattened_df.pivot_table(values='value', index='index_column', columns='category_colu
mn')
print(pivot_df)
```

67. Calculate Exponential Moving Average

Scenario:
You have a DataFrame with stock prices. Calculate the exponential moving average (EMA) with a span of 10 days.

Program:

```
import pandas as pd

# Calculate exponential moving average
df['ema'] = df['price'].ewm(span=10, adjust=False).mean()
print(df)
```

68. Perform Advanced Data Validation

Scenario:
You have a DataFrame with customer data. Validate the data by checking for missing values, duplicates, and invalid entries.

Program:

```
import pandas as pd

# Validate data
missing_values = df.isnull().sum()
duplicates = df.duplicated().sum()
invalid_entries = df[~df['email'].str.contains(r'^[\w\.-]+@[\w\.-]+$')]
print(missing_values, duplicates, invalid_entries)
```

69. Perform Advanced Data Visualization

Scenario:
You have a DataFrame with sales data. Visualize the sales trend over time using a line plot.

Program:

```
import pandas as pd
import matplotlib.pyplot as plt

# Plot sales trend
df.set_index('date')['sales'].plot(kind='line')
plt.show()
```

70. Perform Advanced Data Export

Scenario:
You have a DataFrame with processed data. Export the data to multiple formats (CSV, Excel, and JSON).

Program:

```
import pandas as pd

# Export data to multiple formats
df.to_csv('data.csv', index=False)
df.to_excel('data.xlsx', index=False)
df.to_json('data.json', orient='records')
```