Exploratory Data Analysis (EDA)

1. Data Loading

- **Read CSV File**: `df = spark.read.csv('filename.csv', header=True, inferSchema=True)`
- **Read Parquet File**: `df = spark.read.parquet('filename.parquet')`
- **Read from JDBC (Databases)**: `df = spark.read.format("jdbc").options(url="jdbc_url", dbtable="table_name").load()`

2. Basic Data Inspection

- **Display Top Rows**: `df.show()`
- **Print Schema**: `df.printSchema()`
- **Summary Statistics**: `df.describe().show()`
- **Count Rows**: `df.count()`
- **Display Columns**: `df.columns`

3. Data Cleaning

- **Drop Missing Values**: `df.na.drop()`
- **Fill Missing Values**: `df.na.fill(value)`
- **Drop Column**: `df.drop('column_name')`
- **Rename Column**: `df.withColumnRenamed('old_name', 'new_name')`

4. Data Transformation

- **Select Columns**: `df.select('column1', 'column2')`
- **Add New or Transform Column**: `df.withColumn('new_column', expression)`
- **Filter Rows**: `df.filter(df['column'] > value)`
- **Group By and Aggregate**: `df.groupby('column').agg({'column': 'sum'})`
- **Sort Rows**: `df.sort(df['column'].desc())`

5. SQL Queries on DataFrames

- **Create Temporary View**:

- **SQL Query**: spark.sql('SELECT * FROM view_name WHERE condition')

6. Statistical Analysis

- **Correlation Matrix**: from pyspark.ml.stat import Correlation; Correlation.corr(df, 'column') **Covariance**:
- df.stat.cov('column1', 'column2')
- **Frequency Items**: df.stat.freqItems(['column1', 'column2'])
- **Sample By**: df.sampleBy('column', fractions={'class1': 0.1, 'class2': 0.2})

7. Handling Missing and Duplicated Data

- **Fill Missing Values in Column**: df.fillna({'column': value})
- **Drop Duplicates**: df.dropDuplicates()
- **Replace Value**: df.na.replace(['old_value'], ['new_value'], 'column')

8. Data Conversion and Export

- **Convert to Pandas DataFrame**: pandas_df = df.toPandas()
- **Write DataFrame to CSV**: df.write.csv('path_to_save.csv')
- **Write DataFrame to Parquet**: df.write.parquet('path_to_save.parquet')

9. Column Operations

- **Change Column Type**: df.withColumn('column', df['column'].cast('new_type'))
- **Split Column into Multiple Columns**: df.withColumn('new_col1', split(df['column'], 'delimiter')[0])
- **Concatenate Columns**: df.withColumn('new_column', concat_ws(' ', df['col1'], df['col2']))

10. Date and Time Operations

- **Current Date and Time**: df.withColumn('current_date', current_date())

- **Date Formatting**: `df.withColumn('formatted_date', date_format('dateColumn', 'yyyyMMdd'))`
- **Date Arithmetic**: `df.withColumn('date_plus_days', date_add(df['date'], 5))`

11. Advanced Data Processing

- **Window Functions**: `from pyspark.sql.window import Window; df.withColumn('rank', rank().over(Window.partitionBy('column').orderBy('other_column')))`
- **Pivot Table**: `df.groupBy('column').pivot('pivot_column').sum('sum_column')`
- **UDF (User Defined Functions)**: `from pyspark.sql.functions import udf; my_udf = udf(my_python_function); df.withColumn('new_col', my_udf(df['col']))`

12. Performance Optimization

- **Caching DataFrame**: `df.cache()`
- **Repartitioning**: `df.repartition(10)`
- **Broadcast Join Hint**: `df.join(broadcast(df2), 'key', 'inner')`

13. Exploratory Data Analysis Specifics

- **Column Value Counts**: `df.groupBy('column').count().show()`
- **Distinct Values in a Column**: `df.select('column').distinct().show()`
- **Aggregations (sum, max, min, avg)**: `df.groupBy().sum('column').show()`

14. Working with Complex Data Types

- **Exploding Arrays**: `df.withColumn('exploded', explode(df['array_column']))`
- **Working with Structs**: `df.select(df['struct_column']['field'])`
- **Handling Maps**: `df.select(map_keys(df['map_column']))`

15. Joins

- **Inner Join**: `df1.join(df2, df1['id'] == df2['id'])`

- **Left Outer Join**: `df1.join(df2, df1['id'] == df2['id'], 'left_outer')`
- **Right Outer Join**: `df1.join(df2, df1['id'] == df2['id'], 'right_outer')`

16. Saving and Loading Models

- **Saving ML Model**: `model.save('model_path')`
- **Loading ML Model**: `from pyspark.ml.classification import LogisticRegressionModel; LogisticRegressionModel.load('model_path')`

17. Handling JSON and Complex Files

- **Read JSON**: `df = spark.read.json('path_to_file.json')`
- **Explode JSON Object**: `df.selectExpr('json_column.*')`

18. Custom Aggregations

- **Custom Aggregate Function**: `from pyspark.sql import functions as F; df.groupBy('group_column').agg(F.sum('sum_column'))`

19. Working with Null Values

- **Counting Nulls in Each Column**: `df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in df.columns])`
- **Drop Rows with Null Values**: `df.na.drop()`

20. Data Import/Export Tips

- **Read Text Files**: `df = spark.read.text('path_to_file.txt')`
- **Write Data to JDBC**: `df.write.format("jdbc").options(url="jdbc_url", dbtable="table_name").save()`

21. Advanced SQL Operations

- **Register DataFrame as Table**: `df.createOrReplaceTempView('temp_table')`

- **Perform SQL Queries**: `spark.sql('SELECT * FROM temp_table WHERE condition')`

22. Dealing with Large Datasets

- **Sampling Data**: `sampled_df = df.sample(False, 0.1)`
- **Approximate Count Distinct**: `df.select(approx_count_distinct('column')).show()`

23. Data Quality Checks

- **Checking Data Integrity**: `df.checkpoint()`
- **Asserting Conditions**: `df.filter(df['column'] > 0).count()`

24. Advanced File Handling

- **Specify Schema While Reading**: `schema = StructType([...]); df = spark.read.csv('file.csv', schema=schema)`
- **Writing in Overwrite Mode**: `df.write.mode('overwrite').csv('path_to_file.csv')`

25. Debugging and Error Handling

- **Collecting Data Locally for Debugging**: `local_data = df.take(5)`
- **Handling Exceptions in UDFs**: `def safe_udf(my_udf): def wrapper(*args, **kwargs): try: return my_udf(*args, **kwargs) except: return None; return wrapper`

26. Machine Learning Integration

- **Creating Feature Vector**: `from pyspark.ml.feature import VectorAssembler; assembler = VectorAssembler(inputCols=['col1', 'col2'], outputCol='features'); feature_df = assembler.transform(df)`

27. Advanced Joins and Set Operations

- **Cross Join**: `df1.crossJoin(df2)`

- **Set Operations (Union, Intersect, Minus)**: `df1.union(df2); df1.intersect(df2); df1.subtract(df2)`

28. Dealing with Network Data

- **Reading Data from HTTP Source**: `spark.read.format("csv").option("url", "http://example.com/data.csv").load()`

29. Integration with Visualization Libraries

- **Convert to Pandas for Visualization**: `pandas_df = df.toPandas(); pandas_df.plot(kind='bar')`

30. Spark Streaming for Real-Time EDA

- **Reading from a Stream**: `df = spark.readStream.format('source').load()`
- **Writing to a Stream**: `df.writeStream.format('console').start()`

31. Advanced Window Functions

- **Cumulative Sum**: `from pyspark.sql.window import Window; df.withColumn('cum_sum', F.sum('column').over(Window.partitionBy('group_column').orderBy('order_column')))`
- **Row Number**: `df.withColumn('row_num', F.row_number().over(Window.orderBy('column')))`

32. Handling Complex Analytics

- **Rollup**: `df.rollup('column1', 'column2').agg(F.sum('column3'))`
- **Cube for Multi-Dimensional Aggregation**: `df.cube('column1', 'column2').agg(F.sum('column3'))`

33. Dealing with Geospatial Data

- **Using GeoSpark for Geospatial Data**: `from geospark.register import GeoSparkRegistrator; GeoSparkRegistrator.registerAll(spark)`

34. Advanced File Formats

- **Reading ORC Files**: `df = spark.read.orc('filename.orc')`
- **Writing Data to ORC**: `df.write.orc('path_to_file.orc')`

35. Dealing with Sparse Data

- **Using Sparse Vectors**: `from pyspark.ml.linalg import SparseVector; sparse_vec = SparseVector(size, {index: value})`

36. Handling Binary Data

- **Reading Binary Files**: `df = spark.read.format('binaryFile').load('path_to_binary_file')`

37. Efficient Data Transformation

- **Using mapPartitions for Transformation**: `rdd = df.rdd.mapPartitions(lambda partition: [transform(row) for row in partition])`

38. Advanced Machine Learning Operations

- **Using ML Pipelines**: `from pyspark.ml import Pipeline; pipeline = Pipeline(stages=[stage1, stage2]); model = pipeline.fit(df)`
- **Model Evaluation**: `from pyspark.ml.evaluation import BinaryClassificationEvaluator; evaluator = BinaryClassificationEvaluator(); evaluator.evaluate(predictions)`

39. Optimization Techniques

- **Broadcast Variables for Efficiency**: `from pyspark.sql.functions import broadcast; df.join(broadcast(df2), 'key')`
- **Using Accumulators for Global Aggregates**: `accumulator = spark.sparkContext.accumulator(0); rdd.foreach(lambda x: accumulator.add(x))`

40. Advanced Data Import/Export

- **Reading Data from Multiple Sources**: `df = spark.read.format('format').option('option', 'value').load(['path1', 'path2'])`
- **Writing Data to Multiple Formats**: `df.write.format('format').save('path', mode='overwrite')`

41. Utilizing External Data Sources

- **Connecting to External Data Sources (e.g., Kafka, S3)**: `df = spark.read.format('kafka').option('kafka.bootstrap.servers', 'host1:port1').load()`

42. Efficient Use of SQL Functions

- **Using Built-in SQL Functions**: `from pyspark.sql.functions import col, lit; df.withColumn('new_column', col('existing_column') + lit(1))`

43. Exploring Data with GraphFrames

- **Using GraphFrames for Graph Analysis**: `from graphframes import GraphFrame; g = GraphFrame(vertices_df, edges_df)`

44. Working with Nested Data

- **Exploding Nested Arrays**: `df.selectExpr('id', 'explode(nestedArray) as element')`
- **Handling Nested Structs**: `df.select('struct_column.*')`

45. Advanced Statistical Analysis

- **Hypothesis Testing**: `from pyspark.ml.stat import ChiSquareTest; r = ChiSquareTest.test(df, 'features', 'label')`
- **Statistical Functions (e.g., mean, stddev)**: `from pyspark.sql.functions import mean, stddev; df.select(mean('column'), stddev('column'))`

46. Customizing Spark Session

- **Configuring SparkSession**: `spark = SparkSession.builder.appName('app').config('spark.some.config.option', 'value').getOrCreate()`