Data Analysis — Topper-Style Notes

- **Objective**: Derive insights post-preparation.

Excel Statistical Analysis

- **Correlation Analysis**:
  - Enable Data Analysis ToolPak for advanced features
  - Function: `CORREL(range_x, range_y)` for Pearson correlation
  - Create correlation matrices for multiple variables
  - Visualize with scatterplots and trendlines
  - Interpret correlation coefficients and statistical significance
- **Regression Analysis**:
  - `LINEST(y_vals, x_vals, TRUE, TRUE)` returns full regression statistics
  - Multiple regression with several predictors
  - Examine R-squared, F-test, p-values for model evaluation
  - Analyze residuals for assumption validation
- **Forecasting**:
  - `FORECAST.ETS` for exponential smoothing with seasonality
  - `FORECAST.LINEAR` for simple linear trends
  - Handle seasonal patterns and trend components
  - Validate forecasts with out-of-sample testing
- **Outlier Detection**:
  - IQR method: Q1, Q3, bounds (Q1-1.5*IQR, Q3+1.5*IQR)
  - Z-score method for normal distributions
  - Box plots for visual identification
  - Functions: `QUARTILE.INC()`, conditional formatting

Python Data Analysis (Pandas)

- **File Handling**: Read Parquet files for efficient data processing
- **Data Inspection**: `.head()`, `.info()`, `.describe()` for dataset overview
- **Pivot Tables**: Summarize and reshape data for analysis

```
 import pandas as pd
df = pd.read_parquet('data.parquet')
# Create pivot tables
piv = pd.pivot_table(df, index='category', columns='month', values='amount'
# Convert to percentages
piv_pct = piv.div(piv.sum(axis=0), axis=1)
```

- **Correlation Analysis**: Calculate correlations with significance testing

```
 corr = df[['x','y']].corr()
# Statistical significance testing
from scipy.stats import pearsonr
r, p_value = pearsonr(df['x'], df['y'])
print(f"Correlation: {r:.3f}, p-value: {p_value:.3f}")
```

- **Datetime Processing**: Extract temporal features for analysis

```
 df['month'] = pd.to_datetime(df['timestamp']).dt.to_period('M').astype(str
df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
```

- **Visualization**: Create heatmaps and statistical plots

```
 import seaborn as sns
sns.heatmap(piv_pct, cmap='Blues', annot=True)
```

DuckDB Analysis (High-Performance SQL)

- **Storage Optimization**: Parquet format advantages over CSV/JSON/SQLite
    - Faster read/write operations
    - Smaller file sizes with compression
    - Better data typing and schema preservation
- **Performance Features**:
    - Parallel processing for large datasets
    - Columnar storage for analytical queries
    - Memory-efficient operations (no full dataset loading)
- **Direct Parquet Querying**: Query files without importing

```
 -- Analyze flight data directly from Parquet
SELECT carrier, COUNT(*) as flight_count
```

```
FROM '2015_flights.parquet'
GROUP BY carrier
ORDER BY flight_count DESC;

-- Ranking and window functions
SELECT route, distance,
       RANK() OVER (ORDER BY distance DESC) as distance_rank
FROM flights;
```

- **Hybrid Workflow**: Combine DuckDB and Pandas strengths
    - DuckDB: Large aggregations, joins, filtering
    - Pandas: Flexible transformations, plotting, small datasets
- **Integration Example**:

```
 import duckdb
con = duckdb.connect()
# Fast aggregation in DuckDB
result = con.execute("""
    SELECT carrier, AVG(arr_delay) as avg_delay
    FROM '2015_flights.parquet'
    GROUP BY carrier
""").df()
# Then use Pandas for visualization
```

Mixed Workflow Example

```
 import duckdb, pandas as pd
con = duckdb.connect()
# Read Parquet via DuckDB, then to Pandas for plotting
flights = con.execute("SELECT * FROM '2015_flights.parquet'").df()
summary = con.execute("SELECT carrier, AVG(arr_delay) avg_delay FROM flight
```

Performance notes

- DuckDB excels at large aggregations; Pandas shines for flexible transforms and plotting.
- Parquet read/write is faster and smaller than CSV; avoid CSV in hot paths.

Checks and tips

- Prefer Spearman for monotonic but non-linear relations; visualize first.
- Report effect sizes with p-values; adjust for multiple comparisons.
- Use Parquet + DuckDB for large group-bys/joins; Pandas for flexible transforms/plots.
- Treat outliers with robust stats (median/MAD) before modeling.

Advanced theory and tricky exam asks

- **Correlation pitfalls**: Spurious correlation; nonlinearity; outliers dominating Pearson; consider Spearman/Kendall.
- **Multiple testing**: p-hacking risk; control FDR (Benjamini–Hochberg) or Bonferroni; pre-register hypotheses.
- **Confounding**: Simpson’s paradox; stratify or use regression with controls; causal thinking matters.
- **Effect sizes**: Report along with p-values (Cohen’s d, r); practical vs statistical significance.
- **Parquet internals**: Columnar storage, dictionary/RLE encoding, compression (snappy, zstd) → speed and size gains.
- **Outliers**: IQR rule, Z-scores, robust estimators (median, MAD); winsorization vs removal.

Likely exam asks:

- Choose Pearson vs Spearman given skewed, rank-based data.
- Explain why Parquet is faster for columnar scans and joins.
- Interpret a significant p-value that has a trivial effect size.

Deep dive details

Pearson vs Spearman:

```
 import pandas as pd
from scipy.stats import pearsonr, spearmanr
x = pd.Series([1,2,3,4,5])
y = pd.Series([1,1,2,3,8])  # outlier at end
print(pearsonr(x,y))   # sensitive to outlier
print(spearmanr(x,y))  # rank-based, more robust to outliers
```

Multiple testing control (BH-FDR):

```
import numpy as np
from statsmodels.stats.multitest import multipletests
pvals = np.array([0.001, 0.02, 0.04, 0.2, 0.5])
rej, p_adj, *_ = multipletests(pvals, method='fdr_bh')
```

Parquet compression options:

Codec	Speed	Ratio	Notes
snappy	fast	medium	default, great general choice
zstd	medium	high	better compression; good balance
gzip	slow	high	legacy; slower reads

Robust outlier handling:

```
import numpy as np
vals = df['amount']
mad = np.median(np.abs(vals - np.median(vals)))
z_mad = 0.6745 * (vals - np.median(vals)) / mad
df_robust = df[np.abs(z_mad) < 3.5]
```

Spreadsheet statistics

- Purpose: quick, explainable analytics.
- Why: accessible to non-programmers; good for demos.
- Core: correlation, regression, ETS forecasting, outlier detection.
- Examples:

```
=CORREL(B:B, C:C)
=LINEST(Ys, Xs, TRUE, TRUE)
=FORECAST.ETS(TargetDate, Values, Timeline)
```

- Pitfalls: misuse of linear regression; ignoring residual diagnostics; seasonality assumptions.
- Checklist: visualize residuals; report confidence intervals; validate forecasts out-of-sample.

SQL Data Analysis

- **Purpose**: Scalable data summarization and complex joins
- **Why**: Push computation to the database layer for efficiency
- **Database Connection**: Use SQLAlchemy and Pandas for Python integration

```
 from sqlalchemy import create_engine
import pandas as pd

# Connect to database
engine = create_engine('mysql://user:pass@host/db')
df = pd.read_sql('SELECT * FROM table', engine)
```

- **Core Operations**: GROUP BY, window functions, CTEs, early filtering
- **Advanced Analysis Examples**:

```
 -- User activity analysis
SELECT user_id, COUNT(*) as post_count
FROM posts
GROUP BY user_id
ORDER BY post_count DESC;

-- Correlation calculation in SQL
SELECT
  (COUNT(*) * SUM(age * reputation) - SUM(age) * SUM(reputation)) /
  SQRT((COUNT(*) * SUM(age * age) - SUM(age) * SUM(age)) *
       (COUNT(*) * SUM(reputation * reputation) - SUM(reputation) * SUM(rep
  as correlation
FROM users;

-- Window functions for ranking
WITH summary AS (
  SELECT category, amount, date FROM transactions
)
SELECT category, AVG(amount) as avg_amount,
       AVG(amount) OVER () as overall_avg,
       RANK() OVER (ORDER BY AVG(amount) DESC) as rank
FROM summary GROUP BY category;
```

- **Large Dataset Handling**: Perform calculations on aggregated values rather than full datasets
- **Statistical Analysis**: Calculate correlations, regressions, and user concentration metrics

- **AI Integration**: Use ChatGPT to generate SQL queries and Python code
- **Pitfalls**: Double-counting on joins; inefficient filter placement; memory limitations
- **Checklist**: Validate row counts; use explicit join keys; sample large datasets early

AI-Assisted Data Analysis

- **Purpose**: Accelerate analytical workflows and code generation
- **Why**: Enhance productivity while maintaining analytical rigor
- **Applications**:
  - Generate SQL queries and Python analysis code
  - Create statistical tests and interpretation
  - Draft analytical narratives and insights
  - Suggest visualization approaches
- **Best Practices**:
  - Use AI for scaffolding, not final answers
  - Always verify mathematical calculations independently
  - Request multiple approaches for complex problems
  - Ask for code comments and explanations
- **Integration Workflow**:

```
 # Example: AI-generated analysis code
# Always validate results manually
from scipy.stats import pearsonr

# AI suggests correlation analysis
corr_coef, p_value = pearsonr(df['variable1'], df['variable2'])
print(f"Correlation: {corr_coef:.3f} (p={p_value:.3f})")

# Verify with alternative method
manual_corr = df[['variable1', 'variable2']].corr().iloc[0,1]
assert abs(corr_coef - manual_corr) < 1e-10
```

- **Quality Control**:
  - Cross-validate AI suggestions with established methods
  - Test edge cases and boundary conditions
  - Compare results against known benchmarks
  - Document assumptions and limitations

- **Pitfalls**: Silent mathematical errors; plausible but incorrect reasoning; over-reliance
- **Checklist**: Independent verification; unit testing; baseline comparisons; human oversight

Advanced Excel Analysis

- **Regression Analysis**: Enable Data Analysis ToolPak for comprehensive regression.
- Setup: Input dependent/independent variables; interpret R-squared, F-test, p-values.
- Multiple regression: Handle multiple predictors; check coefficient significance.
- Model evaluation: Examine residuals; validate assumptions; scale interpretation.
- Examples:

```
 =LINEST(y_range, x_range, TRUE, TRUE)  # Returns regression statistics
# Adjusted R-squared in output array position [4,1]
# F-statistic significance in position [4,0]
```

- **Outlier Detection**: IQR method with quartile calculations.
- Process: Calculate Q1, Q3 → IQR → bounds (Q1-1.5*IQR, Q3+1.5*IQR).
- Functions: `QUARTILE.INC()`, conditional formatting, box plots.
- Examples:

```
 =QUARTILE.INC(data_range, 1)  # Q1
=QUARTILE.INC(data_range, 3)  # Q3
=IF(value < lower_bound, "Outlier", IF(value > upper_bound, "Outlier", "Nor
```

- Pitfalls: assuming normal distribution; removing vs. investigating outliers.
- Checklist: visualize with box plots; investigate outlier causes; document decisions.

Geospatial Analysis

- Purpose: analyze location-based patterns and relationships.

- Why: location drives many business and research decisions.
- **Excel Approach**:
    - Basic mapping with geographic data types
    - Distance calculations using coordinates
    - Population density analysis with census data
    - Visualization with map charts and Power BI integration
- **Python Approach** (GeoPandas, Folium):

```
 import geopandas as gpd
import folium
from geopy.distance import geodesic

# Distance calculation
dist = geodesic((lat1, lon1), (lat2, lon2)).kilometers

# Store density within radius
stores_nearby = df[df['distance'] <= radius_km]
density = len(stores_nearby)

# Interactive mapping
m = folium.Map(location=[lat, lon], zoom_start=12)
for idx, row in df.iterrows():
    folium.Marker([row['lat'], row['lon']], popup=row['name']).add_to(m)
```

- **QGIS Approach**:
    - Import shapefiles and KML files
    - Create custom geographic boundaries
    - Spatial joins and buffer analysis
    - Professional cartographic output
- Applications: site selection, coverage analysis, demographic studies.
- Pitfalls: coordinate system mismatches; projection distortions; data quality.
- Checklist: validate coordinates; choose appropriate projections; ground-truth results.

Network Analysis

- Purpose: identify relationships, clusters, and influence patterns.
- Why: understand complex interconnected systems.
- **Python Implementation** (NetworkX, scikit-network):

```
import networkx as nx
from sknetwork import clustering

# Build network from relationships
G = nx.Graph()
G.add_edges_from([(actor1, actor2) for actor1, actor2 in movie_pairs])

# Community detection
communities = clustering.Louvain().fit_transform(adjacency_matrix)

# Centrality measures
betweenness = nx.betweenness_centrality(G)
closeness = nx.closeness_centrality(G)
```

- Applications: social networks, collaboration patterns, influence mapping.
- Metrics: degree centrality, betweenness, clustering coefficient.
- Visualization: node positioning, community coloring, edge weighting.
- Pitfalls: scalability with large networks; interpreting centrality measures.
- Checklist: validate network construction; choose appropriate algorithms; interpret communities meaningfully.

Datasette Analysis

- **Purpose**: Lightweight data exploration and sharing via web interface
- **Why**: Quick data publishing and collaborative analysis without complex setup
- **Key Features**:
  - Instant web interface for SQLite databases
  - SQL query interface with result sharing
  - JSON API for programmatic access
  - Plugin ecosystem for extended functionality
- **Workflow**:

```
# Install and run Datasette
pip install datasette
datasette mydatabase.db
# Access via http://localhost:8001
```

- **Use Cases**:
  - Data sharing with non-technical stakeholders

    - Quick exploratory data analysis
    - Creating data APIs from CSV/Excel files
    - Collaborative data investigation
- **Integration**: Convert CSV to SQLite, then serve with Datasette

```
 import sqlite3
import pandas as pd

# Convert data to SQLite for Datasette
df = pd.read_csv('data.csv')
conn = sqlite3.connect('analysis.db')
df.to_sql('data_table', conn, index=False)
```

- **Advanced Features**: Custom SQL queries, faceted browsing, full-text search
- **Pitfalls**: Limited to SQLite; performance with very large datasets; security considerations
- **Checklist**: Validate data import; test query performance; configure appropriate access controls