Module 2. Data analysis. Tools and technologies

In this chapter, we will learn about different tools and how they can help us find the answers we seek in our data. We will cover Excel, Tableau, and Python to understand how each tool will guide our problem-solving journey deeply. The learning curve that will come along the way to use the platform and the challenges among them. Professionals must master a diverse toolkit of technologies to transform raw information into actionable insights. Excel emerges as the foundational platform for data organization, offering robust capabilities beyond simple spreadsheet management. Its advanced functions can enable users to uncover hidden patterns, while data validation features ensure accuracy and consistency across complex datasets.

Tableau elevates data visualization to an art form, turning complex datasets into interactive, compelling narratives. The platform provides flexibility for real-time and offline analysis. Its sophisticated calculation engine and dynamic parameters enable users to create intricate dashboards that tell comprehensive stories, moving beyond static reporting to interactive exploration. From regional sales mapping to customer behavior tracking, Tableau transforms numbers into strategic insights that drive industry decision-making.

Python offers a library ecosystem that empowers analysts to perform complex computational tasks. The panda's library is a cornerstone, providing intuitive data manipulation capabilities through its dataframe structure. Time series analysis becomes particularly accessible, allowing professionals to uncover temporal trends across various domains. The seamless integration with machine learning libraries like scikit-learn further distinguishes Python, enabling a smooth workflow from data preparation to predictive modeling. Whether analyzing financial trends, user behaviors, or scientific datasets, Python provides a versatile and robust environment for extracting meaningful insights.

These three technologies represent more than just tools; they are gateways to understanding the complex narratives hidden within data. By mastering Excel's organizational techniques, Tableau's visualization prowess, and Python's analytical depth, professionals can transform raw numbers into strategic intelligence that drives informed decision-making in an increasingly data-driven world.

Unit 2.1 Data organization and cleaning

Data organization and cleaning are fundamental steps in any analysis process using Excel. The platform offers robust tools for managing and preparing your data for analysis. Excel's Sort and Filter capabilities are your first defense against data chaos when working with raw data. The sorting functionality extends beyond simple alphabetical or numerical arrangements, allowing multiple sorting levels to help reveal patterns and hierarchies within your dataset. Advanced filtering capabilities enable users to apply complex conditions, enabling them to isolate specific data points or identify outliers requiring attention.

Data validation is crucial for maintaining data integrity throughout your analysis process. By implementing data validation rules, you can ensure that all entries conform to specific formats, ranges or custom criteria. This feature is particularly valuable when working with team members or dealing with data entry processes. For instance, you can create dropdown lists that standardize input options, preventing inconsistencies that might otherwise compromise your analysis. Data validation is not just a tool, but a key practice in data analysis, ensuring the accuracy and reliability of your results.

The Text to Columns feature is another powerful data-cleaning tool in Excel. This functionality proves invaluable when dealing with concatenated data or information imported from other systems. Users can split data using delimiters (commas or tabs) or fixed-width parameters, transforming unwieldy combined data into neatly organized columns. The feature also offers advanced options for handling special characters and applying specific formatting to the resulting columns.

Unit 2.2 Introduction to Excel for data analysis

Microsoft Excel remains one of the most widely used tools for data analysis. It offers a combination of accessibility and robust analytical capabilities. This section explores Excel's advanced features for data analysis and visualization.

2.2.1 Excel functions for analysis

The true power of Excel lies in its extensive library of built-in functions that facilitate sophisticated data analysis. Understanding and effectively utilizing these functions can transform raw data into meaningful insights. Let's explore some of the most impactful categories.

Statistical functions form the backbone of quantitative analysis in Excel. These functions go beyond basic calculations to provide deep analytical capabilities. The AVERAGE function calculates the arithmetic mean of a dataset. At the same time, MEDIAN identifies the middle value in a sorted set of numbers, which is particularly useful when dealing with skewed distributions. The MODE function helps identify your dataset's most

frequently occurring values, offering insights into data patterns and trends. For more advanced statistical analysis, STDEV.P and VAR.P calculate population standard deviation and variance, helping you understand your data's spread and variability.

Lookup functions revolutionize the way we cross-reference data in Excel. VLOOKUP, perhaps the most widely used lookup function, allows you to search for specific values in a table and retrieve related information from adjacent columns. The more versatile INDEX-MATCH combination offers greater flexibility, allowing for both vertical and horizontal lookups without the limitations of VLOOKUP. The newer XLOOKUP function combines the best of both worlds, offering more intuitive syntax and greater functionality, including the ability to look in both directions and handle errors more elegantly.

2.2.2 Excel features for data analysis

1. **Data organization and cleaning**

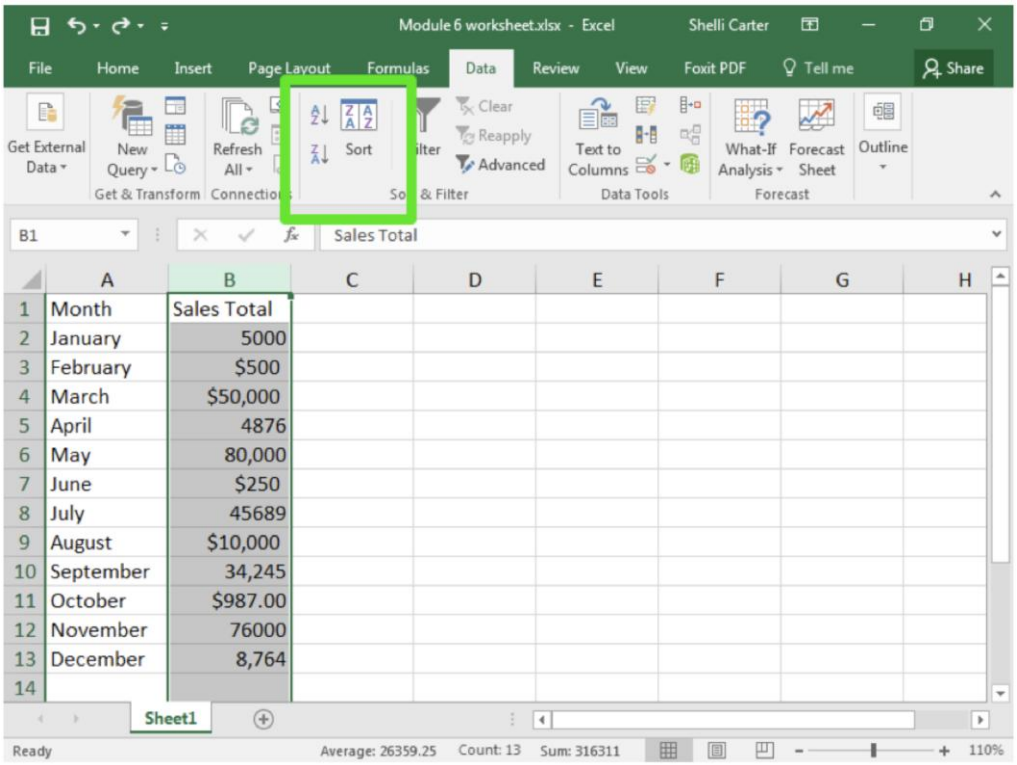
Sort and Filter:

- Custom sorting with multiple levels.

- Advanced filtering with complex conditions.

- Auto-filter for quick data exploration.

Figure 1. Location and use of Sort in Excel



Source: [Untitled image about location and use of Sort in Excel], n. d.

The previous image highlights the location and use of Sort and how you can sort using one or the other column of your choice.

Data validation

This Excel data validation technique allows you to create a dropdown list in a cell by referencing a predefined range of categories. When you select a cell, go to <Data Validation>, choose <List> as the allowed type, and specify a named range (like "Categories") as the source, a dropdown list with those specific options populates.

‘‘‘

Example: setting up dropdown lists.

- Select cell range.

- Data → Data Validation.

- Allow: List.

- Source: =Categories.

‘‘‘

2. Excel functions for analysis

Statistical functions

- AVERAGE(), MEDIAN(), MODE()

- STDEV.P(), VAR.P()

- CORRELATION(), COVAR()

Lookup functions

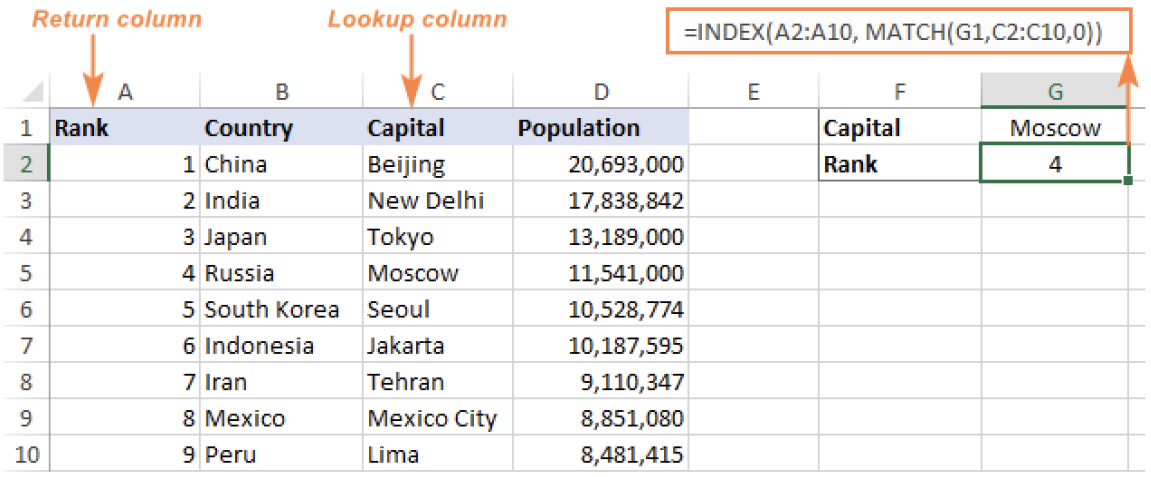
‘‘‘excel

=VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

=INDEX(return_range, MATCH(lookup_value, lookup_range, 0))))

=XLOOKUP(lookup_value, lookup_array, return_array)

Figure 2. Index match function in Excel



	A	B	C	D	E	F	G
1	**Rank**	**Country**	**Capital**	**Population**		**Capital**	Moscow
2	1	China	Beijing	20,693,000		**Rank**	4
3	2	India	New Delhi	17,838,842			
4	3	Japan	Tokyo	13,189,000			
5	4	Russia	Moscow	11,541,000			
6	5	South Korea	Seoul	10,528,774			
7	6	Indonesia	Jakarta	10,187,595			
8	7	Iran	Tehran	9,110,347			
9	8	Mexico	Mexico City	8,851,080			
10	9	Peru	Lima	8,481,415			

Source: [Untitled image about index match function in Excel], n. d.

The image above shows the index match function. INDEX and MATCH is a powerful Excel formula combination that allows flexible lookups across rows and columns. It works by using MATCH to find the position of a value and then INDEX to return the corresponding value from another column or row.

Conditional functions

Excel's IF() is a conditional function that returns different values based on a logical test. `SUMIF() and COUNTIF() are potent functions that perform calculations or count within a range based on specific criteria, making data analysis more efficient.

```excel
=IF(logical_test, value_if_true, value_if_false)

=SUMIF(range, criteria, [sum_range])

=COUNTIF(range, criteria)
```

3. PivotTables

PivotTables are Excel's most powerful feature for data analysis, allowing users to:

- Summarize large datasets.
- Create cross-tabulations.

- Generate dynamic reports.

Creating a basic PivotTable:

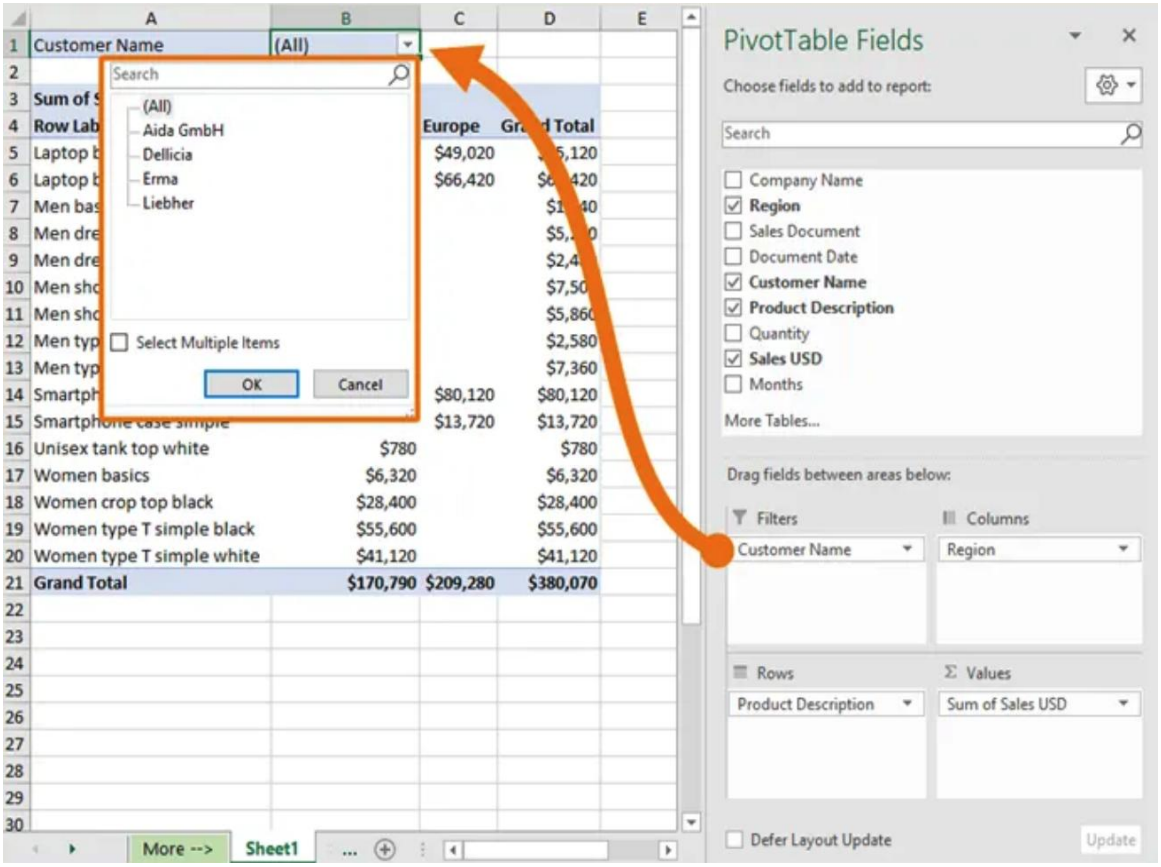
1. Select your date range.

2. Insert → PivotTable.

3. Drag fields to four areas:

    - Filters
    - Columns
    - Rows
    - Values

Figure 3. Pivot Table



Source: Gharani, 2021, https://lc.cx/lwN-aS

A pivot table can help you answer key questions quickly by building rows and columns of your desired metrics and highlighting simple calculations.

Advanced PivotTable features:

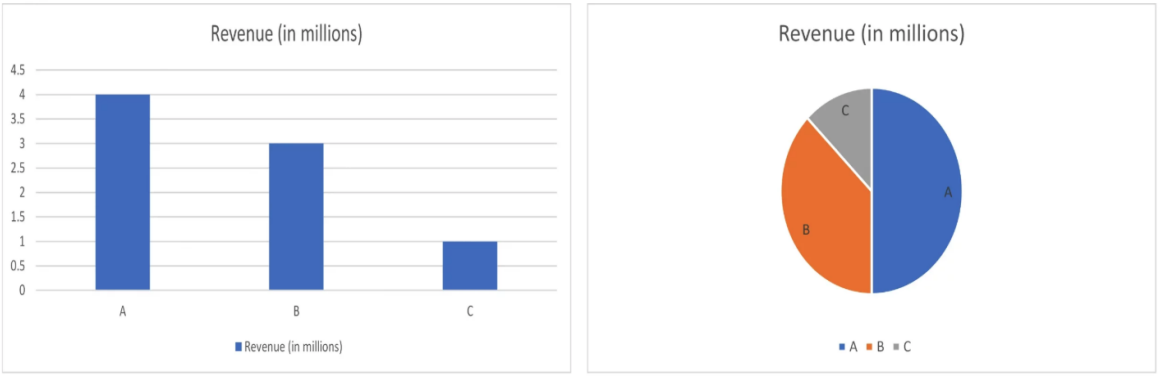
- Calculated fields
- Slicers
- Timeline filters
- Value field settings
- Grouping options

4. Excel charts and visualizations

Basic charts

1. Column charts
   a) Used for comparing categories.
   b) Variations: clustered, stacked, 100 % stacked.
2. Line charts
   a) Ideal for showing trends over time.
   b) Multiple series for comparison.
3. Pie charts
   a) Best for showing composition.
   b) Limited to single data series.

Figure 4. Basic charts



Source: own elaboration.

The image above shows a bar and a pie chart using the same data showing how a bar chart and pie chart can show the same values but in different formats and styles, addressing the power of visualization.

2.2.3 Practical examples

Example 1: sales analysis dashboard

```excel
1. Import sales data
2. Create PivotTable
3. Add calculated fields:
   = [Revenue] - [Cost]
4. Create visualizations:
   - monthly trend chart
   - product mix pie chart
   - regional performance map
```

Example 2: financial modeling

```excel
```

1. Set up assumptions

2. Create projections:

   =FV(rate, nper, pmt, [pv], [type])

3. Sensitivity analysis using Data Table

4. Visualization of scenarios

```

# Unit 2.3 Introduction to Tableau

Tableau is a leading visualization tool that enables users to create interactive and shareable dashboards. Tableau is a powerful visualization platform that empowers users to transform complex data into compelling, interactive dashboards through rich features and capabilities. At the core of Tableau's functionality is its sophisticated data connection methodology, which offers two primary modes of interaction: live connection and data extraction. Live connections enable real-time data updates by executing direct database queries, ensuring that users always have access to the most current information. Conversely, the data extract mode provides offline analysis capabilities, dramatically improving performance through localized data storage and supporting scheduled refreshes that keep information current without constant live querying.

Beyond fundamental charting, Tableau distinguishes itself through advanced analytical features that empower users to perform complex calculations and manipulations. The platform's calculation engine supports intricate computational logic, such as year-over-year growth calculations that dynamically compare performance across temporal dimensions. Parameters introduce another layer of analytical flexibility, allowing users to create dynamic reference lines, implement user-controlled filters, and define flexible calculation inputs that adapt to changing analytical requirements.

Tableau's set and grouping capabilities represent another dimension of its analytical power. Dynamic sets enable users to create fluid data collections that can be modified in real-time, while hierarchical grouping allows for sophisticated data organization strategies. The ability to combine sets further extends analytical possibilities, enabling complex segmentation and filtering approaches that go beyond traditional data manipulation techniques.

Practical implementation of these features comes to life through comprehensive dashboards that demonstrate Tableau's transformative potential. A sales performance dashboard might integrate regional mapping, time series analysis, and product
```

performance metrics, creating a holistic view of organizational performance. Similarly, a customer analysis dashboard could leverage segmentation techniques, loyalty tracking, and interactive filtering to provide actionable insights into customer behavior and trends.

The platform's strength lies in its features and ability to create interconnected, interactive analytical experiences. Tableau transcends traditional reporting tools by enabling users to drill down into data, apply dynamic filters, and rapidly explore complex datasets. It becomes an interactive storytelling platform where data transforms from static numbers into dynamic, compelling narratives that drive strategic decision-making across industries and organizational scales.

2.3.1 Key Tableau features

1. **Data connection**

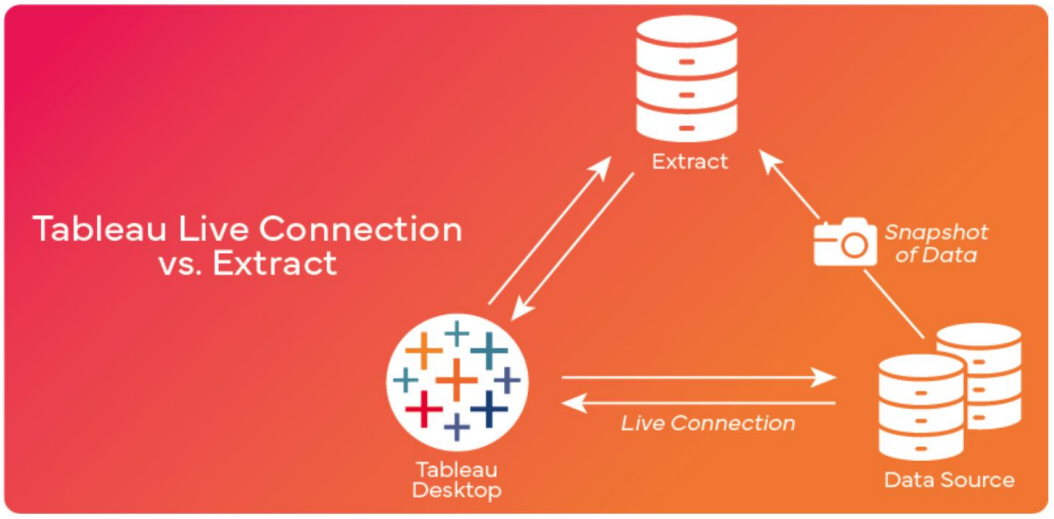
   1.1 Live connection

   - Real-time data updates
   - Direct database queries

   1.2 Data extract

   - Offline analysis
   - Improved performance
   - Scheduled refreshes

Figure 5. Tableau live connection vs. extract

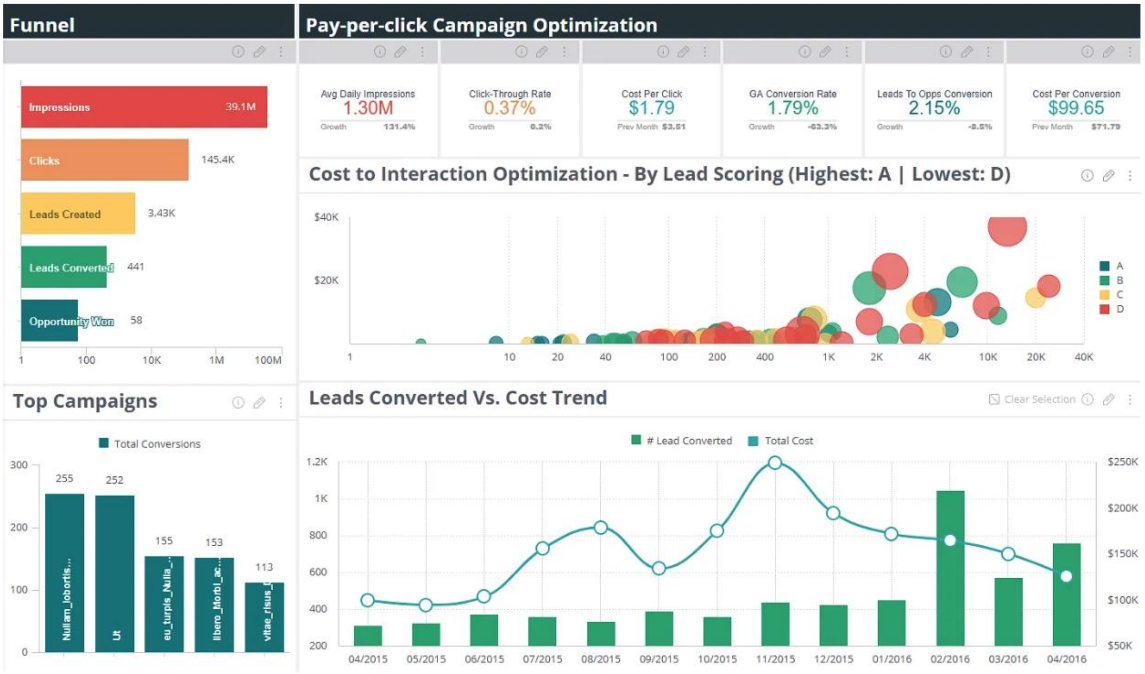


Source: Team Zuar, 2023, https://lc.cx/vGOpmD

Live connections provide real-time access to your data source, ensuring that any changes made to the data are reflected immediately in your visualizations. On the other hand, extracts involve importing and storing a snapshot of the data locally in Tableau, offering enhanced performance and customization capabilities.

2. Basic visualizations
    1. Bar charts
        i. Discrete vs. continuous
        ii. Stacked and grouped options
    2. Line charts
        i. Multiple measures
        ii. Dual-axis options
    3. Scatter plots
    4. Correlation analysis
    5. Size and color encoding

Figure 6. Advanced tableau dashboard



Source: ProjectPro, 2024, https://lc.cx/axk7Ln

Image above represents an advanced tableau dashboard showcasing the trends, highlighting the pay-per-click campaign by showing versatile charts and answering questions about the campaign.

3. Advanced features

Advanced calculations in Tableau are powerful techniques that allow you to create complex, custom calculations beyond simple aggregations. These include the following.

1. Level of detail (LOD) expressions: calculations that can compute values at different granularities, like `{FIXED},` `{INCLUDE},` and `{EXCLUDE},` which let you perform calculations independent of the view's current level of detail.

2. Table calculations: dynamic calculations run after the data has been aggregated, such as running totals, percent of total, rankings, and period-over-period comparisons.

3. Nested calculations: combining multiple functions and logical statements to create intricate computations, like the year-over-year growth example you showed.

4. Calculated fields: custom fields that can transform data, apply complex logic, perform mathematical operations, or create new dimensions and measures using Tableau's calculated field editor.

5. Window calculations: functions that compute values relative to other rows in the current view, enabling analyses like moving averages, running sums, and rankings within a specific data window.

These advanced calculations allow data analysts to perform sophisticated analysis, create custom metrics, and derive deeper insights from their data beyond standard aggregations.

- Calculations

```tableau
// IF
(int [List_2018]) = int('1')
        and (STR(DATEPART('year'),[Session]) = '2018')
    THEN 1
ELSEIF
(int [List_2018]) = int('1')
        and (STR(DATEPART('year'),[Session]) = '2018')
    THEN 1
```

Figure 7. Advanced calculations in Tableau

Source: [Untitled image about advanced calculations in Tableau], n. d.

Previous image highlights the calculation for the best 11 in football and how they can be selected and highlighted in a particular data set.

- Parameters
    - Dynamic reference lines
    - User-controlled filters
    - Calculation inputs
- Sets and groups
    - Dynamic sets
    - Hierarchical grouping
    - Combined sets

2.3.2 Practical examples

Example 1: sales performance dashboard

```tableau
```

1. Connect to sales data.

2. Create worksheets:

   - regional map

   - time series analysis

   - product performance

3. Combine in dashboard.

4. Add interactive filters.

```

**Example 2: customer analysis**

```tableau

1. Customer segmentation.

2. Loyalty analysis.

3. Interactive filtering.

4. Drill-down capabilities.

```

Unit 2.4 Introduction to Python

Python's data analysis capabilities extend far beyond basic statistical computations. When working with large datasets, Python's ecosystem of libraries provides a robust framework for sophisticated analysis. The pandas library is the cornerstone of data manipulation, offering a robust DataFrame structure that makes complex operations intuitive and efficient. Data scientists and analysts can easily leverage pandas' capabilities to handle missing values, reshape data structures, and perform advanced grouping operations.

Time series analysis in Python deserves special attention due to its prevalence in real-world applications. Combining pandas' datetime functionality with specialized visualization libraries enables analysts to uncover temporal patterns and seasonal trends. Whether you're analyzing financial data, sales trends, or user behavior, Python's time
```

series tools provide the flexibility to handle various date formats, resample data at different frequencies, and calculate rolling statistics.

Machine learning integration represents another significant advantage of Python's analytical ecosystem. The scikit-learn library provides a consistent interface for implementing machine learning algorithms, from simple linear regression to complex ensemble methods. This integration allows analysts to seamlessly transition from data preparation to predictive modeling, all within the same programming environment. The library's preprocessing modules facilitate feature scaling, encoding categorical variables, and handling missing values, making preparing data for machine learning applications easier.

2.4.1 Python offers a robust ecosystem for data analysis through various libraries

- Pandas for data manipulation.
- NumPy for numerical operations.
- Matplotlib and Seaborn for visualization.
- Scikit-learn for machine learning.

Data manipulation with Pandas

Basic operations

```python

import pandas as pd

import numpy as np

Reading data

- df = pd.read_csv('sales_data.csv')

Basic operations

- df.head()

- df.describe()

- df.info()
```

Figure 8. Data manipulation with Pandas

```
daily_sleep_duration.reset_index().groupby("user")['duration'].describe()
```

user	count	mean	std	min	25%	50%	75%	max
100059	25.0	6.537000	2.438650	1.225000	5.150000	6.325000	8.233333	11.191667
100063	20.0	7.410306	4.253965	0.308333	4.847917	6.379167	9.754444	17.981944
100107	46.0	6.430072	2.109697	0.250000	6.047917	6.929167	7.654167	9.341667
100117	11.0	13.114394	6.203190	2.783333	8.641667	11.783333	18.204167	22.225000
100121	8.0	8.065625	5.189220	0.025000	6.272917	7.041667	8.995833	17.075000
100126	23.0	16.267705	8.199772	0.091667	11.528750	14.490556	19.786528	40.041389

Source: [Untitled image about data manipulation with Pandas], n. d.

Image above shows the descriptive statistics of the given data set, highlighting some key metrics.

Data cleaning

df.dropna()

df.fill(method='ffill')

‘‘‘

Advanced Pandas operations

‘‘‘python

Group by operations

sales_by_region = df.groupby('region')['sales'].sum()

Pivot tables

pivot_table = pd.pivot_table(df,

values='sales',

index='region',

columns='product',

aggfunc='sum')

Time series analysis

df['date'] = pd.to_datetime(df['date'])

monthly_sales = df.resample('M', on='date')['sales'].sum()

```

**Data visualization**

Matplotlib

```python

import matplotlib.pyplot as plt

# Basic line plot

plt.figure(figsize=(10, 6))

plt.plot(monthly_sales)

plt.title('Monthly Sales Trend')

plt.xlabel('Date')

plt.ylabel('Sales')

plt.grid(True)

plt.show()

Multiple subplots

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

ax1.bar(sales_by_region.index, sales_by_region.values)

ax2.pie(sales_by_region.values, labels=sales_by_region.index)

```
```

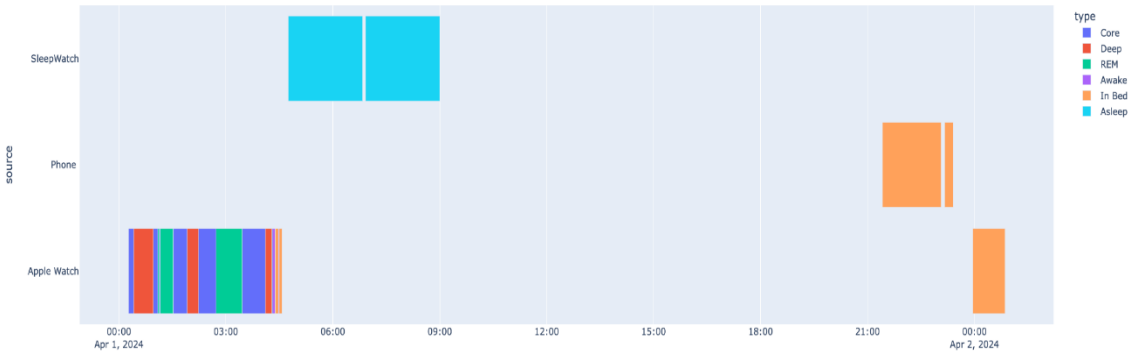
Figure 9. Advanced plot of the types of sleep

```
## Look at top 3
for i in range(3):

    ## Find rows where all to_test conditions are true
    mask_logic = (hk_df[to_test.iloc[i].index] == to_test.iloc[i]).sum(1) == to_test.shape[1]


    ## Gantt chart to see overlapping durations
    fig = px.timeline(hk_df[mask_logic], 'start', 'end', y='source', color='type')

    fig.show()
```



Source: [Untitled image about advanced plot of the types of sleep], n. d.

Image above shows an advanced plot of the types of sleep (deep, rem, core, etc.) at different times during the night.

Statistical analysis

‘‘‘python

from scipy import stats

Hypothesis testing

t_stat, p_value = stats.ttest_ind(group1, group2)

Correlation analysis

correlation = df['sales'].corr(df['marketing_spend'])

‘‘‘

Figure 10. Small script to perform a T-test in Python

```
results = []
alpha = 0.05  # significance level

for metric in test_list:
    # Combined pre and post data
    pre_data = myoton_updated[myoton_updated['time_of_day'] == 'pre'][metric]
    post_data = myoton_updated[myoton_updated['time_of_day'] == 'post'][metric]

    # Paired t-test
    t_stat, p_value = stats.ttest_rel(pre_data, post_data)

    # Print results for each metric
    print(f"{metric}:")
    print(f"  t-statistic = {t_stat:.2f}, p-value = {p_value:.2f}")
    if p_value < alpha:
        print("  Significant difference between pre and post measurements.")
    else:
        print("  No significant difference between pre and post measurements.")
    print()  # Add an extra line for better readability between different metrics

    # Append results to the list
    results.append({
        'Metric': metric,
        't-stat': round(t_stat, 2),
        'p-value': round(p_value, 2),
        'Significant': 'Yes' if p_value < alpha else 'No'
    })

# Convert the results to a DataFrame
results_df = pd.DataFrame(results)

# Display the results as a table
print(results_df)
```

Source: [Untitled image about small script to perform a T-test in Python], n. d.

Previous image highlights how to write a small script to perform a T-test in Python using jupyter notebook.

Machine Learning

“‘python

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

Prepare data

X = df[['marketing_spend', 'price']]

y = df['sales']

Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

Train model

```
model = LinearRegression()

model.fit(X_train, y_train)
```

Make predictions

```
predictions = model.predict(X_test)
```

‘‘‘

Practical examples

Example 1: in this example, we use machine learning to predict the result of a football/soccer match based on shot statistics from the game.

The input data is the following:

- Total shots for each team.
- xG (expected goals) for each shot.

We use the RandomForestClassifier as the model, which works well with this data type and captures the non-linear relationships well.

The model provides probabilities for each possible outcome (win by team A, win by team B, and draw).

Figure 11. Random forest classification

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import joblib
```

Source: [Untitled image about random forest classification], n. d.

The initial steps are to import all the packages needed to build the model. We wanted to use a Random Forest Classifier, so we imported that. The other packages from metrics, etc., are to be used to create specific outputs and visualizations.

Next, we build a soccer match predictor class.

Figure 12. Soccer match predictor class

```
class SoccerMatchPredictor:
 def __init__(self):
 self.model = RandomForestClassifier(n_estimators=100, random_state=42)
 self.scaler = StandardScaler()
```

Source: [Untitled image about], n. d.

The n_estimators is the number of decision trees created in the forest.

random_state = is the seed value used to ensure the reproducibility of the results. You can set this to any number. Machine learning algorithms use randomness in various ways. Setting random_state to any fixed number means you will get the same results every time you run your code. If you don't set random_state, you will get different results each time you train the model.

For example, if you run this code:

 # With random_state

model1 = RandomForestClassifier(random_state=42)

model1.fit(X, y)

predictions1 = model1.predict(X_test)

 # Same code again

model2 = RandomForestClassifier(random_state=42)

model2.fit(X, y)

predictions2 = model2.predict(X_test)

 # predictions1 and predictions2 will be identical

 # Without random_state

model3 = RandomForestClassifier()

model3.fit(X, y)

predictions3 = model3.predict(X_test)

 # Running again

model4 = RandomForestClassifier()

model4.fit(X, y)

predictions4 = model4.predict(X_test)

 # predictions3 and predictions4 might be different

In practice, you would usually:

1. keep random_state fixed during development to ensure reproducible results,
2. experiment with different n_estimators values to find the sweet spot between model performance and training time for your specific problem.

Next, we prepare the features for the model. This is very standard.

Figure 13. Prepare the features for the model

```
def prepare_features(self, df):
"""
Prepare features from raw shot data.

- team_a_shots: List of shot counts for team A
- team_a_xg: List of xG values for team A shots
- team_b_shots: List of shot counts for team B
- team_b_xg: List of xG values for team B shots
- result: 'win', 'draw', or 'loss' from team A's perspective
"""
features = []

for _, row in df.iterrows():
# Calculate statistics for team A
team_a_features = [
len(row['team_a_shots']), # Total shots
sum(row['team_a_xg']), # Total xG
np.mean(row['team_a_xg']), # Average xG per shot
np.max(row['team_a_xg']), # Best chance xG
np.std(row['team_a_xg']) # xG standard deviation
]

# Calculate statistics for team B
team_b_features = [
len(row['team_b_shots']), # Total shots
sum(row['team_b_xg']), # Total xG
np.mean(row['team_b_xg']), # Average xG per shot
np.max(row['team_b_xg']), # Best chance xG
np.std(row['team_b_xg']) # xG standard deviation
]

# Combine features
match_features = team_a_features + team_b_features
features.append(match_features)

return np.array(features)
```

Source: [Untitled image about prepare the features for the model], n. d.

As we are dealing with string outcomes, “win,” “loss,” and “draw,” we convert them into numerical labels.

Figure 14. Convert string outcomes into numerical labels

```
def prepare_labels(self, df):
"""Convert result strings to numeric labels"""
label_map = {'win': 0, 'draw': 1, 'loss': 2}
return df['result'].map(label_map)
```

Source: [Untitled image about convert string outcomes into numerical labels], n. d.

Next, we train the model.

Figure 15. Train the model

```
def train(self, training_data):
"""
Train the model using historical match data
"""
# Prepare features and labels
X = self.prepare_features(training_data)
y = self.prepare_labels(training_data)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42
)

# Scale the features
X_train_scaled = self.scaler.fit_transform(X_train)
X_test_scaled = self.scaler.transform(X_test)

# Train the model
self.model.fit(X_train_scaled, y_train)

# Evaluate the model
y_pred = self.model.predict(X_test_scaled)
print("\nModel Performance:")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=['Win', 'Draw', 'Loss']))

return self
```

Source: [Untitled image about train the model], n. d.

Next, we run the model.

Figure 16. Run the model

```
def predict_match(self, team_a_shots, team_a_xg, team_b_shots, team_b_xg):
"""
Predict the outcome of a single match
"""
# Create a single-row DataFrame with the same structure as training data
match_data = pd.DataFrame({
'team_a_shots': [team_a_shots],
'team_a_xg': [team_a_xg],
'team_b_shots': [team_b_shots],
'team_b_xg': [team_b_xg],
'result': ['win'] # Dummy value, not used for prediction
})

# Prepare features
X = self.prepare_features(match_data)
X_scaled = self.scaler.transform(X)

# Make prediction
prediction = self.model.predict(X_scaled)[0]
probabilities = self.model.predict_proba(X_scaled)[0]

# Map numeric prediction back to result
result_map = {0: 'win', 1: 'draw', 2: 'loss'}
predicted_result = result_map[prediction]

return {
'prediction': predicted_result,
'probabilities': {
'win': probabilities[0],
'draw': probabilities[1],
'loss': probabilities[2]
}
}
```

Source: [Untitled image about], n. d.

How to use this model?

You can use it in multiple ways. Either create some dummy data to test it or if you have historical data, read from a file to run the model.

Figure 17. Use the model in multiple ways

```
if __name__ == "__main__":
 # Create sample training data
 sample_data = pd.DataFrame({
 'team_a_shots': [[1, 2, 3], [2, 3, 4], [1, 1, 2]],
 'team_a_xg': [[0.1, 0.2, 0.3], [0.2, 0.3, 0.4], [0.1, 0.1, 0.2]],
 'team_b_shots': [[2, 2, 1], [1, 2, 1], [3, 3, 2]],
 'team_b_xg': [[0.2, 0.2, 0.1], [0.1, 0.2, 0.1], [0.3, 0.3, 0.2]],
 'result': ['win', 'draw', 'loss']
 })

 # Train the model
 predictor = SoccerMatchPredictor()
 predictor.train(sample_data)

 # Make a prediction for a new match
 new_match_prediction = predictor.predict_match(
 team_a_shots=[1, 2, 3],
 team_a_xg=[0.1, 0.2, 0.3],
 team_b_shots=[2, 2, 1],
 team_b_xg=[0.2, 0.2, 0.1]
 )

 print("\nNew Match Prediction:")
 print(f"Predicted outcome: {new_match_prediction['prediction']}")
 print("Probabilities:")
 for outcome, prob in new_match_prediction['probabilities'].items():
 print(f"{outcome}: {prob:.2%}")
```

Source: [Untitled image about use the model in multiple ways], n. d.

References

Gharani, L. (2021). Excel Advanced Pivot Table Techniques for Serious Data Analysts. *Xelplus*. https://www.xelplus.com/excel-advanced-pivot-tables/

ProjectPro. (2024). 15 Tableau Projects for Beginners to Practice with Source Code. https://www.projectpro.io/article/-tableau-projects-ideas/479

Team Zuar. (2023). Beginner's Guide to Tableau Data Sources. *Zuar*. https://www.zuar.com/blog/guide-to-tableau-data-sources/