Module 2. Machine learning. Part 2

In this module, we will continue with topics related to machine learning. We will discuss reinforcement learning in detail. We will also learn about Principal Component Analysis (PCA) within unsupervised learning. Finally, we will have a detailed example of how to do advanced data analysis on Tableau using its advanced features.

Unit 2.1 Unsupervised learning: dimensionality reduction

Dimensionality reduction is another approach to unsupervised learning. Reduces data complexity while preserving essential patterns. Principal Component Analysis (PCA) is one of the most widely applied dimensionality reduction techniques.

2.1.1 Principal Component Analysis

Principal Component Analysis is a fundamental dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional form while preserving as much variance as possible. PCA is a fundamental unsupervised learning technique for several reasons.

- No labels required. It learns patterns and structure purely from the features.
- Pattern discovery. Reveals underlying patterns and relationships in data.
- Helps visualize high-dimensional data in 2D or 3D, hugely useful for data exploration.

Core concepts

- **Dimensionality reduction**
    - PCA reduces the number of features in a dataset by creating new, uncorrelated variables called principal components.
    - Each principal component is a linear combination of the original features.
    - The components are ordered by the amount of variance they explain in the data.

- **Math behind PCA**
    - PCA begins by standardizing the data (mean centering and scaling).

- It computes the covariance matrix of the standardized features.
- Then, find the eigenvectors and eigenvalues of this covariance matrix.
- The eigenvectors become the direction of the principal components.
- The eigenvalues represent the amount of variance explained by each component.

- **Important notes**
  - Principal components are orthogonal (perpendicular) to each other.
  - They are ordered by decreasing variance.
  - The first principal component captures the maximum possible variance.
  - Each subsequent component captures the maximum remaining variance while being orthogonal to previous components.

- **Limitations**
  - PCA assumes linear relationships between features and may not capture nonlinear patterns effectively.
  - The sensitivity of results depends on feature scaling.
  - Interpretability: principal components can be complex to interpret as they are combinations of original features.
  - PCA assumes variance captures important structure, which may not always be true for all datasets.

Applications PCA

- Reduce dimensionality before applying other ML algorithms.
- Plot high-dimensional data in 2D or 3D to help understand the relationship between features.
- Reduce noise. Lower-order principal components often capture noise.
- Compression. Reduces data storage requirements, speeding up subsequent computations

Applications in sport

Here is a sample of sports questions and problems that PCA can help solve.

- Which player statistics are most important for differentiating players?
- Can we reduce multiple performance metrics into a few key components?
- How do different playing styles cluster together?
- Which combinations of stats best explain player performance variation?

Example

This NBA basketball player analysis example demonstrates the key applications of PCA in sports analytics.

1. **Import the requisite packages**

Figure 1. Import the requisite packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Source: own elaboration.

2. Generate sample synthetic data. This step can be replaced by reading data from a file or database.

    - We generate data for 200 players.
    - Metrics used: points, rebounds, assists, steals, blocks, and the label archetype.
    - We won't use the label in our model. It is just for data generation and model evaluation at the end.

Figure 2. Generate sample synthetic data

```
# Set random seed for reproducibility
np.random.seed(42)
# Generate synthetic player data
n_players = 200

# Create player archetypes to make data more realistic
scorers = pd.DataFrame({
    'points': np.random.normal(28, 3, n_players//4),
    'rebounds': np.random.normal(5, 1, n_players//4),
    'assists': np.random.normal(4, 1, n_players//4),
    'steals': np.random.normal(1, 0.3, n_players//4),
    'blocks': np.random.normal(0.5, 0.2, n_players//4),
    'archetype': 'Scorer'
})

playmakers = pd.DataFrame({
    'points': np.random.normal(18, 3, n_players//4),
    'rebounds': np.random.normal(4, 1, n_players//4),
    'assists': np.random.normal(9, 1, n_players//4),
    'steals': np.random.normal(1.5, 0.3, n_players//4),
    'blocks': np.random.normal(0.3, 0.2, n_players//4),
    'archetype': 'Playmaker'
})

defenders = pd.DataFrame({
    'points': np.random.normal(12, 3, n_players//4),
    'rebounds': np.random.normal(7, 1, n_players//4),
    'assists': np.random.normal(2, 1, n_players//4),
    'steals': np.random.normal(2, 0.3, n_players//4),
    'blocks': np.random.normal(1.5, 0.3, n_players//4),
    'archetype': 'Defender'
})

all_around = pd.DataFrame({
    'points': np.random.normal(22, 3, n_players//4),
    'rebounds': np.random.normal(7, 1, n_players//4),
    'assists': np.random.normal(6, 1, n_players//4),
    'steals': np.random.normal(1.5, 0.3, n_players//4),
    'blocks': np.random.normal(1, 0.3, n_players//4),
    'archetype': 'All-Around'
})
```

Source: own elaboration.

3. Combine all the players, scale the data, and run PCA

Figure 3. Combine all and run PCA

```
# Combine all player types
df = pd.concat([scorers, playmakers, defenders, all_around], ignore_index=True)

# Add some noise and correlations to make data more realistic
df['points'] += 0.3 * df['assists']  # Players with more assists tend to score more
df['steals'] += 0.2 * df['blocks']   # Defensive stats are correlated

# Prepare data for PCA
features = ['points', 'rebounds', 'assists', 'steals', 'blocks']
X = df[features]

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

Source: own elaboration.

4. Function to identify player archetypes.

Figure 4. Function to identify player archetypes

```
# Function to identify player archetype based on PCA coordinates
def identify_archetype(stats):
    # Scale the input stats
    scaled_stats = scaler.transform([stats])
    # Transform to PCA space
    pca_coords = pca.transform(scaled_stats)[0]

    # Simple classification based on location in PC space
    pc1, pc2 = pca_coords[0], pca_coords[1]

    if pc1 > 1 and pc2 > 0:
        return "Scorer"
    elif pc1 < 0 and pc2 > 1:
        return "Playmaker"
    elif pc1 < -1 and pc2 < 0:
        return "Defender"
    elif abs(pc1) < 1 and abs(pc2) < 1:
        return "All-Around"
    else:
        return "Hybrid"
```

Source: own elaboration.

5. Print feature correlations.

Figure 5. Print feature correlations

```
# Calculate and print feature correlations
print("\nFeature Correlations:")
correlation_matrix = df[features].corr()
print(correlation_matrix.round(3))
```

Source: own elaboration.

6. Plot the PCA output.

Figure 6. Plot the PCA output

```
# Create figure with subplots

fig = plt.figure(figsize=(20, 15))

# 1. Scree Plot
plt.subplot(2, 2, 1)
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
plt.plot(range(1, len(explained_variance) + 1), explained_variance, 'bo-', label='Individual')
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, 'ro-', label='Cumulative')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot: Explained Variance by Principal Component')
plt.grid(True)
plt.legend()

# 2. Biplot
plt.subplot(2, 2, 2)
# Plot PC scores
colors = {'Scorer': 'red', 'Playmaker': 'blue', 'Defender': 'green', 'All-Around': 'purple'}
for archetype in df['archetype'].unique():
    mask = df['archetype'] == archetype
    plt.scatter(X_pca[mask, 0], X_pca[mask, 1], c=colors[archetype], label=archetype, alpha=0.6)

# Plot feature vectors
for i, feature in enumerate(features):
    plt.arrow(0, 0, pca.components_[0, i]*3, pca.components_[1, i]*3,
              color='black', alpha=0.5)
    plt.text(pca.components_[0, i]*3.2, pca.components_[1, i]*3.2, feature)

plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('Biplot: PCA Components with Original Features')
plt.legend()
plt.grid(True)
```

```
# 3. Component Loadings Heatmap
plt.subplot(2, 2, 3)
loadings = pd.DataFrame(
    pca.components_.T,
    columns=[f'PC{i+1}' for i in range(len(features))],
    index=features
)
sns.heatmap(loadings, annot=True, cmap='RdBu', center=0, vmin=-1, vmax=1)
plt.title('PCA Component Loadings Heatmap')

# 4. Player Distribution Density Plot
plt.subplot(2, 2, 4)
for archetype in df['archetype'].unique():
    mask = df['archetype'] == archetype
    sns.kdeplot(data=pd.DataFrame({
        'PC1': X_pca[mask, 0],
        'PC2': X_pca[mask, 1]
    }), x='PC1', y='PC2', levels=5, color=colors[archetype], label=archetype)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('Player Distribution Density by Archetype')
plt.legend()

plt.tight_layout()
plt.show()

# Print explained variance and component loadings
print("\nExplained Variance Ratio by Component:")
for i, var in enumerate(explained_variance):
    print(f"PC{i+1}: {var:.3f} ({cumulative_variance[i]:.3f} cumulative)")

print("\nComponent Loadings:")
print(loadings)
```

Source: own elaboration.

7. Output and interpretation

Feature correlations

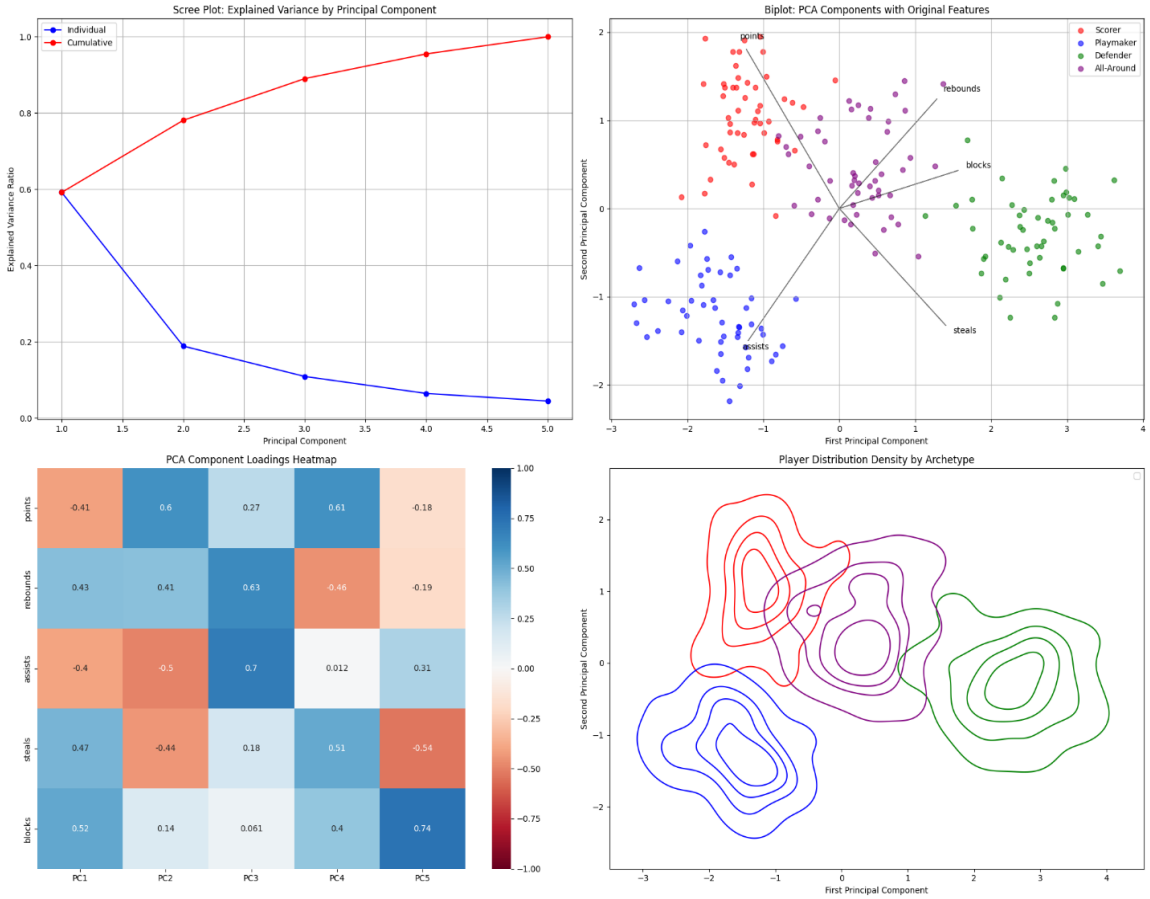
Correlations between the different features used in the analysis show how various skills correlate.

Figure 7. Feature correlations

```
Feature Correlations:
           points  rebounds  assists  steals  blocks
points      1.000    -0.272    0.290  -0.666  -0.490
rebounds   -0.272     1.000   -0.475   0.430   0.648
assists     0.290    -0.475    1.000  -0.310  -0.608
steals     -0.666     0.430   -0.310   1.000   0.645
blocks     -0.490     0.648   -0.608   0.645   1.000
```

Source: own elaboration.

Figure 8. PCA Plots



Source: own elaboration.

Top left: scree plot

Plots the explained variance by component. Blue represents the individual value of each element, and red represents the cumulative explained variance. It shows how much of the total variance each principal component explains. It helps determine how many elements are needed to capture the critical variation.

This example shows that the first two components sufficiently capture the critical variation.

- It is pretty evident that PC1 and PC2 account for almost 80% of the explained variance with PC1 close to 60% and PC2 close to 20%.
- The first three components explain 89% of the variance.
- This suggests we could reduce from 5 dimensions to 2-3 while retaining most information.

Top right: biplot

- This plot shows PCA components with original features.
- Displays both PC scores and feature vectors.

Bottom left: loadings heatmap

- Shows how original features contribute to PCs.

Bottom right: density plot

- Shows distribution of player types in PC space.

Component loadings analysis

PC1 (59.2% variance):

- Strong positive loadings for blocks (0.52), steals (0.47), and rebounds (0.43).
- Negative loadings for points (-0.41) and assists (-0.40).
- This component appears to contrast defensive stats vs. offensive stats.
- This could be interpreted as a "Defensive vs. Offensive Specialist" axis.

PC2 (18.9% variance):

- Strongly favorable loading for points (0.60).
- Strong negative loading for assists (-0.50) and steals (-0.44).
- Moderately favorable loading for rebounds (0.41).
- This seems to separate pure scorers from playmakers.
- This could be interpreted as a "Scorer vs. Facilitator" axis.

Feature correlations

Strong positive correlations:

- Blocks-Rebounds (0.648): players good at rim protection tend to get rebounds.
- Blocks-Steals (0.645): shows overall defensive ability clusters together.

Strong negative correlations:

- Points-steals (-0.666): suggests specialists (either scoring or defense).

- Blocks-assists (-0.608): indicates role separation between playmakers and rim protectors.

Overall insights

- The model reveals natural role separation in basketball.
- Shows trade-off between offensive and defensive specialization,
- Identifies how different skills tend to cluster together.

2.1.2 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent learns optimal behavior through trial and error, receiving rewards or penalties for its actions. Applications include:

- game playing,
- robotics,
- autonomous vehicles,
- resource management.

Key components of the RL framework

1. Agent
    a. The agent is the decision-maker.
    b. It learns from experience and takes actions to maximize rewards.
2. Environment
    a. It is the system the agent interacts with.
    b. The environment responds to agents' actions and provides the state with rewards.
3. State
    a. State is the current situation – it is the available information.
    b. It provides the context for decision-making.
4. Action
    a. Actions are the possible choices and agent's decisions.
    b. Every action affects the environment.
5. Reward
    a. Reward is the feedback signal – immediate or delayed.
    b. It guides the learning process of the agent.

Applications in sports

- Playing calling in NFL.
- Shot selection in basketball.
- Formation and strategy selection in soccer.
- Pitch type selection in baseball.
- Training optimization and workout planning.
- Resource management: substitutes and adjustments.

Example

Let us walk through an example of a reinforcement learning solution for soccer tactical decision-making. The data is simulated; you can plug in actual match data for more realistic outcomes.

1. Include all the requisite packages.

Figure 9. Import all required packages

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from collections import defaultdict
```

Source: own elaboration.

2. Environment design.
    a. State space: score difference, period, team stamina.
    b. Action space: formation, pressing style, attacking intensity, a realistic reward structure based on match outcomes.

Figure 10. Environment design

```
def create_environment():
    """Create the soccer environment configuration"""
    return {
        'score_difference_range': [-2, -1, 0, 1, 2],  # -2 to +2 goals
        'time_periods': [0, 1],                       # First half (0) and Second half (1)
        'stamina_levels': [0, 1, 2],                  # Low, Medium, High
        'actions': [(f, p, a)                         # All possible actions
                    for f in range(3)                 # Formation: defensive, balanced, attacking
                    for p in range(3)                 # Pressing: low, medium, high
                    for a in range(3)]                # Attacking: low, medium, high
    }
```

Source: own elaboration.

3. Initiate state.

Figure 11. Initiate state

```python
def initialize_state():
    """Initialize a new game state"""
    return {
        'score_diff': 0,
        'time_period': 0,    # 0 = First half, 1 = Second half
        'stamina': 2,        # Start with high stamina
        'minutes': 0         # Track actual minutes
    }

def get_state_tuple(state):
    """Convert state dictionary to state tuple"""
    return (state['score_diff'], state['time_period'], state['stamina'])
```

Source: own elaboration.

4. Train agent.

Figure 12. Train agent

```python
def train_agent(env_config, episodes=1000, alpha=0.1, gamma=0.9, epsilon=0.1):
    """Train the agent using Q-learning"""
    Q = defaultdict(lambda: np.zeros(len(env_config['actions'])))
    episode_rewards = []
    win_rates = []
    action_counts = defaultdict(int)

    for episode in range(episodes):
        state = initialize_state()
        total_reward = 0
        done = False

        while not done:
            state_tuple = get_state_tuple(state)

            # Epsilon-greedy action selection
            if np.random.random() < epsilon:
                action_idx = np.random.randint(len(env_config['actions']))
            else:
                action_idx = np.argmax(Q[state_tuple])

            action = env_config['actions'][action_idx]
            action_counts[action] += 1

            next_state, reward, done = take_action(state, action, env_config)
            next_state_tuple = get_state_tuple(next_state)
            total_reward += reward

            # Q-learning update
            best_next_action = np.argmax(Q[next_state_tuple])
            Q[state_tuple][action_idx] = Q[state_tuple][action_idx] + alpha * (
                reward + gamma * Q[next_state_tuple][best_next_action] - Q[state_tuple][action_idx]
            )

            state = next_state

        episode_rewards.append(total_reward)
        win_rates.append(1 if state['score_diff'] > 0 else 0)
        epsilon = max(0.01, epsilon * 0.995)

    return Q, episode_rewards, win_rates, action_counts
```

Source: own elaboration.

5. The agent takes an action and returns a new state, and the reward for the action.

Figure 13. The agent takes an action

```python
def take_action(state, action, env_config):
    """Execute action and return new state, reward, and done flag"""
    new_state = state.copy()
    formation, pressing, attacking = action

    # Calculate energy consumption
    energy_cost = (pressing + attacking) / 4.0
    new_state['stamina'] = max(0, state['stamina'] - energy_cost)

    # Update minutes (each action represents ~5 minutes of play)
    new_state['minutes'] = state['minutes'] + 5

    # Update time period based on minutes
    new_state['time_period'] = 1 if new_state['minutes'] >= 45 else 0

    # Calculate base success probability
    success_prob = 0.5

    # Modify probability based on tactics and time
    if state['score_diff'] < 0:  # Losing
        success_prob += 0.1 if formation == 2 else -0.1
    elif state['score_diff'] > 0:  # Winning
        success_prob += 0.1 if formation == 0 else -0.1

    # Stamina affects success more in second half
    if new_state['time_period'] == 1:
        if new_state['stamina'] == 0:
            success_prob -= 0.3
        elif new_state['stamina'] == 2:
            success_prob += 0.1

    # Determine outcome
    if np.random.random() < success_prob:
        reward = 1
        new_state['score_diff'] += 1
    else:
        reward = -1
        new_state['score_diff'] -= 1

    # Clip score difference
    new_state['score_diff'] = max(-2, min(2, new_state['score_diff']))

    # Check if game is done (90 minutes)
    done = new_state['minutes'] >= 90

    # Final reward based on match outcome
    if done:
        if new_state['score_diff'] > 0:
            reward += 10
        elif new_state['score_diff'] < 0:
            reward -= 10

    return new_state, reward, done
```

Source: own elaboration.

6. Create visualizations.

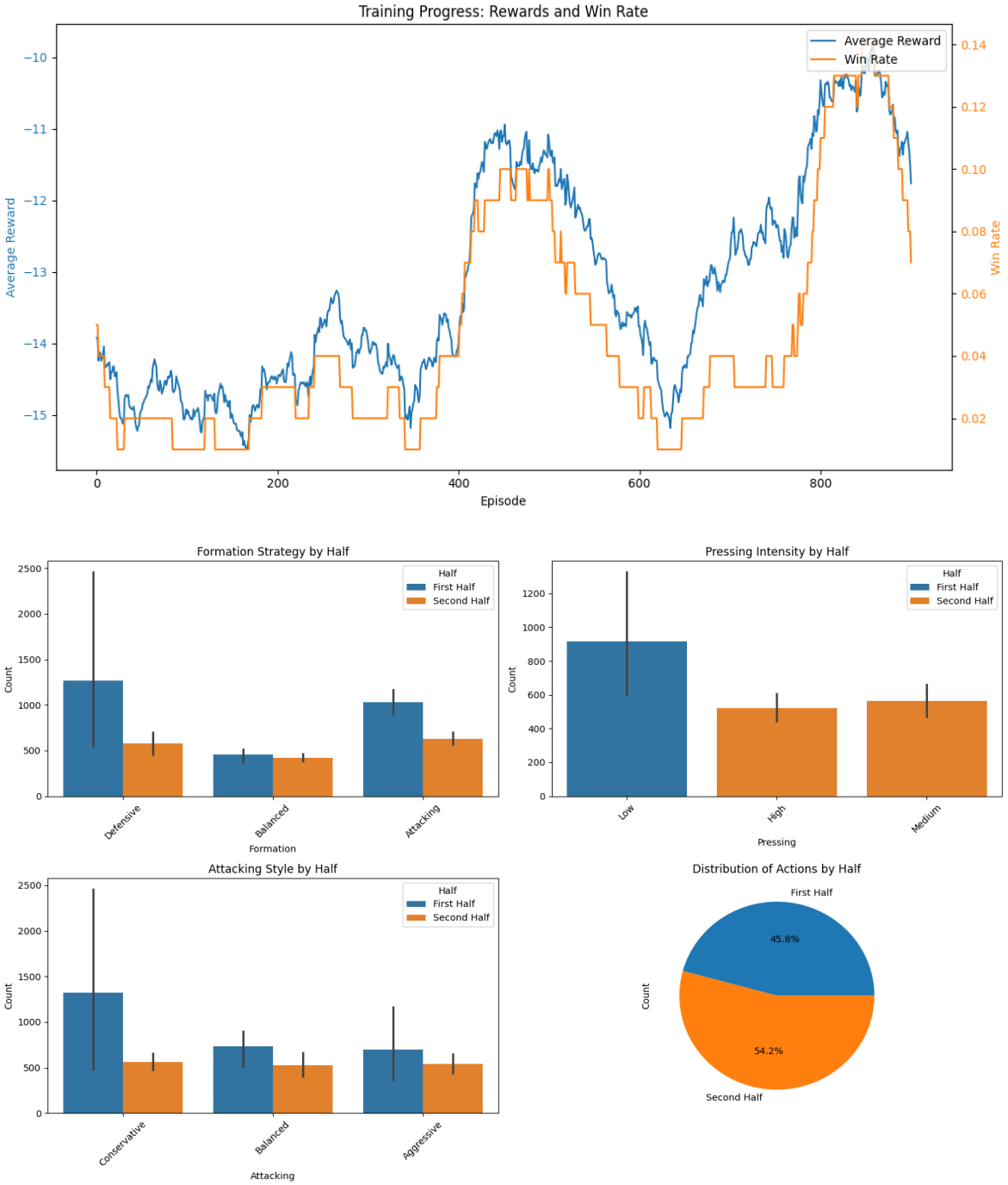
Figure 14. Create visualization

```python
def create_visualizations(episode_rewards, win_rates, action_counts, env_config):
    """Create analysis visualizations with time-based insights"""

    # 1. Combined learning curve and win rate
    fig, ax1 = plt.subplots(figsize=(12, 6))

    # Plot rewards
    color1 = '#1f77b4'  # Blue
    ax1.set_xlabel('Episode')
    ax1.set_ylabel('Average Reward', color=color1)
    rewards_ma = np.convolve(episode_rewards, np.ones(100)/100, mode='valid')
    ax1.plot(rewards_ma, color=color1, label='Average Reward')
    ax1.tick_params(axis='y', labelcolor=color1)

    # Create second y-axis for win rate
    ax2 = ax1.twinx()
    color2 = '#ff7f0e'  # Orange
    ax2.set_ylabel('Win Rate', color=color2)
    win_rate_ma = np.convolve(win_rates, np.ones(100)/100, mode='valid')
    ax2.plot(win_rate_ma, color=color2, label='Win Rate')
    ax2.tick_params(axis='y', labelcolor=color2)

    plt.title('Training Progress: Rewards and Win Rate')
    fig.legend(loc='upper right', bbox_to_anchor=(1,1), bbox_transform=ax1.transAxes)
    plt.tight_layout()
    plt.show()
    # 2. First Half vs Second Half Action Distribution
    action_df = pd.DataFrame(list(action_counts.items()),
                             columns=['Action', 'Count'])
    action_df['Half'] = action_df['Action'].apply(lambda x: 'First Half' if x[1] == 0 else 'Second Half')
    action_df['Formation'] = action_df['Action'].apply(lambda x: ['Defensive', 'Balanced', 'Attacking'][x[0]])
    action_df['Pressing'] = action_df['Action'].apply(lambda x: ['Low', 'Medium', 'High'][x[1]])
    action_df['Attacking'] = action_df['Action'].apply(lambda x: ['Conservative', 'Balanced', 'Aggressive'][x[2]])

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # Formation analysis by half
    sns.barplot(data=action_df, x='Formation', y='Count', hue='Half', ax=axes[0,0])
    axes[0,0].set_title('Formation Strategy by Half')
    axes[0,0].tick_params(axis='x', rotation=45)
    # Pressing analysis by half
    sns.barplot(data=action_df, x='Pressing', y='Count', hue='Half', ax=axes[0,1])
    axes[0,1].set_title('Pressing Intensity by Half')
    axes[0,1].tick_params(axis='x', rotation=45)

    # Attacking analysis by half
    sns.barplot(data=action_df, x='Attacking', y='Count', hue='Half', ax=axes[1,0])
    axes[1,0].set_title('Attacking Style by Half')
    axes[1,0].tick_params(axis='x', rotation=45)

    # Overall strategy distribution
    strategy_counts = action_df.groupby('Half')['Count'].sum()
    strategy_counts.plot(kind='pie', ax=axes[1,1], autopct='%1.1f%%')
    axes[1,1].set_title('Distribution of Actions by Half')

    plt.tight_layout()
    plt.show()
```

Source: own elaboration.

7. Model output.

Figure 15. Model output

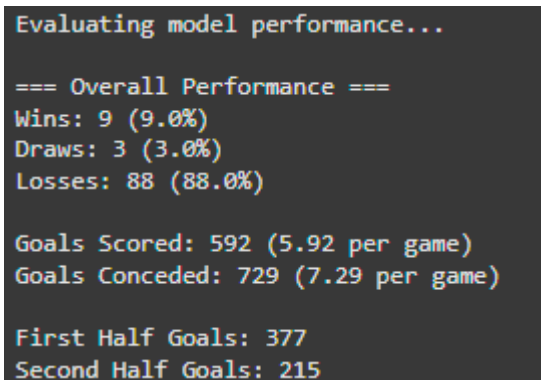


Source: own elaboration.

Analysis and interpretation

The model results indicate that this model requires significant refinement to make it usable.

Figure 16. Evaluating model performance



Source: own elaboration.

1. Abysmal win rate (9%) with a high loss rate (88%).
2. Extremely high-scoring games (13.21 goals per game total).
3. Defensive issues with 7.29 goals conceded per game.
4. More goals in the first half (377) than in the second half (215), suggesting stamina or tactical issues.

Ways to improve the model to make it realistic

1. Adjust reward structure to penalize conceding goals more heavily.
2. Implement better-balanced tactical combinations.

4.2.1 Other ML techniques

4.2.1.1 Neural networks

A neural network is a computing system inspired by biological brains. The fundamental unit is a "neuron" which:

- receives multiple inputs,
- applies weights to those inputs,
- sums them up,
- passes the sum through an "activation function" to produce an output.

A simple example:

- Inputs: $x_1 = 0.5$, $x_2 = 0.8$.
- Weights: $w_1 = 0.4$, $w_2 = 0.6$.

- Sum: $0.5 \times 0.4 + 0.8 \times 0.6 = 0.68$.
- Activation (using ReLU): $\max(0, 0.68) = 0.68$.

These neurons are organized in layers.

- Input layer: receives raw data.
- Hidden layer(s): processes information.
- Output layer: produces results.

Deep learning

Deep learning refers to neural networks with multiple hidden layers (hence "deep"). Each layer learns progressively more complex features.

- The first layers might detect edges in images.
- Middle layers might combine edges into shapes.
- Later layers might recognize complex objects.

2.1.3 Advanced-data analytics in Tableau

Tableau can help teams build and maintain their KPIs and charts. However, to take advantage of Tableau, you must learn and understand its analytics side.

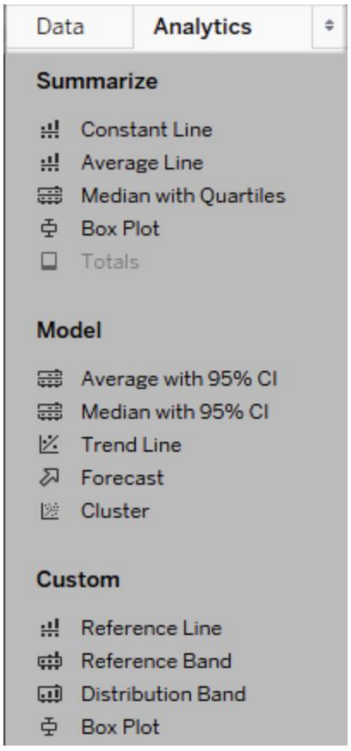
Tableau offers 3 models:

1. Trend line
2. Forecast
3. Clusters

For this course section, we will focus on trendlines and clusters to show how coaches and sports scientists can use these built-in features to their advantage. This will help us understand the trends and explore new questions in the data set.

The Tableau data and analytics section is highlighted below.

Figure 17. Tableau data and analytics section

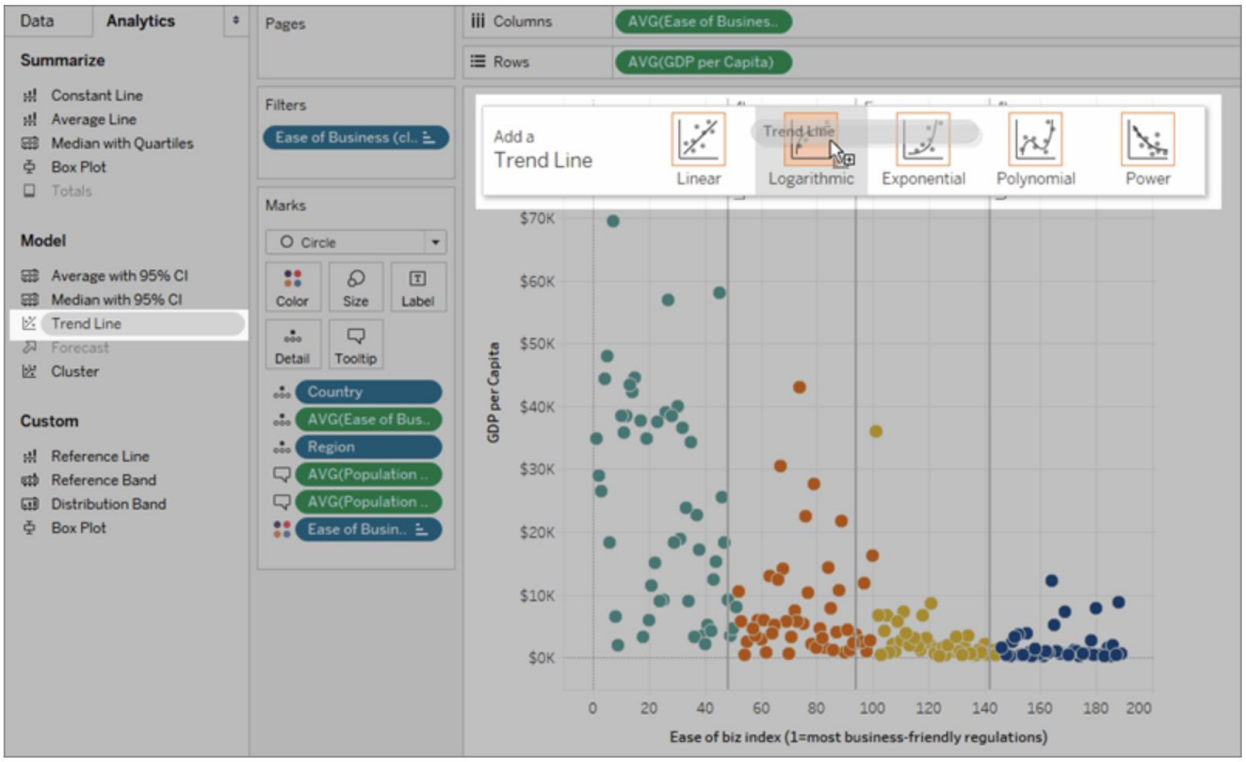


Source: screenshot from Tableau.

Trend lines can be further divided into five types.

1. Linear
2. Logarithmic
3. Exponential
4. Polynomial
5. Power

Figure 18. Five types of trend lines



Source: Tableau, n.d., https://lc.cx/ukbSi0

We discussed them in the previous chapter as tools for analyzing data and adding a layer to the findings.

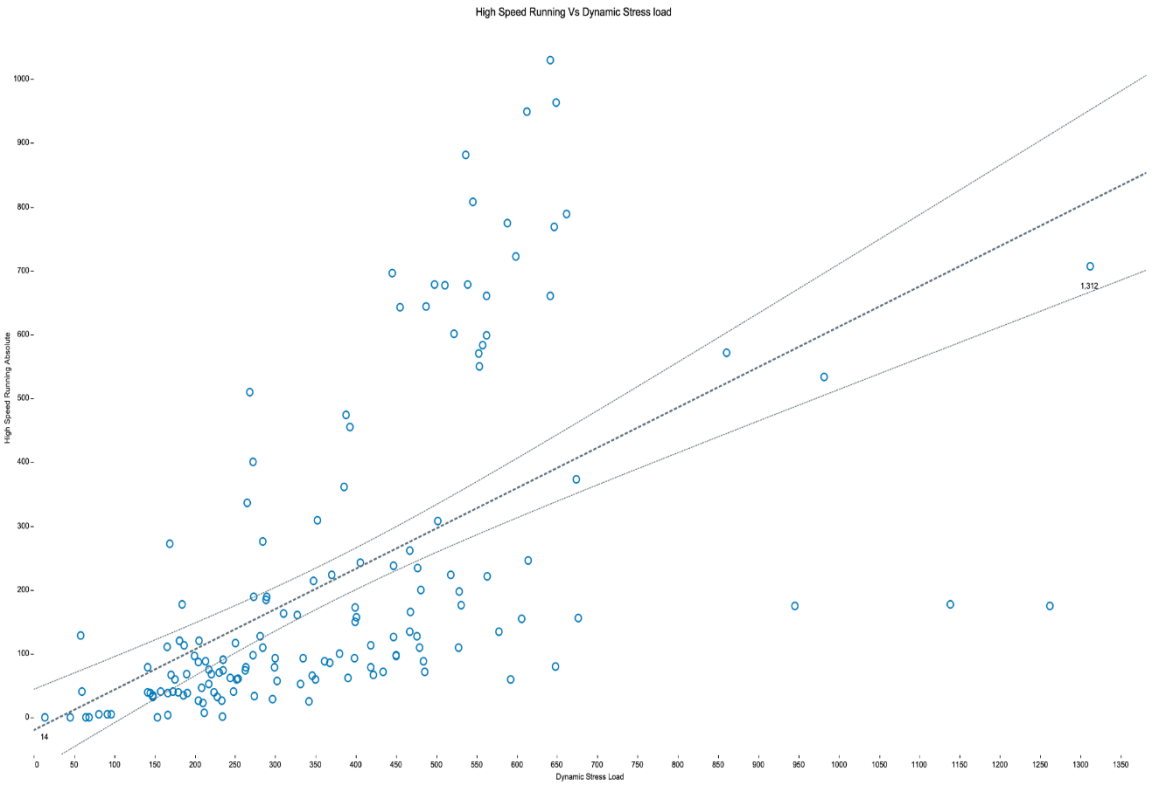
Below is an example to highlight the approach taken by an approach to answer the question using advanced tableau features.

Q: You are a sports scientist and have been asked to show whether covering more distance at high-speed running (> 5.5m/s) causes players more fatigue.

To answer this question, we will use the HSR and Dynamic stress load data collected using GPS (Global positioning device—Statsports) and highlight how a relationship between two metrics can be established using Tableau to identify a trend among athletes.

To begin with, you can build a scatter plot using HSR and Dynamic stress load. We have also added the linear trend line built into Tableau to show how they compare.

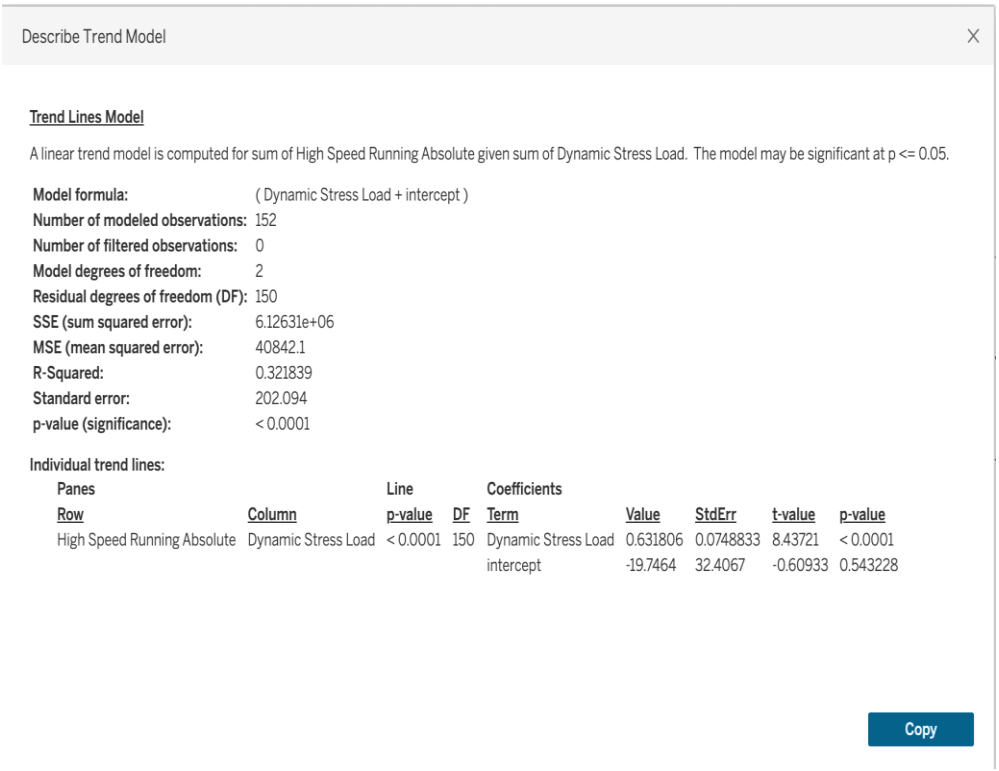
Figure 19. Scatter plot



Source: own elaboration.

To help you understand the trend line, the following figure illustrates how Tableau explains and does it for you.

Figure 20. How Tableau explains and applies the trend line


Describe Trend Model

Trend Lines Model

A linear trend model is computed for sum of High Speed Running Absolute given sum of Dynamic Stress Load. The model may be significant at p <= 0.05.

Model formula:	(Dynamic Stress Load + intercept)
Number of modeled observations:	152
Number of filtered observations:	0
Model degrees of freedom:	2
Residual degrees of freedom (DF):	150
SSE (sum squared error):	6.12631e+06
MSE (mean squared error):	40842.1
R-Squared:	0.321839
Standard error:	202.094
p-value (significance):	< 0.0001

Individual trend lines:

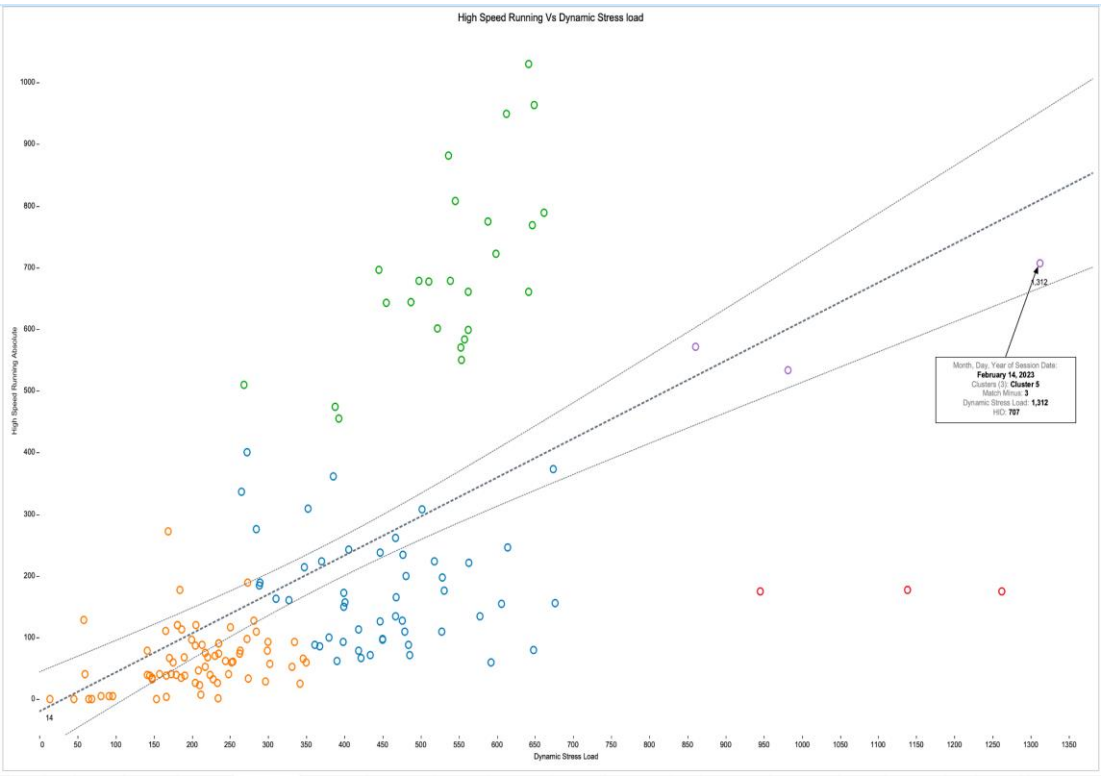
Panes		Line		Coefficients				
Row	Column	p-value	DF	Term	Value	StdErr	t-value	p-value
High Speed Running Absolute	Dynamic Stress Load	< 0.0001	150	Dynamic Stress Load	0.631806	0.0748833	8.43721	< 0.0001
				intercept	-19.7464	32.4067	-0.60933	0.543228

Copy

Source: Tableau, n.d., https://lc.cx/ukbSi0

The next step is to combine cluster analysis with our trend line analysis to reveal interesting patterns in our data. We identified five distinct clusters when examining the relationship between HSR and DSL measurements. Of particular interest are 6 data points that fall into two clusters of 3 points each. These 6 points are at a notable distance from the distribution of the rest of our data (146 data points). This deviation suggests that for identical HSR values, some DSL measurements are recording significantly higher than expected. There could be several factors contributing to these elevated DSL readings.

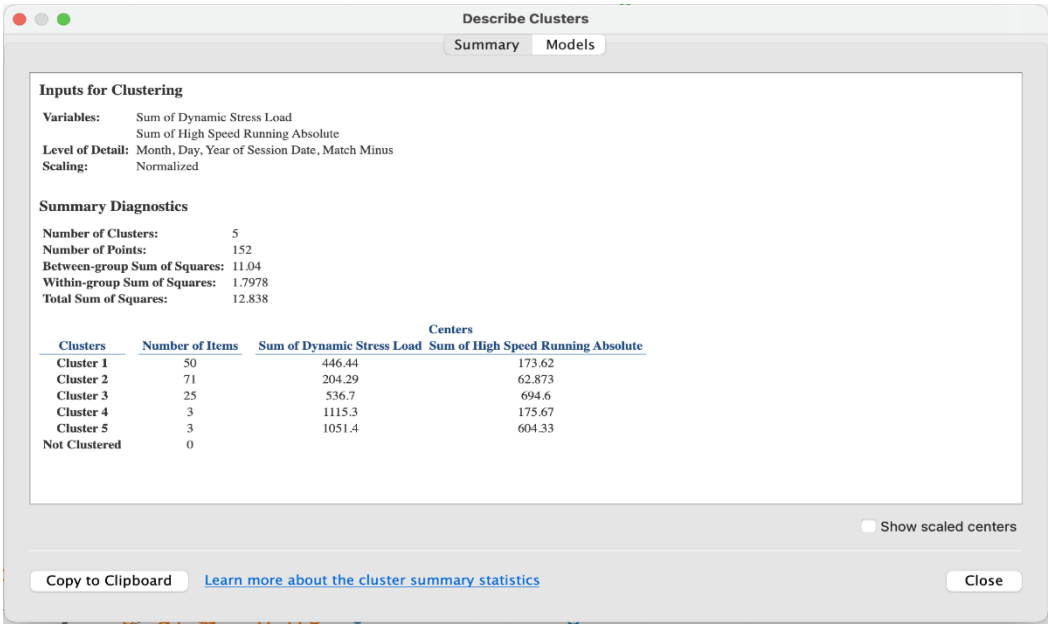
Figure 21. Combine cluster analysis with our trend line analysis



Source: own elaboration.

Here is a description of how Tableau builds and explains its clustering model.

Figure 22. How Tableau builds and explains its clustering model



Source: screenshot of Tableau.

Answer:

The fundamental relationship is clear: as High-Speed Running (HSR) distance increases, Dynamic Stress Load (DSL) typically increases. The trend line shows this and makes intuitive sense—the more high-speed running an athlete does, the more stress they experience.

However, the cluster analysis reveals something more interesting. By dividing the data into five distinct clusters, we can identify what's "normal" versus "unusual" for our athletes. Clusters no. 4 and 5 particularly stand out: instances where players recorded a DSL over 900 while only completing around 400m of HSR.

This is significant because:

- it shows an unusually high-stress load for that amount of running,
- it deviates from the typical relationship between HSR and DSL,
- it might indicate potential issues with how the athlete is moving or responding to the training load.

These outliers could signal:

- unusual movement patterns,
- fatigue affecting running efficiency,
- terrain or drill-specific impacts.

This insight is valuable for medical staff and coaches because it helps identify situations where athletes might be experiencing excessive stress despite not necessarily doing more high-speed running. It's a perfect discussion point for medical meetings, allowing the team to investigate:

- What was the difference between these training sessions?
- How did the athlete feel during and after these sessions?
- Should training be modified to prevent such high-stress loads?

Conclusion

This module taught us about Principal Component Analysis (PCA) and reinforcement learning. Combined with supervised and unsupervised learning from the previous module, these are some of the most powerful machine learning techniques used in data analysis applications in sports and beyond. We've also briefly touched on advanced techniques like Neural Networks and Deep Learning.

While most examples are in Python, we have also examined a detailed example of unsupervised learning—clustering—in Tableau.

References

Tableau. (n.d.). *Add Trend Lines to a Visualization*. https://help.tableau.com/current/pro/desktop/en-us/trendlines_add.htm#Linear

Further reading

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.

Graesser, L. and Keng, W. L. (2019). *Foundations of deep reinforcement learning: Theory and practice in Python*. Addison-Wesley Professional.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2° ed.). Springer.

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). *Mining of massive datasets* (3° ed.). Cambridge University Press.

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement learning: An introduction* (2° ed.). MIT Press.