# Module 1. Introduction to machine learning

BARÇA INNOVATION HUB
Universitas

Machine learning (ML) represents a paradigm shift in how we solve problems with computers. Rather than explicitly programming rules and solutions, ML systems learn patterns from data to make predictions or decisions. This chapter introduces the fundamental concepts, applications, and methodologies that form the foundation of machine learning. We will also look at some examples of applying ML in sports analytics.

# Unit 1.1 What is machine learning?

Machine learning is a subset of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed. The core idea is to develop algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

The theoretical foundation of ML rests on several key principles.

## 1. Statistical learning theory

- This forms the mathematical backbone of ML. It addresses how we can learn patterns from data in a statistically sound way. It helps us understand questions like:

    - How much data do we need to learn effectively?
    - How can we ensure our learning generalizes to new, unseen cases?
    - What is the relationship between training error and actual error?

- The bias-variance trade-off. Simple models may underfit (high bias), while complex models may overfit

(high variance). Finding the right balance is crucial.

## Examples

Imagine you're teaching a computer to recognize cats in photos. If you only show it pictures of white cats, it might learn that "cats must be white" (high bias/underfitting). Conversely, if your model is too complex and learns every whisker pattern it sees, it might fail on new cats because it's too specific (high variance/overfitting). Statistical learning theory helps us find the sweet spot.

**Email spam detection**

- Training data: 10 000 emails labeled as spam/not spam.
- Features: word frequencies, sender info, time sent.
- According to the theory:
  - with 10 000 diverse examples, we can reliably learn common spam patterns,
  - but the rarer spam patterns might need more examples,
  - we need to balance detecting spam (sensitivity) vs. avoiding false alarms (specificity).

## 2. Optimization theory

Optimization is a key component of ML. ML algorithms usually have the following characteristics.

- Define a loss function that measures how wrong our predictions are.
- Using optimization methods (like gradient descent) to minimize this loss.
- Iteratively adjust the model's parameters to find the best solution.

### Examples

Think of a blind hiker trying to find the lowest point in a valley. They can feel the slope under their feet and move downhill but might get stuck in a local depression.

**Training a neural network to predict house prices**

- The loss function is the Mean Squared Error (MSE) between predicted and actual prices.

- Gradient Descent: adjusting weights iteratively.

    - If the predicted price is $100 000, but the actual price is $125 000.
    - The algorithm adjusts weights to reduce this error.
    - It takes small steps to avoid overshooting the optimal solution

## 3. Information theory

Information theory helps us understand the answers to the following questions.

- How much information is contained in our data?
- What makes the features informative or redundant?
- How to measure uncertainty in our predictions?
- The principles behind compression and encoding of information.

**Example**

Consider language translation. Some words carry more information than others. "The" appears often, but tells us little, while "quantum" is rarer but more informative.

**Image compression**

- Original image: high-resolution photo (25MB).
- Information theory shows:
    - large areas of the sky can be stored efficiently (low information),
    - complex textures need more data (high information),
    - we can compress the image (the low-information parts) while preserving all the essential details.

## 4. Computational learning theory

Computational learning theory answers the key questions about “learnability”:

- What types of patterns can or cannot be learned?

- How many examples do we need to learn effectively?
- What are the computational resources required?

### Examples

Think of teaching a child to recognize shapes. How many triangles must they see before reliably identifying new ones?

**Example: credit card fraud detection**

Learning complexity

- Simple fraud patterns (large, unusual purchases) need only a few examples.
- Complex patterns (sophisticated fraud networks and rings) need more data.
- Minimum samples needed for reliable detection.
- Indicates when we need complex models.

CONTINUE

# Unit 1.2 Terms and definitions in ML

We have introduced some new terms in this module so far. ML has a lot of terms and definitions that you need to be familiar with. You will see a lot more new terms in the rest of this course. Here is a summary of the most important and frequently used ones.

### Model

A mathematical representation that learns patterns from data. Think of it as a function that takes inputs and produces outputs. For example, a house price prediction model might take square footage and location as inputs and output an estimated price.

### Data

The raw information from which the system learns. This could be numbers, text, images, or other structured or unstructured data.

### Labels

Labels are the answers or outcomes we want our model to predict based on the features.

### Training

The process of teaching a model to make predictions by showing examples.

### Parameters and hyperparameters

| | |
|---|---|
| Parameters | the values the model learns during training (like weights in a neural network). |

| | |
|---|---|
| Hyperparameters | configuration settings we choose before training (like learning rate or tree depth). |

### Features

The input variables used to make predictions are the individual measurable properties of the observed phenomenon. For example, in a house price prediction system, features might include square footage, number of bedrooms, and location.

In an email spam detector:

- word frequencies
- email length
- sender information

- time sent
- presence of specific phrases

### Loss/cost function

Measures how wrong the model's predictions are compared to actual values. Here are the most common loss functions.

- Mean Squared Error (MSE) for regression.
- Cross-entropy loss for classification.
- Hinge loss for support vector machines.

### Gradient descent

An optimization algorithm that iteratively adjusts parameters to minimize the loss function. It works through:

- calculating the gradient (direction of steepest increase),
- taking steps in the opposite direction,
- repeating until reaching a minimum.

### Training, validation and test sets

| | |
|---|---|
| Training set | data used by the ML algorithm/model to learn patterns in the data. |

| | |
|---|---|
| Validation set | data used to tune hyperparameters. |
| Test set | data kept separate to evaluate the final performance of the model. |

## Overfitting and underfitting

**Overfitting** _

the model learns noise in training data and performs poorly on new data.

**Underfitting** _

the model is too simple to capture essential patterns.

## Regularization

Regularization is the process of preventing overfitting by the model.

| | |
|---|---|
| L1 (Lasso) | this regression can zero out irrelevant features. |

| | |
|---|---|
| L2 (Ridge) | ridge regression prevents weights from becoming too large. |
| Dropout | randomly deactivates neurons in neural networks. |

## Bias and variance

- Bias is the error from oversimplified assumptions.

- Variance is the error from sensitivity to small fluctuations in training data.
- The bias-variance tradeoff is fundamental to model selection.

### Cross-validation

Cross-validation is a technique to assess model performance by:

- dividing data into k folds,
- training on k-1 folds,
- testing on the remaining fold,
- repeating k times.

### Generalization

The model's ability to perform well on new, unseen data. Good generalization means:

- captures accurate underlying patterns,
- isn't memorizing training data,

- performs consistently across different datasets.

### Model evaluation

Metrics different measures of performance.

| |
|---|
| **Accuracy** _<br><br>proportion of correct predictions. |
| **Precision** _<br><br>true positives / (true positives + false positives). |
| **Recall** _<br><br>true positives / (true positives + false negatives). |
| **F1 Score** _<br><br>harmonic mean of precision and recall. |

**ROC curve** —

plots actual positive rate vs. false positive rate.

### Dimensionality reduction

Techniques to reduce the number of features while preserving important information.

- Principal component analysis (PCA)
- t-SNE
- Autoencoders

### Ensemble methods

Combining multiple models to improve performance.

| | |
|---|---|
| Random forests | multiple decision trees. |
| Boosting | sequential training of weak learners. |

| | |
|---|---|
| Bagging | parallel training on bootstrap samples. |

CONTINUE

# Unit 1.3 Types of machine learning

There are three broad types of machine learning.

1. Supervised learning.
2. Unsupervised learning.
3. Reinforced learning.

Other types of ML, like deep learning, don't fall under this classification. We will touch upon them briefly in the next module.

We will detail supervised and unsupervised learning in this module with walkthroughs using examples from sports.

We will explore reinforcement learning in the next module.

### 1.3.1 Supervised learning

Supervised learning is a machine learning paradigm where the algorithm learns from labeled training data to predict new, unseen data. The term "supervised" comes from the idea that the algorithm learns under supervision, with correct answers (labels) provided during training. There are two main types:

| | |
|---|---|
| Classification | predicts discrete categories (e.g., spam vs. not spam email, identifying species of flowers). |

| | |
|---|---|
| Regression | predicts continuous values (e.g., house prices, temperature forecasting). |

**Framework**

- Input space X: features (also known as predictors).
- Output space Y: target variables or labels.
- Training data D.
- Hypothesis H: set of possible functions mapping X to Y.
- Objective/goal: find the function $h \in H$ that minimizes error on unseen data.
- Loss functions.

- Regression: mean square error (MSE), mean absolute error (MAE).
- Classification: cross-entropy loss, hinge loss.

## Typical applications of supervised learning in sports

- Predicting player statistics.
- Player recruitment.
- Transfer market valuation.
- Win/loss probability.
- Point spread forecasting.
- Talent ID.
- Play calling prediction.

**Example**

We will use supervised learning to predict the expected points scored by players in an NBA match. The model uses gradient boost regression. We will explain the code block by block, followed by an analysis of the results.

First, we import all the specific packages needed for this analysis and then we generate random basketball data.

**Figure 1. Import all the necessary specific packages**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import matplotlib.pyplot as plt
import seaborn as sns
```

Source: own elaboration.

**Figure 2. Generate random basketball data**

```python
def generate_sample_data(n_samples=1000):
    """Generate realistic sample NBA player data"""
    np.random.seed(42)

    data = {
        'minutes_per_game': np.random.normal(25, 5, n_samples),
        'field_goal_percentage': np.random.normal(45, 5, n_samples),
        'three_point_percentage': np.random.normal(35, 5, n_samples),
        'free_throw_percentage': np.random.normal(75, 8, n_samples),
        'rebounds_per_game': np.random.normal(5, 2, n_samples),
        'assists_per_game': np.random.normal(3, 2, n_samples),
        'previous_season_ppg': np.random.normal(12, 5, n_samples),
        'games_played': np.random.normal(70, 10, n_samples)
    }

    df = pd.DataFrame(data)

    # Generate target (points per game) with realistic relationships
    df['points_per_game'] = (
        0.4 * df['minutes_per_game'] +
        0.3 * df['field_goal_percentage'] / 100 * df['minutes_per_game'] +
        0.2 * df['previous_season_ppg'] +
        0.1 * df['free_throw_percentage'] / 100 * df['minutes_per_game'] +
        np.random.normal(0, 2, n_samples)
    )

    return df
```

Source: own elaboration.

---

## Prepare data

- Specify the feature space X.
- Target variable y = points_per_game.
- Return data split into training and test sets. The training set is 80% of the dataset. The test set is 20% of the dataset.

**Figure 3. Prepare data**

```python
def prepare_data(df):
    """Prepare data for training"""
    features = [
        'minutes_per_game', 'field_goal_percentage', 'three_point_percentage',
        'free_throw_percentage', 'rebounds_per_game', 'assists_per_game',
        'previous_season_ppg', 'games_played'
    ]

    X = df[features]
    y = df['points_per_game']

    return train_test_split(X, y, test_size=0.2, random_state=42)
```

Source: own elaboration.

## Train the model

- We are using gradient descent regression (gradient boost regressor).

**Figure 4. Train the model**

```python
def train_model(X_train, y_train):
    """Train the model and return scaler, model, and cross-validation scores"""
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)

    model = GradientBoostingRegressor(
        n_estimators=100,
        learning_rate=0.1,
        max_depth=4,
        random_state=42
    )

    model.fit(X_train_scaled, y_train)

    cv_scores = cross_val_score(
        model, X_train_scaled, y_train,
        cv=5, scoring='r2'
    )

    return scaler, model, cv_scores
```

Source: own elaboration.

## Evaluate the model

- Calculate R-squared, RMSE, and MA.

## Figure 5. Evaluate the model

```python
def evaluate_model(model, scaler, X_test, y_test):
    """Evaluate model performance"""
    X_test_scaled = scaler.transform(X_test)
    y_pred = model.predict(X_test_scaled)

    metrics = {
        'r2_score': r2_score(y_test, y_pred),
        'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
        'mae': mean_absolute_error(y_test, y_pred)
    }

    return metrics, y_pred
```

Source: own elaboration.

## Figure 6. Create different visualizations

```python
def create_visualizations(model, X_test, y_test, y_pred):
    """Create and display all visualizations"""
    # 1. Actual vs Predicted Plot
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
             'r--', lw=2)
    plt.xlabel('Actual Points Per Game')
    plt.ylabel('Predicted Points Per Game')
    plt.title('NBA Player Points Prediction: Actual vs Predicted')
    plt.tight_layout()
    plt.show()

    # 2. Feature Importance Plot
    feature_importance = pd.DataFrame({
        'feature': X_test.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=True)

    plt.figure(figsize=(10, 6))
    plt.barh(range(len(feature_importance)), feature_importance['importance'])
    plt.yticks(range(len(feature_importance)), feature_importance['feature'])
    plt.xlabel('Feature Importance')
    plt.title('NBA Player Points Prediction: Feature Importance')
    plt.tight_layout()
    plt.show()
    # 3. Prediction Distribution
    plt.figure(figsize=(10, 6))
    plt.hist([y_test, y_pred], label=['Actual', 'Predicted'],
             bins=20, alpha=0.5)
    plt.xlabel('Points Per Game')
    plt.ylabel('Frequency')
    plt.title('Distribution of Actual vs Predicted Points')
    plt.legend()
    plt.tight_layout()
    plt.show()
     # 4. Error Distribution
    errors = y_test - y_pred
    plt.figure(figsize=(10, 6))
    sns.histplot(errors, kde=True)
    plt.xlabel('Prediction Error')
    plt.ylabel('Frequency')
    plt.title('Distribution of Prediction Errors')
    plt.axvline(x=0, color='r', linestyle='--', label='Zero Error')
    plt.legend()
    plt.tight_layout()
    plt.show()

    # 5. Error vs Predicted
    plt.figure(figsize=(10, 6))
    plt.scatter(y_pred, errors, alpha=0.5)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Predicted Points Per Game')
    plt.ylabel('Prediction Error')
    plt.title('Prediction Error vs Predicted Value')
    plt.tight_layout()
    plt.show()
```

Source: own elaboration.

**Figure 7. Analyze the model**

```python
def analyze_model(model, scaler, X_train, X_test, y_train, y_test, y_pred):
    """Analyze model performance in detail"""
    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
    from scipy import stats

    # Basic metrics
    metrics = {
        'R-squared': r2_score(y_test, y_pred),
        'RMSE': np.sqrt(mean_squared_error(y_test, y_pred)),
        'MAE': mean_absolute_error(y_test, y_pred),
        'Training samples': X_train.shape[0],
        'Test samples': X_test.shape[0]
    }
        # Feature importance analysis
    feature_importance = pd.DataFrame({
        'feature': X_test.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)

    # Prediction error analysis
    errors = y_test - y_pred
    error_stats = {
        'Mean Error': errors.mean(),
        'Error Std Dev': errors.std(),
        'Error Skewness': stats.skew(errors),
        'Error Kurtosis': stats.kurtosis(errors)
    }
        # Residual normality test
    _, normality_p_value = stats.normaltest(errors)
    # Performance at different prediction ranges
    y_ranges = pd.qcut(y_test, q=4)
    range_performance = pd.DataFrame({
        'Actual': y_test,
        'Predicted': y_pred,
        'Range': y_ranges
    }).groupby('Range').agg({
        'Actual': ['mean', 'count'],
        'Predicted': ['mean', 'std']
    })
        # Feature correlation analysis
    feature_correlations = X_test.corrwith(pd.Series(y_test)).sort_values(ascending=False)

    # Print comprehensive analysis
    print("\n=== Model Performance Metrics ===")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.3f}")

    print("\n=== Feature Importance ===")
    for idx, row in feature_importance.iterrows():
        print(f"{row['feature']}: {row['importance']:.3f}")

    print("\n=== Error Analysis ===")
    for stat, value in error_stats.items():
        print(f"{stat}: {value:.3f}")
    print(f"Error Normality p-value: {normality_p_value:.3f}")

    return metrics, feature_importance, error_stats, range_performance, feature_correlations
```

Source: own elaboration.

## Model output plots

Top left

Correlation between Predicted PPG and Actual PPG.



Top Right

Feature importance. Minutes per game is by far the most important feature.

Bottom left

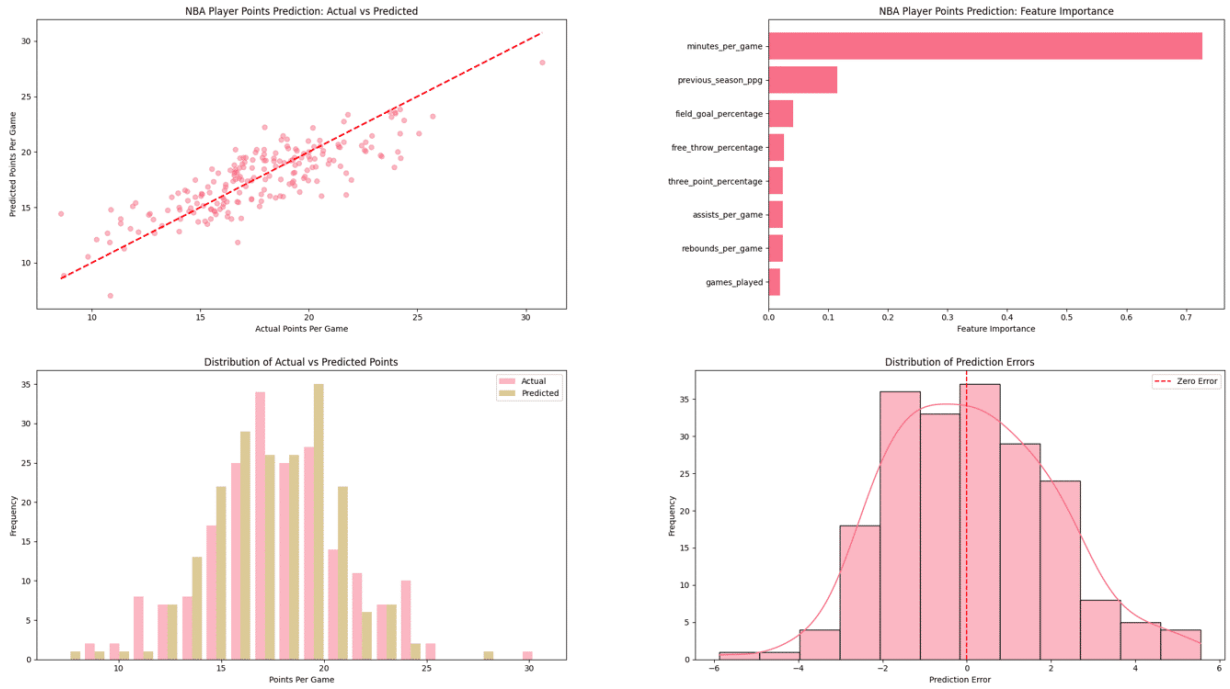
Actual vs predicted points at different points per game.



Bottom right

Frequency histogram of prediction errors, with a dotted red line indicating “zero error”.

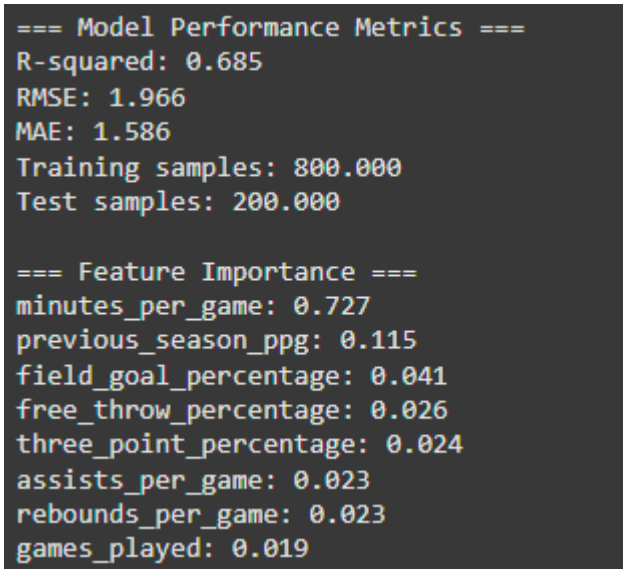
## Figure 8. Model output plots




Source: own elaboration.

---

## Figure 9. Model characteristics




Source: own elaboration.

## Model interpretation

- $R^2$ of 0.685 indicates the model has moderate predictive power.
- RMSE of ~2.0 points = reasonable accuracy.
- Minutes per game dominates the model (72.7% importance, 0.8 correlation).
- The model under-predicts scores above 20+ and over-predicts low scores (below 15.6).
- The error distribution is nearly “normal”.

This experimental model is suitable for baseline predictions but has room for improvement, particularly for players outside typical scoring ranges.

### Unsupervised learning

Unsupervised learning is a branch of machine learning where algorithms find patterns and structures in unlabeled data. Unlike supervised learning, there are no predefined correct answers - the algorithm discovers inherent groupings and relationships within the data. It's like finding structure in data without a teacher. Key approaches include the following.

- Clustering: groups similar data points together (e.g., customer segmentation, image compression). Below are some of the most widely used clustering methods.
    - K-means
    - Hierarchical
    - DBSCAN
    - Spectral clustering
- Dimensionality reduction: reduces data complexity while preserving essential patterns. A couple of the

most widely applied dimensionality reduction techniques are:

- principal component analysis (PCA),
- t-SNE (t-Distributed Stochastic Neighbor Embedding).

- Anomaly detection: identifies unusual patterns or outliers in data.

## Framework

1. Pattern recognition

    1. Identify natural groups within the data.
    2. Discover relationships without prior labels.

2. Dimensionality

    1. Identify feature space relationships.
    2. Apply dimensionality reduction techniques to reduce the dimensions in data.

3

Distance metrics: different ways to measure the "distance" between two points in an n-dimensional space. In this case, you may treat this as "How far are two neighbors in a sample space? This distance determines whether two points (players or entities) belong to the same group. A couple of the most used distance metrics:

1. Euclidean distance
2. Manhattan distance

## Applications of unsupervised learning in sports

These are a few common scenarios where unsupervised learning is significantly applied in sports data.

- Identifying player types and styles.
- Team playing style clustering.
- Discover similar players and player comparison.
- Movement pattern recognition.

**Example: clustering American Football (NFL) players focusing on offensive skill positions**

Quarterback (QB), running back (RB), wide receiver (WR), and tight end (TE) are offensive skill positions in American Football. In this example, we will focus on QB, WR and RB positions, as well as players who are dual threats, WRs that run, RBs that catch, mobile QBs who can also run.

We generate sample data of the following metrics for players across these three positions.

- Passing yards
- Rushing yards
- Receiving yards
- Total touchdowns
- Yards per play

**Figure 10. Step 1: import all the necessary packages**

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

Source: own elaboration.

## Figure 11. Step 2: generate sample data

```python
def generate_nfl_data(n_players=200):
    """Generate more realistic NFL player statistics"""
    np.random.seed(42)

    # Define player type distributions
    n_elite_qb = int(n_players * 0.1)     # 10% Elite QBs
    n_avg_qb = int(n_players * 0.15)      # 15% Average QBs
    n_elite_rb = int(n_players * 0.15)    # 15% Elite RBs
    n_avg_rb = int(n_players * 0.15)      # 15% Average RBs
    n_elite_wr = int(n_players * 0.2)     # 20% Elite WRs
    n_avg_wr = n_players - n_elite_qb - n_avg_qb - n_elite_rb - n_avg_rb - n_elite_wr  # Remaining WRs

    # Generate player statistics
    players = {
        'Player_ID': range(n_players),
        'Passing_Yards': np.concatenate([
            np.random.normal(4500, 300, n_elite_qb),     # Elite QBs
            np.random.normal(3200, 250, n_avg_qb),       # Average QBs
            np.zeros(n_elite_rb + n_avg_rb + n_elite_wr + n_avg_wr)  # Non-QBs
        ]),
        'Rushing_Yards': np.concatenate([
            np.random.normal(300, 50, n_elite_qb),       # Elite QBs (mobile)
            np.random.normal(150, 30, n_avg_qb),         # Average QBs
            np.random.normal(1300, 150, n_elite_rb),     # Elite RBs
            np.random.normal(600, 100, n_avg_rb),        # Average RBs
            np.random.normal(100, 30, n_elite_wr),       # Elite WRs
            np.random.normal(50, 20, n_avg_wr)           # Average WRs
        ]),
```

```
    'Receiving_Yards': np.concatenate([
        np.zeros(n_elite_qb + n_avg_qb),            # QBs
        np.random.normal(400, 50, n_elite_rb),       # Elite RBs
        np.random.normal(200, 30, n_avg_rb),         # Average RBs
        np.random.normal(1200, 150, n_elite_wr),     # Elite WRs
        np.random.normal(700, 100, n_avg_wr)         # Average WRs
    ]),
    'Total_TDs': np.concatenate([
        np.random.normal(38, 5, n_elite_qb),         # Elite QBs
        np.random.normal(22, 4, n_avg_qb),           # Average QBs
        np.random.normal(12, 2, n_elite_rb),         # Elite RBs
        np.random.normal(6, 1, n_avg_rb),            # Average RBs
        np.random.normal(10, 2, n_elite_wr),         # Elite WRs
        np.random.normal(5, 1, n_avg_wr)             # Average WRs
    ]),
    'Yards_Per_Touch': np.concatenate([
        np.random.normal(7.5, 0.5, n_elite_qb),      # Elite QBs
        np.random.normal(6.5, 0.5, n_avg_qb),        # Average QBs
        np.random.normal(5.0, 0.3, n_elite_rb),      # Elite RBs
        np.random.normal(4.2, 0.3, n_avg_rb),        # Average RBs
        np.random.normal(12.5, 1.0, n_elite_wr),     # Elite WRs
        np.random.normal(10.5, 1.0, n_avg_wr)        # Average WRs
    ]),
    'Total_Touches': np.concatenate([
        np.random.normal(600, 50, n_elite_qb),       # Elite QBs (passes + rushes)
        np.random.normal(500, 40, n_avg_qb),         # Average QBs
        np.random.normal(280, 30, n_elite_rb),       # Elite RBs
        np.random.normal(180, 20, n_avg_rb),         # Average RBs
        np.random.normal(100, 15, n_elite_wr),       # Elite WRs
        np.random.normal(70, 10, n_avg_wr)           # Average WRs
    ])
}

df = pd.DataFrame(players)

# Clean up any negative values
for col in df.columns:
    if col != 'Player_ID':
        df[col] = df[col].clip(lower=0)

return df
```

Source: own elaboration.

---

In the above section, we created sample data of all the metrics we use for the players that we will use in the analysis.

### Analyze clusters

This section of the code analyzes the different clusters and examines the ranges of other metrics for each cluster. It returns the means of all the metrics for each cluster (cluster_means).

We also create various scatter plots (which we analyze in the output section).

**Figure 12. Analyze clusters**

```python
def analyze_clusters(df, labels):
    """Create detailed visualizations of clusters"""
    df_analysis = df.copy()
    df_analysis['Cluster'] = labels

    # Calculate cluster means
    cluster_means = df_analysis.groupby('Cluster').mean()

    # Create visualizations
    plt.figure(figsize=(20, 15))

    # 1. Passing vs Rushing Yards
    plt.subplot(3, 2, 1)
    scatter = plt.scatter(df['Passing_Yards'], df['Rushing_Yards'],
                          c=labels, cmap='viridis', alpha=0.6)
    plt.xlabel('Passing Yards')
    plt.ylabel('Rushing Yards')
    plt.title('Player Clusters: Passing vs Rushing')
    plt.colorbar(scatter, label='Cluster')

    # 2. Receiving vs Rushing Yards
    plt.subplot(3, 2, 2)
    scatter = plt.scatter(df['Receiving_Yards'], df['Rushing_Yards'],
                          c=labels, cmap='viridis', alpha=0.6)
    plt.xlabel('Receiving Yards')
    plt.ylabel('Rushing Yards')
    plt.title('Player Clusters: Receiving vs Rushing')
    plt.colorbar(scatter, label='Cluster')

    # 3. Yards Per Touch vs Total Touches
    plt.subplot(3, 2, 3)
    scatter = plt.scatter(df['Total_Touches'], df['Yards_Per_Touch'],
                          c=labels, cmap='viridis', alpha=0.6)
    plt.xlabel('Total Touches')
    plt.ylabel('Yards Per Touch')
    plt.title('Player Clusters: Efficiency vs Volume')
    plt.colorbar(scatter, label='Cluster')

    # 4. TDs Distribution
    plt.subplot(3, 2, 4)
    sns.boxplot(data=df_analysis, x='Cluster', y='Total_TDs')
    plt.title('Touchdown Distribution by Cluster')

    # 5. Heatmap of cluster centers
    plt.subplot(3, 2, (5, 6))
    sns.heatmap(cluster_means, annot=True, cmap='coolwarm', center=0, fmt='.1f')
    plt.title('Cluster Centers Heatmap')

    plt.tight_layout()
    plt.show()

    return cluster_means
```

Source: own elaboration.

**Figure 13. Interpret clusters**

```python
def interpret_clusters(cluster_means):
    """Provide detailed interpretation of each cluster"""
    interpretations = []

    for cluster in cluster_means.index:
        stats = cluster_means.loc[cluster]

        # More nuanced player type determination
        if stats['Passing_Yards'] > 3000:
            if stats['Rushing_Yards'] > 250:
                role = "Dual-Threat QB"
            else:
                role = "Pocket QB"
        elif stats['Rushing_Yards'] > 800:
            if stats['Receiving_Yards'] > 300:
                role = "Pass-Catching RB"
            else:
                role = "Power RB"
        elif stats['Receiving_Yards'] > 800:
            if stats['Yards_Per_Touch'] > 11:
                role = "Deep Threat WR"
            else:
                role = "Possession WR"
        else:
            role = "Utility Player"

        interpretations.append({
            'Cluster': cluster,
            'Predicted Role': role,
            'Key Characteristics': [
                f"Pass Yards: {stats['Passing_Yards']:.0f}",
                f"Rush Yards: {stats['Rushing_Yards']:.0f}",
                f"Rec Yards: {stats['Receiving_Yards']:.0f}",
                f"TDs: {stats['Total_TDs']:.1f}",
                f"Yards/Touch: {stats['Yards_Per_Touch']:.1f}",
                f"Touches: {stats['Total_Touches']:.0f}"
            ]
        })

    return pd.DataFrame(interpretations)
```

Source: own elaboration.

Based on the cluster, we interpret and label the different clusters.

Main function to run the code:

- Generate data.
- Find the optimal number of clusters.
- Analyze and interpret clusters.

**Figure 14. Interpret and label the different clusters**

```python
def main():
    # Generate and prepare data
    df = generate_nfl_data()

    # Scale features
    scaler = StandardScaler()
    features = ['Passing_Yards', 'Rushing_Yards', 'Receiving_Yards',
                'Total_TDs', 'Yards_Per_Touch', 'Total_Touches']
    scaled_data = scaler.fit_transform(df[features])

    # Find optimal clusters and perform clustering
    kmeans = KMeans(n_clusters=6, random_state=42)  # Using 6 clusters for main position groups
    labels = kmeans.fit_predict(scaled_data)

    # Analyze results
    cluster_means = analyze_clusters(df[features], labels)
    interpretations = interpret_clusters(cluster_means)

    print("\n=== Cluster Interpretations ===")
    print(interpretations)

    print("\n=== Detailed Statistical Profiles ===")
    print(cluster_means.round(1))

if __name__ == "__main__":
    main()
```

Source: own elaboration.

## Output

**Figure 15. Data clusters**

| Cluster | Predicted Role |
|---|---|
| 0 | Pass-Catching RB |
| 1 | Utility Player |
| 2 | Dual-Threat QB |
| 3 | Utility Player |
| 4 | Deep Threat WR |
| 5 | Pocket QB |

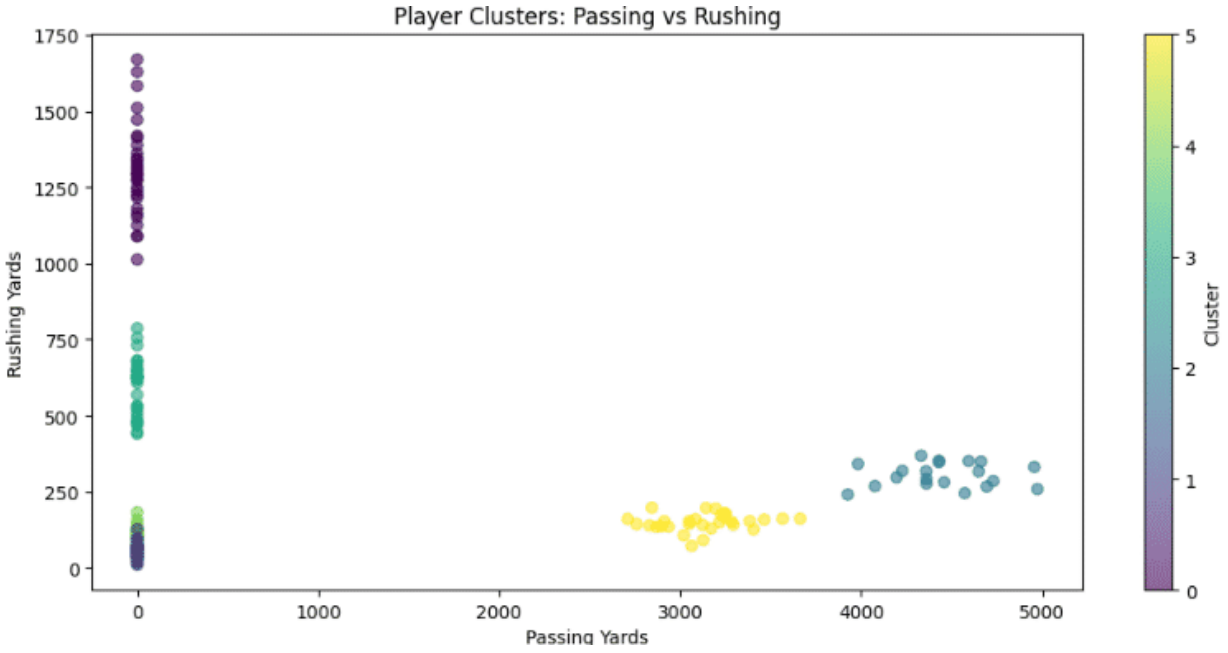
Source: own elaboration.

---

The data has 5 clusters. We have labeled each cluster with a specific archetype of NFL player.

## Passing yards vs. rushing yards

**Figure 16. Passing yards vs. rushing yards**




Source: own elaboration.

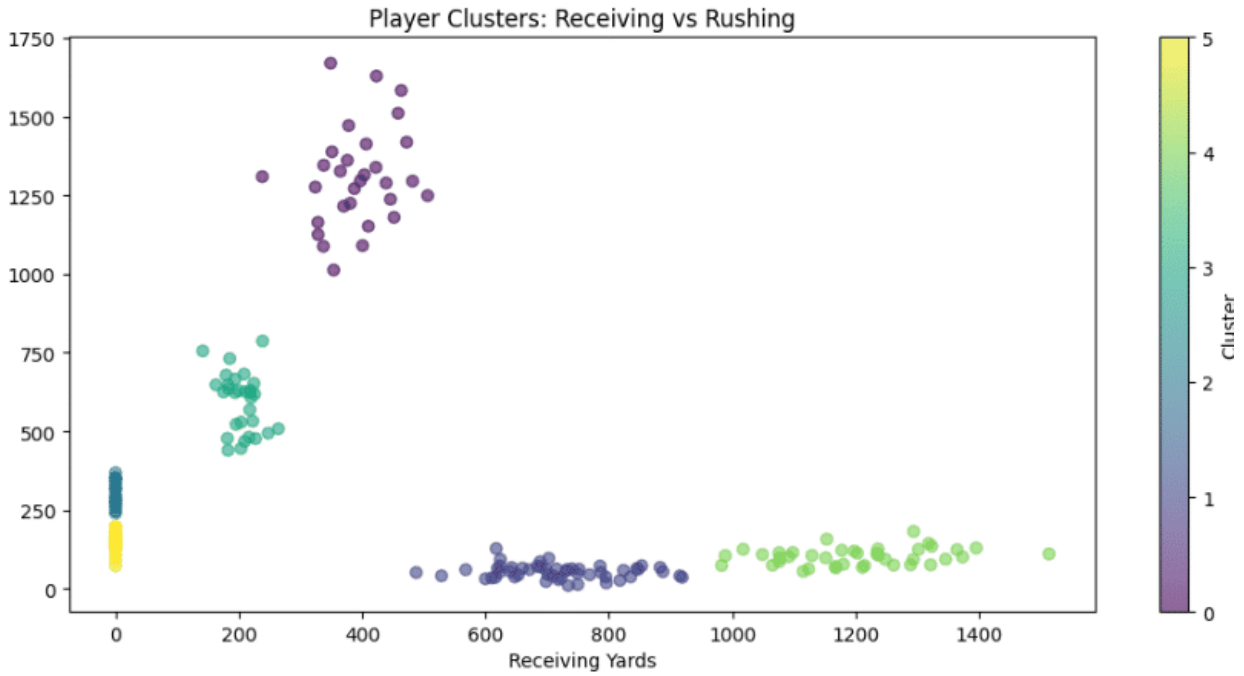
Cluster 1, 3, 4 are non-QBs. They have 0 yards passing. Clusters 2 and 5 are QBs. 2 is mobile QBs who accumulate more passing yards than those in cluster 5, who are primarily pocket passers.

### Receiving yards vs. rushing yards

QBs in 2 and 5 have zero receiving yards.

Cluster 0 – Passing-catching RBs accumulate yards through receiving and rushing.

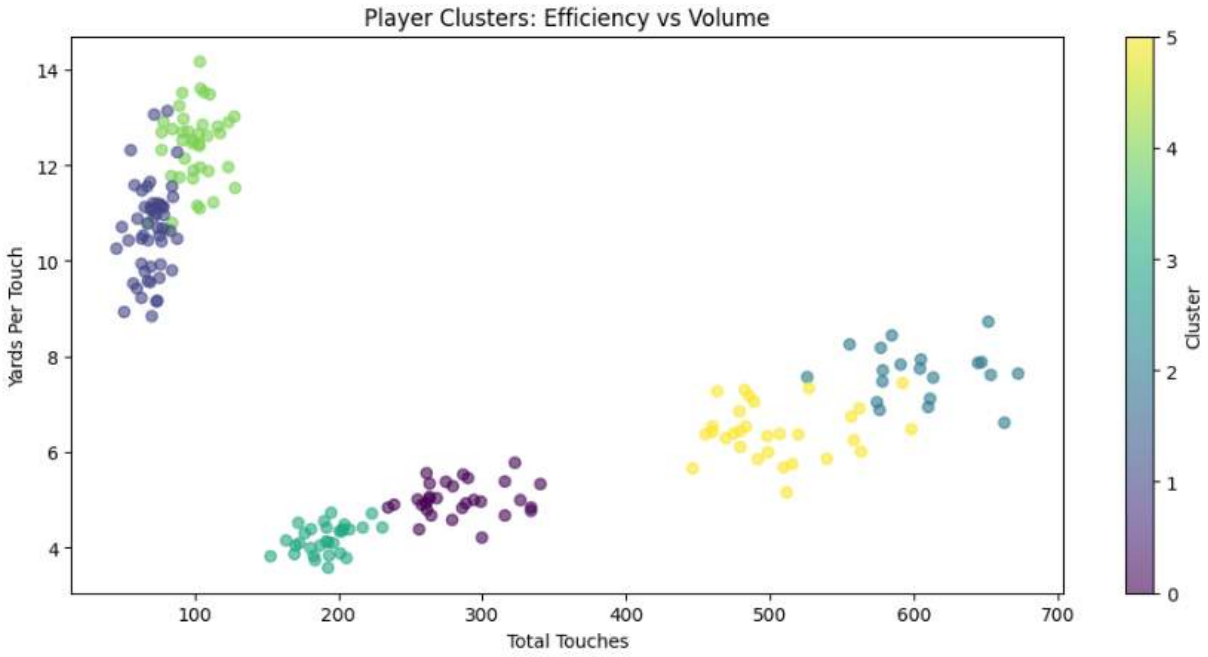
**Figure 17. Receiving yards vs. rushing yards**




Source: own elaboration.

## Efficiency vs. volume

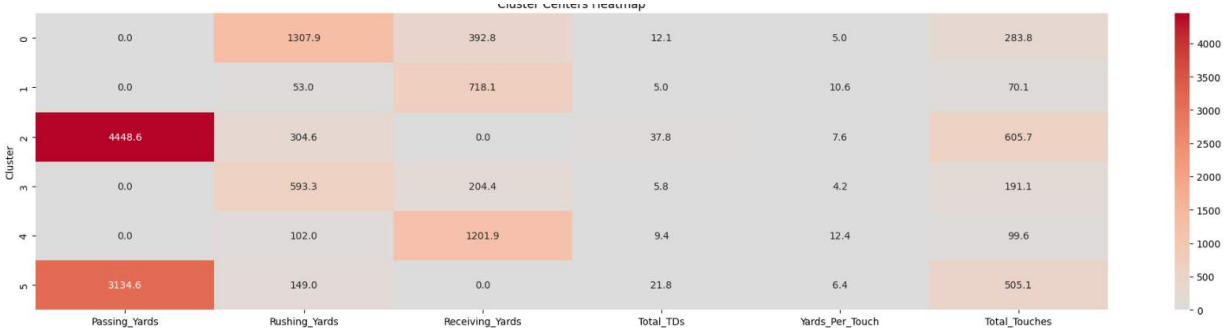
**Figure 18. Efficiency vs. volume**




Source: own elaboration.

Finally, we publish the **cluster centers** (or means) of all the metrics for all five clusters.

**Figure 19. Publish the cluster centers**



Source: own elaboration.

## Interpretation

1 **Cluster 2 - Dual-threat QB:**

- Best performing QBs with 4,449 passing yards.
- Significant rushing ability (305 yards).
- High TD production (37.8 TDs).
- High volume (606 touches).
- Represents elite QBs like Josh Allen, Patrick Mahomes.

2 **Cluster 5 - Pocket QB:**

- Traditional QBs with 3,135 passing yards.
- Limited rushing (149 yards).
- Good TD production (21.8 TDs).
- High volume (505 touches).
- Represents traditional QBs like Kirk Cousins.

3 **Cluster 0 - Pass-catching RB:**

- Strong rushing (1,308 yards).
- Good receiving (393 yards).
- Solid TD production (12.1 TDs).
- High volume (284 touches).
- Represents backs like Christian McCaffrey, Austin Ekeler.

4 **Cluster 3 - Power RB:**

- Moderate rushing (593 yards).
- Limited receiving (204 yards).
- Lower TD production (5.8 TDs).
- Moderate volume (191 touches).
- Represents rotational/backup RBs.

5 **Cluster 4 - Deep threat WR:**

- Significant receiving (1,202 yards).
- Highest yards per touch (12.4).
- Good TD production (9.4 TDs).
- Efficient with touches (100 touches).
- Represents receivers like Tyreek Hill, Justin Jefferson.

6

**Cluster 1 - Possession WR:**

- Good receiving (718 yards).
- Efficient (10.6 yards/touch).
- Lower TD production (5.0 TDs).
- Limited volume (70 touches).

## Overall summary

- Clear separation between QBs (Clusters 2 and 5).
- Distinct RB profiles (Clusters 0 and 3).
- Different WR types (Clusters 1 and 4).
- Volume vs. Efficiency trade-offs visible.
- TD production aligns with role expectations.

The model has identified the main NFL offensive player archetypes and their characteristic statistical profiles.

### Conclusion

In this module, we learned the basics and the theory behind the fascinating world of machine learning. We have also looked at two of the most widely used ML techniques, supervised and unsupervised learning, with detailed examples in sports.

## Further reading

**Naranjo-Campos, F., Victores, J., and Balaguer, C.** (2024). Expert-Trajectory-Based Features for Apprenticeship Learning via Inverse Reinforcement Learning for Robotic Manipulation. *Applied Sciences, 14*(23), 11131.

**Software Development Solutions.** (n.d.). *Revolutionizing Industries with Machine Learning: Unveiling Cutting Edge Applications.* https://www.software-development-solutions.com/post/revolutionizing-industries-with-machine-learning-unveiling-cutting-edge-applications

**Tech Target.** (n.d.). *What is Machine Learning.* SIG.Org. https://sig.org/what-machine-learning

CONTINUE

# References

**Aughrey, R**. (2011). Applications of GPS Technologies to Field Sports. International Journal of Sports Physiology and Performance, 6, 295-310. PMID: 21911856

**Barnes, C., Archer, D. T., Hogg, B., Bush, M., & Bradley, P. S**. (2014). The Evolution of Physical and Technical Performance Parameters in the English Premier League. International Journal of Sports Medicine, 35(13), 1095-1100. doi: http://dx.doi.org/10.1055/s-0034-1375695

**Gabbett, T.** (2013). Influence of playing standard on the physical demands of professional rugby league. Journal of Sports Sciences, 31(10), 1125-1138. doi: 10.1080/02640414.2013.773401

CONTINUE